



Data Analytics

▼ Lezione 1 - 04/03/2024

Bisogna sempre utilizzare gli strumenti in maniera critica, cioè curare il metodo, il ragionamento
Le tecnologie sono importanti, ma non quanto il metodo

Fare data analytics serve a ricavare delle informazioni utili a supportare chi ha il dominio di quei dati (supportare chi deve prendere decisioni)

I DATI

Il dato digitale si è diffuso in maniera esponenziale negli ultimi due decenni

Il dato viene prodotto da qualsiasi attività della vita quotidiana

Esempio: voti esami universitari

In passato solo la segreteria studenti poteva lavorare con dati digitali perché veniva utilizzato il libretto cartaceo, quindi né i professori né gli studenti erano attori del sistema

Invece, ora è il professore che verbalizza i voti e può visualizzare tutta la storia delle sedute di esame che ha sostenuto

Una situazione analoga si ha anche per lo studente, il quale può consultare il proprio libretto digitale senza passare per la segreteria

I dati memorizzati in questo modo sono quelli più fruibili e più puliti

Tuttavia, fare data analytics vuole dire che spesso ci si trova davanti a dati che sono sporchi e difficili da trattare

L'analisi dei dati è utile perché si possono ricavare informazioni che permettono di migliorare le decisioni da prendere nella realtà

I dati digitali rappresentano un'approssimazione della realtà

In generale, il dato riguarda qualcosa che è già successo

Per questo motivo, utilizziamo i dati della storia per migliorare l'approccio al futuro

I dati **NON** sono solo quelli che produciamo in prima persona in un sistema chiuso

Esempio sistema chiuso: esse3 a cui accediamo con username e password

I dati vengono prodotti anche in sistemi aperti, cioè in maniera **NON controllata**

Ad **esempio**, le recensioni sono dati che possono andare in ingresso ad un sistema di data analytics.

Nel caso della moda, si analizzano i trend. Infatti, oggi quando si lancia un capo viene subito fatta l'analisi di tendenza. Ciò permette di non produrre in massa un capo prima di lanciarlo sul mercato, ma di avviare la produzione solo se si è accertato il buon andamento del trend

Il settore logistico è quello più in crescita grazie all'analisi dei dati, dato che è il settore dove circola una quantità di dati enorme

Esempio caso medico sanitario: è molto importante l'analisi di questi tipi di dati a causa della molteplicità dei dati dato che ci sono dati acquisiti in maniera precisissima (analisi, emocromo), mentre ci sono campi come il "diario" che sono di difficile comprensione. Inoltre, oggi esistono anche le recensioni per medici

I **dati** in sé per sé **NON hanno significato**

Esempio: 38,5 sul termometro

38,5 è un numero intero che di per sé non significa nulla

Dai dati, si estrae un'**informazione quando ad essi applichiamo la nostra la conoscenza** pregressa sui fenomeni

La **conoscenza** è ciò che **estraiamo dall'informazione mediante elaborazione e ragionamenti**

Esempio: sappiamo che se la temperatura corporea è più alta di 37° allora è febbre

Analizziamo i dati per **estrarre regole** da un contesto **NON** noto

CONOSCENZA = REGOLA

Una volta estratta una regola, bisogna chiedersi se effettivamente è utile per quell'applicazione
Ciò si capisce grazie alla padronanza che si ha del dominio applicativo

Grazie all'analisi dei dati possiamo osservare e misurare fenomeni della realtà

Un **primo passo** da fare per comprendere un fenomeno è quello di applicare la **statistica descrittiva**

La **statistica descrittiva** è il modo più semplice per conoscere il dato, quindi per posizionarsi nel contesto del dominio applicativo

Grazie all'analisi dei dati, siamo in grado di risolvere vari problemi (favorire la ricerca scientifica, previsione eclissi, trovare il biglietto più economico, prevedere il churn)

Per analizzare i dati ci sono **due metodi** possibili

1. bisogna **capire** fino in fondo il dominio applicativo (metodo Binotto)
2. si raccolgono dati e tramite AI si ricavano regole, "**senza capire**" (**mg**); ciò che si ottiene è una decisione **black box**, cioè il passato ha messo in evidenza un certo pattern e **NON** si "perde tempo" ad osservare il fenomeno. Però per quest'approccio serve una **notevole quantità di dati**

BIG DATA

Le principali caratteristiche che distinguono i big data sono:

- **ampiezza** della conoscenza sulla realtà osservata (**esempio**: genoma umano)
- **profondità** della conoscenza
- **tempo** (**esempio**: come cambia nel tempo parte del genoma di una persona, utile per capire l'evoluzione di malattie neurodegenerative)

▼ Lezione 2 - 07/03/2024

Oggi Python, con Chiara

Non c'è soluzione univoca il gioco è trovare la soluzione migliore per noi

Python è sia imperativo (prevede l'esecuzione di istruzioni una dietro l'altra senza salti, procedurali) sia funzionale (codice strutturato, funzioni)

È molto flessibile, infatti permette di fare anche OO

Prevede una tipizzazione dinamica, non si indica il tipo di una variabile, però ciò è un'insidia

Succede un disastro se in una variabile int ci finisce una string

È un linguaggio pseudo compilato

Lo script .py viene trasformato dal bytecode compiler in un file .pyc, in pratica in linguaggio c, e poi si compila di nuovo

L'interprete collega le funzioni di libreria nello script scritto da noi

L'interprete viene costruito a priori, cioè ce n'è uno solo per tutti gli script che facciamo

Il gestore di pacchetti anaconda permette di manipolare le librerie con una GUI

I for iterano su tutti gli elementi di un iterabile

Un iterabile può essere una lista di elementi o un range

È possibile aggiungere un comportamento else ad un ciclo while, si entra nel else appena non si verifica più la condizione del while

Python utilizza l'indentazione (1 tab = 4 spazi) e non le parentesi graffe per distinguere l'inizio e la fine di un blocco

La keyword `def` permette di definire una funzione

Python si incappa se usiamo camel case o se mettiamo nomi con l'iniziale maiuscolo

È possibile definire dei valori di default per gli argomenti delle funzioni, ma vanno sempre alla fine

Se c'è un parametro di default si può omettere quando si richiama

Python consente di definire piccole funzioni di una sola riga, dette lambda function

Queste però sono funzioni semplici, tipicamente una sola istruzione matematica

Una funzione dichiarata successivamente non può essere chiamata sopra perché è procedurale

Le stringhe vengono trattate come liste di carattere

Le stringhe sono immutabili

`String.digits` ci restituisce i numeri da 0 a 9 come se fossero string

Si può utilizzare `in` per capire se c'è un carattere nella stringa

Anche la tupla è oggetto immutabile

Può contenere oggetti eterogenei

Vediamo le liste

Le liste sono mutabili

Si definiscono con [] e i valori si separano con ,

Si può accedere agli elementi in entrambi i versi, ad esempio se si accede all'elemento -1 allora si accede all'ultimo elemento

Le liste si passano per riferimento e non per valore

Python permette list comprehension

Si possono anche mettere filtri nella comprehension

▼ Lezione 3 - 11/03/2024

DATA MINING

I passi da compiere per fare data analytics sono:

1. **dati**
2. **dati selezionati**
3. **dati processati**
4. **dati trasformati**
5. **pattern**
6. **conoscenza**

Per ottenere della **conoscenza** bisogna fare degli studi sui dati

Soltanamente i dati vengono acquisiti da **più sorgenti** (più database)

Esempio: studio che riguarda il flusso degli studenti iscritti ad Unisannio

Potrei utilizzare l'interno database dell'Unisannio oppure un database esterno sulla popolazione che ci permette di fare ricerche più esterne(provincia, distribuzione su regione)

Le **fonti** di dati **esterne** devono essere **integrate** nel dataset

DATASET: insieme di dati che ci interessa per fare analisi

Per essere utili i dati **NON** devono essere necessariamente di grandi dimensioni

Una realtà viene descritta mediante **attributi** (colonne)

Più attributi ci sono, più abbiamo informazioni su quel **modello**, quindi è sempre **più preciso**

Invece, più **righe** permettono di avere più **storia** da analizzare

Gli algoritmi **NON** possono macinare quantità di dati indefinite, inoltre è possibile che un'**eccessiva quantità di dati alterino il comportamento** del modello

I dati vanno guardati con attenzione, infatti vengono eseguiti un processo di **selezione** e uno di **preprocessing**

SELEZIONE

Si prendono gli attributi del modello e con capacità di analisi e con qualche strumento (statistico o di ML) riusciamo a fare una **selezione** degli **attributi migliori** per **descrivere** quel determinato **contesto**. È possibile fare anche un processo di selezione sulle istanze (**esempio**: mi interessa solo la popolazione di ingegneria di Unisannio)

La domanda che dobbiamo porci è: che tipo di informazione/conoscenza devo ricavare da certi dati?

È importante conoscere l'obiettivo dell'analisi

Dopo la selezione, c'è una parte di **preprocessing** dei dati

Spesso all'interno dei database i dati vengono inseriti in maniera **incompleta, imprecisa, ridondante, o con errori di digitazione**

I dati vanno **puliti**

Talvolta è possibile che non riusciamo a fare la pulizia, ma possiamo risolvere anche eliminando alcune istanze

| Dove non si è certi di portare una riga corretta, la si può eliminare

I dati possono essere **trasformati**

Esempio: si potrebbero fondere più attributi che rappresentano lo stesso comportamento

Oppure se ci sono due attributi che sono molto correlati (si verifica con un test statistico), facendo l'analisi si nota che i due attributi fanno convergere allo stesso risultato, per cui se ne può portare appresso uno solo

DATA MINING: si applicano algoritmi per ricavare conoscenza (modello di predizione, pattern comportamentale, ecc), tutto però dopo aver fatto la parte di selezione, preprocessing

Quando si fa **data mining** si usano degli algoritmi di una libreria Python dove è molto richiesta l'intelligenza umana

L'analista diventa forte quando è in grado di **analizzare bene** i dati **prima di darli in ingresso ad un algoritmo**

Se si va di fretta perché ci scocciamo, il risultato sarà STM

Paradossalmente la parte che viene dopo l'applicazione dell'algoritmo è più automatica
Invece è molto più difficile la fase di selezione e preprocessing

TECNICHE DI ANALISI

Esistono 3 tipologie di tecniche di analisi:

- **regole di associazione**, permettono di identificare delle **associazioni frequenti** in un insieme di elementi all'interno di un dataset (**esempio**: molto utilizzata per il carrello della spesa, chi compra un tavolo compra anche un Lollone)
- **classificazione**, consente di creare **modelli predittivi** di **comportamento rispetto** ad un **modello della realtà** che andiamo a costruire. Potremmo avere un insieme di dati che vengono classificati rispetto ad una **classe di appartenenza** per creare un modello di predizione per i nuovi dati che arrivano per una classe non nota (**esempio**: capire se un nuovo cliente può essere fedele o meno, sistema anti spam delle mail)
- **clustering**, ci serve a **segmentare/suddividere** alcuni **elementi omogenei** all'interno di un insieme di dati (**esempio**: tipologie di clienti, studenti fuoricorso o non). Come si segmenta? Si deve **controllare la distanza** tra gli **attributi**. Si crea un **sottoinsieme** in cui ci sono **punti vicini tra loro e sufficientemente lontano dagli altri**

PROCESSO

Innanzitutto, bisogna **documentarsi** bene sul **dominio applicativo**

Bisogna **pulire** i dati

Bisogna **integrare** i dati provenienti da diverse sorgenti

Dopodiché si passa alla **selezione** dei dati

Poi si passa alla fase di **data trasformation**

Si **sceglie** il tipo di **analisi da fare** (se fare regole di associazione, classificazione, clustering, regressione lineare), in particolare bisogna capire quale **analisi** è **migliore** per il nostro dataset

In base all'analisi scelta, bisogna scegliere l'**algoritmo** da utilizzare

Si procede con la fase di **data mining** e poi bisogna **valutare i risultati**

Per la valutazione, si chiede all'algoritmo dei casi in cui noi sappiamo già la risposta ma che lui non ha mai visto

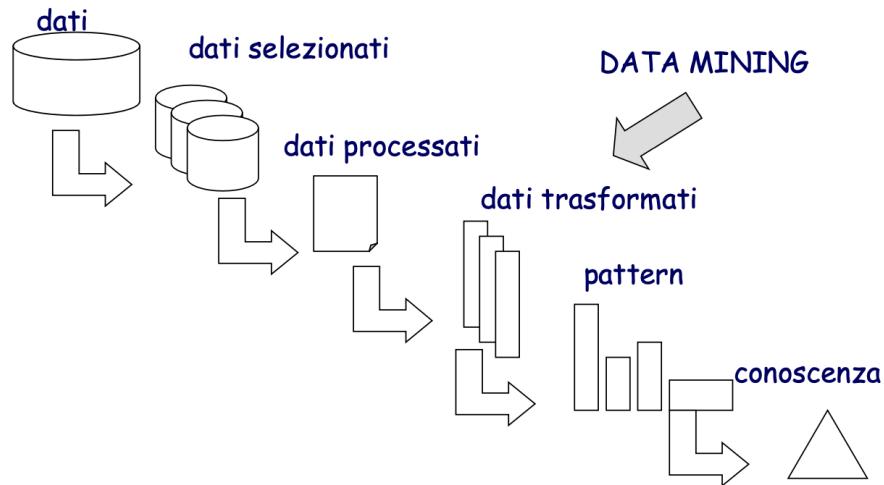
Quindi abbiamo bisogno di dati e di misure (corrette o non corrette)

Abbiamo bisogno di **dati** sia per **addestrare** sia per **testing**

Se testassimo con dati per addestramento, avremmo la certezza che funzioni bene

La tecnica più semplice per dividere il dataset è fare 70-30

L'importante è che il modello non abbia mai visto i 30



PRE-ELABORAZIONE DI DATI

Istanze e attributi

I dati devono essere **necessariamente denormalizzati** per fare data mining

Dobbiamo avere **una sola tabella**, la **ridondanza regna**

All'interno del dataset, tutte le **colonne** sono gli **attributi**

Le **righe** sono le **istanze** (o **tuple**)

I dati devono essere **denormalizzati** perché ci servono subito, **NON** possiamo pensare di fare un join prima di conoscere tutte le informazioni di uno studente perché tipicamente le istanze in un dataset sono tante

Da un punto di vista di performance si perde tantissimo se si hanno dati normalizzati

Quindi si **accetta la ridondanza** (occupiamo più spazio su disco) perché il **calcolo** è più **veloce**

La **classificazione** è una delle tecniche più utilizzate

Esempio: compagnie assicurative per l'assegnazione delle classi di rischio

Dataset = tabella denormalizzata

Gli **attributi** possono essere **distinti** sul livello di misura:

- **nominali**: ogni valore è **singolo** e **distinto** per ogni attributo, l'unica operazione consentita è **l'uguaglianza**, esempio: tipo di auto

- **ordinali:** ci sono ancora **etichette** che rappresentano un significato, ma è possibile una **relazione d'ordine**. Da un punto di vista logico sono consentite anche operazioni di **confronto (minore e maggiore)**, oltre all'**uguaglianza**. **Esempio:** freddo, tiepido, caldo
- **intervallo:** è consentita l'operazione di **sottrazione** per vedere la **distanza**, mentre le altre operazioni **NON** hanno senso. Inoltre, **NON** esiste un **valore zero significativo**
- **ratio:** **tutte le operazioni** aritmetiche sono **consentite**, **esiste** un valore **zero ben definito**

Per **velocizzare** le operazioni si fa un **encoding**, ad **esempio** posso assegnare a freddo il valore 0, a tiepido 1 e caldo 2 così da capire più velocemente chi viene prima e chi dopo

È importante capire il tipo di dato che si utilizza per comprendere le operazioni si possono fare

Gli **algoritmi** di **data mining** trattano **attributi**:

- **nominali:** detti anche **categoriali** o **discreti**, sono quelli che hanno le **etichette/label**
- **numerici:** abbiamo noi la **responsabilità** di **NON far fare all'algoritmo operazioni** che **NON** hanno **senso sui dati in questione**

I dati nel mondo reale sono **sporchi**. Per **sporchi** si intende che sono:

- **incompleti:** mancano il valore di alcuni attributi o mancano del tutto alcuni attributi di interesse
- **inaccurati:** dati che contengono valori che sono più o meno palesemente errori (esempio: età = 150)
- **inconsistenti:** due filiali dello stesso negozio che usano due codici diversi per rappresentare lo stesso prodotto

Tutte queste inesattezze concorrono a produrre spazzatura come analisi finali

garbage in - garbage out

Le tecniche principali per la pulizia dei dati sono:

- **data cleaning:** riempire i valori mancanti, "lisciare" i dati rumorosi, rimuovere valori non realistici.
- **data integration:** al dataset si aggiungono dati che provengono da fonti esterne
- **data transformation:** ad **esempio**, si potrebbe sostituire un valore palesemente errato con la media
- **data reduction:** potrebbe essere necessario ridurre la mole dei dati, ad **esempio** quando il **dataset** è **sbilanciato**. Per **correggere lo sbilanciamento** si può attuare il

campionamento. Si può o **sovracampionare** sulla **classe minoritaria** oppure **campionare** sulla **classe maggioritaria**. Con questa tecnica si corregge anche la **ridondanza**

È importante avere classi bilanciate altrimenti l'algoritmo **NON** apprendere correttamente dalle classi

DATA CLEANING

Le attività eseguite sono:

- **riempire gli attributi** che hanno **valori mancanti**
- **identifichiamo gli outliers** (**dati molto diversi dai valori attesi**, valore molto fuori scala), per vedere questa cosa si fanno dei grafici e subito salta all'occhio l'anticonformista
- **si lisciano i dati rumorosi**
- **si correggono le inconsistenze**

DATI MANCANTI

I possibili approcci sono per risolvere il problema dei dati mancanti sono:

- **ignorare le istanze con valori mancanti**, però non è un buon approccio se la percentuale dati mancanti è alta
- **riempire i valori mancanti manualmente**, molto utilizzata in un dominio medico
- utilizzare un **valore costante**

I **dati numerici** potrebbero essere corretti con la media, oppure utilizzando modelli di predizione (**esempio**: numeri di prodotti venduti da un agente, con la regressione lineare si può predire la quantità di merce venduta)

DATI INACCURATI

Per i dati inaccurati ci vuole più pazienza

Bisogna utilizzare opportuni **filtri** oppure sviluppare opportuni **grafici**

Spesso sono **più difficili da individuare che da correggere**

Esempio: se trovassimo "coca" anziché "coca cola" allora la correzione sarebbe facile, ma se si tratta di qualche medicina allora è un problema

I grafici che si utilizzano sono **istogrammi, grafici di dispersione**

Gli **istogrammi** sono utili per capire il **bilanciamento dei dati**

Per gli **outliers** sono meglio i **grafici di dispersione**

Quando abbiamo necessità di correggere dati in cui sospettiamo che ci siano errori, ma non riusciamo ad identificarli facilmente, allora possiamo correggere gli errori su **tutta la colonna di attributi** utilizzando la **tecnica del binning**

Esistono due versioni di **binning**:

- **equi-depth binning:** consideriamo tutti i valori con ripetizioni dell'attribuito e li ordiniamo in ordine crescente. Si fissa un valore di correzione **d (depth)** e si divide l'intervallo di tutti i valori in intervalli più piccoli consecutivi disgiunti, detti **bin**, di ampiezza più o meno pari a d. Ci saranno $\frac{n}{d}$ intervalli. La funzione che si applica è detta di **smoothing**. Sostituiamo ogni valore del bin con valore della funzione di smoothing. In questo modo correggiamo l'**intera colonna**, perciò si dice "lasciare" dato che non si tratta di una correzione secca su un solo valore. Quindi, l'errore è stato corretto rispetto a tutti i valori. La **profondità** si sceglie **in base all'ampiezza iniziale** dei dati
- **equi-width binning:** cambia solo la costruzione e dei bin, mentre lo smoothing si fa allo stesso modo

Le funzioni di **smoothing** sono:

- **media**, ogni valore appartenente all'intervallo singolo è sostituito con la media di quell'intervallo
- **mediana**, ogni valore appartenente all'intervallo singolo è sostituito con la mediana di quell'intervallo
- **boundaries (gli estremi)**, sostituisco ogni valore con l'estremo più vicino

▼ Lezione 4 - 14/03/2024

Oggi un po' di Python per fare esercitazione **pre-elaborazione**

DICTIONARY

Negli altri linguaggi si chiama **mappa**

In generale, si tratta di una **collezione di coppie chiave-valore**

Esercizio: **frequenzimetro**

```
def frequenza_di_comparsa(s):
    occorrenze = {}
    for c in s:
        if c == ' ':
            continue
        elif c in occorrenze:
            occorrenze[c] += 1
        else:
```

```
    occorrenze[c] = 1  
return occorrenze
```

FILES

In Python esiste una **funzione built-in** che ci consente di aprire un file, ritorna un object che è proprio il file letto

A seconda del tipo di file bisogna fare elaborazioni differenti

`open()` apre qualsiasi tipo di file anche se ci sono librerie che permettono di lavorare più facilmente alcuni tipi di file

`csv` è ottima per i file excel

`pandas` invece per i file di testo

Ovviamente ci sono due modalità di scrittura: **write** e **append**

Il costrutto `with` serve per indicare che una volta finite le operazioni dopo la open, il file viene chiuso automaticamente

```
with open('test.txt') as f:  
    f.write(contenuto)
```

Il **modulo os** contiene molte funzioni per manipolare file e processi

ESERCITAZIONE 1

L'obiettivo dell'esercitazione è il **data cleaning**

Dobbiamo lavorare in excel e descrivere tutti i passi per correggere gli errori

▼ Lezione 5 - 18/03/2024

CAPIRE I DATI: STATISTICA VS MACHINE LEARNING

Sono due discipline che prevedono l'uso di dati ma con scopi diversi

Il **Machine Learning** è basato sui big data e su algoritmi di analisi che consentono al sistema di apprendere dai dati stessi. Il ML quindi, impara dai dati e modifica i processi di conseguenza, non si ferma all'osservazione

Un **modello statistico** tradizionale si basa su **campioni di dati**, dalla numerosità non eccessivamente elevata, sui quali si formulano assunzioni **a priori**, ossia derivate dal

ragionamento a partire da proposizioni autoevidenti

Si può dire che l'**analisi statistica** è per lo più **descrittiva**

Le **relazioni** tra i dati devono essere prima ipotizzate per **verificarle** con un'analisi statistica

Invece, quando si utilizzano modelli di **ML** si è liberi dal fare ipotesi perché il **modello impara da solo**, l'**ipotesi è tirata fuori dal modello stesso**

L'**insieme di dati** in **statistica** è detto **popolazione**, invece in **ML** è detto **dataset**

In **ML** si parla di **big data** e di **modelli di auto-apprendimento** che ci consentono di fare previsioni del comportamento futuro di un fenomeno

La **statistica** è una scienza utilizzata per **studiare fenomeni collettivi** che possono essere **misurati** e di cui è possibile fare una **descrizione quantitativa**

La statistica richiede l'osservazione di fenomeni collettivi e ne fa una sintesi

La statistica si divide in:

- **statistica descrittiva:** è una tappa obbligatoria, serve per capire il dominio, l'ampiezza e l'andamento dei dati sia **calcolando indicatori** sia facendo **rappresentazioni grafiche**
- **inferenza:** si cercano **relazioni** all'interno dei dati facendo **a priori** delle **ipotesi**

Alla base della statistica ci sono due concetti:

- **evento:** fenomeno da osservare a cui si associano delle misure dato che occorrono misure quantitative (deve essere dominio discreto)
- **spazio dei campioni:** insieme di tutti i possibili risultati osservabili di un evento

Altre definizioni:

- **carattere:** è la **caratteristica di interesse** che viene rilevata/**misurata/osservata**
- **modalità:** manifestazione con cui si presenta il carattere, possono essere **numeriche** o **NON numeriche**; devono essere:
 - **esaustive:** devono includere **tutti** i modi di essere del carattere considerato
 - **non sovrapponibili:** esattamente individuate o mutuamente escludenti

Il **carattere** può essere:

- **quantitativo:** esprimibile mediante **numeri** e sono **misurabili**; a sua volta può essere:

- **discreto:** assume valori all'intero di un intervallo finito di valori, esprimibile tramite numeri interi
- **continuo:** assume un numero illimitato di valori, esprimibile tramite numeri reali
- **qualitativo:** quando viene espresso con una **descrizione**, quindi in modalità non misurabile; a sua volta può essere:
 - **ordinale:** le modalità seguono un ordine naturale in successione
 - **sconnesso:** non esiste un ordine tra le modalità

Esempio: un insegnante chiede a 50 alunni dove sono stati in vacanza, dando loro delle alternative di risposta; il **carattere** rilevato è il **tipo di vacanza** e le **modalità** sono le alternative di risposte date agli alunni: nessun viaggio, mare, montagna, campagna, città d'arte e altro

Continuiamo con altre definizioni:

- **popolazione:** si indica con N e rappresenta il numero totale delle unità statistiche considerate
- **campione:** si indica con n e rappresenta il sottoinsieme della popolazione su cui intendiamo effettuare le analisi
- **unità statistica:** unità elementare su cui vengono osservati i caratteri; può essere:
 - **naturale:** ad esempio un essere umano, un'auto
 - **convenzionale:** ad esempio la famiglia

INDICI DI POSIZIONE

Gli indici di posizione servono a **sintetizzare la posizione centrale di una distribuzione statistica** sostituendo tutti i dati **con un solo valore**. Forniscono un'indicazione sull'ordine di grandezza dei dati in nostro possesso

I principali indici di posizione sono:

- **media**
- **mediana:** divide in due parti di uguale numerosità i dati
- **moda:** è il valore che si presenta con maggiore frequenza all'interno della variabile

Per utilizzare la **mediana**, i **dati vanno necessariamente ordinati** (indifferente se ordine crescente o decrescente)

Se **n è dispari**, allora la **mediana** è proprio il **valore centrale**, cioè quello di posizione $n/2$

Se **n è pari**, allora per calcolare la **media** bisogna calcolare la media aritmetica dei due valori centrali, cioè quelli di posizione $n/2$ e $(n/2) + 1$

La **moda** può essere utilizzata **solo se** si tratta di una variabile **nominale o categorica**. Inoltre è possibile anche averne più di una

Valutare la moda permette di capire facilmente se la **distribuzione** della variabile è **omogenea**

Quando e quale indicatore di posizione centrale usare?

La **media** va utilizzata quando si studiano quantità che non presentano valori anomali, quindi quando i dati sono abbastanza **omogenei tra loro**

La **mediana** si utilizza spesso quando NON è possibile usare la media

La **moda** si può usare quando abbiamo variabili nominali o categoriche e abbiamo necessità di conoscere la caratteristica più diffusa

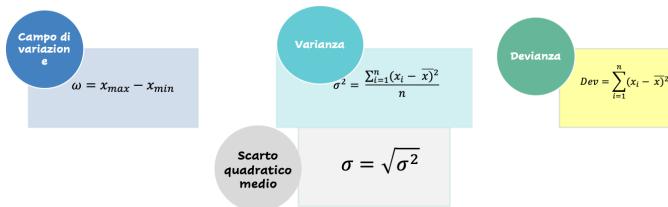
INDICI DI VARIABILITÀ

Gli indici di posizione permettono di capire la **tendenza** dei dati, ma **NON** sono sufficienti a **caratterizzarli**

Gli indici di variabilità possono assumere **solo valori positivi**

Di questo tipo sono:

- **campo di variazione:** differenza tra il valore **massimo e minimo**
- **varianza:** media del quadrato degli scarti dalla media, da questa si ricava anche lo scarto quadratico medio
- **devianza:** è la somma dei scarti della media al quadrato (**numeratore della varianza**)



Questi indici sono utilizzati per capire la **variabilità dei dati** nella popolazione, per **comprendere la dispersione dei dati**

Tra gli indici di variabilità, il più utilizzato è la varianza

RAPPRESENTAZIONE GRAFICA

Le più famose sono box-plot e istogramma

BOX-PLOT

Il box plot è un grafico che fornisce nello stesso momento informazioni sulla posizione, sulla variabilità e sulla forma della distribuzione dei dati

È costituito da **tre elementi**:

- una linea in corrispondenza della **mediana** che rappresenta il **centro** della distribuzione
- un rettangolo o scatola (**box**) che va **dal primo al terzo quartile**
- due segmenti, detti **baffi**, che indicano gli **estremi** della distribuzione (min e max)

In presenza di valori anomali è possibile trovare anche un quarto elemento: alla **fine dei baffi** troveremo dei punti che indicano i **valori anomali**. Per poter capire quando un valore viene considerato **"anomalo"** basta semplicemente **calcolare il recinto interno** (r_1, r_2) e il **recinto esterno** (R_1, R_2). Tutte le **osservazioni all'esterno** del **recinto interno** sono dette **"osservazioni distanti"**, mentre quelle all'esterno del **recinto esterno** sono dette **"osservazioni molto distanti"**; entrambi i tipi di osservazioni costituiscono i valori anomali della distribuzione.

$$\begin{aligned}r_1 &= Q_1 - 1.5 * DQ; & r_2 &= Q_3 + 1.5 * DQ \\R_1 &= Q_1 - 3 * DQ; & R_2 &= Q_3 + 3 * DQ\end{aligned}$$

Dove Q_1 è il valore del **primo quartile**, Q_3 il valore del **terzo quartile** e DQ è la **differenza interquartile**, ossia $Q_3 - Q_1$

I **quartili** sono valori che ripartiscono la distribuzione della variabile in quattro parti di uguale numerosità. In particolare, la **mediana** non è altro che il **2° quartile**, quindi divide in due parti di uguale numerosità. Il **quartile zero** coincide con il **valore minimo** della variabile, invece il **4°** coincide con il suo **valore massimo**

Ogni quartile contiene il 25% del numero totale di dati

I box-plot sono particolarmente utili quando studiamo la stessa variabile osservata però in momenti differenti

ISTOGRAMMA

È il grafico mediante il quale viene rappresentata la distribuzione in classi di un carattere quantitativo

È composto da tanti rettangoli adiacenti quante sono le classi di cui è composta la variabile e viene costruito seguendo due criteri:

- le **basi** dei rettangoli rappresentano le classi
- l'**area** dei rettangoli è data dalla frequenza delle classi

Per ciascuna classe quindi va calcolata l'**ampiezza** della **base** come differenza fra gli estremi della classe e l'**altezza** del rettangolo o **densità** data dal rapporto tra la frequenza relativa e l'ampiezza

Nella maggior parte degli istogrammi è possibile notare una più o meno forte **asimmetria** tra i vari rettangoli, ciò è dovuto alla presenza di una **classe modale**, ossia una classe con **maggior frequenza**

Se, invece, i valori sono distribuiti in modo **simmetrico**, allora la curva rappresentata dall'istogramma può essere ricondotta ad una **gaussiana**

ALTRE RAPPRESENTAZIONI GRAFICHE

Esistono anche altre rappresentazioni grafiche come i grafici a barre, a torta e a dispersione

Tuttavia, agli ingegneri non piacciono i grafici a torta perché non consentono il confronto immediato tra le classi

▼ Lezione 6 - 21/03/2024

ANALISI ESPLORATIVA

Si intende l'insieme di attività volte a studiare:

- la **distribuzione**
- le **proprietà**
- **relazioni** tra attributi

Esempio: data di nascita ed età sono due attributi in forte relazione

È importante capire le relazioni tra i dati soprattutto quando si lavora con **grandi quantità di dati**. Infatti in questi casi, bisogna effettuare una **selezione** degli attributi per rendere i calcoli meno complessi

ANALISI DEI TIPI

Il primo obiettivo dell'**analisi esplorativa** consiste nel capire:

- la **dimensionalità** del dataset (numero di righe e colonne)
- il **tipo associato** ad ogni **colonna** (intero, decimale, data, ecc)

- la **distribuzione** dei **null** (o **valori mancanti**) per ogni colonna

In Python possiamo usare il metodo `info()` della libreria `pandas` che ci consente di ottenere tutte queste informazioni con una sola riga di codice, quindi ricava un **summary** dei dati

```
# Esempio

import pandas as pd
import seaborn as sns
taxis = sns.load_dataset('taxis')
taxis.info()
print(taxis.head())
```

Il **dataframe** è **taxis** e su questo invochiamo il metodo `info()`

`info()` crea una tabella che ci permette di ricavare la **dimensionalità** del dataset, il **tipo** associato alle colonne e quanti valori **NON nulli** ci sono

È una stampa **quantitativa**

Tuttavia è sempre opportuno andare a verificare i risultati del metodo `info()` (*fidarsi è bene non fidarsi è meglio*)

Il metodo `head()` permette di visualizzare il contenuto delle **prime 10 righe**

In realtà, `head()` accetta come parametro un **int** che permette di **scegliere il numero** di **righe** da visualizzare (**di default è 10**)

ANALISI DELLE RELAZIONI

Il passo successivo è **calcolare gli indici statistici** in modo da farci un'idea della distribuzione e dell'eventuale **presenza di outlier**

In Python tutto ciò è rapido grazie al metodo `describe()` di pandas

```
taxis.describe()
```

Ci basta eseguire questo metodo per ottenere il **numero di valori non-null**, la **media aritmetica**, la **deviazione standard**, **massimo**, **minimo** e **quartili** di **tutte le colonne**

Inoltre, se come parametro passiamo `include='all'`, otterremo informazioni anche per le variabili categoriche come **moda** e numero di **valori univoci**

Un ulteriore obiettivo dell'analisi esplorativa è indagare sulla presenza di **relazioni e associazioni tra le colonne**.

La **correlazione** è un indice che dice quanto il variare di una variabile incide sulla variazione di un'altra

È importante capire se due variabili sono **correlate** perché, se lo fossero, allora sappiamo che avranno un andamento **simile**

Può essere anche una **correlazione negativa**

Ciò si può analizzare facilmente con il metodo `corr()` di pandas

```
taxis.corr()
```

Questo metodo ci restituisce la **matrice di correlazione** tra le colonne numeriche del DataFrame utilizzando il **coefficiente di correlazione di Pearson** (tramite il parametro `method`) però si può calcolare il coefficiente di Spearman o Kendall Tau)

La matrice di correlazione avrà **tante righe e tante colonne quante** sono le **variabili** analizzate e l'intersezione riga-colonna è proprio il coefficiente di correlazione della coppia di variabili

Sulla **diagonale** della matrice, ovviamente, troviamo sempre 1 perché è il coefficiente di correlazione con la variabile stessa

Una cosa carina che si può fare con Python è produrre un risultato graficamente, in particolare una **heatmap**

```
mat = taxis.corr()
plt.figure()
sns.heatmap(mat, annot=True)
```

Graficata in questo modo la matrice fa capire subito il grado di correlazione tra le variabili

In generale, **questa matrice** è uno strumento che ci serve a capire quali variabili portare avanti nel processo di analisi dei dati

Quindi, **permette** di fare una **selezione** delle **features**

Esempio: se ci sono variabili **fortemente correlate**, allora queste daranno lo stesso contributo se dovessimo risolvere un problema con un modello di predizione

Un algoritmo per un **modello predittivo** impara di più se le variabili **NON** sono molto **correlate**

Solitamente, si sceglie una **soglia** per il coefficiente di correlazione e, sulla base di questa, si scelgono **quali variabili includere** nel calcolo del **modello** di apprendimento

Il calcolo del modello di apprendimento **migliore NON** avviene subito, al primo tentativo, ma si lavora **empiricamente**, per esperimenti

Nessuno può sapere qual è **a priori** la **soglia giusta** da stabilire, ma si capisce empiricamente

RAPPRESENTAZIONE GRAFICA

Rappresentare graficamente le variabili permette di avere una visione **più veloce e d'impatto** delle caratteristiche dei dati con cui si lavora

Con le librerie pandas e seaborn possiamo facilmente fare grafici accurati e professionali

Per l'**istogramma** c'è il metodo `catplot()` di seaborn

Il **box-plot** è una tipologia di rappresentazione grafica di una variabile statistica quantitativa tra le **più utili** in quanto riassume alcuni dei più importanti indici statistici della variabile, mettendone in luce anche la **presenza di eventuali outlier**

Per disegnare il box-plot di una variabile dobbiamo calcolare:

- **mediana**
- **primo e terzo quartile**
- **valori soglia** (detti anche **valori limite**) per individuare gli outlier, cioè:
 - primo quartile – $1.5 * (\text{terzo quartile} - \text{primo quartile})$
 - terzo quartile + $1.5 * (\text{terzo quartile} - \text{primo quartile})$
- **outlier**

A questi eventualmente si può aggiungere anche la media aritmetica

Il box plot può essere disegnato anche per **più variabili** dello stesso **dataset**

Se il risultato è **compatto** e schiacciato, allora ciò ci fa capire che ci sono degli **outlier** che **aumentano** di molto la **dimensione dell'asse**

Inoltre, è possibile rappresentare anche un box-plot su **una variabile** ma diviso in **categorie differenti**

In Python è molto semplice realizzarli grazie al metodo `boxplot()` di pandas

▼ Lezione 7 - 25/03/2024

OPEN DATA

Il concetto da cui partiamo è quello di **open-source**

L'**apertura** è un concetto basato sulla trasparenza e sull'accesso **senza restrizioni** all'**informazione** e alla **conoscenza**

La filosofia open è comune a diversi movimenti:

- Open Content
- Open Source

- Open Access
- Open Education

Con **Open Science** ci riferiamo alla possibilità di accesso per tutti ai dati utilizzati nella ricerca scientifica, in modo che si possa ripetere lo studio e magari anche migliorarlo

In generale, si sta diffondendo sempre di più la possibilità di accedere gratuitamente ad articoli scientifici, mentre fino a pochi anni fa i materiali accessibili sul sito IEEE erano tutti a pagamento

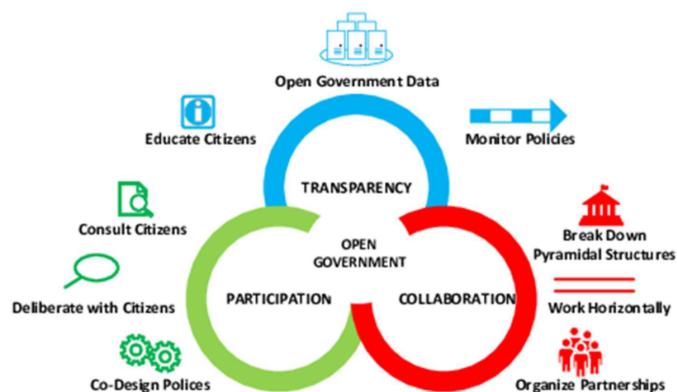
Gli **open data** sono dati liberamente accessibili a tutti, privi di brevetti o altre forme di controllo che ne limitino la riproduzione e le cui restrizioni di copyright eventualmente obbligano a **citare la fonte** o al rilascio delle **modifiche** (l'**elaborazione** di quei dati)

Il concetto di open data è molto legato alla disciplina dell'**open government**, quindi più alla pubblica amministrazione che al privato

Nel amministrazione pubblica, c'è la responsabilità di raccogliere i dati e renderli fruibili alla cittadinanza

Con **open government** si intende la possibilità che **qualunque cittadino** ha di poter fare **analytics** sui dati messi a disposizione da un'azienda

Purtroppo però, siamo ancora lontani ad avere tutte le aziende pubbliche che lavorano in questo modo



L'open government **NON** può esistere senza gli **open data**

Un passo importante per la distribuzione dei dati pubblici è avvenuto durante il periodo del covid, infatti la protezione civile forniva dati in modo da renderli fruibili per qualcuno che ne ha condiviso i risultati (**esempio**: bollettino medico giornaliero)

Con "qualcuno" si intende molte persone in grado di fare questo tipo di analisi, cioè persone che hanno messo a disposizione le proprie conoscenze informatiche per fare analisi dei dati

C'era proprio una repository su GitHub

PROGETTARE OPEN DATA

Un **dato** si definisce **aperto** se **chiunque** è in grado di **utilizzarlo, riutilizzarlo, ridistribuirlo** e soggetto alla richiesta di attribuzione e **condivisione** allo stesso modo

Gli open data devono essere:

- **rintracciabili**, quindi **indicizzati** dai **motori di ricerca**
- **disponibili** in un **formato aperto**, standardizzato e leggibile da una macchina
- rilasciati attraverso **licenze libere** che non impediscono la diffusione e il riutilizzo citandone la fonte

Se il dato è **leggibile** sia dall'**uomo** che dalla **macchina**, allora è **veramente accessibile**

Progettare open data significa:

- **organizzarli** in **dataset**
- **verificarne** il **contenuto**, cioè fare un'attività di pre-elaborazione perché devono essere **privi** di **errori**
- eventualmente lavorare per **completare** i dati
- verificare la **consistenza** dei dati (no duplicati, ecc)

Il **processo** di **produzione** dei dati segue il modello del **continuous improvement**, in particolare quello di **plan-do-check-act**:

- **plan**: bisogna fare una cernita dei dati, cioè analizzarli per scegliere quali devono diventare open
- **do**: si pubblicano i dati
- **check**: si monitorano i dati, bisogna seguire ciò che viene fatto con quei dati

- **act:** bisogna aggiornare i dati con una cadenza temporale fissata e agire in base al monitoraggio

CARATTERISTICHE DEGLI OPEN DATA

Gli open data devono essere:

- **completi**, cioè devono comprendere tutte le componenti che consentano di esportarli, utilizzarli, integrarli e aggregarli con altre risorse per diffonderli successivamente e allo stesso tempo indicare anche le metodologie di realizzazione adottate
- **primari**, cioè devono essere dati che si presentano ad un livello basso di granularità, quindi devono essere dati **NON elaborati**
- **tempestivi**, gli utenti devono poter accedere ai dati presenti in rete in modo rapido per massimizzare l'uso delle risorse
- **accessibili**, tutti gli utenti devono avere la possibilità di fruire i dati senza alcun pagamento o richiesta ufficiale
- **leggibili da una macchina**, ad **esempio** se ho i dati in formato pdf, questi non possono essere utilizzati, occorre un dataset
- **NON discriminatori**, cioè **NON** devono esserci **meccanismi di controllo** degli **accessi**
- **NON proprietari**, cioè il dato deve essere direttamente interpretabile, quindi gli utenti devono poter utilizzare e processare i dati attraverso programmi e applicazioni non proprietarie
- **liberi da licenze**, cioè **NON** devono sottintendere né brevetti, né copyright, per questo motivo sono molto legati alle aziende pubbliche
- **riutilizzabili**, cioè gli utenti devono essere messi in condizione di riutilizzare e integrare i dati fino a creare nuove risorse
- **ricercabili**, cioè gli utenti devono avere l'opportunità di ricercare con facilità e immediatezza dati e informazioni di proprio interesse
- **permanenti**, la loro validità deve essere mantenuta nel tempo (**continuous improvement**)

Da un punto di vista giuridico, c'è un equilibrio difficile tra trasparenza e privacy

Con **trasparenza** indichiamo il concetto che obbliga a condividere più informazioni possibili
Ciò però rende complicato l'equilibrio con il concetto di
privacy in quanto bisogna rispettare quella del cittadino coinvolto nella produzione dei dati

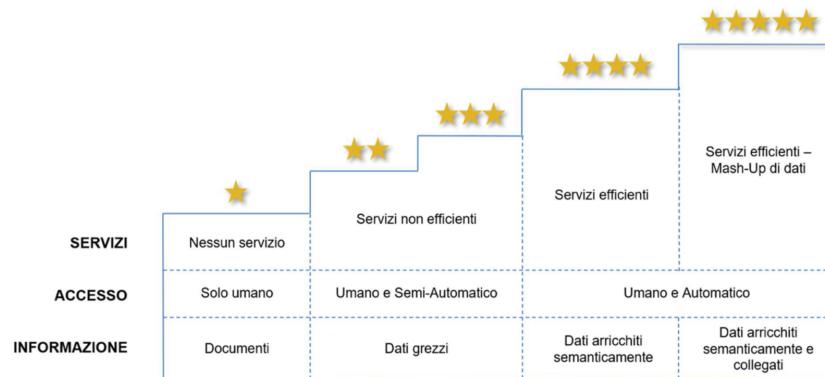
Esempio: sanità

La struttura ospedaliera deve rilasciare dati relativi a quante operazioni chirurgiche sono state fatte, ma senza dare la possibilità di ricondursi ad una persona in particolare

SISTEMA DI CLASSIFICAZIONE DEGLI OPEN DATA

Questo sistema ci dice quanto siamo vicini all'ottimo dell'open data così come ci aspettiamo che venga rilasciato. Ci sono 5 livelli:

1. è il livello base, cioè il dato **esiste** ma **NON** è **organizzato** nella forma che dovrebbe essere, rispetta **pochi requisiti**
2. dati **strutturati** ma codificati in un formato **NON liberamente accessibile**, in un **formato proprietario**
3. a questo livello si arriva alla sufficienza perché i dati sono **strutturati** ma codificato con un **formato NON proprietario**
4. dati **strutturati** ma codificati con un **formato NON proprietario dotati di URL** a cui vengono aggiunti **metadati**
5. **linked open data**



È importante la classificazione perché da questa si determina l'utilizzo che si può fare di determinati dati

Esempio: dati a livello base (1 stella), nella maggior parte dei casi si ha a che fare con file che sono immagini, quindi comprensibili solo da un umano e quindi non può essere fatto nessun tipo di servizio

L'intelligenza umana è fondamentale, quindi anche per modelli a 4 e 5 stelle c'è bisogno dell'intervento umano

DAI DATI GREZZI AI LINKED OPEN DATA: FORMATI E TIPOLOGIE DI DATI

Più i dati sono manipolabili dalla macchina, più riusciamo ad analizzarli

Formato	Può essere estratto il testo?	Machine Readable?	Specifiche disponibili?	Formato aperto?
Plain Text (.txt)	Si	Si	Si	Si
Comma Separated Value (.csv, .txt)	Si	Si	Si	Si
Hyper Text markup Language (.html, .htm)	Si	Si	Si	Si
Extensible Markup Language (.xml)	Si	Si	Si	Si
Microsoft Word (.doc, .docx)	Si	No	Si	No
Microsoft Excel (.xls, .xlsx)	Si	Si	Si	No
Resource Description Framework (.rdf)	Si	Si	Si	Si
Portable Document Format (.pdf)	No	No	Si	Si
Open Document Format (.odt, ods, ...)	Si	Si	Si	Si
Image files (.jpg, .tiff, .tif, ...)	No	No	Si	Si

La direttiva sull'open government del dicembre 2009 afferma che le agenzie devono pubblicare quanto più possibili dati in un formato open, cioè un formato indipendente rispetto alla piattaforma, leggibile dall'elaboratore e reso disponibile al pubblico senza che sia impedito il riuso dell'informazione veicolata

In generale, **csv** è il formato più **semplice**

DIFFUSIONE DEI DATI OPEN

Gli **open data** fanno spesso riferimento a informazioni rappresentate in forma di **database**

Spesso si cerca di indirizzare l'accesso a questi dati a livello governativo

Viene fatto attraverso portali che collegano i vari livelli, dall'ente centrale alla singola azienda locale

In Italia ci si è adeguati con un po' di ritardo, quando è nato il progetto per il rilascio degli open data dati.gov.it

Esistono normative che impongono il rilascio e le licenze di rilascio di dati

Ciò che si sta sviluppando oggi è l'allineamento con i dati degli altri stati europei

L'obiettivo europeo è la condivisione di dati guidata con modelli di qualità, altrimenti non sarebbe possibile integrazione tra i dati prodotti dai singoli membri

OPEN DATA MATURITY REPORT

Ogni anno viene redatto un report che consente di valutare il grado di maturità dei dati prodotti da ciascun Paese

La **qualità** si valuta mediante 4 parametri:

- **policy**, cioè si cerca di capire la strategia utilizzata per il rilascio degli open data

- **impatto**, si analizza la volontà e capacità di creazione e uso dei dati
- **portale**, si analizza il portale attraverso cui i dati vengono distribuiti, se sta funzionando
- **quantità**, si esaminano le misure del singolo gestore del portale adottate per verificare la qualità dei metadati associati ai dati. I metadati sono importanti perché consentono l'arricchimento semantico, altrimenti senza questi i dati sono spesso sterili

COME FUNZIONANO GLI OPEN DATA: I METADATI

I metadati sono informazioni a **corredo** dei dataset resi disponibili come dati aperti, quindi consentono una maggiore comprensione dei dati e aiutano la ricerca e la scoperta degli stessi mediante la loro aggiunta a **cataloghi**

Arricchiscono i dati consentendone un uso migliore (ad **esempio** per indicizzarli meglio) e inoltre aggiungono informazioni di tipo **semantico**

Nel rilascio degli open data, i **metadati** sono **obbligatori**

I metadati obbligatori sono:

- **publisher**: colui che pubblica il dataset
- **creator**: colui che produce il dato, spesso coincide con il publisher
- **rightsHolder**: indica il soggetto o l'organizzazione che detiene e gestisce i diritti sul dataset
- **titolo**
- **descrizione**: descrizione in linguaggio naturale del dataset
- **modified**: data dell'ultima modifica
- **frequenza di aggiornamento**
- **licenza**: indica la licenza utilizzata
- **keyword**: fondamentali per indicizzare i dati dai motori di ricerca

Esempio: se come comune mi affido ad un ente terzo per la creazione del dato, io comune sono solo publisher

In generale, **più metadati** aggiungiamo, **più il legame** tra gli open data rilasciati diventa **forte**

Quando si aggiungono metadati è importante aggiungere:

- **contestualizzazione geografica**: dove è stato prodotto il dataset
- **contestualizzazione temporale**: quando è stato prodotto il dataset

Dato che i dati vengono intercettati spesso da una macchina, la possibilità che esistano metadati che permettono la contestualizzazione automatica, allora la macchina può capire quando e dove il data set è stato prodotto

QUALITÀ DEL DATABASE

Deve:

- essere **completo**, cioè per ogni record dovrebbero essere compilati tutti i campi
- essere **corretto**, cioè si devono limitare il più possibile gli errori
- essere **coerente**, cioè le informazioni riportate non devono contraddirsi
- deve usare **classificazioni standard**
- **NON** deve riportare **errori di formattazione**

Nel modello degli open data, i beneficiari di questi siamo tutti noi

I dati sono a disposizione del cittadino, ma in generale possono avere anche un ruolo per lo sviluppo economico o del territorio

▼ Lezione 8 - 04/04/2024

Quasi tutti gli algoritmi di ML rientrano in **3 categorie**:

- **regole di associazione**: algoritmi utilizzati per cercare degli schemi, regole associative, dipendenze di dati identificati da comportamenti frequenti (**esempio**: carrello della spesa). Spesso usato per fare campagne di marketing mirato
- **classificazione**: utilizzati per creare dei modelli per fare una **predizione** di un **comportamento**. Questo comportamento di solito è etichettato con una **label** in una variabile detta **target**
- **clustering**: analizza i dati di tipo numerico/categorico e ha l'obiettivo di **segmentare i dati**. Gli algoritmi di clustering sono di tipo **NON supervisionato** perché non c'è una variabile **target di riferimento**. È la famiglia di algoritmi che misura la distanza tra tutte le istanze del dataset

REGOLE DI ASSOCIAZIONE

Dato un dataset, questi algoritmi ci aiutano ad **individuare** delle **regole forti** (le più rappresentative) che rappresentano comportamenti modellati dalla regola stessa

Le **regole** possono essere utilizzate per fare delle **previsioni**

Per cercare regole associative, si parte **in assenza di ipotesi**, quindi **NON** c'è una **regola da verificare**, ma è dai **dati** che viene ricavata

Per fare ciò i dati devono essere organizzati in un modo preciso

Dati del problema:

- **I** = insieme di **items** (esempio: tutti i prodotti venduti da un supermercato)
- **T** = **transazioni**, rappresentano l'insieme degli item che vengono acquistati (esempio: singolo scontrino che passa alla cassa il quale contiene un sottoinsieme degli item del supermercato)
- **D** = **base di dati**, cioè un insieme di transazioni

Ogni riga rappresenta una transazione, ogni colonna un item

Regola di associazione: $X \Rightarrow Y$

Supporto S: è una misura che indica la frequenza di quanto vengono acquistate insieme X e Y rispetto al totale delle transazioni, si calcola:

$$\frac{\#\text{trans. contenenti } X \cup Y}{\#\text{trans. in } D}$$

Questa misura ha una rilevanza statistica

Confidenza C: esprime la significatività dell'implicazione

$$\frac{\#\text{trans. contenenti } X \cup Y}{\#\text{trans. contenenti } X}$$

Quando andiamo alla ricerca delle regole associative, dobbiamo determinare tutte le **regole con supporto e confidenza superiori** ad un **soglia**

Il problema è che la **soglia** va decisa sempre **empiricamente**, non c'è nessuno che a priori ci dice qual è il valore

Questo è un po' la storia del ML, cioè bisogna fare delle prove per capire come ottimizzare al meglio il nostro studio

Esempio:

TRANSACTION ID	OGGETTI ACQUISTATI
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Assumiamo:

- supporto minimo: 50%
- confidenza minima: 50%

Regole ottenute:

- A \Rightarrow C: supporto 50% e confidenza 66.6%
- C \Rightarrow A: supporto 50% e confidenza 100%

APPLICAZIONI: Market Basket Analysis

Spesso la regola di associazione da cui si parte è:

* \Rightarrow uova

oppure

Latte \Rightarrow *

Le domande corrispondenti a queste regole sono:

- cosa si deve promuovere per aumentare le vendite di uova?
- quali altri prodotti devono essere venduti da un supermercato che vende latte?

* è l'**incognita**

Per risolvere il problema, utilizziamo un approccio *divide et impera*:

- andiamo a trovare tutti gli insiemi di item che hanno un supporto minimo (questi insiemi vengono detti **frequent itemsets**)
- generiamo delle **regole** a partire dai frequent itemsets

Questa decomposizione del problema è normalizzata dall'algoritmo **APRIORI** (ideato nel 1994)

Esempio:

Passo 1: estrazione frequent itemsets		
TRANSACTION ID	OGGETTI ACQUISTATI	
1	A,B,C	
2	A,C	
3	A,D	
4	B,E,F	
supporto minimo 50%		
FREQUENT ITEMSET	SUPPORTO	
{A}	75%	
{B}	50%	
{C}	50%	
{A,C}	50%	

Passo 2: estrazione regole

- confidenza minima 50%
- Esempio: regola A \Rightarrow C
 - supporto {A,C} = 50%
 - confidenza = supporto {A,C}/supporto{A} = 66.6%
- regole estratte
 - A \Rightarrow C supporto 50%, conf. 66.6%
 - A \Rightarrow C supporto 50%, conf. 100%

Qui finisce il ruolo dell'IA, perché c'è bisogno di qualcuno che interpreti le regole estratte

Infatti, **NON sempre** tutte le **regole** con **supporto e confidenza superiori alla soglia** sono **interessanti**

Esempio: scuola con 5000 studenti

- 60% (3000) gioca a pallacanestro
- 75% (3750) mangia cereali a colazione
- 40% (2000) gioca a pallacanestro e mangia cereali a colazione

Con supporto minimo 40% e confidenza minima 60%, la regola:

gioca a pallacanestro \Rightarrow mangia cereali

- supporto = 2000/5000 = 0.4
- confidenza = 2000/3000 = 0.66 > 0.6

Questa è una regola fuorviante perché il 75% degli studenti mangia cereali

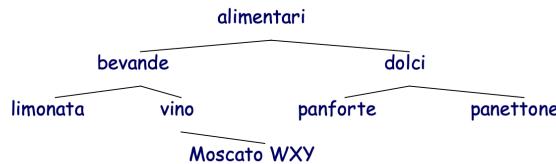
Introduciamo una nuova misura per misurare il fenomeno:

$$\frac{\text{supporto}(A, B) - \text{supporto}(B)}{\text{supporto}(A)}$$

Gli **item** possono avere una **dipendenza gerarchica** e questa può essere utilizzata per **generalizzare** delle **regole**

Il problema è che spesso la macchina NON conosce l'item **generalizzato**

Per cui è l'analista che deve costruire una nuova colonna, quindi deve mettere l'intelligenza per costruire questa nuova regola



- $vino \Rightarrow dolci$, valida anche se:
 - $bevande \Rightarrow dolci$ non ha confidenza suff.

Per utilizzare l'approccio della **generalizzazione**, bisogna **pre-elaborare** il dato

In generale, la generalizzazione è molto utile, ad **esempio** un supermercato che ha milioni di record, è molto comodo sostituire coca cola, aranciata, vino, acqua con la generalizzazione "bevande"

STRUTTURA E VALUTAZIONE

Date due proposizioni Y e Z, una **regola** è un **implicazione** del tipo $Y \Rightarrow Z$

$Y \Rightarrow Z$ significa che se **Y è vera**, allora anche **Z è vera**

Una **regola** si dice **probabilistica** se la **validità** di **Z** è **associata** ad una **probabilità p**, cioè se **Y è vera**, allora anche **Z è vera con probabilità p**

Talvolta le regole prodotte si limitano a rispecchiare circostanze evidenti, mentre altre volte diventano una **scoperta** all'interno di un dominio applicativo (**capovolgono il legame causale**)

MATRICE DEGLI ITEM

È una matrice **booleana**, infatti la codifica che viene fatta si chiama **One-hot encoding** proprio perché è una matrice binaria

Il **numero di colonne dipende dal numero di item**

Estendere la matrice vuol dire **aggiungere colonne**

METRICHE PER LE REGOLE

Frequenza empirica $f(L)$: è il numero di transazioni che contengono l'oggetto L

$$f(L) = \text{card}\{T_i : L \supseteq T_i, i = 1, \dots, m\}$$

Confidenza p: è una probabilità e, in particolare, è un **rappporto di frequenze**

$$p = \text{conf}\{L \Rightarrow H\} = \frac{f(L \cup H)}{f(L)}$$

L viene detto **corpo** della **regola**, mentre H è detto **testa** della **regola**

Supporto s: è dato dalla **frequenza empirica** $f(L \cup H)$ **diviso m**(cardinalità di T, insieme delle transazioni)

$$s = supp\{L \Rightarrow H\} = \frac{f(L \cup H)}{m}$$

GENERAZIONE ITEM SET FREQUENTI

Il **supporto** di una regola **dipende** solo dall'insieme $L \cup H$ (**itemset**) e **NON** dall'effettiva **distribuzione** degli oggetti tra **corpo** e **testa**

Infatti, se il supporto di un itemset è **minore** della soglia di supporto minimo fissata, allora la regola va **eliminata**

GENERAZIONE DELLE REGOLE

Dopo aver generato tutti i **frequent itemset**, bisogna **identificare** le **regole forti**, cioè quelle regole che superano la soglia di confidenza fissata

Gli oggetti in ciascun itemset vanno separati secondo tutte le combinazioni di corpo e testa per verificare se la confidenza della regola supera una soglia p_{min} (è un valore di probabilità)

Se l'itemset è **frequente**, allora si possono ricavare le **regole forti**

Però **NON sempre** le regole **forti** sono **significative** e/o particolarmente interessanti

Introduciamo una nuova metrica

INDICE DI LIFT

È un indice che ci permette di valutare la significatività di una regola ed è il **rapporto** tra **confidenza** e **frequenza**

$$l = lift\{L \Rightarrow H\} = \frac{conf(L \Rightarrow H)}{f(H)} = \frac{f(L \cup H)}{f(L)f(H)}$$

Questo indice consente di **neutralizzare** le **frequenze elevate**

L'obiettivo è trovare regole che hanno valore di **lift alto**

Valori di $l > 1$ indicano che la **regola è più efficace** nel **predire la probabilità** che la **testa** sia **contenuta** in una **generica transazione** di quanto lo sia la sua frequenza

Valori di $l < 1$ indicano che la **regola che nega la testa è più efficace** della regola **iniziale**

ALGORITMO APRIORI

In generale, un dataset formato a partire da n oggetti, può contenere fino a 2^{n-1} itemset frequenti

Nelle applicazioni pratiche n è almeno nell'ordine di alcune decine e un metodo di generazione esaustivo degli itemset è impraticabile

Il presupposto teorico alla base dell'algoritmo Apriori è:

se un **itemset** è **frequente**, allora **anche** i suoi **sottoinsiemi** sono **frequenti**; di conseguenza se un itemset **NON** è **frequente**, allora **neanche** gli insiemi che lo contengono sono **frequenti**

L'algoritmo **ricerca** gli **itemset frequenti** in maniera **iterativa**

Quindi, all'**inizio** parte da **itemset formati da un solo elemento**

Il **numero** delle **iterazioni** è $k_{max} + 1$, dove k_{max} indica la **cardinalità massima** di un **itemset frequente**

I passi che vengono effettuati sono:

1. si **calcola** la **frequenza relativa** di **ciascun oggetto** e si **scartano** quelli con **frequenza minore** della **soglia** di supporto s_{min} . Si pone $k = 2$
2. si **generano** gli **itemset** di **ordine k** a partire da quelli di **ordine $k - 1$**
3. si **calcola** il **supporto** di **ciascun itemset** e si **scartano** quelli con **supporto inferiore** ad s_{min}
4. l'algoritmo si **arresta** se **NON** è stato **generato alcun k-itemset**, altrimenti si pone $k = k + 1$ e si procede al **passo 2**

Quindi, la **ricerca termina** quando **k NON cresce più**, cioè quando **NON** si **riescono a creare itemset con supporto maggiore** della soglia oppure quando **k = numero item**

GENERAZIONE REGOLE FORTI

Anche qui vengono eseguiti alcuni step:

1. si effettua una **scansione** della **lista** degli **itemset frequenti** generati nella prima fase. Se la **lista è vuota** si **arresta**, altrimenti sia B il **successivo itemset**, questo **viene tolto** dalla **lista**
2. si **suddivide** l'itemset B in L e $H = B - L$ in **tutte le combinazioni possibili**
3. per **ciascuna regola candidata** $L \Rightarrow H$ si calcola la **confidenza**

$$p = conf\{L \Rightarrow H\} = \frac{f(B)}{f(L)}$$

4. se $p > p_{min}$ la **regola viene inserita** nella **lista delle regole forti**, altrimenti viene eliminata

▼ Lezione 9 - 11/04/2024

ESERCITAZIONE POSTI LETTO

Utilizziamo un open dataset, un CSV sui posti letto negli ospedali italiani preso da dati.gov.it per due motivi:

- il **dataset** contiene **informazioni concrete** sulla sanità nazionale e sappiamo che la data science offre il meglio quando è applicata a dati provenienti dal mondo reale
- i **dati** sono **disordinati, incompleti** e non documentati abbastanza. La prima cosa da accettare per un esperto di dati è che questi sono spesso sparsi e sporchi. Parte del lavoro (forse è la cosa più importante) consiste proprio nel compattarli e pulirli

Dopo aver letto il file CSV utilizzando `pandas`, con l'attributo `dtypes` andiamo a controllare che pandas abbia dedotto correttamente il tipo di dato per ogni colonna

Notiamo che alcune colonne numeriche sono scambiate per testuali (object) e ciò è dovuto al fatto che ci sono dei valori come "**N.D.**" in colonne che dovrebbero contenere solo numeri

Il dataset contiene informazioni relative agli ultimi anni, ma noi limitiamo l'analisi all'anno 2014 e per iniziare facciamo un'**analisi esplorativa** concentrandosi sulla variabile più importante del dataset, cioè "**Totale posti letto**"

Ovviamente è anche utile fare una serie di operazioni di **analisi descrittiva** sul dataset che con pandas è un'operazione rapida grazie al metodo `describe`

Inoltre potremmo sfruttare `pyplot` per realizzare degli istogrammi, ad **esempio** andando a vedere come sono distribuiti i posti letto per regione oppure la distribuzione del numero di posti letto

Da queste analisi, notiamo che la **media** rispetto a questo dataset **NON** è molto **significativa** dato che la **varianza** è molto **elevata**

Ciò è dovuto al fatto che gran parte degli ospedali hanno meno di 100 posti letto, mentre sono pochi quelli che ne hanno 800

Da questo capiamo che la **media** e la **varianza** di una variabile possono dare dei falsi indizi, per cui bisogna **osservare** la **distribuzione** dei **dati**

Invece analizzando il numero di posti letto per regione, notiamo che risultano esserci più posti letto nelle regioni con numero di abitanti più elevati

Per vedere se quest'osservazione è significativa, una cosa che potremmo fare è andare a vedere qual è la **densità dei posti letto rispetto alla popolazione** delle regioni

Però abbiamo il problema che nel nostro dataset non c'è l'informazione relativa alla popolazione, per cui dobbiamo andare a unire il dataset che abbiamo con un altro che contiene i dati relativi alla popolazione

▼ Lezione 10 - 15/04/2024

Abbiamo visto che la ricerca delle **regole associative** serve a valutare se esistono delle dipendenze

Abbiamo utilizzato la **matrice delle transazioni**, composta da 0 e 1

Il modello per ricercare le regole viene costruito in base al dominio

CLASSIFICAZIONE E PREDIZIONE

Classificazione e **predizione** sono processi che consistono nel **creare modelli** che possono essere utilizzati per:

- **descrivere degli insiemi di dati**
- **fare previsioni future**

CLASSIFICAZIONE

In generale, quando abbiamo a che fare con dei dati che **NON** sono **numerici**, si utilizza la **classificazione**

Il **processo** di **classificazione** può essere visto come un processo a tre fasi

Fase 1: ADDESTRAMENTO (TRAINING)

In questa fase si **produce** un modello da un **insieme** di dati di **addestramento**

Per **addestrare** il modello è **necessario** avere un **insieme** di **dati** di cui si **conoscono** già le **classi di appartenenza** delle **singole istanze**

La **classe** per ogni istanza si trova in uno **specifico attributo**, detto **class label attribute** (**attributo classificatore**)

Per addestrare i modelli però è necessario avere un numero minimo di dati per ogni classe, o meglio, devo avere un dataset **bilanciato**

Infatti a volte bisogna utilizzare delle tecniche di **rebalancing**

Proprio perché la classe di ogni istanza è fornita in input nella fase di addestramento, si parla di **apprendimento supervisionato**

Fase 2: STIMA DELL'ACCURATEZZA

Si stima l'accuratezza del modello utilizzando un **insieme** di **test**

Il modello viene quindi verificato fornendo dei dati di cui si conosce la risposta corretta e andando poi a verificare se il modello ha classificato correttamente questi dati

Per questo il modello viene chiamato **oracolo**

N.B: è **fondamentale** valutare il modello con dei dati che **NON** sono quelli di **addestramento**

Soltamente, in questa fase si spende molto tempo, infatti si cerca di cambiare qualche parametro per cercare di ottimizzare il modello, ma ogni cambiamento adoperato sul modello fa sì che bisogna verificare nuovamente l'accuratezza

Fase 3: UTILIZZO DEL MODELLO

Si **classificano** le istanze di **classe ignota**

In pratica, in questa fase siamo di fronte a istanze di cui si conoscono tutti gli attributi tranne quello di **classificazione**

Per utilizzare l'algoritmo ci sono vari metodi, per **esempio** in cloud

In ogni caso, ciò che otteniamo è una **componente** che è **indipendente** dalla macchina e dal software

Questa componente sarà sempre in aggiornamento

CLASSIFICAZIONE CON ALBERI DI DECISIONE

Molto spesso questo tipo di modello funziona bene

Com'è fatto l'albero?

Un **albero di decisione** è un albero in cui:

- i **nodi interni** rappresentano un **test** sugli **attributi**
- i **rami** rappresentano i **risultati del test** al **nodo padre**
- le **foglie** rappresentano una **classe** o una **distribuzione di probabilità** per le classi

Per l'**addestramento** del modello sono previste due fasi:

1. **costruzione dell'albero**
2. **sfoltoimento (pruning)** dell'albero dove si vanno ad identificare e rimuovere quei rami che rappresentano rumore nei dati o outliers

Finita la fase di **pruning**, si può passare all'utilizzo dell'albero, andando a **controllare i valori** degli **attributi** della nuova **istanza seguendo il percorso** che parte **dalla radice** fino ad arrivare **ad una foglia**

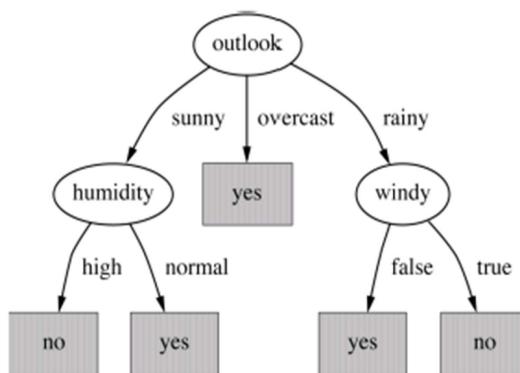
Si arriva alla **foglia** perché questo nodo **rappresenta** la **classe** che il **modello andrà a predire**

Esempio:

Il set di dati weather-nominal

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

Con questo set di addestramento si avrà un albero del tipo:



ALGORITMO ID3

Per ottenere l'albero si utilizza l'**algoritmo ID3**

È uno degli algoritmi storici per l'induzione di alberi di decisione

È un algoritmo **greedy** perché in ogni momento che fa la "mossa", **massimizza la "bontà"** dell'albero

Con quest'algoritmo, l'albero viene costruito in maniera **top-down** usando una politica **divide et impera**

Si parte con un **albero costituito** dalla **sola radice** (alla radice sono assegnate tutte le istanze di addestramento)

Si **sceglie** un **attributo**

Si creano tanti nodi (figli della radice) quanti sono i possibili valori dell'attributo scelto e si distribuiscono le istanze nel ramo corrispondente (le istanze di addestramento sono assegnate al figlio appropriato)

Si procede ricorsivamente usando come radici i nuovi nodi (è come se avessimo costruito una sorta di dataset più piccolo)

CONDIZIONI DI ARRESTO

Ci si ferma quando:

- **tutte le istanze di un nodo appartengono alla stessa classe**, quindi il nodo diventa una foglia con la corrispondente classe assegnata
- **NON ci sono più attributi sui quali dividere l'albero**, quindi il nodo diventa una foglia a cui viene assegnata la classe più comune tra le istanze ad esso assegnate

In alternativa, invece che **assegnare** una **classe** ad un **nodo**, si può assegnare una **distribuzione di probabilità** dell'attributo classe, relativamente alle istanze ad esso assegnate

COME SCEGLIERE L'ATTRIBUTO

Questa è la parte più complessa

Le possibilità sono:

- calcolo del **guadagno di informazione** (metodo utilizzato dall'algoritmo ID3)
- calcolo del **gain ratio**

Sono entrambe misure numeriche che indicano quanto l'attributo è discriminante all'interno del dataset

ENTROPIA

L'entropia è un concetto legato al grado di "disordine" in un sistema

Questo concetto viene utilizzato in vari settori delle scienze, tra cui la **teoria dell'informazione**

L'entropia è quindi legata al concetto di **misura dell'informazione**

Esperimento 1: 4 possibili risultati equiprobabili

Se volessimo memorizzare il risultato su un elaboratore dovremmo utilizzare 2 bit per risultato

Probabilità	Risultati	Codice binario
$\frac{1}{4}$	a	00
$\frac{1}{4}$	b	01
$\frac{1}{4}$	c	10
$\frac{1}{4}$	d	11

Esperimento 2: 4 possibili risultati **NON** equiprobabili

In questo caso si avrà un codice con numero di bit variabili

Probabilità	Risultati	Codice binario
$\frac{1}{2}$	a	0
$\frac{1}{4}$	b	10
$\frac{1}{8}$	c	110
$\frac{1}{8}$	d	1110

La lunghezza media qui è: $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 4 = \frac{15}{8}$

Per il secondo esperimento servono più bit perché **contiene "meno informazione"** rispetto al primo esperimento

È possibile fare meglio per l'esperimento 2? A ciò risponde la **teoria dell'informazione**

ENTROPIA DI UN ESPERIMENTO FINITO

Sia X un esperimento con un numero finito di possibili risultati e_1, \dots, e_i

La probabilità che l'evento e_i si verifichi è p_i

L'entropia dell'esperimento X è:

$$H(X) = H(p_1, p_2, \dots, p_q) = - \sum_{i=1}^q p_i \log(p_i)$$

H è **continua** sulle p_i , quindi piccole modifiche delle probabilità causano piccole modifiche dell'incertezza dell'esperimento

Se X e X' hanno q e q' risultati equiprobabili e $q < q'$, allora:

$$H(X) = H\left(\frac{1}{q}, \dots, \frac{1}{q}\right) < H\left(\frac{1}{q'}, \dots, \frac{1}{q'}\right) = H(X')$$

Più risultati possibili abbiamo, **maggior incertezza** dell'esperimento

GUADAGNO DI INFORMAZIONE

Siano date le classi C_i (che troverò in un sottoinsieme di istanze) con $i = 1, \dots, m$

La classe C_i contiene s istanze. Sia A un attributo con valori $\{a_1, \dots, a_v\}$ e s_{ij} sia il numero di istanze in C_i con $A = a_j$

L'entropia iniziale è: $I(s_1, s_2, \dots, s_m) = - \sum_{i=1}^m \frac{s_i}{s} \log_2 \frac{s_i}{s}$

Possiamo calcolare l'entropia condizionata rispetto ad un attributo:

$$E(A) = \sum_{j=1}^v \frac{s_{1j} + \dots + s_{mj}}{s} I(s_1, s_2, \dots, s_m)$$

Il **guadagno di informazione** per **A** è: $Gain(A) = I(s_1, s_2, \dots, s_m) - E(A)$

cioè è proprio la differenza tra l'entropia iniziale e l'entropia condizionata.

In pratica andiamo a **misurare quanto** l'attributo **A** è in grado di ridurre il disordine all'interno del sistema.

Questo si vuole capire perché quando vogliamo **costruire l'albero di classificazione**, bisogna **scegliere l'attributo** basandoci su quello che ha il **guadagno di informazione maggiore**.

Tuttavia, il **gain** ha un **difetto**, cioè **predilige gli attributi con molti valori diversi**.

Ad **esempio**, è un errore considerare dei codici univoci, perché saranno per forza sempre diversi, ma non porteranno alcuna informazione.

C'è il **rischio di overfitting**.

GAIN RATIO

Se non ci è ancora chiaro se considerare o meno un attributo, si utilizza un altro indice

Il **gain ratio** è una leggera modifica del guadagno di informazione che tiene conto del **numero di valori** degli **attributi**

In particolare, prende in considerazione l'**informazione intrinseca** (numero di valori diversi) del partizionamento creato dall'attributo

L'**informazione intrinseca** di un partizionamento è l'**entropia** delle istanze

Il **gain ratio** di un attributo si definisce come: $GRatio(A) = \frac{Gain(A)}{Intrinsic(A)}$

Il gain ratio potrebbe **sovra-compensare** se si sceglie **un solo attributo** perché la sua **informazione intrinseca** è bassa

Il problema potrebbe essere risolto scegliendo l'attributo con gain ratio maggiore tra tutti quelli con guadagno di informazione superiore alla media

SFOLTIRE GLI ALBERI

L'albero generato può essere troppo specifico per l'insieme dei dati di addestramento (fenomeno detto **overfit**)

Se accade ciò, significa che ci sono troppi rami e alcuni riflettono soltanto il rumore dei dati di ingresso, quindi il risultato è una scarsa accuratezza sulle istanze nuove

Sfoltire l'albero serve a rimuovere questi rami poco affidabili

Esistono due approcci:

- **pre-pruning**
- **post-pruning**

PRE-PRUNING

Consiste nell'**interrompere la costruzione dell'albero prima di costruirlo**

Esempio: ci si può fermare quando il guadagno di informazione dell'attributo scelto è comunque sotto una certa soglia

Il metodo di pre-pruning ha vari problemi, tra cui è difficile capire come scegliere il valore di soglia e l'**early stopping** (un'interruzione prematura della costruzione dell'albero)

POST-PRUNING

Costruisce tutto l'albero e poi rimuove i rami peggiori

È il metodo più comune

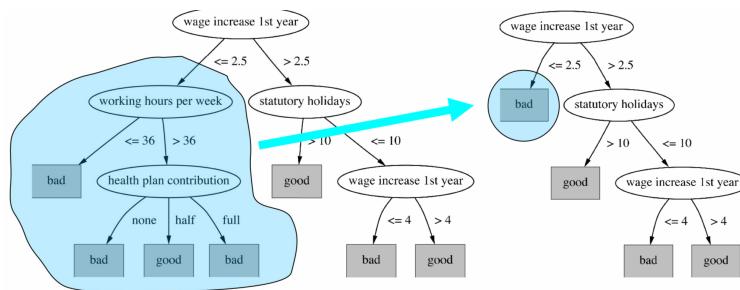
I punti importanti sono:

- le **operazioni** da applicare sull'albero **per sfoltirlo** (**subtree-replacement** o **subtree-raising**)
- il **criterio** per decidere **dove sfoltirlo**

SUBTREE-REPLACEMENT

L'idea è **rimpiazzare un sottoalbero con una foglia**

Peggiora le prestazioni dell'albero sull'insieme di addestramento, ma può migliorarle su un insieme di dati indipendente

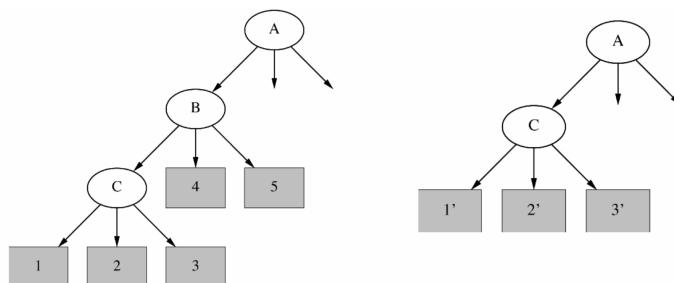


SUBTREE-RAISING

È un'operazione molto più costosa e dalla utilità più discussa

Si va ad agganciare un sottoalbero, all'albero finale

In pratica un nodo intermedio viene eliminato e i nodi foglia del suo sottoalbero vanno riclassificate nel nuovo sottoalbero



Dobbiamo capire **quando effettuare il post-pruning**

Spesso si parla di **reduced-error pruning**, cioè si utilizza un insieme di dati indipendente sia da quello di testing che da quello di training

Si calcola il **tasso di errore** dell'albero **prima e dopo** l'operazione di **pruning** e si sceglie di effettuarlo se il **tasso di errore diminuisce**

▼ Lezione 11 - 17/04/2024

MARKET BASKET ANALYSIS

Nella lezione precedente abbiamo visto l'algoritmo apriori che permette di individuare gli itemset più frequenti

Il dataset sui dati alimentari è

[groceries.csv](#)

Contiene un insieme di transazioni con gli articoli comprati

Le librerie che ci utilizziamo sono:

- NumPy
- Pandas
- Matplotlib
- Seaborn
- CSV Reader
- MLXtend (contiene l'implementazione dell'algoritmo apriori)

Da MLXtend importiamo anche **TransactionEncoder**

Questa è la libreria che ci consente di rappresentare i dati nella forma di una **matrice booleana**

Dopodiché leggiamo il dataset in una **lista**

NON utilizziamo il metodo `read_csv()` di pandas si aspetta un numero di colonne che sia sempre consistente

In questo caso, essendo un dataset strutturato, **ogni riga ha un numero diverso di elementi**

A questo punto, possiamo preparare il **dataframe** per le **transazioni** nel formato richiesto dall'algoritmo **Apriori**

Dobbiamo trovare gli elementi unici, cioè una lista senza duplicati i cui oggetti saranno le colonne del dataframe

Inizialmente creiamo il dataframe **vuoto** e successivamente lo andiamo a popolare di 0 o 1 a seconda se l'oggetto compare o non nella transazione

Costruiamo il dataframe andando ad aggiungere una riga alla volta

Questo è un metodo efficiente in termini di risultato ma non lo è in termini temporali

In MLXtend esiste un algoritmo per fare ciò che è ottimizzato

Per usare il metodo della libreria utilizziamo la classe `TransactionEncoder`

Questa classe permette di costruire una struttura dati che contiene 0 e 1 a seconda se l'articolo è contenuto nella transazione

```
from mlxtend import TransactionEncoder

encoder = TransactionEncoder()
transactions = encoder.fit(groceries).transform(groceries)

transactions = transactions.astype('int')

df = pd.DataFrame(transactions, columns=encoder.columns_)
```

Dopo aver creato il dataframe, possiamo applicare l'algoritmo Apriori, il quale permette di calcolare i frequent itemset, sulla base del supporto minimo (soglia)

L'istruzione sarà:

```
frequent_itemset = apriori (df, min_support = 0.02, use_colnames = True)
```

Aggiungiamo la colonna **length** ad `frequent_itemset`, cioè aggiungiamo la lunghezza usando una **lambda function**

```
frequent_itemset['length'] = frequent_itemset['itemset'].apply(lambda x: len(x))
```

Come troviamo gli articoli più venduti?

Per rispondere facciamo un ordinamento e mettiamo i livelli di supporto in ordine decrescente

Il risultato del sort lo otteniamo dal metodo `sort_values` applicato su `frequent_itemset`

Se invece vogliamo sapere solo gli itemset che hanno un supporto maggiore di una soglia, possiamo fare una query, quindi creare un sottodataset

Dopo trovati gli itemset frequenti, da questi vanno **estratte le regole**

Infatti, per calcolare le regole associative con i livelli di supporto, confidenza e lift c'è bisogno di avere degli itemset frequenti

Per calcolare le `association rules` esiste il metodo `association_rules`

L'output di questo metodo sono proprio le **regole** (fatte da **antecedente, conseguente, supporto antecedente, supporto conseguente, supporto della regola, confidenza, lift**)

L'**indice di lift** è quello che ci permette di capire se si tratta di una **regola forte**

Più lift è grande, più è forte la regola

Per visualizzare meglio le regole estratte, possiamo anche fare un ordinamento su queste per valore di confidenza, lift, ecc

Inoltre, è possibile anche fare filtri concatenati per selezionare ad **esempio** le regole con un supporto minimo e un certo valore di lift

▼ Lezione 12 - 22/04/2024

Riprendiamo a parlare della **costruzione** degli **alberi**

TRATTAMENTO DATI NUMERICI

Vediamo come comportarci quando gli alberi contengono valori **NON discreti**

Dobbiamo fare attenzione quando abbiamo dati numerici **continui**, come ad **esempio** la temperatura perché se facessimo un test sulla temperatura avremmo 3 archi uscenti

Dobbiamo **discretizzare** l'attributo andando a **selezionare** delle **fasce di appartenenza**

Vogliamo quindi ottenere dei **test** per **selezionare e splittare** un **attributo continuo** in uno **discreto**

Si dice che **tra due valori consecutivi** c'è uno **split-point**, cioè un punto che permette di **separare** l'insieme delle **istanze** in **due sottoclassi**

Quindi, dopo aver individuato gli split-point, creiamo dei **test**

Per individuare gli split-point andiamo a considerare l'**andamento** dei **valori** per quell'**attributo**

Esempio: **temperatura**

Innanzitutto, dobbiamo capire com'è distribuito il valore della temperatura

Infatti, se facessimo split-point $> 41^\circ$ e **NON** ci fossero *temperature* $> 41^\circ$, allora **NON** avrebbe senso fare quel test

Quindi potrebbe essere utile un **box plot**, perché ci permette di visualizzare l'andamento dell'attributo

Gli **attributi continui** vanno **necessariamente discretizzati**

Inevitabilmente, ciò andrà a **modificare** il valore dell'**entropia**

Quindi l'**entropia media** dell'attributo va calcolata **considerando il test effettuato**

Poi in base al **guadagno d'informazione**, andremo a decidere come modificare l'albero

MODELLO A REGOLE

Ora, vogliamo costruire un **albero di classificazione** che produce una **classificazione** di **un'istanza NON nota attraverso regole** e **NON** producendo un albero da navigare

NON sono **regole associative**

Una **regola di classificazione** permette di fare una **predizione**, è una formula logica:

| **if <antecedente> then <conseguente>**

Antecedente: è una serie di **test** più o meno complesso

Conseguente: è la **classe** da **assegnare** alle **istanze** che **soddisfano l'antecedente**, è l'output della predizione

Esempio: If outlook=sunny and humidity=high then play=yes

Il problema è come costruire le regole

Esiste un **collegamento** semplice tra **albero di classificazione** e **regole di classificazione**

Quindi, se andassimo a **percorrere** un **albero dalla radice fino alla foglia**, otterremo un **insieme di test**

Creare un percorso da una radice ad una foglia vuol dire **ottenere una regola di classificazione**

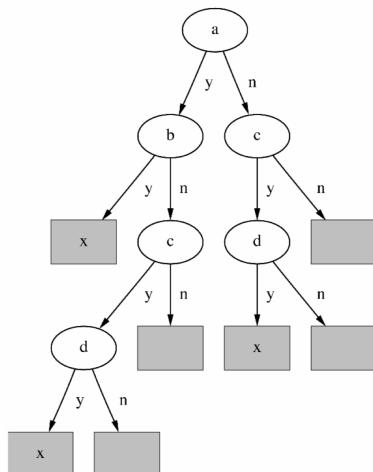
Prendendo **tutti** i possibili **cammini** da un albero, otteniamo un **modello a regole**

Come sull'albero di classificazione, è possibile fare uno **sfoltimento** delle **regole**

Tuttavia, **costruire un modello a regole a partire da un albero NON** è la cosa **migliore**

L'inverso non è mai utilizzato, cioè **NON** si crea un albero a partire da un modello a regole perché si otterrebbe un albero molto complesso

Infatti, anche con solo due regole si ottiene un albero complesso



Albero ottenuto da:

if a and b then x
if c and d then x

In ogni caso, la **conversione va evitata** il più possibile e quindi utilizzare **algoritmi di costruzione opportuni**

La conversione va evitata perché la **spiegabilità** di un **modello** è importante

Esempio: le reti neurali **NON** sono spiegabili

Quindi si vede che il modello ha imparato in quel modo ma non si sa il perché

Le **regole spesso** sono **più spiegabili rispetto all'albero** proprio perché quest'ultimi sono spesso complessi

Tuttavia, avendo a disposizione un albero potremmo ricavare le regole al solo scopo di **spiegare** come si è arrivati al risultato

Per costruire regole si utilizzano algoritmi detti **di copertura**

ALGORITMI DI COPERTURA

Sono detti **di copertura** perché man mano gli algoritmi incorporano sempre più istanze

Si parte da una **regola semplice** e man mano diventa **sempre più complessa** fin quando il nostro insieme di regole non avrà **coperto tutte le istanze**

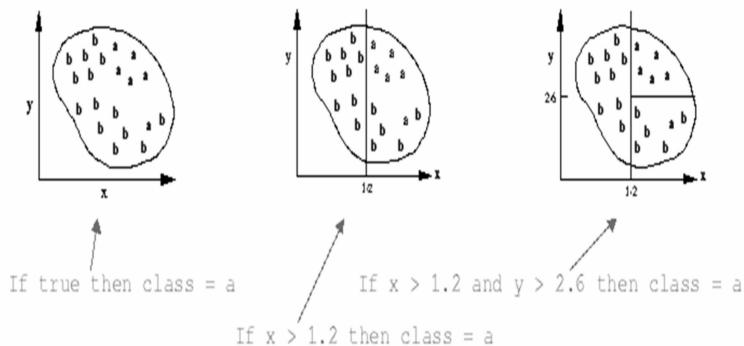
Quante regole bisogna costruire? **Una per ogni classe target**

Esempio: se la variabile **target** è **booleana**, allora **2 regole**

Le regole quindi vengono **raffinate**

Esempio: generazione di una regola

Immaginiamo che le istanze siano in un piano cartesiano



Possiamo dire inizialmente: `if true then class = a`

Questa regola **copre tutte** le istanze però avrà margine di **errore alto**

Supponiamo ora di fare un test su un attributo: `if x>1.2 then class = a`

Abbiamo costruito una regola che **NON** ha **incluso** tutte le **istanze**, ma allo stesso tempo che ha un margine di **errore minore**

Possiamo **raffinare** ulteriormente la regola aggiungendo un altro test: `if x >1.2 and y>2.6 then class = a`

Ora abbiamo un'**accuratezza elevata**, ma **NON** ha **copertura totale**

Spesso ci si ferma ad un certo livello di approssimazione, ma con intelligenza, cioè bisogna sapere che cosa è si lasciato qualcosa fuori

METRICHE

Come capiamo se un modello è buono o no? Abbiamo bisogno di misure

Sia:

- **R** una **regola**
- **t** il **numero** di istanze che **soddisfano** la **premessa (antecedente)** (dubbio: rispetto alle istanze totali di quella classe)
- **p** il **numero** di istanze che **soddisfano** la **premessa e conclusione (conseguente)**

Definiamo:

- **Copertura:** corrisponde al rapporto tra le istanze correttamente predette della classe a (che quindi rispettano l'antecedente) e il numero totale di istanze della classe a.
- **Accuratezza:** rapporto tra p e t in percentuale

Questi parametri vengono **misurati** in **maniera incrementale** per permettere di capire quando fermare il processo di raffinamento

Inizialmente si parte da una **regola di base senza condizioni**

Man mano si **migliora la regola aggiungendo** dei **test** che ne **massimizzano l'accuratezza**

A questo punto, bisogna scegliere **su quali attributi** fare il **test** per **creare** la parte **antecedente** della **regola**

Bisogna utilizzare meccanismi che massimizzano la **copertura** e l'**accuratezza**

La parte **antecedente** della regola viene costruita in maniera **incrementale**

Esempio: lenti a contatto

Sono tutti attributi **nominali** (i **modelli a regole** funzionano **bene** con questi attributi)

Questo modello ci dice quali modelli di lenti a contatto sono consigliate sulla base di alcuni attributi

La variabile **target** ha 3 possibili valori: **hard, soft, none**

La **classe target** è la classe su cui voglio fare la **predizione**

L'**oracolo** è l'**insieme** dei **dati** di una **variabile target**, quindi si **conosce** già il **valore**

L'oracolo è l'equivalente dei test di ingegneria del software

COME COSTRUIAMO LE REGOLE?

Fissiamo un **conseguente**, cioè la **variabile target**

Ora decidiamo il **test** da fare e utilizziamo una **misura di confronto**

Faccio **ogni possibile test** e calcoliamo la **copertura**

Il test che sceglio è quello con il **valore** di **copertura più alto**

Nel caso dell'**esempio delle lenti a contatto** è "astigmatismo = yes"

Quindi la **prima forma** della **regola** è:

| *if astigmatismo = yes then recommendation = hard*

A questo punto, dobbiamo **raffinare** la **regola**

Quindi effettuiamo i **test** sul **sottoinsieme filtrato**

Il **valore di copertura più alto** è dato da **"Tear Prod. Rate = normal"**

Ci sono 6 istanze che superano i nostri due test, ma sono solo 4 quelle che verificano il conseguente, per cui il valore di **accuratezza** è $\frac{4}{6}$ (più alto di prima)

Possiamo fare ulteriori test per raffinare ancora la regola

Proviamo altri test e notiamo che sono due che hanno un valore di copertura più alto, cioè sia **"age = young"** sia **"spectable prescription = myope"** cioè sono rispettivamente $\frac{2}{2}$ e $\frac{3}{3}$

Prendiamo il **secondo test** perché garantisce **copertura maggiore**

La regola diventa:

if astigmatismo = yes and prod rate = normal and spectable prescription = myope then reccomendation = hard

Ora l'**accuratezza** è **pari a 1**

Possiamo quindi provare a definire altre regole, andando a scegliere un'altra classe target e utilizzando questo metodo incrementale

ALGORITMO PRISM

L'algoritmo appena descritto si chiama algoritmo **PRISM** e **genera solo regole perfette**, cioè con **accuratezza del 100%**

È un algoritmo lungo, richiede tempo di calcolo proprio perché produce regole con accuratezza totale

Pseudocodice:

```
Per ogni classe C
    inizializza E con l'insieme di tutte le istanze
    creare una regola R con una parte sinistra vuota che predice C
        finché R è perfetta (o non ci sono più attributi da usare)
            per ogni attributo A non menzionato in R, e ogni valore v
                considera di aggiungere la regola A=v al lato sinistro di R
                seleziona il valore v che massimizza l'accuratezza
                    (in caso di più valori massimali, considera quello che
                     massimizza anche il supporto)
                aggiungi A=v alla regola R
            rimuovi le istanze coperte da R in E
```

Metodi come PRISM sono detti **separate and conquer**, cioè:

- viene identificata una regola
- le istanze coperte dalla regola vengono separate dal resto

- le istanze che rimangono vengono conquistate

La **differenza** con i metodi **divide and conquer** (come gli alberi di decisione) è che i sottoinsiemi separati non devono più essere considerati

CLASSIFICATORI BAYESIANI

La terza famiglia dei modelli di classificazione è di tipo **statistico**

I **metodi statistici NON** permettono di **predire la classe esatta**, ma predicono una **probabilità** che un'**istanza appartiene ad una certa classe**

Anche questi sono modelli **incrementali**, cioè **ogni istanza** dell'insieme di addestramento **modifica** in maniera incrementale la **probabilità** che un'**ipotesi sia corretta**

Infatti, questi modelli vengono molto utilizzati quando si hanno dataset variabili proprio perché se cambiano i dati, il modello si aggiorna di conseguenza

Sono modelli **dinamici**

Quindi, ci consentono di utilizzare un **modello più fine** nella **predizione** e nell'**accuratezza**

La **conoscenza già acquisita** può essere **combinata facilmente** con le **nuove osservazioni**

I classificatori Bayesiani utilizzano il teorema di Bayes

Per cui, sia X un'istanza da classificare e C_1, \dots, C_n le possibili classi, la probabilità che X appartenga alla classe C_i è:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

$P(C_i)$ è la probabilità della singola classe

Si sceglie la classe C_i che massimizza $P(C_i|X)$

$P(X)$ è uguale per tutte le classi per cui non occorre calcolarla

$P(C_i)$ si può calcolare facilmente sull'insieme dei dati di addestramento

CLASSIFICATORI NAIVE

Come si calcola la probabilità condizionata?

Per fare questo calcolo facciamo un'assunzione, cioè assumiamo l'**indipendenza degli attributi**

Quindi, se X è composta dagli attributi $A_1 = a_1, \dots, A_m = a_m$, otteniamo

$$P(X|C_i) = \prod_{j=1}^m P(A_j = a_j|C_i)$$

- Se A_j è **categorico**, $P(A_j = a_j|C_i)$ viene stimato come la frequenza relativa delle istanze che hanno valore $A_j = a_j$ tra tutte le istanze di C_i
- Se A_j è **continuo**, si assume che $P(A_j = a_j|C_i)$ segua una distribuzione gaussiana, con media e varianza stimata a partire dall'istanze di classe C_i , cioè come se fosse una variabile gaussiana con media e varianza note

▼ Lezione 13 - 29/04/2024

Oggi facciamo anche una parte esercitativa in Python

Prima però dobbiamo definire le **metriche** per gli **algoritmi di classificazione**

Andiamo quindi a misurare l'**accuratezza** dei **classificatori**

Supponiamo di avere un insieme di dati e di voler costruire un modello previsionale (a che classe appartengono i dati)

Dopo aver costruito il modello, dobbiamo testarlo

Abbiamo visto che i **dati** vanno **divisi** per l'**addestramento** e per il **test**

Ma come gestiamo i dati?

METODO HOLDOUT

È un metodo con cui si divide l'insieme di dati in una parte usata per l'addestramento e una per il testing

Quindi, una frazione dei dati a disposizione viene messa da parte per fare il test

La **tipica divisione** che si fa è **70%** per il **training set** e **30%** per il **test set**

Si riservano **più dati all'addestramento**

Ciò che è importante è che la **suddivisione** tra **training** e **test** set deve essere **rappresentativa** di **tutte le classi**

Quindi la **divisione** deve essere effettuata con un **criterio di rappresentatività delle classi**

Ciò viene detto **stratificazione**

La **stratificazione** è **facile** quando il **dataset** è **bilanciato**, cioè quando le istanze sono distribuite tra tutte le classi

Esempio dataset sbilanciato: dataset con le classi di rischio delle associazioni, 3 classi (alto, medio, bassa)

Ci sono 100mila istanze e solo 1000 sono di "bassa"

In un caso come questo, l'**addestramento** va gestito per **evitare** che il **modello impari a classificare solo le classi dominanti**

Esistono librerie di Python che risolvono tutti questi problemi, ma in ogni caso dobbiamo avere chiari questi concetti

Ci dobbiamo soffermare sul metodo e non sulla tecnica

Ad **esempio**, creare un'istogramma sulla variabile target ci può far capire se il dataset è bilanciato

CROSS VALIDATION

È una tecnica alternativa al metodo holdout

Si tratta di una tecnica **iterativa**, cioè l'**insieme di dati** viene **diviso in k parti** e l'**addestramento** del modello viene fatto **k volte**

Supponiamo di dividere il dataset in 5 gruppi e sia $k=1$

Addestriamo il modello su i gruppi 2-3-4-5 e testiamo con il gruppo 1

Il **vantaggio** è che si ha la **massima rappresentazione dei dati** perché ogni istanza alla fine di tutto sarà stata utilizzata sia nella **fase di training** sia in quella di **test**

Infine, si valuta l'**accuratezza** del modello facendo una **media** delle **metriche** ottenute ad ogni **iterazione**

Da numerosi esperimenti si è visto che un valore di k ideale è 10

Infatti, la tecnica viene spesso chiamata **10 fold cross validation**

In realtà, il **valore di k ideale dipende dai dati a disposizione**

Dobbiamo quindi provare con valori diversi di k per dire qual è il **folding ideale**

LEAVE ONE OUT CROSS VALIDATION

Questa tecnica un modo per estremizzare la cross validation

Con questa tecnica si sceglie **$k=N$ (numero totale di istanze)**

In questo modo si ha il vantaggio che si fa il massimo uso dei dati a disposizione

Tuttavia, è una tecnica estremamente costosa e richiede di avere a disposizione risorse computazionali adeguate

Viene spesso utilizzata quando il dominio di applicazione è estremamente critico, le risorse di calcolo a disposizione sono numerose e si hanno pochi dati

MATRICE DI CONFUSIONE

Gli errori commessi dal classificatore possono essere visualizzati in una **matrice di confusione**

Questa matrice avrà **tante righe e colonne quante** sono le **classi della variabile target**

Nell'**intersezione riga-colonna** si trovano i **valori della classe i predetti come classe j**

I **valori corretti** si trovano **solo** sulla **diagonale principale**

Quindi per avere una buona situazione, i **valori più alti** devono **trovarsi** sulla **diagonale principale**

Controllando i valori sulla matrice si capisce anche dove intervenire per migliorare il modello

Inoltre, va anche anche considerata la semantica del significato

PRESTAZIONI

Le **metriche** che si utilizzano sono:

- **ACCURACY:** dice con che accuratezza complessivamente sono state predette le classi, è il **rappporto tra predizioni corrette e il numero totale di predizioni**
- **Precision:** dice **quanta informazione di classe 1 predetta come classe 1 è effettivamente di classe 1**, è un'informazione relativa alla singola classe
- **RECALL:** è la **quantità di informazione di classe 1 predetta correttamente**
- **F-MEASURE:** è la **media armonica di precision e recall**, si usa per sintetizzare in un'unica metrica precision e recall

		PREDETTE	
		1 (yes)	0 (no)
VERE	1	TP true positive	FN false negative
	0	FP false positive	TN true negative

Le formule sono:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F\text{-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

ESERCITAZIONE IN PYTHON - ALBERI DI DECISIONE

L'algoritmo è suddiviso in 3 fasi:

- **preparazione** dei dati per l'addestramento
- creazione del modello previsionale (**addestramento**)
- verifica del modello (**test**)

IRIS

Prediamo un dataset di default dalla libreria `sklearn`

```
from sklearn import datasets  
iris = datasets.load_iris()
```

Importiamo "iris"

È un dataset di 130 istanze che sulla base delle caratteristiche di un fiore, ne predice

Le caratteristiche si soffermano sulle dimensione del petalo e del sepalo

Importiamo `numpy`

Costruiamo il nostro **dataset** andando ad **isolarcì** la nostra **variabile target**

```
X = iris.data[:, [2,3]]  
y = iris.target
```

Costruiamo quindi `x` e mettiamo tutte le righe delle colonne 2 e 3

Ad `y` invece assegnamo la **variabile target** (l'ultima colonna di iris è in `y`)

Facciamo la suddivisione tra **training** e **test set**

Per utilizzare il metodo holdout, la funzione è `train_test_split`

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,  
random_state=0)
```

`test_size = 0.3` significa che per il **test set** prendiamo il **30% del dataset di partenza**

`random_state` è l'**indicatore di randomizzazione** che in questo caso **lasciamo a 0** perché il **dataset** è già **bilanciato**

Possiamo vedere:

- `X_test` come le **domande**
- `y_train` sono le **etichette** dell'insieme di train
- `y_test` è l'**oracolo**

Per costruire il classificatore, dobbiamo costruire il modello

Quindi, importiamo il `DecisionTreeClassifier` da `sklearn`

Scegliamo un `max_depth` pari a 5

```
from sklearn.tree import DecisionTreeClassifier  
clf = DecisionTreeClassifier(max_depth=5)
```

Il modello poi si crea con la funzione `fit` (fa il training)

```
clf.fit(X_train, y_train)
```

Dopo quest'istruzione, `clf` **NON** è più un modello **vuoto**, ma uno **addestrato**

Per **validare** si usa la funzione `predict` passando come parametro `X_test`

```
y_pred = clf.predict(X_test)
```

Il **risultato della predizione** lo mettiamo in `y_pred`

Con `y_test` e `y_pred` possiamo costruire la **matrice di confusione**

Con il metodo `accuracy_score` (importato da `sklearn.metrics`) calcoliamo le metriche

Il risultato del modello va accompagnato con le informazioni relative alle condizioni di come si è arrivati a quel risultato

WINE

Prendiamo un altro dataset

Importiamo "wine" da datasets

Questo ha 13 attributi che consentono di classificare il tipo di vino

In particolare, fornisce dati derivanti da analisi chimiche

Il dataset lo importiamo in un dataframe di pandas

Utilizziamo i metodi `head` e `describe` per farci un'idea dei dati con cui dobbiamo lavorare e **conteggiamo i NaN**

Questi **passaggi** sono da fare **sempre, indipendentemente dal dominio applicativo**

Verifichiamo i valori della variabile target e li contiamo per **vedere se il dataset è bilanciato o meno**

Costruiamo una **matrice di correlazione** per andare ad analizzare quali sono gli attributi più correlati alla variabile target

Inoltre, questa ci permette di vedere come sono correlati tra loro gli attributi

In questo modo possiamo scegliere quali attributi portare nell'analisi

L'analisi si fa partendo dalla correlazione tra **variabile target e altre variabili**

Poi si va a vedere anche la **correlazione** tra il **sottoinsieme** che ho scelto

Infatti se, ad **esempio**, ho scelto a e b che sono molto correlati con la variabile target e allo stesso tempo sono molto correlati tra loro, scelgo di continuare l'analisi con uno solo di quegli attributi

Solitamente si esclude quello **meno correlato** con la **variabile target**

Possiamo definire una funzione di **feature selection** che ci **scarta** dal dataset tutte le **feature** che hanno una **correlazione rispetto alla variabile target minore di una soglia**

```
def remove_unrelated_variability_with_target(df_model, target, threshold):
    corr = df_model.corr()
    corr_target = abs(corr[target])
    relevant_features = corr_target[corr_target > threshold]

    print('\nFeatures correlation with target > ', threshold)
```

```

print(relevant_features)
print('\nFeatures correlated: ', len(relevant_features))

relevant_features_col = relevant_features.keys().tolist()

new_df = df_model[relevant_features_col]

return new_df

```

L'**output** di questa funzione sarà un **sottoinsieme** delle **feature** su cui andare a costruire il modello

È importante utilizzare una funzione perché spesso è necessario cambiare la **threshold** per fare più prove

Utilizziamo anche una funzione per la **classificazione**

```

def classification(df):
    x = df.drop('target', axis=1)
    y = df['target']

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=
0.2, shuffle=True)

    print("\nBefore y_train counts:\n", y_train.value_counts())
    print("\nBefore y_test counts:\n", y_test.value_counts())

    smote = SMOTE(sampling_strategy='not majority', random_state=32)

    x_train, y_train = smote.fit_resample(x_train, y_train)
    x_test, y_test = smote.fit_resample(x_test, y_test)

    print("\ny_train counts:\n", y_train.value_counts())
    print("\ny_test counts:\n", y_test.value_counts())

    classifier = DecisionTreeClassifier()
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict(x_test)

```

La funzione **SMOTE** in questo caso va a **campionare** sulla classe **NON maggioritaria** per andare a **bilanciare il dataset**

Qui non era strettamente necessario, ma lo facciamo per capire che c'è quest'opportunità

Visualizziamo con un'istogramma il risultato della classificazione

```

plt.hist([y_pred, y_test], bins=[0, 1, 2, 3, 4, 5], label=['y_pred', 'y_test'],
         align='left')

plt.xticks([0, 1, 2])
plt.yscale('linear')
plt.title('Decision Tree Classifier')
plt.legend()
plt.show()

```

Dall'istogramma capiamo se le predizioni sono corrette

Alla fine andiamo a vedere l'**accuracy**

▼ Lezione 14 - 02/05/2024

CLASSIFICATORI BAYESIANI

Si tratta di metodi **statistici** di classificazione

Questi tipi di classificatori **predicono la probabilità** che una data istanza **appartenga** ad una classe

Si **stima la probabilità** che un'istanza **appartenga** ad una certa classe in base a dati **valori** e attributi dell'istanza **rispetto alle features del modello**

Sono **metodi incrementali**, cioè ogni volta che si introduce **una nuova istanza** nel modello, questa va a **modificare i valori di probabilità** che un'ipotesi sia corretta

Esempio: questi modelli sono spesso utilizzati per il riconoscimento delle mail di spam

Per il **calcolo della probabilità**, questi modelli utilizzano il **teorema di Bayes**

Sia X un'istanza da classificare e C_1, \dots, C_n le possibili classi, la probabilità che l'istanza sia di classe C_i si calcola:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

Si sceglie la classe C_i che **massimizza** $P(C_i|X)$

$P(X)$ è **uguale** per tutte le classi (quindi non occorre calcolarla)

$P(C_i)$ si calcola sull'insieme dei **dati di addestramento**

Quando si costruisce il modello, si danno in **input** le **features** ma **NON** la **classe** di **appartenenza**

Esempi:

- $P(C_i)$ è la probabilità che C_i è una mela indipendentemente dal colore
- $P(X|C_i)$ è la probabilità che X è rossa e rotonda dato che è una mela
- $P(X)$ è la probabilità che un frutto sia rotondo (non la calcoliamo perché è uguale per tutte le classi)

Come si calcola $P(X|C_i)$?

CLASSIFICATORI NAIVE BAYES

L'assunzione che viene fatta è che **gli attributi siano tutti indipendenti tra loro**

Quindi $P(X|C_i)$ sarà il **prodotto delle singole probabilità**

Se X è composta dagli attributi $A_1 = a_1, \dots, A_m = a_m$, otteniamo:

$$P(X|C_i) = \prod_{j=1}^m P(A_j = a_j|C_i)$$

Viene fatta una distinzione se l'**attributo** è categorico o continuo:

- se è **categorico**, $P(A_j = a_j|C_i)$ viene stimato come la frequenza relativa delle istanze che hanno valore $A_j = a_j$ tra tutte le istanze di C_i
- se è **continuo**, si assume che $P(A_j = a_j|C_i)$ segua una distribuzione Gaussiana, con media e varianza stimata a partire dalla istanze di classe C_i

PROBLEMA DELLE FREQUENZE NULLE

Se ci fosse un attributo che **NON** si **verifica mai** per una data classe, la **probabilità** sarebbe **sempre 0** dato che viene calcolata come prodotto delle singole probabilità

Per risolvere questo problema si usa **sommare 1** ai conteggi per le coppie (attributo, valore) che hanno probabilità 0

VALORI MANCANTI

Per le istanze di **addestramento** se abbiamo istanze con valori mancanti, tipicamente le librerie non considerano queste istanze

VANTAGGI E SVANTAGGI

L'assunzione di indipendenza **semplifica** il calcolo

L'evoluzione di questi modelli sono **le reti Bayesiane** che consentono di **considerare le relazioni causali** tra gli attributi

In realtà, si è visto che anche quando l'ipotesi di indipendenza non è soddisfatta, **il classificatore naive Bayes fornisce spesso ottimi risultati**

METODI BASATE SULLE DISTANZE

A partire dai **valori delle singole istanze**, vengono **calcolate le distanze tra quelle da classificare e quelle che già appartengono al modello**

Quindi questi modelli **memorizzano tutte le istanze del set di addestramento** e rimandano tutte le elaborazioni al momento in cui una **nuova istanza** deve **essere classificata**

Il modello non è pronto, si parla di **lazy evaluation**

Un **esempio** di metodo basato sulle distanze è il **k-nearest neighbor**

K-NEAREST NEIGHBOR

Viene chiamato così perché il **valore** viene **predetto** in base alle **istanze più vicine**

La **predizione** sarà la **classe prevalente** tra le **4/5 istanze più vicine** (tipicamente)

Il **numero di istanze vicine da considerare** è proprio il valore di **k**, per cui è a tutti gli effetti un parametro del modello

Anche questo è un metodo **incrementale**

Hanno **accuratezza maggiore** perché viene **sfruttata tutta la conoscenza possibile**, detto **spazio delle ipotesi**

In ogni caso, l'**output** sarà la **classe** a cui è più probabile che l'istanza appartenga, basata sulla maggioranza delle **k istanze più vicine**

Il modello per fornire le risposte, utilizza il concetto di **distanza**

Supponiamo che le istanze siano formate da n attributi (escluso l'attributo classe)

Ogni istanza è un **punto** in uno **spazio n-dimensionale**

Infatti, il modello prevede di costruire la formula per calcolare la distanza considerando ogni istanza come un punto in uno spazio n-dimensionale

La **distanza** si **calcola** con la solita **formula**:

$$d(X, Y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Nasce però la necessità di **normalizzare i dati** in modo da dare a **tutti gli attributi lo stesso peso**
Invece, con gli **alberi** gli attributi **NON** hanno tutti la **stessa importanza**, anzi diamo più
importanza a quelli che hanno guadagno di informazione maggiore

Quando gli attributi sono **nominali** come calcoliamo la distanza?

Si utilizza una **regola booleana**, cioè se l'attributo che arriva è **uguale ad una classe**, allora la
distanza è 0, altrimenti è 1

Per gli **attributi ordinali**, ad **esempio** la temperatura che può essere alta, media e bassa, bisogna
convertire i valori in un tipo **numerico**, quindi **NON** va **considerato** come **nominale**

Questo perché la distanza tra **bassa** e **alta NON** è la stessa tra **media** e **alta**

Ricapitolando, data una **nuova istanza da classificare** si **considerano** le **k** **istanze dell'insieme di addestramento più vicine** e la **classe predetta** è quella più comune tra queste **k** **istanze**

Lo **svantaggio** di questo metodo sono:

- i **costi computazionali** sono **alti**
- gli **attributi irrilevanti vengono pesati come gli altri**, è necessario quindi andare a considerare la correlazione in modo da eliminare alcuni attributi e selezionare un sottoinsieme di attributi importanti, rischiamo quindi di portare nell'analisi attributi che diventano **confondenti**, introducono **entropia** nel modello

Nell'albero emergerà l'attributo più rilevante perché nella costruzione viene calcolato il guadagno

Il **calcolo della distanza** viene fatto su dei **campioni**, cioè su un **sottoinsieme rappresentativo** del **dataset di addestramento**, quando questo è enorme
Per fortuna, questo processo lo effettua la libreria

PREDIZIONE

All'inizio del corso abbiamo visto la **differenza tra classificazione e predizione**

Si parla di **classificazione** quando abbiamo **classi** e **attributi nominali**

Invece, si parla di **predizione** quando si ha a che fare con **classi** e **attributi numerici**

Con gli attributi **numerici** si può fare sia **classificazione** che **predizione**

La **predizione** spesso si fa con il **metodo più semplice** possibile: la **REGRESSIONE**

Molti problemi di predizione si risolvono con la **regressione lineare**

REGRESSIONE LINEARE BIVARIATA

È il metodo più semplice di regressione

Date delle **istanze** con **attributi numerici** X e Y , si vuole determinare Y come **funzione lineare** di X

$$Y = a + bX$$

I **coefficienti** a e b sono detti **coefficienti di regressione**

Per **determinarli** si usa il metodo dei **minimi quadrati** (cioè si **minimizza** la **funzione di errore**)

$$\sum_i (a + bX_i - Y_i)^2$$

dove X_i e Y_i sono i valori di X e Y per l'istanza i -esima

ALTRI TIPI DI REGRESSIONE

- Regressione **lineare multipla**: $Y = a + b_1X_1 + b_2X_2$
- Regressione **non lineare**: $Y = a + b_1X + b_2X^2 + b_3X^3$
Spesso è possibile ricondurre il tutto a un problema di regressione lineare introducendo delle nuove variabili, ad
esempio poniamo $X_1 = X$, $X_2 = X^2$, $X_3 = X^3$ e il problema diventa $Y = a + b_1X_1 + b_2X_2 + b_3X_3$

Riassumendo, la classificazione è un problema studiato diffusamente, principalmente in statistica, apprendimento automatico, reti neurali, data mining, ecc

Tuttavia, la **scalabilità** è uno dei problemi fondamentali nell'uso dei classificatori per grandi insiemi di dati

L'**adattabilità del modello** è **implicita** nei modelli **bayesiani** e **basati su distanza**

Invece un **modello ad albero**, una volta che viene costruito, questo è **chiuso**

Esempio: se quindi ho costruito un modello ad albero con 1000 istanze, ma dopo 6 mesi ne ho 100000, il modello costruito prima non serve più a nulla

Per cui quando si **costruiscono modelli**, bisogna riflettere anche sul **futuro**, cioè interrogarsi su **quanto frequentemente cambiano i dati**, quindi se ha senso costruire un modello chiuso o meno

COMBINAZIONE DI CLASSIFICATORI

Spesso si utilizzano tecniche per **combinare classificatori**, in particolare **quando con uno solo NON si ottengono risultati interessanti**

Analogia con i medici:

supponiamo di voler diagnosticare una malattia, possiamo rivolgerci a vari medici invece che ad uno solo.

Due tecniche:

- **bagging:** prendo le risposte di tutti i medici e considero come diagnosi valida quella prodotta in maggioranza
- **boosting:** peso la diagnosi di un medico in base agli errori che egli ha commesso in precedenza

Composizione semplice ⇒ bagging

Composizione pesata ⇒ boosting

BAGGING

Sia S un insieme di s istanze (il training set) e supponiamo di avere C_T classificatori, la stessa istanza s_i deve essere utilizzata da tutti i classificatori

Per **ogni istanza provo tutti i classificatori e scelgo la risposta che occorre con più frequenza**

Idealmente i **diversi classificatori** devono **imparare dagli stessi dati** e poi si effettua il **voting**

Però questo **sistema è pesante** soprattutto quando il **dataset è grande**, per cui si utilizza un **sottoinsieme** delle **istanze** per l'addestramento

Questo **sottoinsieme** deve essere **rappresentativo** e, inoltre, **bisogna essere sicuri** che **una parte del sottoinsieme sia dato in pasto a tutti gli altri classificatori, altrimenti si avrà un apprendimento esclusivo**

Il campionamento delle istanze si dice che viene effettuato **"con rimpiazzo"**, cioè **un'istanza deve finire in tutti i classificatori**

In pratica, l'**istanza** è come se venisse **duplicata** in modo che sia presente in più sottoinsiemi

BOOSTING

Questa tecnica si **applica ad algoritmi di classificazione** che possono **supportare istanze pesate**

Ogni istanza ha un peso w , inizialmente uguale per tutte

Si **addestra** il classificatore C_t **tenendo conto dei pesi attuali**

Poi si **modificano i pesi** nelle **iterazioni successive** in modo tale che durante l'apprendimento di C_{t+1} **contino di più le istanze classificate incorrettamente** da C_t

▼ Lezione 15 - 06/05/2024

CLUSTERING

In generale, quando abbiamo a disposizione una serie di informazione, dobbiamo capire quale famiglia di algoritmi applicare

Un'analisi di **clustering** ha l'obiettivo di **suddividere l'insieme di dati in gruppi omogenei**

L'**obiettivo** è quindi realizzare una **raggruppamento** tale che gli **oggetti** che **appartengono** allo **stesso cluster** siano **simili**, mentre quelli che **NON appartengono** devono essere **dissimili**

Quindi l'**analisi di somiglianza** deve essere fatta sia **internamente** che **esternamente** al cluster

A differenza della classificazione, in questo caso **NON** esiste una **variabile target**, infatti gli algoritmi di clustering sono detti **unsupervised**

I dati vengono **segmentati** sulla **base** delle loro **proprietà**

BONTÀ DEL CLUSTERING

Due **misure** di riferimento:

- **distanza intraclasse:** la **distanza** nello **stesso cluster** deve essere **minimizzata**
- **distanza interclasse:** la **distanza** tra **diversi cluster** deve essere **massimizzata**

In generale, deve esserci un'**alta similarità intraclasse e bassa similarità interclasse** per essere un buon algoritmo di clustering

Gli **algoritmi** di **clustering** si **basano** sulla **distanza** tra le **istanze**

Il **problema** è che spesso **NON** sono così immediate le **distanze** tra le istanze del nostro dataset, infatti spesso **NON** sappiamo **a priori** **quante classi** troviamo proprio perché sono algoritmi **unsupervised**

Per capire qual è la soluzione migliore, dobbiamo utilizzare le misure per le distanze

Un **buon metodo** di **clustering** richiede:

- **scalabilità** (anche se è un obiettivo di tutti gli algoritmi che lavorano con i dati)
- **abilità di trattare con tipi di attributi differenti** (formati e/o scale diverse)

- **scoprire cluster con forme arbitrarie**, non deve essere un vincolo iniziale
- **minima conoscenza del dominio applicativo**
- **robusto al rumore e outlier**
- non deve essere sensibile all'ordinamento dei record in input, quindi **la storia temporale di acquisizione dati NON deve influenzare l'apprendimento**
- deve poter lavorare con grande quantità di dati
- acquisire i vincoli dagli utenti
- deve produrre dati interpretabili e usabili

TIPI DI DATI

Per effettuare il clustering, si parte da una matrice (con n righe che rappresentano gli oggetti e p colonne che rappresentano gli attributi o features)

Da questa, viene calcolata una **matrice di dissimilarità o delle distanze**

Viene calcolata la **distanza** $d_{i,j}$ per ogni coppia di **oggetti** i e j

Se $d = 0$, allora gli **oggetti** sono **simili**

In generale, **più grande è la distanza più gli oggetti sono dissimili**

La **matrice** delle **distanze** è una **matrice triangolare bassa** perché le **distanze** nella maggior parte dei casi sono **speculari**

Quindi, la **similarità** può essere espressa in **funzione** della **distanza**

Per variabili numeriche la formula è:

$$sim(i, j) = \frac{1}{1 + d(i, j)}$$

COME CALCOLARE LA DISTANZA

La **funzione dipende dal tipo di variabile**

Inoltre, è possibile anche dare dei **pesi** alle **variabili** che possono essere associati al piano semantico dei dati

Tuttavia, se un dato è più importante di un altro lo capisco se ho conoscenza del dominio applicativo

Prima di tutto, dobbiamo **standardizzare (normalizzare i dati)**

Esempio: voglio analizzare la media voto degli esami sostenuti da uno studente, ma ho a disposizione altri dati, come il numero di esami sostenuti

Così come sono non possono essere sommati tra loro perché non sono sulla stessa scala

Ci sono vari modi per **normalizzare**

- **min-max normalization:** dati tutti i valori possibili del dominio dell'attributo, il valore viene scalato in base al rapporto massimo meno minimo

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- **Z normalization:** si utilizza la deviazione standard e la media per calcolare lo z-score, la formula è

$$z = \frac{x - \mu}{\sigma}$$

dove:

- x = valore della variabile
- μ = media
- σ = deviazione standard

DISTANZA PER VARIABILI NUMERICHE

Ci sono varie formule

La formula più generica è quella di **Minkowski**, dove si sceglie **q = numero di attributi**:

$$\left(\sum_i^n |x_i - x_j|^q \right)^{\frac{1}{q}}$$

Se **q = 1**, si parla di distanza di **Manhattan**

Quando i **valori** degli **attributi** di tutte le istanze sono **binari**, allora si parla di **distanza di Hamming**

Se **q = 2**, otteniamo la **distanza euclidea**

N.B: NON tutte le **distanze** sono **simmetriche**

Esempio: differenza tra date

Ovviamente questa cosa non ce la dice nessun algoritmo, dobbiamo essere noi a capirlo

Esempio: prendiamo due insiemi A e B

$$A = \{1, 2, 3, 4\} \text{ e } B = \{2, 3, 4\}$$

Calcolando la distanza $d(A, B) = |A - B| = |\{1\}| = 1$

DISTANZA PER VARIABILI BINARIE

Un metodo è utilizzare una **tabella di contingenza**, la quale, dati due vettori binari, dice **quanti 0 e 1 in comune ci sono** e quanti no

p è il risultato delle somme delle righe e delle colonne sulla tabella

A partire da questa tabella si calcola la misura di **similarità di Jaccard**

Questo indice viene utilizzato per il calcolo della similarità quando si hanno variabili binarie

ALTRI MODI PER IL CALCOLO DELLA DISTANZA

Un altro modo per calcolare la distanza è utilizzare il **coseno**

Si usa quando si ha a che fare con **documenti** e quindi un **vettore** può **rappresentare la frequenza di una parola** in un documento

Concettualmente è uguale alla formula di Jaccard ma cambia il tipo di variabile, infatti qui si tratta di **variabili numeriche**

Un'altra misura per calcolare la distanza è la **matrice di correlazione statistica**

La **correlazione misura la relazione lineare tra due oggetti x e y, rappresentati con vettori di n misure numeriche**

Nella formula appare anche una misura nota come **covarianza**

Le **misure dell'indice variano da 1 a -1**, indicando **correlazione positiva o negativa**

Se si **correlano negativamente**, saranno **più distanti**, quindi in questa misura è importante il segno

Con uno **scatter plot** è possibile visualizzare graficamente la misura di correlazione

DISTANZA PER VARIABILI NOMINALI

Si vedono quanti **match** e quanti **mismatch** ci sono

p = numero colonne

m = numero di match

quindi vado a vedere se corrispondono
se da una parte ho rosso e anche da un'altra allora c'è match

DISTANZA PER VARIABILI ORDINALI

Vanno convertite in una scala numerica e poi vanno standardizzati

DISTANZA PER VARIABILI NUMERICHE RATIO-SCALED

Si tratta di valori numerici misurati su una **scala NON lineare** di valori

Un modo per risolvere è utilizzare una funzione di **regressione**

DISTANZA PER VARIABILI DI TIPO DIVERSO

La complicazione si ha quando bisogna mettere insieme le **misure di distanze tra attributi** che sono **di tipo diverso**

Quando si hanno variabili di tipo diverso, non è possibile usare una sola formula di distanza (come la distanza euclidea) per tutte le variabili senza tenere conto della loro natura diversa

Bisogna invece trovare un modo per combinare le diverse misure di distanza, ognuna adatta al tipo di variabile in questione, in una formula complessiva che rappresenti correttamente la distanza tra due punti

Dati x_i e x_j caratterizzati da p attributi, bisogna utilizzare una **formula pesata** per **rappresentare gli effetti delle distanze** calcolate con metodi diversi

Consideriamo δ come un peso che può essere 0 o 1 a seconda di alcune condizioni specifiche

δ può essere 0 se:

- x_{if} o x_{jf} non esistono nei due oggetti i e j : cioè che per queste variabili non si dispone di valori in uno o entrambi i punti, per cui **NON** si può **calcolare** una **distanza** per questo attributo
- se $x_{if} = x_{jf} = 0$ e l'attributo f è binario asimmetrico, quindi se entrambi i valori sono zero, non si considera una distanza significativa per questo attributo

Altrimenti $\delta = 1$, quindi la distanza tra i punti deve essere considerata

Per rendere più efficiente l'esecuzione di questo tipo di algoritmi, si potrebbe ridurre la dimensionalità per avere una serie di vantaggi computazionali e temporali, ma per farlo è necessario entrare nel dominio dei dati

Similarità e dissimilarità vanno sempre calcolate perché si deve calcolare **sia la distanza intraclasse sia quella interclasse**

METODI DI CLUSTERING

Ci sono vari algoritmi:

- **partitional**: metodo iterativo, in cui si segmenta e si spostano di conseguenza le istanze
- **hierarchical**: creano la **decomposizione** a partire da un determinato **criterio** (ad esempio la semantica degli attributi)
- **density-based**: tutte le **istanze** vengono **suddivise in gruppi che garantiscono di massimizzare la densità all'interno dei gruppi**
- **grid-based**
- **model-based**: si basano su metodi **statistici**

PARTITIONAL VS HIERARCHICAL

Nel caso **partizionale** **NON** c'è **nessuna relazione a priori tra i cluster**, mentre in quello **gerarchico** esiste una **relazione** tra i cluster

Infatti, quelli **gerarchici** sono organizzati in una **relazione di dipendenza**, si rappresentano graficamente con un diagramma Dendrogram

Nel caso **gerarchico**, una **stessa istanza** può **appartenere a più cluster** (**NON** è esclusivo)

Invece il **partizionale** è **esclusivo**

CENTER BASED

Sono i cluster che vengono costruiti a partire da un insieme di punti tali che per ogni punto appartenente ad un cluster, questo punto è più vicino (più simile) al centro del proprio cluster rispetto a tutti gli altri centri

È un tipo di cluster che si basa su un **centroide**

Il **centroide** è il **punto** che rappresenta il **centro** del cluster. Può essere calcolato come la media aritmetica di tutte le coordinate dei punti dati nel cluster

I **punti** dati vengono **assegnati al cluster** il cui **centroide** è **più vicino**, secondo una specifica metrica di distanza (ad **esempio**, distanza euclidea)

L'algoritmo di clustering **iterativamente aggiorna i centroidi e riassegna i punti** dati **finché** non viene raggiunta una **convergenza**, cioè i centroidi non cambiano più significativamente

CONTIGOUS CLUSTER

È un cluster costituito da una serie di punti i quali sono tutti connessi tra loro direttamente o indirettamente attraverso una serie di altri punti nel cluster

In pratica, esiste un percorso continuo che collega qualsiasi coppia di punti all'interno del cluster senza passare attraverso punti al di fuori del cluster

ALGORITMO K-MEANS

L'obiettivo è calcolare **k cluster**

Dato il valore di k, devo trovare un **partizionamento in k cluster che massimizza i criteri scelti**

Viene misurata **iterativamente la distanza intracluster** con metodi di ricerca euristicci

È un **algoritmo center-based**

All'inizio si **scelgono a caso k centroidi** e si vanno a posizionare **le istanze vicino ai centroidi**

Si calcolano le **distanze**

Dopodiché viene **riposizionato il centroide**, cioè si ricalcola il centro all'**interno** del **cluster** e si riposizionano le istanze

E così via...

Ci si ferma quando la misura del centroide non cambia più, si dice che si è stabilizzata la suddivisione

La distanza del singolo punto (istanza) va calcolata k volte, perché va calcolata la distanza dal centroide

▼ Lezione 16 - 09/05/2024

Con Chiara

Per il preprocessing possiamo utilizzare il `LabelEncoder`

NORMALIZZAZIONE

La applichiamo quando gli attributi hanno un range di valori molto vario tra loro

La normalizzazione riporta i valori in un range prefissato

Una delle normalizzazioni più utilizzate è quella **min-max**

Un altro tipo di normalizzazione è la Z-Score, la quale calcola il nuovo valore tenendo conto della media e della varianza

Quale tipo di normalizzazione è corretto usare dipende dai dati

Esempio: la min-max normalization è fortemente influenzata da outliers

ESERCITAZIONE CRM

Il dataset che andiamo ad utilizzare contiene informazioni sui clienti di una piccola azienda

Gli attributi riguardano acquisti e dati personali dei clienti

Gli attributi sulle dimensioni hanno subito una **one hot encoding**

L'obiettivo è prendere il dataset ed analizzarlo per fare un lavoro di classificazione

Dobbiamo quindi descrivere la variabile target, gestire il dataset se sbilanciato

In generale quando ci troviamo davanti una problematica dobbiamo descriverla e poi descrivere il modo per risolverla

Il codice del cliente lo andremo ad eliminare per fare la classificazione perché non aggiunge nessuna informazione utile ai fini della classificazione

Dopodiché inizia la fase di pre-processing

Dobbiamo verificare la consistenza degli attributi, cioè se ci sono dati mancanti, NaN

Uno dei modi per trattare dati mancanti è con `fillna(0)`, ma non sempre è quello giusto

Infatti alcune volte potremmo andare a sostituire con la media, moda, mediana, ecc

Inizia la fase di analisi

Andiamo ad estrarre informazioni utili dal dataset attraverso istogrammi, boxplot

Costruiamo la matrice di correlazione e l'istogramma della correlazione di tutte le features con la variabile target

Questo ci permette di capire se è possibile fare un'operazione di **selezione delle features**

Dal dataset filtrato andiamo a vedere quanto si è speso per l'acquisto fatto a rate e non

La matrice di correlazione si può plottare con seaborn oppure con matplotlib

Dalla matrice notiamo che gli attributi sono tutti estremamente correlati tranne `first_amount_spent` e `number_of_product`

Però è logico che più compro più spendo, quindi non abbiamo nessuna conoscenza nuova

Possiamo vedere che nella fascia di età media, le persone spendono leggermente di più

Possiamo passare alla fase di classificazione

Utilizziamo il metodo **hold-out**, andiamo quindi a **dividere il dataset** in un set per il training e uno per il test

Dobbiamo separare la colonna target

Addestriamo il modello attraverso un meccanismo di voting

`fit` è il metodo di addestramento e infatti passiamo le feature di train e le label

Dalla stampa ottenuta vediamo che l'accuratezza dei modelli è molto varia tra loro

L'obiettivo è quindi trovare qual è il classificatore che si comporta meglio con il dataset

Se trovassimo un accuratezza del 100%, dobbiamo essere sospettosi, perché in quel caso il modello si **overfittato**, cioè troppo legato ai dati che immettiamo

Ci sono anche altre metriche per capire se tenere o scartare il modello

Con la matrice di confusione andiamo a vedere se le predizioni sono corrette o false

Sulle **righe** abbiamo le **etichette vere** e sulle **colonne** quelle **predette**

Sulla **diagonale principale** ci sono le **predizioni azzeccate**, nelle altre posizioni che non sono sulla diagonale principale ci sono le predizioni sbagliate

L'f-score ci dà una misura globale

Una funzione molto utile è `classification_report`

Con questa possiamo andare a vedere come si comporta il classificatore rispetto alle classi

Nell'esempio vediamo che per la classe 1 il classificatore funziona molto bene

Quindi questa cosa è utile per vedere se c'è un bias, o sbilanciamento del dataset

Support ci indica quante istanze ci sono per la relativa classe

HOLD OUT

È il metodo utilizzato per dividere i dati in due insiemi, uno per il train e uno per il test

Il problema che potrebbe esserci è che il campione potrebbe non essere rappresentativo

Ci sono alcune tecniche di hold out che utilizzano la **stratificazione**, cioè si cerca di **mantener le proporzioni** delle classi all'interno del train set

Tuttavia, il problema rimane nel caso in cui le istanze delle classi sono poche

CROSS VALIDATION

L'hold out è una tecnica di valutazione in cui si divide il dataset in due parti

La cross validation funziona un po' diversamente

Si divide l'insieme di dati in k parti

Iterativamente, per k volte, il training sarà costituito da k-1 parti e il testing sarà costituito dalla restante parte

Ad ogni iterazione il modello viene addestrato e testato e poi si cambia il training set

Questo si fa per far sì che il modello veda tutte le parti del dataset, o in generale per evitare overfitting

Tipicamente il valore di k che si sceglie è 10

Ad ogni iterazione verranno calcolate le metriche

Alla fine una media di tutte le metriche

Un'estensione della cross validation è la **leave one out cross validation**, ma non è molto utilizzata perché si sceglie **k uguale al numero delle istanze**

Quindi la computazione è estremamente onerosa

La funzione `precision_recall_fscore_support` calcola tutte le metriche **globali**, cioè viene fatta la media tra le singole metriche delle classi

In generale, trattiamo i dati mancanti perché alcuni classificatori non riescono a gestire i dati mancanti

C'è la possibilità di usare un **imputer**, cioè un modello che si addestra sui valori e poi va a trasformare i dati con una strategia che definiamo noi (può essere media, moda, mediana)

Nel caso in cui avessimo una **variabile target fortemente sbilanciata**, dobbiamo bilanciarla

Le strategie possibili sono due:

- **sovracampionamento**: si creano istanze sintetiche a partire da quelle che esistono già
- **sottocampionamento**: si prende un numero minore di istanze

SMOTE è una tecnica di sovracampionamento

Viene specificata una strategia, ad **esempio** **not_majority** crea istanze solo per la classe minoritaria

LE 3 ESERCITAZIONI

1. Premio dell'assicurazione sanitaria dei clienti
2. Rilevamento frodi nei pagamenti online
3. Marketing bancario

Il consiglio è che se intendiamo fare operazioni all'interno del dataset, dobbiamo vedere che effetti hanno queste modifiche sulle predizioni

Ad **esempio**, dobbiamo vedere se è un'operazione giusta andare a ribilanciare

I dataset delle esercitazioni hanno tutti problemi differenti

▼ Lezione 17 - 16/05/2024

ESERCITAZIONE CLUSTERING

Ricordiamo che siamo nell'ambito dell'**apprendimento NON supervisionato**, quindi i dati **NON** hanno un'**etichetta**

Le principali applicazioni del **clustering** riguardano la **ricerca** di **pattern** comuni che permettono di raggruppare diverse tipologie di dati

Quindi possiamo associare una **label** in base al gruppo a cui appartengono

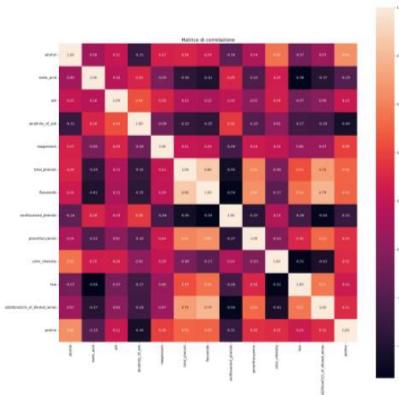
Il clustering consiste nel lavorare senza avere alcuna conoscenza pregressa sui dati

L'esercitazione sul clustering prevede l'utilizzo dell'algoritmo **K-means**

Il dataset che utilizziamo è quello dei vini (13 features)

Dato che vogliamo fare clustering, non andiamo a considerare la **label** che è già presente nel dataset

Dopo aver visualizzato e appreso le prime informazioni sul dataset, costruiamo la **matrice di correlazione** con: `sns.heatmap()`



I valori **più chiari** si riferiscono a valori di **correlazione più elevati**

Dopodiché andiamo a fare il **PAIRPLOT**

È molto simile alla matrice di correlazione

Ad ogni coppia di attributi corrisponde uno **scatterplot**, il quale permette di visualizzare graficamente la relazione tra le due **features**

Andiamo poi a visualizzare la distribuzione con un bloxplot

Notiamo che i **range di valori** assunti dalle features sono **molto diversi**

Per una singola feature (proline) la distribuzione è **molto ampia**

Questo è un problema quando si fa clustering

Per evitare problemi legati alla diversa scala delle variabili nella pratica si usa standardizzare il dataset

Andiamo quindi a scalare i dati (si potrebbe fare anche normalizzazione)

Utilizziamo lo StandardScaler la cui formula è:

$$x_{std} = \frac{x - \mu}{\sigma}$$

- x è il valore attuale
- μ è la media della variabile
- σ è la deviazione standard

Sintatticamente si istanzia allo stesso modo del `labelEncoder`

A questo punto facciamo un altro **bloxplot** per **verificare se il range di variabilità** è più o meno **uniforme**

In questo modo riusciamo a vedere anche **eventuali outlier** (che in realtà **NON lo sono** perché sono comunque vicini alla distribuzione) che prima non riuscivamo a visualizzare

Ora entra in gioco l'algoritmo **K-means**

Lo istanziamo e scegiamo **un valore a caso di cluster da fare**

Addestriamo il modello sul **dataset scalato**

Estraiamo i **centroidi** con il metodo `cluster_centers_`

I centroidi sono rappresentati come **coordinate**

```
kmeans_model = KMeans(n_clusters=4)
kmeans_model.fit(scaled_dataframe)
centroids = kmeans_model.cluster_centers_
```

Stampando lo shape di `cluster_centers_` notiamo che ci sono **4 centroidi con 13 coordinate**

Aggiungiamo quindi al dataframe scalato la label che specifica il cluster di appartenenza

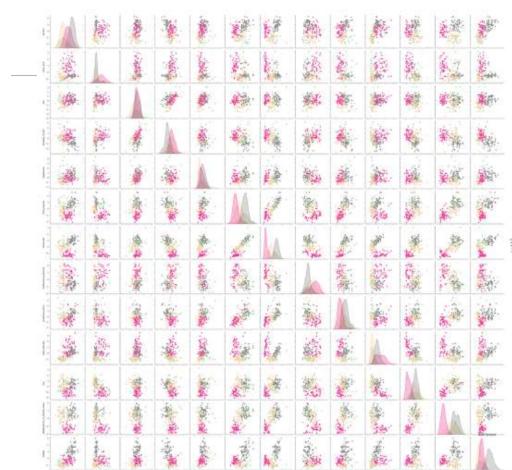
Andiamo a verificare visivamente i cluster ottenuti, con degli **scatterplot**

Scegliamo a 2 a 2 diverse feature e **plottiamo** la distribuzione

Chiara ha scelto le distribuzioni più diverse tra loro

`hue` indica in base a quale variabile bisogna dare il colore

SLIDE PAIRPLOT 2:



Distinguiamo molto bene il cluster rosso, qualche volta anche quello verde

Dobbiamo fare **altre ricerche** per capire se il **numero di cluster è buono**

Facciamo una scelta matematica, valutando l'**indice di Silhouette**

$$s = \frac{b - a}{\max(a, b)} [-1, 1]$$

Questo indice ha un range di **variabilità compreso tra -1 e 1** e **indica** quanto un **cluster è puro**

Cluster puro: distanza intraclass è minima e contemporaneamente la distanza interclasse è massima

Andiamo a rilanciare il K-means tante volte quanto il k_to_test e ogni volta cambiamo il valore del numero di cluster

Alla fine scegliamo il **k** che ci dà lo **score migliore**

Come cosa aggiuntiva, possiamo anche graficare l'indice di silhouette

▼ Lezione 18 - 20/05/2024

DATA WAREHOUSE

Per realizzare un **data warehouse** si lavora in maniera **differente** rispetto ad un **database tradizionale**

Il **magazzino** deve contenere la **storia dell'azienda**, non è che ogni mattina devo andare a consultarlo per capire cosa fare

L'obiettivo è creare uno **strumento** che consente di avere informazioni sulla **storia dell'azienda**

Esempio: si vuole conoscere l'andamento dei voti degli studenti del corso di laurea degli ultimi 15 anni

Questa interrogazione **NON** viene effettuata su un **database** di tipo **operativo**, ma su uno **warehouse**

NON è possibile fare una **query** su un **database relazionale** per andare a prendere la storia degli ultimi 15 anni

Essendo

milioni di record, serviranno **tantissimi join** per arrivare ad avere la **risposta**

In passato, dato che la moltiplicazione dei dati disponibili era più veloce rispetto all'evoluzione del software, si improvvisavano soluzioni
Magari il caporeparto si faceva il suo foglio excel, il marketing un altro
Quindi, spesso non c'era allineamento tra i report

Si è avuta l'esigenza di avere strumenti che siano dedicati proprio a contenere il **magazzino dei dati aziendali** (ciò permette anche di **snellire il database operazionale**)

Quando inseriamo dati in un **warehouse**, allora siamo consapevoli che stiamo creando un qualcosa di storico, nel senso che serviranno ad **analizzare la storia dell'azienda**

Le **interrogazioni su database operazionali** sono **puntuali e attuali**, inoltre **rappresentano un modello di dati** che fa funzionare il sistema software nel **momento in cui viene fatta l'interrogazione**

Invece, le **interrogazioni su data warehouse** sono **più complesse** perché hanno l'**obiettivo di analizzare l'andamento nel tempo**

In teoria, per **ogni sistema software** dovrebbe esistere un **data warehouse**

Per **costruire** un sistema di data warehouse serve:

- un luogo dove archiviare i dati con un modello diverso rispetto a quello relazionale
- un sistema che permette di interrogare questo nuovo modello di dati

Quando si **realizza** un **data warehouse** viene fatta un'**integrazione** con dei **dati esterni**, perché solitamente si **memorizzano** anche **dati di contesto** che **NON appartengono al database operativo** dell'azienda

Serve uno **strumento di analisi** (uno strato software) dove è possibile **includere** dei modelli di **machine learning** per fare delle **predizioni oltre a fare report**

Però, è necessario progettare bene il data warehouse

TERMINOLOGIA

- **Data Warehousing:** strumenti e tecniche di definizione, costruzione, mantenimento del data warehouse
- **Decision Support System (DSS):** sistemi informatici usati nel processo decisionale come supporto all'estrazione delle informazioni da basi di dati organizzate
- **Data Mining:** strumenti e tecniche per l'estrazione dai dati di informazioni "nascoste" (predizioni, regole, suggerimenti) a partire dai dati nei database

DIFFERENZA TRA DATA WAREHOUSE E DATA MINING

Nel data mining spesso non si conoscono le regole che verranno fuori dai dati e/o cosa dedurre (**esempio** il clustering), quindi il data mining ci aiuta a vedere informazioni nascoste

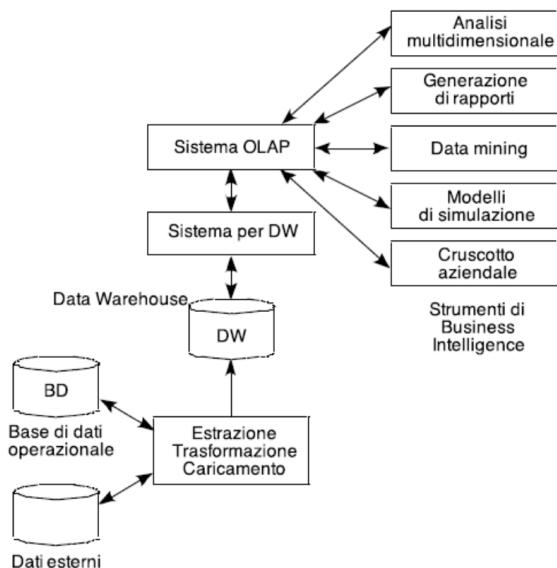
In un sistema di data warehouse invece è diverso, infatti già in fase di progettazione e costruzione del modello bisogna conoscere a che tipo di query/interrogazione si vuole rispondere

In generale, **un data warehouse viene costruito sulla base delle query a cui si vuole rispondere**, quindi i dati vengono organizzati per rispondere a quella specifica domanda

Possiamo dire che per un **data warehouse si parte dalla domanda**, mentre per il **data mining si parte dai dati** e grazie all'analisi di questi si ottengono informazioni

Il **database operazionale** e i **dati esterni** sono **due sorgenti di informazioni** che vengono **immagazzinati** in un **data warehouse** dopo essere stati **estratti, trasformati e caricati**

ETL (Extract, Transform, Load): sono le funzioni che vengono implementate per **estrarre dati** da un database operazionale, per **trasformarli** e **caricarli** in un data warehouse



Al di sopra del repository di data warehouse, si costruisce il sistema software

Ancora più in alto, viene costruito un **sistema OLAP**, il quale è il sistema che **permette** di fare **query** su un data warehouse

Le **caratteristiche principali** di un **data warehouse** sono:

- fornisce la base di conoscenza per **anticipare gli avvenimenti futuri**
- ha una struttura diversa rispetto ai database relazionali, perché **prima di organizzare** i dati si **individuano i soggetti di analisi**, cioè le **domande** a cui si vuole rispondere, infatti tutto ciò che non serve non viene caricato nel data warehouse
- i sistemi warehouse vengono **utilizzati a livello decisionale** dall'azienda
- **storicità**, è la caratteristica fondamentale, infatti **copre un arco temporale più esteso** rispetto alle basi di dati relazionali
- i **dati** sono in **forma aggregata**, ma è possibile avere diversi livelli di aggregazione
- l'**accesso interattivo** è **SOLO in lettura**, si accede in scrittura solo quando viene popolato, dopodiché **NON** si toccano più (**DIFERENZA FONDAMENTALE, chiede all'ESAME e nessuno lo sa**)

MODELLO MULTIDIMENSIONALE

La forma di rappresentazione più utilizzata è il modello **multidimensionale**

Questa è una struttura dei dati che **NON è più una tabella**

Per il tipo di analisi da fare, **è una struttura dati più intuitiva** e ci **consente di fare le operazioni di aggregazione e disaggregazione** in maniera più efficiente

L'analisi gira intorno al soggetto da analizzare, cioè andiamo a costruire il modello sulla base del soggetto

Quindi, si vede ciò che è memorizzato nel database relazionale del soggetto e si prende ciò che serve per costruire il **modello multidimensionale**

Bisogna **valutare in che misura il soggetto partecipa agli eventi**, cioè alle informazioni contenute nel database relazionale

Inoltre, bisogna tenere conto che **uno stesso record di un database relazionale può essere utile per più soggetti**, per cui le **stesse informazioni** possono finire nel data warehouse **più volte**

Quindi, anche qui c'è **ridondanza**

Consideriamo lo **spazio delle informazioni** come un **insieme di matrici multidimensionali**:

- **ogni matrice rappresenta un tipo di evento**
- **ogni elemento della matrice rappresenta un singolo evento descritto da un insieme di coordinate**
- **ogni coordinata rappresenta un soggetto** per le analisi da condurre su quella tipologia di evento

IPERCUBO

È il modello multidimensionale per eccellenza

Esso è una matrice multidimensionale che rappresenta una tipologia di eventi

L'elemento ottenuto specificando un **valore** per **ogni coordinata** si chiama **fatto elementare**

Il **fatto** contiene **misure** numeriche che lo **quantificano**

Ogni **coordinata** dell'iper cubo viene detta **dimensione**

La **misura DEVE essere quantitativa**, quindi è un numero e **NON** può essere una label

Nella descrizione del modello deve esserci l'indicazione di cosa effettivamente rappresenta la misura

Disponiamo i dati in questo modo perché così quando arriva una query, il **dato è già memorizzato nella misura** e **NON** bisogna fare una ricerca nelle tabelle

L'iper cubo va costruito in modo da avere le risposte pronte

Un **fatto** è un **evento** che accade nell'ambito dell'attività e che si ha interesse a misurare

Le **caratteristiche del fatto** sono:

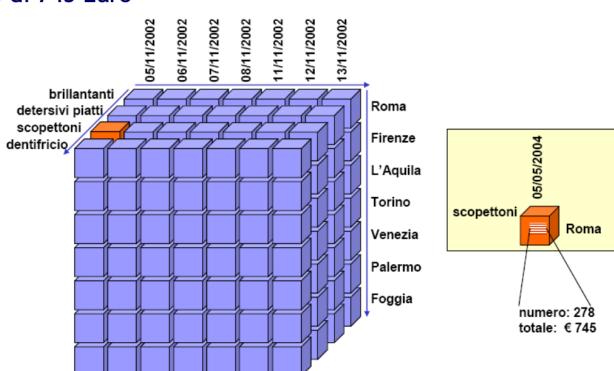
- **dimensioni**, è una scelta vincolante perché servono a collocare il fatto nel **tempo** e nello **spazio aziendale**
- **misure**, quantificano il fatto
- **informazioni descrittive**, vengono aggiunte come note di progettazione

Per **identificare in maniera univoca** un **fatto** serve una **ennupla di coordinate** (perché n dimensioni)

Esempio di **FATTO**:

Il giorno 02/05/2004 a Roma
sono stati venduti
278 scopettoni
con un incasso di 745 Euro

Cubo multidimensionale



La **misura** viene **memorizzata** a livello **più basso di disaggregazione**

Cosa possiamo dire sulle due misure?

Non è necessario memorizzare il totale perché si può calcolare anche a partire dal numero di prodotti venduti

È una **scelta progettuale**

DIMENSIONI

Per quanto riguarda le **dimensioni**, spesso c'è il problema di capire se un **attributo numerico** è un **fatto** o è un **attributo** di una dimensione

In generale, si utilizza questa strategia:

- se è una **misura** che **varia continuamente nel tempo** ⇒ **fatto**
- se è una descrizione discreta di qualcosa che è **ragionevolmente costante** ⇒ **attributo**

Le **dimensioni** che vengono utilizzate sono spesso in comune tra vari contesti applicativi

In genere, il **numero di attributi** per **ogni dimensione** è molto **elevato**

Inoltre, spesso gli **attributi** sono collegati secondo una **gerarchia**

Il **tempo** è **presente** in **ogni DW** in quanto virtualmente ogni DW rappresenta una serie temporale

Inoltre, va gestito con attenzione

Esempio: il giorno può assumere un significato differente se è lavorativo o festivo

Per cui non basta la data, ma servono informazioni aggiuntive

Esempio: non si possono paragonare le vendite dei gelati in estate con quelle dell'inverno

Alcuni tipici attributi della dimensione tempo sono:

- tempo-k (può essere un campo di tipo data in SQL)
- giorno della settimana
- n giorno nel mese
- n giorno in anno
- n settimana in anno
- mese
- stagione
- periodo fiscale

MISURE

È la **caratteristica numerica del fatto** che ne descrive aspetti quantitativi rilevanti per l'analisi

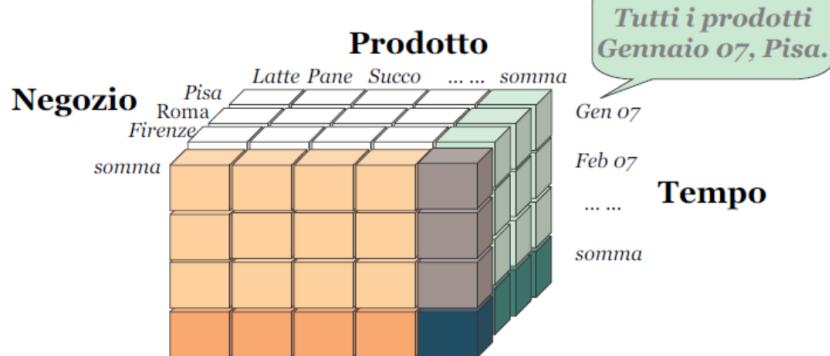
Ogni **fatto** può avere **più misure**

Le misure possono essere:

- **effettive**, cioè memorizzate sulla base di dati
- **calcolate** a run-time utilizzando i valori delle misure effettive
- **implicite**, cioè indicano la presenza o l'assenza di un fatto (simile a booleano)

ESEMPIO: VENDITE

- **Tabella dei fatti:** Vendita
- **Dimensioni:** {tempo, prodotto, negozio}
- **Misura:** numero di unità vendute



Il negozio si troverà in mezzo tra città e cliente

Il collegamento che c'è è proprio la gerarchia

AGGREGABILITÀ

Le **misure NON** sono **sempre aggregabili lungo tutte le dimensioni**

Dai **fatti elementari** si possono derivare **fatti sintetici** se **NON** si considera una o più coordinate

Le **misure dei fatti sintetici** sono ottenute **aggregando** le **misure** dei **fatti elementari** tramite opportuni **operatori** (somma, media, max, min)

Gli **operatori** devono essere **coerenti** con il significato della misura

Infatti, per ogni **coppia (misura, dimensione)** possono essere definiti **operatori di aggregazione diversi**

Quando si parla di **aggregabilità** si fa riferimento alla **possibilità di utilizzare qualsiasi operatore di aggregazione di una misura rispetto ad una dimensione**

Invece, si parla di **additività** quando è possibile utilizzare l'operatore di "somma"

Esempio: "totale prodotti venduti" è aggregabile rispetto alla dimensione "tempo" con la somma

Misura di livello: valore proprio del fatto, è valido nel momento in cui viene registrato il fatto e non può essere sommato nel tempo (**esempio:** la giacenza di un articolo in magazzino)

Misura unitaria: valore relativo ad uno dei soggetti, valido nel momento in cui viene registrato il fatto (**esempio:** percentuale di sconto). In pratica è una misura che viene applicata nel momento in cui viene registrato il fatto e non ha una validità generale. **Mai additiva**

Misura di flusso: valore proprio del fatto rapportato ad un intervallo temporale di riferimento, **è additiva lungo qualsiasi dimensione** (**esempio:** il numero di prodotti venduti in una giornata)

Aggregabilità: esempio

Articolo		Deposito	Data	Misura (Giacenza)
PP1007015	Pannello di Polistirolo 100x70x1,5	Materia Prima	13/02/05	100
PP1007015	Pannello di Polistirolo 100x70x1,5	Ricezione Merce	13/02/05	20
VA1010	Vite in acciaio 10mmx1	Materia Prima	13/02/05	24002
	...			
PP1007015	Pannello di Polistirolo 100x70x1,5	Materia Prima	14/02/05	110
PP1007015	Pannello di Polistirolo 100x70x1,5	Ricezione Merce	14/02/05	0
VA1010	Vite in acciaio 10mmx1	Materia Prima	14/02/05	23870
	...			

La misura Giacenza è **(PP1007015, Materia Prima, 13/02/05).GIACENZA = 100**

Additiva rispetto alla dimensione deposito

(PP1007015, *, 13/02/05). GIACENZA = 120

Non additiva rispetto alla dimensione tempo

(PP1007015, Materia Prima, *). GIACENZA = 100+110?FALSE

(PP1007015, Materia Prima, *). GIACENZA = MEDIA(100,110)=105

Non aggregabile rispetto alla dimensione Articolo

(* , Materia Prima, 13/02/05). GIACENZA = 24102

ADDITIVITÀ DEI FATTI

Esempio: incasso, unità vendute

Sono additivi in quanto si possono aggregare sommando rispetto ad ogni dimensione (somma incassi/unità su tempo, somma incassi/unità su prodotti, ecc)

SEMIADDITIVITÀ DEI FATTI

Si verifica quando le misure sono aggregabili parzialmente

Esempio: "numero clienti" non è un fatto additivo

È possibile effettuare la **somma** di "numero clienti" su "tempo" e su "dipartimenti", ma su "prodotto" genera **problem**i

Supponiamo che i clienti che hanno acquistato carne siano 20, mentre quelli che hanno acquistato pesce siano 30

Il numero di clienti che hanno comprato carne o pesce è un qualunque numero tra 30 e 50

La **soluzione** per gestire la **semiadditività** è cambiare la **granularità** del **database**, andando a realizzare il **dettaglio preciso** della **transazione**

Tutte le **misure** che **memorizzano** un'**informazione statica** sono **semiadditive** rispetto al **tempo** (esempio: bilanci finanziari, temperatura di una stanza)

Ciò che si può **sempre fare** è calcolare la **media** su un certo **periodo di tempo**

NON ADDITIVITÀ DEI FATTI

Esistono anche casi estremi in cui le **misure NON** sono **additive**

Esempio: il costo unitario non ha senso sommarlo a meno che non diventi un incasso (cioè moltiplicato per la quantità di oggetti venduti)

Ricapitolando, le **dimensioni rappresentano l'insieme dei soggetti** a cui si vuole rapportare i fatti di analisi

L'insieme delle **dimensioni** di un **fatto** è importante perché:

- **fissa il livello di dettaglio** delle informazioni che si possono estrarre
- influenza sulla complessità del modello informativo
- determina la granularità con cui è possibile misurare il fatto

Esempio: se il fatto è una vendita e le sue dimensioni sono data, cliente, articolo, la granularità minima ci consente di sapere la vendita di un articolo A in un certo giorno al cliente X ma non ci consente di sapere se X ha acquistato prima 3 pezzi e poi 2

Il **dominio** della **dimensione DEVE essere finito**, infatti se il dominio è continuo o infinito deve essere reso **discreto e finito**

▼ Lezione 19 - 27/05/2024

Riprendiamo con i **data warehouse**

Le **query** eseguite sono diverse rispetto ai database operazionali e inoltre la **modalità di accesso è solo in lettura**

I dati sono rappresentati in maniera diversa, infatti si utilizza il **modello multidimensionale**

MODELLO MULTIDIMENSIONALE

È scelto come forma di rappresentazione perché consente di organizzare meglio le query rispetto al soggetto di analisi

Questo modello viene utilizzato per ottenere informazioni di tipo **quantitativo**

La misura ci dice in che modo i diversi soggetti concorrono nell'evento

Ogni matrice rappresenta un tipo di evento

Si possono includere più matrici multidimensionali

Il modello multidimensionale di riferimento è l'**iper cubo**

Bisogna specificare le dimensioni della matrice (**esempio**: cliente, articolo, tempo)

Il fatto elementare è il valore che si ottiene specificando le coordinate per ogni dimensione

La misura è sempre quantitativa

DIMENSIONI

Sono tutte presenti nel modello

Si può prevedere a priori che alcune possono essere trascurate, cioè quando si fa una query non vengono specificate

GERARCHIE

Si definisce gerarchia l'**insieme di attributi dimensionali collegati gerarchicamente ad una dimensione**

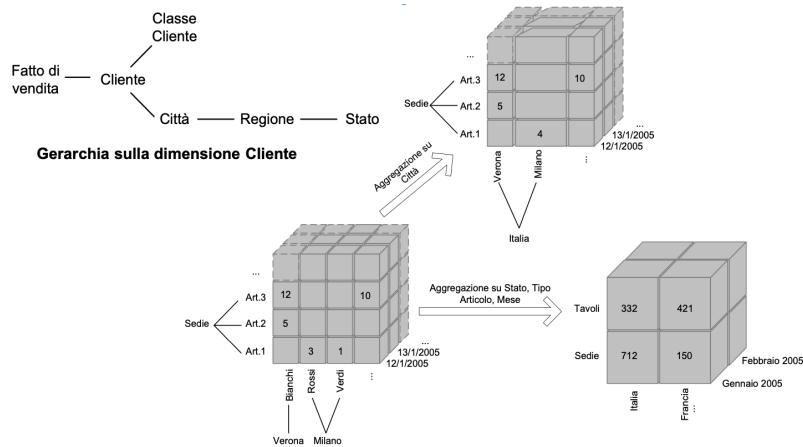
Per **ogni dimensione individuiamo** un **insieme di attributi**, non uno singolo

Gli **attributi dimensionali**:

- sono utilizzati per **aggregare i fatti elementari**
- sono determinati **univocamente dal valore della dimensione**
- rappresentano classificazioni della dimensione

L'analisi dei fatti di un cubo può essere condotta attraverso viste che utilizzano attributi dimensionali

Esempio:



Un cliente può essere visto rispetto alla città o regione o stato

Gli **attributi devono essere legati tra loro**

La **gerarchia consente** di fare le **operazioni di analisi (aggregazione e disaggregazione)**

DIMENSIONI OPZIONALI

Un attributo dimensionale potrebbe anche non essere definito per alcuni valori della dimensione

L'opzionalità deve essere nota per **mantenere consistente** la base di dati a supporto del data warehouse

GERARCHIA CONDIVISA

Può esistere anche una **gerarchia condivisa**

È la gerarchia utilizzata in fatti diversi per descrivere il ruolo di una certa classe di soggetti oppure è utilizzata per descrivere nello stesso fatto ruoli diversi di una stessa classe di soggetti

CARATTERISTICHE STRUTTURALI DELL'IPERCUBO

Sono:

- **multidimensionalità:** si accede alle informazioni tramite ennuple di coordinate
- **granularità:** è il grado minimo di aggregazione dei dati nel DW, corrisponde ad un fatto elementare
- **arco temporale:** è l'intervallo di tempo coperto dai dati nel data warehouse

- **profondità storica:** capacità di mantenere anche l'informazione storica sui soggetti, oltre al valore attuale

CARATTERISTICHE FUNZIONALI DELL'IPERCUBO

Sono:

- **integrazione dei dati:** i dati provengono da più sorgenti
- **accessibilità:** devono avere uno strato software per fare query, ottenere report
- **flessibilità e sintesi:** hanno una buona capacità di assecondare l'utente nell'articolare richieste, aggregare dati e mettere in relazione misure secondo criteri di selezione non prefissati
- **correttezza:** quando si trasferiscono i dati nel data warehouse si verifica se rappresentano effettivamente la realtà
- **completezza:** il sistema ha la capacità di rappresentare tutti gli eventi importanti per le analisi

DATA WAREHOUSE E DATA MART

Essendo un **data warehouse** un **unico contenitore**, anche se **costituito** da **più ipercubi**, dato che può raggiungere dimensioni estremamente elevate, spesso nelle analisi l'utente opera su una **porzione** del DW, detta **data mart** che **contiene** solamente i **dati di suo interesse**

Data mart: porzione del DW che contiene **solo i dati** che servono a **rispondere** ad un certo **tipo** di **analisi**

Attenzione: si possono creare data mart solo dopo avere creato un DW e, inoltre, NON è che aggregando più data mart si ottiene un data warehouse

Nel **data warehouse** vanno individuati **tanti ipercubi quanti sono i soggetti di analisi**

ARCHITETTURA DEI SISTEMI DI DATA WAREHOUSING

Dobbiamo prevedere tutte le componenti che ci servono a mettere insieme l'usabilità delle informazioni memorizzate

I **livelli sempre presenti** sono:

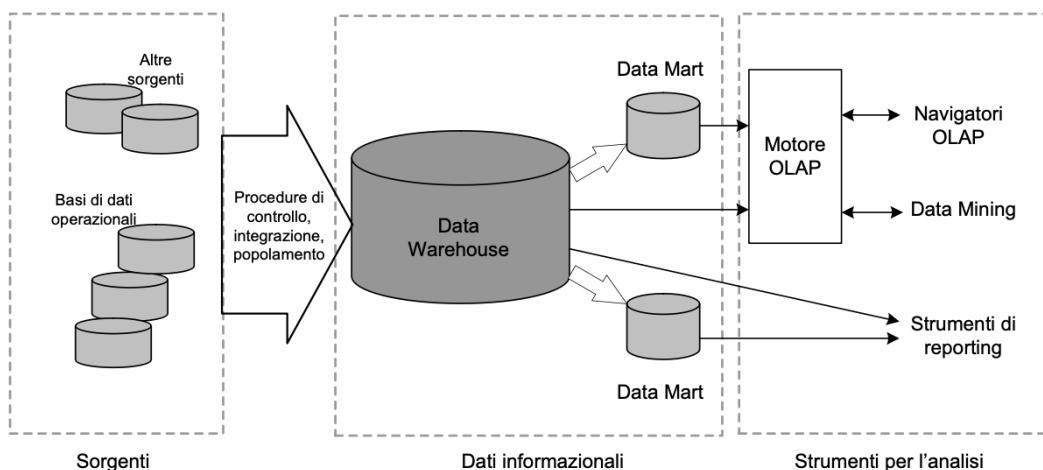
- **sorgenti**, cioè le basi di dati di origine (operazionali o esterne)
- **staging area** (è opzionale), l'area utilizzata per archiviare i risultati intermedi della trasformazione dei dati
- **data warehouse**, base di dati centrale che contiene tutti i dati necessari all'analisi articolati su un modello unificato concettualmente multidimensionale
- **data mart**, basi di dati multidimensionali su cui si appoggia l'analisi

Le architetture nei sistemi di DW si differenziano per il **numero di livelli di memorizzazione** delle informazioni a partire dalla fonte fino al DW:

- **architettura a due livelli:** sorgenti, data warehouse e data mart
- **architettura a tre livelli:** in più hanno la **staging area**

In **entrambe** le architetture, ci sono gli strati **software** per le procedure **ETL** e gli **strumenti** per fare **analisi** (esempio: reporting, tipicamente per riportare il risultato dell'analisi su un browser, motore OLAP)

La parte di **ricerca delle regole, clustering**, può essere **applicata a valle di un sistema di data warehouse aziendale**



MODELLO CONCETTUALI PER IL DW: IL DFM

La modellazione di un data warehouse avviene attraverso un modello chiamato **DIMENSIONAL FACT MODEL (DFM)**

Il modello è uno solo per ogni ipercubo

Quindi, il **singolo DFM rappresenta la specifica progettuale di 1 ipercubo**

Più DFM rappresentano un data warehouse

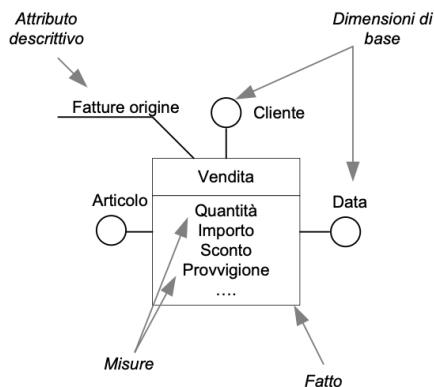
SCHEMA DI FATTO

Il **fatto** è rappresentato con un **rettangolo**, il quale contiene il **nome** del fatto e le sue **misure**

Ci sarà un **pallino** collegato direttamente al rettangolo **per ogni dimensione** dell'ipercubo (dimensione di base)

Possono essere specificati attributi descrittivi che vengono collegati con una linea orizzontale al rettangolo, questi attributi non vengono utilizzati nelle query

Le misure si trovano all'interno del rettangolo



Le **gerarchie** dimensionali sono rappresentate come **alberi** che hanno **radice** nelle **dimensioni di base**, infatti gli **attributi dimensionali** sono i **nodi** dell'albero

Esempi di gerarchie:

- legati al tempo (data esatta, data mese, anno)
- cliente-area geografica (cliente, città, regione, stato)
- cliente-professione
- cliente-livello di scolarizzazione

Nei fatti compaiono i **membri di livello più basso**

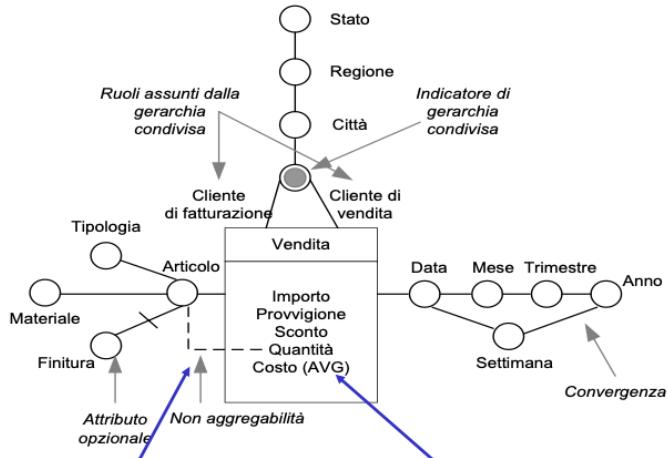
Quindi nel **tempo**, il primo cerchio che si trova è la **data**, poi c'è il **mese** e poi l'**anno**

Se è presente un **trattino** sull'**arco**, allora l'**attributo è opzionale**

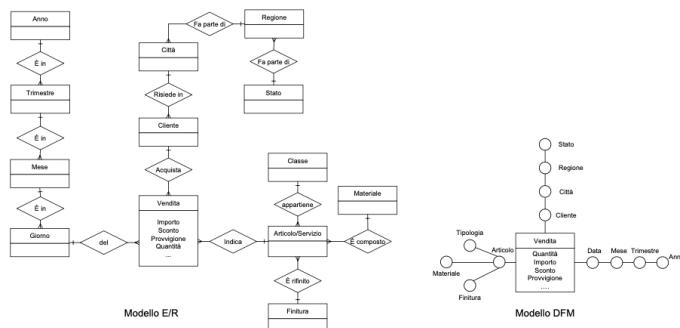
Quando **NON** viene specificato l'**operatore di aggregazione**, allora è **additivo**

Se invece **NON** è un attributo **additivo**, è **necessario specificare quali operatori** di aggregazione si possono utilizzare

Inoltre **NON tutti** gli attributi sono **aggregabili**, ciò si indica con una **linea tratteggiata**



Esempio: corrispondenze con il modello E-R



PROGETTAZIONE CONCETTUALE DI UN DFM

Gli approcci per progettare a livello concettuale un DFM sono due:

- **basato sui requisiti:** è la strategia ideale, viene stilato un progetto i cui requisiti devono essere estrapolati da interviste condotte presso l'utente, però in questo modo viene rimandato il problema del collegamento tra lo schema concettuale prodotto e le sorgenti operazionali che poi andranno utilizzate
- **basato sulle sorgenti:** prima di realizzare lo schema, ci si fa un'idea sui dati che si hanno a disposizione e si osserva anche la struttura delle sorgenti, infatti il modello concettuale del DFM viene derivato proprio dagli schemi che ha il database operazionale (spesso dall'E-R)

PASSI DI PROGETTAZIONE

Bisogna definire i fatti

Per ogni fatto

1. **si costruisce l'albero degli attributi:** si potrebbe pensare di astrarre il modello E-R che si ha a disposizione e collegare direttamente gli attributi, cioè ci si libera del vincolo di capire se un attributo è un'entità o una relazione

2. **editing dell'albero degli attributi:** è un processo di analisi puro in cui si modifica l'albero sulla base dell'obiettivo che ha l'analisi. Questo passaggio è abbastanza soggettivo, mentre gli altri passi sono più forzati
3. **definizione delle dimensioni**
4. **definizione delle misure**
5. **creazione dello schema di fatto**

Esempio: Vendite, dall'E-R a modello concettuale del DFM

Come prima cosa, dobbiamo **individuare i concetti di primaria importanza**, cioè i **fatti**

I **fatti** sono **concetti di interesse primario** per il processo decisionale

Sullo **schema E-R**, un **fatto** può **corrispondere** ad un'**entità** oppure ad un'**associazione** n-aria tra entità

Invece, sullo **schema relazionale**, un **fatto** corrisponde ad una **relazione**

Ogni **fatto identificato** diviene la **radice** di un **nuovo schema**

A questo punto, **creiamo l'albero degli attributi**

All'inizio, sarà un **albero** base **con solo la radice** a cui poi saranno **collegati tanti nodi quanti sono gli attributi diretti** dell'entità o relazione e **tanti sottoalberi quanti sono le entità o relazioni collegate alla foglia**

La costruzione dell'albero degli attributi è **meccanica**

La **radice** sarà proprio il **soggetto di analisi**

EDITING DELL'ALBERO

In genere, non tutti gli attributi dell'albero sono d'interesse per il DW, quindi l'albero può essere manipolato per eliminare i livelli di dettaglio non necessari

Ci sono due strategie:

- **potatura:** si decide di **eliminare** un **sottoalbero** perché gli attributi ad esso collegato non sono di interesse per l'analisi
- **innesto:** viene utilizzato quando, sebbene un vertice esprima un'informazione non interessante, è necessario **mantenere** nell'**albero** i suoi **descendenti**. In pratica, si prende una parte inferiore di un sottoalbero e si porta ad un livello diverso

Se un vertice opzionale viene innestato, tutti i suoi figli ereditano il trattino di opzionalità

DEFINIZIONE DELLE DIMENSIONI

Le **dimensioni** devono essere scelte nell'albero degli attributi tra i **vertici figli della radice**; possono corrispondere ad attributi discreti o a intervalli di valori di attributi discreti o continui. La loro scelta è cruciale per il progetto poiché definisce la granularità degli eventi primari.

IL DATA WAREHOUSE SENZA IL TEMPO NON ESISTE, NON SERVE A NIENTE

Quindi se il tempo NON c'è, va inserito

DEFINIZIONE DELLE MISURE

Se tra le dimensioni compaiono tutti gli attributi che costituiscono un identificatore dell'entità fatto, allora le misure corrispondono ad **attributi numerici** che siano **figli della radice** dell'albero. In caso contrario le misure devono essere definite applicando, ad attributi numerici dell'albero, funzioni di aggregazione.

In questo modo si crea lo **schema di fatto**

Gli attributi che non verranno usati per l'aggregazione possono essere contrassegnati come descrittivi

Ricapitolando, i passaggi **soggettivi** sono:

- **scegliere il nodo radice**
- **editing dell'albero**

▼ Lezione 20 - 30/05/2024

Per potere effettivamente creare un DW, dobbiamo scegliere quali tecnologie utilizzare

In particolare, bisogna scegliere quale **DBMS** utilizzare per **ospitare** il nostro **iper cubo**

NON esiste un'unica soluzione

Esistono DBMS che sono in grado di ospitare **direttamente la matrice**

Oppure la **matrice (iper cubo)** può anche essere **simulata** e quindi è possibile utilizzare un DBMS **relazionale**

Un altro aspetto da considerare è che se si utilizza un DBMS **proprietario multidimensionale**, allora ci sarà un **linguaggio proprietario** per le **query**

Invece, utilizzando un **DBMS relazionale** è possibile fare le **query tradizionali SQL**

Ovviamente le due scelte progettuali hanno vantaggi e diversi

ARCHITETTURE OLAP

L'obiettivo è fare analisi OLAP

OLAP (On-Line Analytical Processing): si tratta di un insieme di tecniche software per l'analisi interattiva e veloce di grandi quantità di dati, che è possibile esaminare in modalità piuttosto complesse

Questi sistemi si riveleranno molto utili per l'ottenimento di informazioni di sintesi. Queste ultime avranno lo scopo di supportare e migliorare i processi decisionali aziendale

ROLAP

R = Relational

Questa tipologia di sistema OLAP è la scelta più **immediata** perché **tutti** i programmatore conoscono i **database relazionali** e **SQL**

Vantaggi:

- **minima occupazione di spazio**
- elevata **conoscenza** degli **strumenti relazionali** da parte degli operatori

Svantaggi:

- l'esecuzione delle **query** è poco **efficiente**
- per migliorare la **velocità** è necessario **aumentare la complessità**

MOLAP

M = Multidimensional

La **struttura** dei **fatti** viene realizzata **direttamente** con un database di tipo **multidimensionale** e l'accesso è **posizionale**

Le **interrogazioni** sono **immediate**, ma il **linguaggio** è **proprietario**

Vantaggi:

- elevata **efficienza delle query**, infatti se il modello è fatto bene, bisogna leggere un solo dato, senza fare raggruppamenti
- **stretta aderenza al modello concettuale** perché si realizza proprio l'**iper cubo**

Svantaggi:

- **spazio occupato notevole**
- **NON** c'è uno **standard**, Oracle ha la sua soluzione, Microsoft Server la propria

Esistono dei tentativi di standardizzazione, ma non ancora non si è arrivati alla definizione di uno standard come SQL

HOLAP

H = Hybrid

È la soluzione intermedia

Per costruire il **data warehouse** viene usato il modello **ROLAP**

Invece per i **data mart** viene utilizzato il modello **MOLAP**

Si utilizza **MOLAP** per i **data mart** perché questi sono segmentati dato che contengono meno dati (solo quelli di interesse per un tema), per cui l'**occupazione di spazio** dell'ipercubo è limitata

Lo **schema concettuale** di un modello **MOLAP** di solito viene definito con un linguaggio **XML**

COME SI SIMULA UNO SCHEMA MULTIDIMENSIONALE IN UN DATABASE RELAZIONALE?

L'obiettivo è simulare un ipercubo su una base di dati relazionale

Esistono vari schemi, il più famoso è quello **a stella**

SCHEMA A STELLA

Viene creata **una tabella per ogni fatto** e ci sarà **un campo per ogni misura e una chiave esterna per ogni dimensione prevista nell'ipercubo**

Quindi ci saranno tante tabelle quante sono le dimensioni

Nella **tabella delle dimensioni** ci sarà **un attributo dimensionale per ogni elemento della gerarchia**

La **gerarchia** è quindi **denormalizzata** dentro la tabella, la radice è l'attributo a granularità più bassa

Il **vantaggio** di questo schema è che garantisce la **massima velocità** nel **reperimento** delle informazioni (**NON** bisogna fare **tanti join**)

Gli **svantaggi** sono la **ridondanza** (gerarchie sono duplicate), **spazio occupato**, complessità del ragionamento

SCHEMA A FIOCCO DI NEVE

Rispetto allo schema a stella, questo schema **riduce la denormalizzazione** della **tabella delle dimensioni** andando a **rendere esplicite** alcune **gerarchie**

Vantaggi:

- c'è una chiara separazione logica sui soggetti
- minor sensibilità alle variazioni logiche delle gerarchie nel tempo

Svantaggi:

- velocità di risposta alle interrogazioni minore

Con questo schema si parla di **COSTELLAZIONE**

Le **tabelle dimensionali** possono essere **condivise da più tabelle dei fatti**

ANALISI OLAP

Abbiamo visto le strutture multidimensionali perché l'obiettivo è fare delle **interrogazioni OLAP**, le quali **consentono di esplorare i dati guidati da ipotesi**

In ogni sessione di analisi, le interrogazioni operano per differenza rispetto all'interrogazione precedente

In pratica, spesso viene fatta **una prima interrogazione e poi si va ad approfondire una dimensione**

Per questo motivo, spesso si parla di **passo di navigazione**

I risultati di queste analisi spesso sono riportati in **tabelle** o in forma **grafica**

OPERATORI OLAP

- **DRILL DOWN**: scende di dettaglio lungo la gerarchia perché **aggiunge** una **dimensione** di analisi
- **ROLL UP**: operazione opposta al drill down, va ad aggregare una gerarchia, cioè va ad **eliminare** una delle **dimensioni** di analisi
- **SLICE**: fissa il valore di una delle dimensioni di base per analizzare la porzione di dati filtrati così ottenuta, è **un filtro**
- **DICE**: fissa due dimensioni, molto simile a slice, ma anziché prendere una fetta, va a prendere un **sottocubo**, un sottoinsieme di agenti e di articoli, è **un filtro su due dimensioni**
- **PIVOT**: va a **invertire** la **relazione tra le dimensioni** realizzando una rotazione del cubo nell'analisi

CICLO DI VITA DEI SISTEMI DI DATA WH

Ricordiamo che non si tratta di un sistema statico, la costruzione è **dinamica e iterativa**

Si costruisce **un primo ipercubo relativo al fatto più significativo** e poi piano piano vengono **integrati altri ipercubi**

Se necessario vengono creati i data mart