

Riconoscimento e Recupero - Appunti

Lezione 1

Pattern Recognition:

Estrarre informazioni, sono tecniche che sono alla base di molti programmi.

Cos'è la pattern recognition?

Il procedimento che porta l'essere umano a rispondere a domande del tipo: individuare classi di oggetti, riconoscere oggetti, distinguere oggetti, è la pattern recognition.

Riconoscere, associare, distinguere e identificare. Sono tutte informazioni legate al concetto di **categoria**. Il processo che ci porta a questo è la Pattern Recognition, legato al concetto di classe, categoria, gruppo → Prendo in ingresso un insieme di dati, effettuo un'analisi di tali dati per rispondere ad una domanda legata al concetto di categoria.

Pattern: immagine, suono, odore, è il dato che viene analizzato, l'entità di interesse.

Definizione

Il processo che prende in input dati grezzi ed effettua un'azione sulla base della categoria dei dati.

Il processo quindi interagisce con il pattern, come fa l'uomo automaticamente (ma lo fa tramite processi complicati che non sono ora completamente chiari)

Spostandosi nella prospettiva informatica: **voglio che questo sia automatico** → scrivere un algoritmo che è in grado di fare pattern recognition. Il problema è stato studiato a lungo, ma cose che per l'uomo sono banali per una macchina sono molto difficili.

Perché?

C'è molta variabilità → completamente diversi per dare una definizione corretta e universale.

Oggetti della stessa classe possono essere molto diversi ma oggetti di classi diversi possono anche essere molto simili.

Gli esseri umani hanno sensori migliori, anni e anni di evoluzioni hanno portato i nostri sensi a sviluppare il modo migliore per poter analizzare i dati → devo quindi avere dei **sensori** in grado di riconoscere i pattern. I "nasi elettronici" ad esempio sono lontanissimi dall'accuratezza del nostro naso: ad esempio ci sono dei macchinari in grado di riconoscere la qualità dell'olio, ma non si riesce ancora a riconoscere tutta la gamma degli odori.

Gli esseri umani inoltre hanno informazione d'insieme, globale. Il calcolatore no, vede ogni pixel, vede ogni numero, ma non riesce a vedere l'insieme. **Il calcolatore non vede un contesto, ma matrici di numeri e questo è estremamente difficile.**

Scrivere un algoritmo che riconosce le immagini è molto più complicato → devo considerare molte variabili.

Esempi classici

Distinguere diverse persone sulla base del volto, pattern: la parte dell'immagine che contiene la faccia. Riconoscere che anche cambiando le variabili di una persona (umore, espressione, ecc.): l'uomo ha la capacità di riconoscere i pattern anche tramite informazioni parziali, cosa che il calcolatore non ha.

Riconoscimento del parlato, delle impronte digitali o riconoscimento di gesti.

Esempio in cui il calcolatore è migliore dell'uomo: biometria basata sul riconoscimento delle impronte digitali. In realtà negli ultimi anni ci sono diversi casi, soprattutto perché il calcolatore non si stanca (task che richiedono tempo e fatica).

Biometria è la scienza che permette di riconoscere un individuo sulla base di tratti caratteristici.

Dopo l'11 settembre c'è stato un boom di investimenti su quel versante.

Altro esempio: riconoscimento di scene o di categorie di luoghi. Classificazione di video, (primo es. quello della classificazione dei pixel in bianco e nero a seconda del si muovono o meno - secondo es. fare tracking dei movimenti, es. riconoscere se mi trovo in una situazione anomala tipo tentato furto di auto in un parcheggio).



Problema principale:

Capire e modellare i diversi pattern di un problema (cercare di caratterizzarli in termini di classi/gruppi/categorie) → **paradigma principale** (come lo risolvo?): **apprendimento da esempi**. La conoscenza deriva da un insieme di esempi campionati dal problema (training set).

Per un calcolatore più che la definizione, funziona **l'esempio**. La cosa fondamentale è che, dati gli esempi, il calcolatore ha costruito la sua conoscenza, sa riconoscere anche pattern al di fuori del training set: ha imparato.



Obiettivo principale

GENERALIZZAZIONE: capacità di generalizzare e riconoscere anche oggetti non presenti nel training set.

Si fa solitamente un'operazione per costruire un modello (disegnare un algoritmo), si fa un'operazione di **ottimizzazione**.

Devo inventare una funzione:

model ← **max E(T, P, Theta)** dove *T* è training set, *P* è informazioni a priori e *Theta* è parametri.

Il tutto viene formulato tipicamente con un algoritmo.

Devo

- definire la E:
 - compromesso tra la capacità di spiegare il training set e la complessitàes. la funzione E da trovare per fittare il polinomio, ad esempio trova la funzione minima che si avvicini ai punti del training set.)
 - ottimizzare la E
-
- Aspetti pratici
 - accuratezza
 - requisiti computazionali (tempo e spazio)
 - flessibilità
 - usabilità (quindi l'accettabilità è anch'essa un requisito fondamentale)

Problemi principali in PR

- Classificazione
- Detection
- Clustering

Occorre costruire un modello a partire dai dati.

Aspetti principali

La realizzazione di un sistema di pattern recognition implica la soluzione di alcuni problemi:

- **Rappresentazione**

Come rappresento in modo digitale gli oggetti? Devo trovare un modo per farlo.

- **Costruzione del modello**

Supponendo di aver raccolto un training set, devo capire come costruire il modello

- **Testing**

Dal problema faccio **campionamento**, che mi permette di portare l'oggetto all'interno del calcolatore (foto, scansione, spettrometro, ecc.) → operazione di pulizia

Quando costruisco il modello, uso delle informazioni a priori, faccio un addestramento e ottengo dei modelli. Nel testing uso dei modelli addestrati per estrarre le informazioni necessarie.

Rappresentazione



Obiettivo: trovare una rappresentazione digitale per gli oggetti del problema in esame.

Cos'è una *feature*?

Una singola misura, l'insieme delle misure è il pattern. (Es. un'immagine che è pattern, ogni pixel è feature). Un'immagine è una matrice, un insieme di pixel (puntini) che posso vedere come un insieme di valori. Il valore mi indica quanta luce colpisce quel punto. A seconda della risoluzione ho un numero di puntini, raffinando la rappresentazione posso estrarre sempre più feature (es. larghezza, altezza), estraendo queste feature ho molte più informazioni su cui confrontarmi → questa operazione si chiama **Preprocessing** (es. riduco le dimensioni)

Costruzione del modello



Problema: costruire un modello in grado di spiegare i dati del training set (generalizzare).

Lo scopo fondamentale è astrarre.

Paradigma di apprendimento da esempi (basato su training set) si basa su:

- le misure

Training Set

Il training set deve essere:

- largo (molti pattern)
- completo (tutte le categorie sono rappresentate)
- variabile (deve tenere in considerazione la variabilità dei pattern nelle categorie)

Non basta un buon metodo, serve un training set adeguato.

Esempio: **classificazione**, nel training set so esattamente per ogni oggetto a che classe appartiene. Vado ad estrarre le feature per ogni training set e posso rappresentare il tutto nello spazio. Costruire un classificatore vuol dire essere in grado di dividere le due classi. Per poter poi classificare un oggetto, estraggo le feature nello stesso modo in cui facevo col training set e uso le feature come avevo fatto in precedenza, nel modello dato dal training set. pattern recognition supervisionata (ho info a priori)

L'obiettivo ovviamente è classificare correttamente.

Esempio: **detection**, voglio capire se appartiene o meno ad una determinata classe. La costruzione del modello è simile alla classificazione.

Esempio: **clustering**, insieme di oggetti su cui non ho alcuna informazione a priori, devo riuscire a distinguere delle classi tra quegli insiemi, scoprire i gruppi in cui posso dividerli. Il problema è molto più difficile: pattern recognition non revisionata (no info a priori), non posso misurare la correttezza del risultato.

Il concetto di cluster è molto vago: dipendentemente dalle misure di similarità utilizzate, così cambia il risultato. Ci sono varie sottocategorie che sono possibili e buone. Il problema è NP-completo.

L'informazione a priori è fondamentale per ottenere risultati che abbiano un senso.

Quindi la costruzione del modello può essere *supervisionato* o *non supervisionato*.

Interpretazione dei risultati

Una volta costruito il modello, devo usarlo.

Il focus è l'interpretabilità: dei metodi e delle soluzioni. Soprattutto nel contesto della biomedicina, l'interpretabilità è fondamentale, bisogna sempre cercare di usare e produrre metodi e soluzioni interpretabili.

La soluzione che ha valore è la soluzione in cui l'utente è a proprio agio e capisce cosa sta succedendo, soprattutto nel contesto biomedicale.

Ad alto livello pattern recognition e machine learning sono la stessa cosa. In realtà ci sono delle differenze: pattern recognition viene dalla fisica (dati reali e focus sulle soluzioni), machine learning viene dalla matematica (il focus è sulle proprietà dei metodi).

Pattern Recognition e Bioinformatica

Perché ha senso usare queste tecniche nella bioinformatica?



LA MOTIVAZIONE PRINCIPALE: la caratterizzazione di una popolazione in termini di gruppi/classi/categorie può essere utilizzata per inferire alcune proprietà di oggetti sconosciuti guardando ad oggetti conosciuti nello stesso gruppo.

Problemi classici di bioinformatica:

- in bioinformatica ci sono molti problemi di clustering, classificazione e detection
- Possibilità di derivare modelli per i dati tramite esempi
- Ci sono problemi di classificazione che possono essere automatizzati
- **Gene Search**

Io devo estrarre un gene dall'interno del genoma ed è un problema di detection.

- **Microarray** per capire l'espressione di un determinato gene

Analisi dell'espressione e della regolazione genica

Di solito vengono fatti diversi esperimenti, differenti condizioni di crescita, soggetti diversi e malattie diverse. Posso fare classificazione di campioni.

Clustering

Trovare geni con pattern di espressione simili.

Si hanno poi delle matrici in cui ogni riga ha un diverso livello di espressione. Questa matrice viene analizzata tramite classificazione di campioni, oppure tramite clustering (ad esempio coregolazioni, ecc.)

- **Motif Discovery**

Vado a cercare all'interno della mia proteina modelli che si ripetono (pattern simili nelle sequenze). Anche questo è un problema di detection

- **Protein Remote Homology Detection**

Caratterizzare proteine omologhe ma hanno corrispondenza di sequenza molto bassa. Devo quindi caratterizzare ogni classe di sequenze omologhe (in senso remoto)

- **Filogenesi**

Una tecnica di riconoscimento di pattern che ci permette di analizzare più organismi. Devo inferire le relazioni a livello genomico tra le specie.

- **Genomica comparativa**

Filogenomica ha lo stesso obiettivo della filogenesi ma viene effettuata a livello di genoma

Altre motivazioni:

Costruire un modello a partire da un insieme di dati. Posso usare questi modelli ad esempio per generare dati simulati (es. problema scarsità dei dati)

Ci son problemi dell'automazione che possono essere automatizzati. Posso imparare come uno specialista esegue determinate operazioni e mi addestro tramite un training set "annotato".

Esempio: qualità degli spot dei microarray.

Approccio tipico: annotazione manuale da parte di esperti. Costruisco un modello in grado di replicare questo comportamento.

Input: immagine, vado ad estrarre le feature che mi interessano. Costruisco quindi un classificatore che stabilisca la qualità.

Lezione 2

Rappresentazione

Cosa vuol dire fare rappresentazione = Voglio rappresentare gli oggetti in modo digitale, voglio una rappresentazione che il calcolatore possa capire.

Per fare rappresentazione devo prima fare campionamento.

Campionamento

Trovare una rappresentazione digitale dell'oggetto, raccogliere i dati.

Dato grezzo: il dato che esce direttamente dal sensore. Questo è già una rappresentazione, ma ovviamente migliorabile.

Esempio: costruire un modello di determinati pesci.

Esempio: supponendo di voler fare filogenesi, per rappresentazione intendo tirare fuori una sequenza (un file di testo) che mi indichino i componenti della sequenza. Da organismo → a sequenza: il campionamento consiste nel sequenziare.

Esempio: classificazione del volto (3 diverse tipologie di sensori: fotocamer tradizionale, 3D e telecamera a infrarossi)

In entrambi i casi però l'obiettivo è: partire da un oggetto e arrivare ad una rappresentazione digitale.

Devo tener conto anche del fatto che l'informazione può essere o meno **interessante** (es. fotocamera a infrarosso per il riconoscimento dei volti).

Problemi da tenere in considerazione:

- **Frequenza di campionamento**

Il campionamento dipende strettamente dalla problematica (e tipicamente è deciso dall'esperto).

Il problema vero e proprio: possiamo avere rumore o info di scarto per le informazioni che ci arrivano dai dati grezzi.

Soluzione:

Elaborare i dati che vengono dal sensore. Estraggo delle feature e costruisco il pattern.

La **feature** è una misura che io faccio, dai dati che derivo. Il **pattern** è l'insieme di features.

Per risolvere un problema posso avere opzioni diverse, con pro e contro. Ogni volta dipende dalle cose di cui ho bisogno, capire qual è l'opzione corretta da applicare.

Classificazioni più ricche possono essere più difficili da estrarre e da modellare.

esempio: rappresentazione compatta per distinguere i pesci vs modellamento della forma dei pesci (non più uno spazio vettoriale)

Classificatore più semplice

Nearest Mean Classifier

Classifica in base alla media più vicina

La scelta della feature è chiaramente cruciale (deve essere rilevante, discriminante, misurabile)

Feature

È una misura che io effettuo e può essere di diversi tipi (discreta, continua, valori binari, valori nominali).

Costruzione del pattern

Mettere assieme le varie features.

Esempio: è diverso avere $[h, l]$ o $[l, h]$ o ancora $[h * l]$. **Il pattern è come metto assieme varie features.**

Tipi di Pattern

- **Vettori - dati vettoriali**

Abbiamo un insieme prefissato di features, messe in ordine, fondamentale ma arbitrario.

L'oggetto viene rappresentato come un punto in uno spazio d-dimensionale, detto "spazio delle features", dove d è il numero delle features (con 2 lo posso vedere, con 3 lo posso immaginare,...)

La scelta è cruciale e bisogna scegliere adeguatamente.

Usando molte features, gli spazi diventano molto grandi e possiamo avere problemi: **Curse of dimensionality**

Più sono le misure che faccio, migliore è la mia capacità di riconoscere l'oggetto: per il calcolatore è un po' diverso. Aggiungendo delle features il mio problema si incasina! ho problemi computazionali ma non solo:

Ad esempio: usando 100 features, userei 100 oggetti per stimare 100 valori. Comincio ad avere dei problemi nella stima.

- **Sequenze**

Si presentano in forma ordinata e sequenziale (uno dopo l'altro): l'ordine è fondamentale e non è arbitrario.

Ad esempio: sequenze temporali, l'evoluzione e l'ordine sono dati dal tempo, oppure sequenze non temporali: nucleotidiche e amminoacidiche (ordine fondamentale ma non dato dal tempo)

- **I grafi**

Rappresentano un insieme di nodi collegati da archi.

Esempio: modellare le diverse parti di un corpo umano.

Esempio: protein-protein interaction, oppure percorsi metabolici

- **Gli insiemi**

Pattern estremamente non strutturato, una collezione non ordinata di dati a cardinalità variabile

Vettori vs altri pattern

Tipicamente pattern compressi sono più espressivi, ma molte tecniche si applicano meglio ai vettori

Preprocessing

- Concetto di scala
- Data standardization
- Data trasformation
 - riduzione della dimensionalità

Scala

Significatività relativa dei numeri.

Riconoscere la scala è fondamentale anche per capire la relazione tra due pattern.

Esempio: **distanza euclidea**

Soffre se le features sono a scala diversa: problema infatti è che spesso le variabili che descrivono un oggetto non sono nella stessa scala

La distanza euclidea viene usata in tantissimi algoritmi. In che modo soffre?

Con 3 punti in uno spazio bidimensionale ad esempio:

Scala x : $[0 - 2]$

Scala y : $[0 - 20]$

A : $[1, 10]$

B : $[1, 12]$

C : $[2, 10]$

Calcolando la distanza euclidea:

$$d(A, B) = 2$$

$$d(A, C) = 1$$

A è più simile a C , ma sul grafico A è più simile a B quindi la risposta non è vera: A e C sono più lontani di A e B . Qual è il problema? **La differenza in una direzione sulle scale usate nel grafico sono troppo differenti.**

Come risolvere il problema della scala?

Data Standardization

Produce dati senza dimensionalità, facendo in modo che tutta la conoscenza su scala e locazione dei dati venga persa dopo la standardizzazione

Approcci di Data Standardization

Notazioni:

- **Dataset X**

$$X = \begin{bmatrix} x_{1,1} & x_{...} & x_{1,n} \\ x_{...,1} & x_{...} & x_{...} \\ x_{d,1} & x_{...} & x_{d,n} \end{bmatrix}$$

n punti in uno spazio d -dimensionale

$$A = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} 4 & 5 \end{bmatrix}$$

$$C = \begin{bmatrix} 7 & 8 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$$

- **Media** lungo la direzione j (j ccva da 1 a n)

$$\bar{x}_j = \frac{1}{N} \sum_{i=1}^N x_{ji}$$

- **Range**: massima estensione delle feature


$$R_j = \max_{x_{ji}} - \min_{x_{ji}}$$

- **Deviazione standard**: dispersione rispetto alla media

$$\sigma_j = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_{ji} - \bar{x}_j)^2}$$

Lezione 3

Data trasformation

 Cercare di creare una rappresentazione più adeguata.

Differenza con la standardizzazione: nella data standardization tutte le operazioni sono implementate dimensione per dimensione, nella data transformation le operazioni agiscono su tutte le dimensioni contemporaneamente (tipicamente in modo lineare)

Esempio più semplice: supponendo di avere un punto $o \begin{bmatrix} x & y \end{bmatrix}$ suppongo di avere un nuovo spazio in cui la dimensione è $\begin{bmatrix} x + y & x - y \end{bmatrix}$, la prima nuova feature non dipende solo dalle x del dataset ma anche dalla y .

Costruisco nuove feature sfruttando le precedenti

Esempio

Schizofrenia: esiste un cambio strutturale nel cervello? In questo caso la domanda è posta in questi termini: abbiamo un po' di pazienti e le loro risonanze magnetiche e l'obiettivo era discriminare tra sani e malati.

Provo a vedere se riesco, visualizzando i dati, a fare una discriminazione. Non posso fare visualizzazione solo dei volumi.

Approccio classico:

Trasformazione lineare dello spazio delle features.

Ogni nuova feature è una combinazione lineare di tutte le feature precedenti

Una combinazione lineare si può scrivere come prodotto tra vettori.

La trasformazione lineare dello spazio delle features è quindi una moltiplicazione tra matrici

$$Y = A^T * X$$

Questa matrice mi dice come fare a trasformare i dati.

X contiene gli oggetti del problema (per colonna), A contiene i vettori utilizzati per la trasformazione (una colonna per ogni dimensione), Y è lo spazio dopo la trasformazione

Per fare la riduzione della dimensionalità ho due approcci:

- **non supervisionati** → uso solo i dati (cerco di comprimere)
- **supervisionati** → uso anche l'informazione di cosa voglio fare
 - Es. Problema di classificazione, non voglio estrarre in generale features, ma estrarne di funzionali al mio task e uso questa informazione all'interno del modo in cui riduco la dimensione.

Ci sono moltissime tecniche che fanno questa cosa:

vediamo la **PCA**.

Principal Component Analysis

Approccio lineare non supervisionato.

Obiettivo: trovare la matrice A ($Y = A^T * X$)

Supponiamo di avere una nuvola di punti

La componente principale ha più valore: ha una varianza più ampia. Perché? Due punti che hanno una componente dello stesso valore, una volta effettuata la riduzione delle dimensioni, collasano sulla stessa rappresentazione. Ciò succede di meno se ci sono dati più variabili → con una varianza maggiore.

Idea principale: l'informazione più importante da mantenere è la varianza dei dati!

- Estrae le direzioni di massima varianza dei dati.
- Mantiene anche la maggior aderenza ai dati originali
- Minimizza lo scarto quadratico medio tra i dati originali e quelli ricostruiti

La direzione migliore per la **pca** è quindi quella a massima varianza (ragionando anche sul range si arriva allo stesso risultato)

esempio

Compressione a perdita: perdo informazioni nella compressione (per ridurre al massimo la descrizione dell'oggetto) → es. jpeg

La differenza tra l'immagine originale e quella compressa, mi misura quanta informazione ho perso. Se misuro l'errore quadratico medio, la PCA è la scelta migliore per minimizzare lo scarto quadratico medio (non posso fare meglio della PCA)

Come funziona la PCA?

La **prima** componente che estrae è quella a massima varianza.

La **seconda** componente principale è quella ortogonale alla prima, di massima varianza.

Come si realizza PCA?

Esiste un **algoritmo** specifico che ci permette di trovare direttamente la matrice di trasformazione A che si basa sulla creazione di autovalori e autovettori della matrice di trasformazione.

- Dataset X

$$X = \begin{bmatrix} x_{1,1} & x_{...} & x_{1,n} \\ x_{...,1} & x_{...} & x_{...} \\ x_{d,1} & x_{...} & x_{d,n} \end{bmatrix}$$

n punti in uno spazio d -dimensionale

$$A = \begin{bmatrix} 3 & 4 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 7 & 8 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

$$X = \begin{bmatrix} 3 & 7 & 1 & 3 \\ 4 & 8 & 2 & 4 \\ 0 & 3 & 4 & 2 \end{bmatrix}$$

Obiettivo: trovare uno spazio ottimale di dimensione L , con $L < d$

Lo spazio è definito come:

$$Y = A^T * X$$

quindi **trovare A**.

Nel particolare quindi voglio ridurre la dimensionalità a L dimensioni.

Nel nostro caso ho $d = 3$, se voglio arrivare a $L = 2$, voglio:

$$Y = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$$

Riduco a due le dimensioni, mantenendo n invariato ovviamente.

A dovrà quindi essere:

$$A = \begin{bmatrix} a^1 & b^1 \\ c^1 & d^1 \\ e^1 & f^1 \end{bmatrix}$$

Algoritmo PCA

1. Primo passo:

calcolare la media lungo ogni direzione

$$U_m = \frac{1}{N} \sum_{n=1}^N x_{m,n}$$

Questo vettore U contiene tutti gli elementi:

$$U = \begin{bmatrix} u_1 \\ \dots \\ u_d \end{bmatrix}$$

2. Secondo passo:

centrare i dati

Voglio far sì che la media diventi zero

$$B = X - Uh$$

dove $h = [1 \quad \dots \quad 1]$ e sono n valori

3. Terzo passo:

trovare la matrice di covarianza

$$C = \frac{1}{N-1} B * B^T$$

$$\frac{1}{N-1} \sum (x - \mu_x)(y - \mu_y) \leftarrow \text{per ricordare la covarianza}$$

4. Quarto passo:

trovare autovalori e autovettori di C

Definizioni per ricordare: $Cx = \lambda x$, con $x \neq 0$, allora x = autovalore di C e λ = autovettore di C .

$$C = dxd$$

ho d autovalori $\lambda_1, \dots, \lambda_d$

e d autovettori v_1, \dots, v_d

5. Quinto passo:

ordinare gli autovalori

6. Sesto passo:

costruire la matrice A

La matrice si ottiene con gli L autovalori corrispondenti agli L autovalori più grandi

Le colonne di A in numero sono pari alle dimensioni che voglio ottenere (L), invece le righe sono sempre d .

Vantaggi e Svantaggi di PCA

Vantaggi

- Migliore tecnica lineare di riduzione della dimensionalità di un insieme di dati
 - migliore in senso di “errore quadratico medio”
- I parametri del modello possono essere ricavati direttamente dai dati
 - Io ho un algoritmo e se lo seguo ottengo la matrice, tanti altri metodi non hanno questa possibilità (hanno bisogno di ottimizzazione o addirittura operazioni più complesse)
- La proiezione nello spazio è un'operazione molto veloce (moltiplicazione di matrici)
- Tecnica molto conosciuta e utilizzata (permette di avere una prima idea sui dati)

Svantaggi

- Alto costo computazionale per il calcolo dei parametri del modello (soprattutto in caso di dimensionalità elevata)
- Non è chiaro come questa tecnica possa gestire il caso di dati incompleti
 - Grosso problema soprattutto in medicina
- PCA non tiene conto della densità di probabilità dello spazio considerato (viene considerata solo la vicinanza del vettore trasformato al vettore originale)
- Non è detto in tutti i casi che le direzioni a varianza maggiore siano le direzioni ottimali

Feature Selection

Approccio alternativo alla riduzione della dimensionalità

Scelgo semplicemente le feature.

- **Vantaggio:**
spesso alcune features sono ridondanti, non informative o addirittura dannose
- **Svantaggio:**
spesso è computazionalmente oneroso, occorre trovare il miglior sottoinsieme

Problemi da risolvere

- scegliere il criterio di ottimalità: informazione (come si misura?), varianza, capacità di classificazione (solo per problemi supervisionati)
- come trovare il sottoinsieme ottimale senza provarli tutti (troppo oneroso computazionalmente)

Ho un criterio di ottimalità (es. l'accuratezza del nearest mean classifier), faccio un ranking e trovo la migliore, poi formo le coppie tenendo fissa la prima. Ad ogni passo faccio la scelta migliore localmente → **greedy**

Forward Sequential Feature Selection: algoritmo greedy per trovare l'insieme ottimale di features

Schema:

- si valuta il criterio per tutte le features singolarmente
- si sceglie la feature che massimizza il criterio (chiamata f_1)
- si valuta il criterio per tutte le coppie (f_1, f_x) – cioè tenendo fissata f_1
- si sceglie la coppia che massimizza il criterio (la coppia (f_1, f_2))
- si valuta il criterio per tutte le terne (f_1, f_2, f_x) – cioè tenendo fissate f_1 e f_2
- ...

Algoritmo greedy – ad ogni istante la scelta migliore: computazionalmente efficiente ma sub ottimale

Esempio neuropatologie: Da un certo punto in poi aggiungere feature semplicemente crea rumore (disturba il classificatore).

Lezione 4

Una volta costruita una rappresentazione, dobbiamo costruire un modello.

Classificazione

Abbiamo un insieme di addestramento → come ho rappresentato gli oggetti e un insieme di etichette: ho un insieme di oggetti del problema per cui conosco la classe vera, cosa voglio fare?

Costruire un modello: in grado di discriminare tra le due classi.

Nello spazio delle feature visualizzo le dimensioni e gli oggetti nello spazio.

Uso i modelli per classificare un oggetto: il modello può essere assegnato ad una classe, una categoria.

1. Come costruire il modello?

2. Come classifico gli oggetti in base al mio modello?

1. Troviamo un sistema di classificazione

Ho qualcosa che viene applicato alla rappresentazione dell'oggetto in ingresso e mi ritorna la classe: l'output è discreto, è la classe.

Ho una funzione $f(x) = y$, dove y è l'output discreto.

Voglio trovare una funzione che sbagli il meno possibile (errore di generalizzazione) → ricordiamo che il modello deve essere in grado di generalizzare.

Non devo avere errore di classificazione (un oggetto di una classe 1 viene classificato come di classe 2)

Costruire la funzione $f \rightarrow$ **TEORIE DELLA DECISIONE**

è un insieme di regole, approcci e metodi che ci permettono di costruire un classificatore:

- come vengono caratterizzate le entità
- come viene determinata la regola di decisione
- come possono essere interpretate le soluzioni

TEORIE PIÙ IMPORTANTI

- Teoria di Bayes
- Statistical Learning Theory - più complessa

Non c'è una chiara separazione tra le teorie

Teorema di Bayes

- Approccio molto utilizzato e storicamente provato (ci sono molti risultati teorici: importante nel contesto della qualità dell'algoritmo).
 - Il problema è posto in termini probabilistici (probabilità di una classe, di fare un errore, ecc.)
 - Tutte le probabilità sono conosciute

Come funziona?

Uso le probabilità per costruire il classificatore.

Def: ω è la classe dell'oggetto

ω_1 = prima classe

La regola di classificazione mi permette di rispo

💡 Dato un oggetto x , lo assegno a ω_1 , a ω_2 o a ω_3 ? In altre parole: a quale classe deve essere assegnato x ?

Ci sono diverse probabilità importanti all'interno della teoria della decisione di Bayes:

- **Probabilità a priori**

codifica l'informazione a priori, capacità descrittiva molto bassa

es. rappresenta la probabilità dello stato che è nota a priori (non ho osservato nulla)

- **Probabilità condizionale**

mi dice la probabilità all'interno di una determinata classe

es. se x è lo stipendio, la distribuzione dello stipendio è fatta in un certo modo

quindi se sto considerando solo i calciatori, è come si distribuiscono gli stipendi, quindi distribuisco gli oggetti all'interno di una classe

è molto più descrittiva di quella a priori, ma non tiene conto di quest'ultima → non abbastanza descrittiva

- **Probabilità a posteriori**

Ho un modo di descrivere il problema che tiene conto di tutte le probabilità.

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

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

Dato che $P(A, B)$ e $P(B, A)$

$$P(B)P(A|B) = P(A)P(B|A)$$

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

$$p(\omega_j|x) = \frac{p(x|\omega_j)p(\omega_j)}{p(x)}$$

La regola di decisione di Bayes si basa sulla probabilità a posteriori, e permettere di classificare un oggetto x : si può dimostrare che la regola di decisione di Bayes minimizza la probabilità di errore.

Regola di decisione di Bayes:

💡 Dato x , decisi ω_1 se $p(\omega_1|x) < p(\omega_2|x)$, ω_2 altrimenti.

Equivalentemente:

- l'evidenza rappresenta un fattore di scala che descrive quanto frequentemente si osserva un pattern x
- non dipende da ω_1 o da ω_2 , quindi è ininfluente per la regola di decisione

$$posterior = \frac{likelihood \times prior}{evidence}$$

La regola utilizza le posterior (non note), per stimare le posterior si possono usare le likelihood e le prior.

Approcci Generativi

Un modello per ogni classe

Parentesi: la densità normale

è analiticamente trattabile, fornisce la migliore modellazione di problemi sia teorici che pratici. Il teorema del Limite centrale asserisce che "sotto varie condizioni, la distribuzione della somma di d variabili aleatorie indipendenti tende ad un limite"

/formule

La varianza misura l'altezza della campana, posso avere una gaussiana anche multidimensionale (formula simile -ma con matrici e vettori)

Densità normale multivariata

Approccio generativo → voglio stimare le proprietà di una classe: es- la mia classe segue una distribuzione gaussiana, stimo i parametri (media e deviazione standard): modello le classi quindi generativo, e approccio parametrico

Nel caso bidimensionale i due parametri sono media e varianza.

Matrice di covarianza

Una matrice 2x2 nel caso bidimensionale; misura quanto varia una feature rispetto a quanto varia un'altra feature - cattura la relazione che c'è tra i due parametri e mi dà un'idea della forma.

Nel caso in cui i parametri siano scorrelati, la covarianza è 0.

$$M = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}$$

Ho zero perché non ho relazioni tra le due covarianze.

I numeri che trovo sulla diagonale sono la varianza di x e quella di y. In un caso generico ho:

$$M = \begin{bmatrix} \sigma_x & \sigma_{x,y} \\ \sigma_{x,y} & \sigma_y \end{bmatrix}$$

Quando assumo che una classe ha distribuzione gaussiana posso avere un'idea di come sono correlate le feature. Quando so la forma diminuiscono anche i parametri che devo stimare.

-
- Stimo le probabilità a priori e le probabilità condizionali
 - Stime parametriche:
 - stimo i parametri data la forma della pdf
 - es. nella gaussiana stimo la media
 - Stimo la probabilità a priori
 - Si combinano a formare le probabilità a posteriori
 - Rappresenta il vero problema, poiché voglio stimare i parametri sconosciuti della funzione conosciuta
1. Stimola forma
 2. Ne stimo i parametri

Costruire un classificatore bayesiano con stima parametrica

Si stima dal training set la probabilità a priori per ogni classe

Per la probabilità condizionale: si decide o si stima, la forma per ogni classe (ad esempio gaussiana) & si stimano i parametri a partire dai dati di training (un insieme di parametri per ogni classe)

- Si classifica con la regola di Bayes

Stima non parametrica

Con la stima parametrica assumo che la forma delle densità di probabilità sia nota, ma questa assunzione non può sempre essere fatta.

Se la scelta della forma è sbagliata, la stima sarà povera.

Soluzione: metodi non parametrici

Approcci discriminativi

Modello direttamente alla fine

Lezione 5

Teoria della decisione di Bayes

Rappresentare tutto il problema di rappresentazione in termini probabilistici. La regola di decisione di Bayes si basa su alcune probabilità, in particolare su quella posteriore $p(\omega_j|x)$.

Vado a vedere la classe la cui posteriore è massima, e le assegno x_0 - *argomento massimo*.

es. $\max[10, 20, 30]$ $\arg\max = 3$ (la posizione dell'argomento massimo).

$$\frac{p(x|\omega_j)p(\omega_j)}{p(x)}$$

$p(\omega_j)$ ho una probabilità a priori: quanto è probabile osservare in natura una determinata classe

$p(x|\omega_j)$ mi dà l'informazione della distribuzione degli oggetti per la classe.

Contesto generale:

funzioni discriminanti lineari - posso formulare un classificatore, e assegno un oggetto alla classe la cui $g(x)$ è massima, dove $g(x)$ è una funzione discriminante.

Passiamo in questo contesto così da poterlo interpretare in maniera delle geometria: supponendo di avere uno spazio delle feature, posso isolare una regione dello spazio dove ad esempio la prima funzione $g_1(x)$ è più grande della seconda $g_2(x)$, e altre regioni in cui succede il contrario.

Posso creare una linea intorno alla quale $g_1(x) = g_2(x)$.

Questo ci permette di capire che posso affrontare il problema in due modi:

- andando a calcolare $g_1(x)$ e $g_2(x)$

Un modo generico di formulare un classificatore: calcolo $g_1(x)$ e $g_2(x)$ e assegno l'oggetto alla classe che risponde più fortemente alla funzione.

- andando a stimare la funzione $g_1(x) - g_2(x)$, ovvero $g_1(x) - g_2(x)$: in maniera più intelligente

$$g(x) = g_1(x) - g_2(x)$$

Ragionando dal punto di vista di Bayes:

posso definire $g_1(x)$ come voglio, ma se in particolare la descrivo come posterior:

- $g_1(x)$ è la mia Posterior ($p(\omega_1|x)$): calcolare le funzioni equivale ad applicare la regola di decisione di Bayes:

Assegno x ad ω_1 se $p(\omega_1|x) \geq p(\omega_2|x)$

Due strade che corrispondono a due percorsi diversi.

- posso cercare di stimare le due funzioni: $p(\omega_1|x)$ e $p(\omega_2|x)$
→ **APPROCCIO GENERATIVO**
- posso cercare di stimare direttamente $g(x)$, il "confine" → **APPROCCIO DISCRIMINATIVO**

Approccio Generativo

Devo stimare $p(\omega_1|x)$ e $p(\omega_2|x)$: dato che è difficile uso:

$$\frac{p(\omega_1|x) = p(x|\omega_1)p(\omega_1)}{p(x)}$$

$$\frac{p(\omega_2|x) = p(x|\omega_2)p(\omega_2)}{p(x)}$$

che è più facile da stimare.

Devo stimare quindi: $p(\omega_1)$ e $p(x|\omega_1)$

Come?

Apprendimento da esempi.

- stimare la probabilità a priori: il modo più semplice è vedere quanto rappresentata è la classe all'interno del training set, uso la cardinalità delle classi del training set
- stimare la probabilità di x dato ω_1 : come si distribuiscono gli oggetti all'interno delle classi?

Due possibilità:

- **approccio parametrico:** conosco la forma, il problema è stimare i parametri. Devo stimare quale specifica delle forme descrive meglio la mia classe. Devo stimare la media e la varianza della funzione.

Assumo una determinata forma (es. gaussiana), e poi ne stimo i parametri.

Problema: se sbaglio la forma della distribuzione, la stima che faccio è pessima! In molti problemi di riconoscimento non è possibile assumere la forma.

In alcuni casi posso provare a stimare la forma.

- **approccio non parametrico:** non faccio nessuna assunzione sulla forma della distribuzione della probabilità.

Se voglio stimare la probabilità di un punto in una regione dello spazio, vado a vedere cosa succede nell'intorno: analizzo le singole regioni dello spazio. Es. se nell'intorno non ho punti di training set, la probabilità sarà bassissima. Vado a ragionare per regione nell'intorno del punto e nella quantità

Passo 1:

Sia $x_{\{0\}}$ un punto generato con la probabilità $p(x)$. Si definisce P come la probabilità che il punto appartenga alla regione.

Ragioniamo come in un caso monodimensionale, in cui la regione è semplicemente un intervallo, un segmento di una certa lunghezza.

$$P(x_0 \in R) = \sum_{x=2, x=3, x=4} p(x) = \sum_{2 \leq x \leq 4} p(x)$$

ragionando nel continuo arrivo a:

$$P = Prob(x_0 \in R) = \int_R p(x) dx$$

 **Teorema:**

Dati N punti generati con la probabilità $p(x)$, se k punti cadono all'interno della regione, allora $\frac{k}{N}$ è uno **stimatore corretto e consistente di P .**

Passo 2:

Se

- $p(x)$ è continua
- $p(x)$ non varia molto all'interno della regione R

allora possiamo scrivere:

$$P = \int_R p(x) dx \approx p(x_0) V$$

Dove V è il volume della regione R e x_0 è un qualsiasi punto interno alla regione. Allora:

$$\frac{K}{N} = p(x_0) V \rightarrow \forall x_0 \in R, p(x_0) = \frac{K}{NV}$$

Sto facendo due assunzioni che sono in contrasto tra di loro: la prima è meglio se la regione R è ampia, la seconda se invece la regione è piccola.

Quindi ora la regola è:

$$p(x_0) = \frac{K}{NV}$$

Ho due strade per calcolare un punto:

- Ho x_0 : **definisco la regione R** e conto quanti punti nel mio training set cascano nella regione. Fisso quindi R , calcolo K e ottengo $\frac{K}{NV}$

Più sono i punti nel volume fissato V , più alta è la probabilità.

CLASSIFICATORE: Parzen Windows

- Fisso K :** Qual è il senso? Se voglio almeno un tot di punti ma per farlo devo avere un volume molto grande e la probabilità sarà grossolana.


Più grande è la regione che devi considerare per trovare K punti, più bassa è la probabilità.

CLASSIFICATORE: K-Nearest Neighbor

K-Nearest Neighbor

Classificatore famoso, generativo, non parametrico, semplice, intuitivo e molto utilizzato.

Si basa sul concetto di similarità.

 **Idea:** se devo classificare un oggetto i , vado a vedere gli oggetti che gli sono più simili.

Il Nearest Neighbor:

l'oggetto più simile tra tutti a quello che sto analizzando. Classifico l'oggetto come il suo nearest neighbor. ($K=1$)

Posso generalizzare la cosa:

supponendo di avere un training set (insieme di oggetti della classe 1, insieme della classe 2 e il mio x_0 da classificare).

- fisso un k
- vedo quali sono i k punti più vicini all'oggetto di test nel training set
- classifico questo oggetto a seconda dei suoi vicini

Questo classificatore implementa la regola di Bayes dove le probabilità a posteriori sono stimate in modo generativo e in modo non parametrico.

Vantaggi:

- tecnica semplice e flessibile
- estremamente intuitiva
- funziona anche per dati non vettoriali
- ragionevolmente accurata
- ci sono pochi parametri da aggiustare
- sono stati dimostrati molti risultati teorici su questa tecnica

XAI (explainable artificial intelligent) ha dato una "seconda vita" a KNN

Il classificatore

$$p(x_0) = \frac{K}{NV}$$

Notazioni:

- $X = \{(x_1, y_1), \dots, (x_n, y_n)\}$
dove x_i è l'i-esimo oggetto e y_i la sua etichetta
- c è il numero di classi
- N_j è il numero di oggetti in X che appartengono alla classe ω_j
- $N_j = |\omega_j|$
- $\sum_{j=1}^c N_j = N$
- K è il parametro di KNN

Goal: classificare x_0 , con la regola di Bayes

Devo calcolare $p(\omega_1|x)$ e $p(\omega_2|x)$

Quindi scompongo:

$$p(\omega_j|x) = \frac{p(x_0|\omega_j)p(\omega_j)}{p(x_0)}$$

$p(x_0)$ = considero i K punti più vicini a $x_0 \rightarrow$ regione R con volume V

$$p(x_0) = \frac{K}{NV}$$

$p(x_0|\omega_j) = \frac{K_j}{N_j V}$ dove K_j è il numero di punti in $R \in \omega_j$

$$p(\omega_j) = \frac{N_j}{N}$$

Quindi:

$$\frac{\frac{K_j}{N_j V} \frac{N_j}{N}}{\frac{K}{NV}} \rightarrow \frac{K_j}{K}$$

Per ogni classe prendo il massimo. Dei K punti presi, K_1 appartengono alla classe 1, ... K_j appartengono alla classe j . Prendo la classe più frequente all'interno dell'insieme/regione R .

Svantaggi:

- tutti i punti del training set devono essere mantenuti in memoria
- la scelta è molto locale, prendendo quindi pochi punti nello spazio
- la scelta della misura di similarità è fondamentale: sbagliandola, sbaglio tutto
- a seconda della matrice utilizzata, potrebbe essere necessario preprocessare lo spazio
- la scelta di K è spesso cruciale:
 - legato al grado di non-linearità che voglio

Scegliere K

Vado a vedere cosa succede nella regione, o lo faccio provando diversi valori e guardando l'andamento del classificatore, oppure empiricamente la radice quadrata degli oggetti.

Genericamente si usa un K dispari.

Ridurre la dimensione del training-set

Cancello punti che non hanno effetto sul confine di decisione (condensing) - non è detto però che questi punti non siano importanti per il testing set

Lezione 6

Bayes - dove siamo arrivati?

$\operatorname{argmax} P(\omega_i | x)$

- approccio generativo visto finora: condizione a priori
- approccio discriminativo: direttamente il confine

Classificatori discriminativi

I classificatori discriminativi sono i classificatori più utilizzati e funzionano molto bene:

- concettualmente sono semplici
- sono molto potenti
- sono veloci

SVM

Nella versione semplice, le Support Vector Machines, sono un **iperpiano (caso multidimensionale)**

Suddividono lo spazio in due regioni: sono classificatori binari. Abbiamo quindi solo due classi.

La retta

- $y = ax + b$ - definizione classica
- $ax - y + b = 0$ - ora proviamo a scriverlo come se fosse un vettore
- $\begin{bmatrix} x & y \end{bmatrix}$

$$\begin{bmatrix} a & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + b = 0$$

Posso chiamare $\begin{bmatrix} a & -1 \end{bmatrix}$, w e $\begin{bmatrix} x \\ y \end{bmatrix}$, $z + b = 0$

ovvero

$$w^T \cdot z + b = 0$$

Per semplicità chiamiamo z , x :

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

possiamo dividere lo spazio in due regioni:

$$w^T \cdot x + b > 0$$

$$w^T \cdot x + b < 0$$

Come uso la retta per fare classificazione?

Se ho due coordinate $z = \begin{bmatrix} x \\ y_1 \end{bmatrix}$, per classificare il punto z , devo vedere: $\text{sgn}(w^T \cdot x + b)$


A seconda del segno lo classifico nella classe sopra o sotto. Lì dove è uguale a zero, è esattamente sul confine.

Nel caso monodimensionale, **la retta è una soglia**. Se ho un solo elemento: $w = [w_1]$, $x = [x_1]$ devo vedere se $w^T \cdot x + b > 0$; ciò equivale a guardare se x è maggiore di una certa soglia

Come trovo l'iperpiano ottimale?

Cambiando i pesi, cambio radicalmente la retta, devo trovare un vettore di pesi che mi vada bene.

Introduciamo il concetto di:

 **Dati linearmente separabili**: esiste una retta per cui tutti gli oggetti sono linearmente separabili se esiste un iperpiano (una retta nel 2D) per cui le classi risutino essere perfettamente separate.

Ci sono un'infinità di rette che dividono dati linearmente separabili. L'assunzione fatta dalla SVN è che la retta migliore sia quella che si pone a metà tra le due classi: va a massimizzare il **margin** μ

Il margin è la distanza dei punti più vicini alla retta, di entrambe le classi, sommati tra di loro. Non posso però semplicemente massimizzare il margin, devo aggiungere dei vincoli, devo far sì che la retta separi le due classi.

Vincoli: i punti devono stare dalla parte giusta!

$$\begin{array}{ll} x_1 & y_1 = +1 \\ x_2 & y_2 = +1 \\ x_3 & y_3 = -1 \\ x_4 & y_4 = -1 \end{array}$$

I miei 4 vincoli sono questi. Deve essere vero che: se $y_i = +1$ allora $w^T \cdot x + b > 0$, se invece $y_i = -1$, allora $w^T \cdot x + b < 0$, ho quindi 4 punti su cui basare la mia retta.

Con i punti del training set io posso sfruttarli per creare la retta.

Se ho 100 punti, avrò 100 vincoli del tipo se $y_i = \dots$

Mi serve un ulteriore passaggio - **modo compatto ed elegante**:

$$y_i(w^T \cdot x_i + b) > 0$$

Addestrare una SVM vuol dire stimare i parametri.

Posso rinforzare il vincolo \rightarrow voglio che sia un po' più grande di 0. Perché? Se una soluzione è maggiore di 1, sicuramente è maggiore di 0.

Quale effetto ha?

Crea una zona di incertezza: $-1 < w^T \cdot x + b < 1$

Se un punto si trova nella zona di incertezza, non sono del tutto sicuro della mia classificazione.

Si ottiene quindi un problema di ottimizzazione quadratica con vincoli:

$$\rho = \frac{2}{\|w\|}$$

Vincoli: per ogni $(x_i, y_i) \in D : y_i(w^T \cdot x_i + b) \geq 1$

Trovo infine w e b che minimizzano:

$$\frac{1}{2} \|w\|^2 \rightarrow \frac{1}{2} w^T \cdot w$$

Vincoli: per ogni $(x_i, y_i) \in D : y_i(w^T \cdot x_i + b) \geq 1$

Voglio trovare il miglior w e il miglior b: ho un vincolo per ogni punto. Se io trasformo con i moltiplicatore di Lagrange, trovo un coefficiente α per ogni vincolo.

$L(w, b, \alpha)$

Alla fine trovo i coefficienti α_i per cui la funzione risulta ottimizzata:

$$w = \sum_{i=1}^N \alpha_i y_i x_i$$

Dopo l'ottimizzazione: tutti gli α sono uguali a zero tranne i punti più vicini al margine: **SUPPORT VECTORS**



Dati non linearmente separabili: non esiste nessuna retta tale per cui le due classi siano separate

Cosa succede? Non esiste un iperpiano in grado di dividere in due i miei punti.

Supponendo di applicare la formulazione equivalente: non ho una parte giusta e una parte sbagliata perché non esiste una soluzione alla mia ottimizzazione.

Cosa faccio? Devo rilassare i vincoli.

Posso "allargare" la mia valutazione:

$$w^T \cdot x_i + b \geq 1 - \xi_i$$

Il valore di ξ_i indica la posizione del vettore rispetto all'iperpiano:

- $\xi_i = 0$ classificazione corretta
- $0 < \xi_i \leq 1$ classificazione corretta ma nella zona del margine
- $\xi_i > 1$ classificazione errata

Voglio fare ovviamente in modo che ξ siano i più piccoli possibile, ξ_i sono **SLACK**

VARIABLES:

Voglio massimizzare il margine e minimizzare il costo - esiste un parametro C che mi regola esattamente il rapporto tra queste due formule:

$$\frac{1}{2} w^T \cdot w + C \sum \xi_i$$

Il parametro C è definito dall'utente, in base alle esigenze! Pesa l'errore e controlla l'overfitting.

Posso scegliere un C molto alto con un training set altamente affidabile, per cui gli errori mi peseranno tantissimo. Un C molto basso però mi privilegia il margine al contrario.

Ottimizzando la soluzione è sempre:

$$w = \sum_{i=1}^N \alpha_i y_i x_i$$

Il trucco di Kernel

Ci sono dei casi in cui una retta fa un lavoro pessimo.

Come posso risolverlo? Mi baso su due ragionamenti:

Se non riesco a risolvere il problema in uno spazio bidimensionale, posso provare a risolverlo in uno spazio a **dimensionalità elevata**. Supponendo di dare un'ulteriore coordinata potrei avere un'informazione aggiunta che mi permette di discriminare.

Come faccio?

Devo inventare una funzione che trasformi il mio oggetto in uno spazio di dimensionalità elevata.

Esempio: spazio monodimensionale in cui i punti sono disposti su una retta.

Invento:

$$\phi_1(x) = x$$

$$\phi_2(x) = x^2 \text{ (esempio, lo devo inventare)}$$

Adesso riesco a risolvere il problema con una retta.

Problema: curse of dimensionality

Problema: devo definire il mapping

Fare testing

$$\left(\sum_{i=1}^N \alpha_i y_i x_i \right) \cdot x + b$$

diventa:

$$\left(\sum_{i=1}^N \alpha_i y_i \phi_i \right) \cdot \phi(x)$$

L'effettivo kernel trick: se io devo testare l'oggetto non devo fare il mapping, devo fare il prodotto scalare tra le due funzioni.

Esistono delle funzioni, chiamate **kernels**, che calcolano il prodotto scalare in uno spazio indotto dalla funzione ϕ

Le funzioni kernel hanno una serie di proprietà matematiche che mi permettono di garantire che sono il prodotto scalare di uno spazio multidimensionale.

Molte volte non è più importante calcolare il mapping.

Cosa vuol dire costruire una SVM in uno spazio multidimensionale? Nello spazio originale ho un classificatore non lineare, senza dover fare nulla.



Una funzione kernel è una funzione che mi opera una curva di separazione senza che io debba effettivamente fare nulla, risolvendo in uno spazio multidimensionale, molto spesso non calcolato, la funzione che io non riuscivo a calcolare nelle dimensioni iniziali.

Concettualmente il kernel è una misura di similarità.

Anche il training set può essere riformulato per non calcolare esplicitamente il mapping, ma solo il prodotto scalare, che può essere ridefinito tramite i kernel.

Ogni volta che trovo un prodotto scalare posso quindi usare un kernel e lavorare in uno spazio a dimensionalità elevata... senza in realtà lavorarci.

- Occorre ovviamente scegliere la funzione kernel.
- Attenzione al costo C

Come abbiamo formulato il problema? Un problema di ottimizzazione convessa: ho una

soluzione ottimale SEMPRE. È convessa e ciò vuol dire che esiste un unico minimo e lo posso trovare

Vantaggi

- SVM hanno un'interpretazione geometrica semplice ma mancano completamente di interpretabilità
- Posso ottenere un classificatore non lineare in maniera ottimale
- La teoria è robusta ed elegante
- I support vectors hanno e danno una rappresentazione compatta del training set

Svantaggi

- Scelta del kernel e dei parametri è cruciale
- A volte il training è molto lungo
- Il training non è incrementale (se arriva un nuovo oggetto devo costruire un nuovo classificatore)
- SONO CLASSIFICATORI BINARI (e se ho tre classi?)
- Soluzioni:

ONE VS ALL

Costruire una support vector machine in grado di classificare una classe rispetto alle altre classi

es. ° vs not(°)

questo genere di schema ha dei problemi di ambiguità (delle zone grigie)

ONE VS ONE Confronto a coppie

Costruisco delle support vector machine che confrontano tutte le classi tra di loro

Se la maggior parte assegna ad una determinata classe, questa vince

Se ho infinite classi? Ho infinite coppie

Un'altra soluzione è fare una specie di torneo tra le classi.

Lezione 7

Validazione dei classificatori

Cercare di capire come va un classificatore. Se ho un problema, con il mio training set, dove ho deciso quali feature (ridotte con pca) utilizzare, quale classificatore (KNN) e voglio capire se ho operato delle buone scelte: voglio capire la sua capacità di generalizzazione (di misurarla)

Cosa faccio di solito per validare un classificatore?

Ci sono tanti criteri: velocità, compattezza, **accuratezza del classificatore**.

Lo valutiamo quindi in termini di errori di classificazione.

Cosa vuol dire testare?

Vuol dire calcolare l'errore di generalizzazione di un classificatore.

Usiamo il testing set per "testare" le capacità discriminative del classificatore costruito. Devo quindi avere un insieme di test separati: un testing set separato.

Come ottengo un testing set?

L'idea è quella di riestrarre altri esempi dal problema (nuovo campionamento) - non è mai fattibile (molto dispendioso e lungo).

Soluzione realmente adottata:

Suddivido l'insieme a disposizione in due parti:

- Utilizzo una parte per costruire e addestrare il classificatore
- Utilizzo l'altra per testarlo

Un classificatore costruito con meno oggetti è di certo peggiore rispetto ad uno costruito con più oggetti: cerco sempre di massimizzare gli oggetti che uso per costruire il classificatore, al contempo però il testing set risulta troppo piccolo, quindi la stima dell'errore è estremamente grossolana. Anche il testing set dovrebbe esser il più grande possibile.

Di solito si usa una tecnica chiamata **CROSS VALIDATION** per decidere a priori come descrivere i set:

- esempio: dividere a metà funziona benissimo se il dataset è sufficientemente grande
HOLDOUT
 - Potrei dividere a metà in maniera randomica e poi fare la stima di tre divisioni
AVERAGED HOLDOUT
- **LEAVE ONE OUT**: usata molto spesso, soprattutto quando ho pochi oggetti. Costruisco il classificatore con tutti gli oggetti tranne il primo e vedo il classificatore cosa mi dice sul primo. Poi prendo tutti tranne il secondo, poi tutti tranne il terzo, ecc.
 - Così facendo costruisco 100 classificatori diversi per 100 oggetti. Avrò una classe predetta per ogni oggetto
 - Questa è la scelta migliore per massimizzare la dimensione del training set in ogni tornata e testare il classificatore su più oggetti possibili.
 - Si utilizza spesso in casi medici

Altri strumenti:

- Learning Curves: variazione dell'errore al variare della dimensione del training set
- Matrici di confusione: l'errore di classificazione non ci permette sempre di capire o confrontare completamente due classificatori, per comprendere la differenza uso le matrici di confusione che mi dicono quanti errori faccio rispetto alla classe che esamino.
 $A(i,j)$? numero di elementi della classe i che sono assegnati alla classe j
 - True Positive, False Negative, False Positive e True Negative
 - Posso calcolare da questo diversi indici: precisione, recall, ecc. che mi danno un'idea diversa di come opera il classificatore.

Significatività statistica

Quando confronto dei numeri ho bisogno di sapere se la differenza è significativa o meno, ad esempio supponendo di usare l'errore di classificazione.

Clustering

È un problema completamente non supervisionato, ha una grande rilevanza scientifica. Seguiamo quindi una pipeline e chiuderemo con la validazione di un sistema di clustering.

Il clustering è studiato in molte aree scientifiche, ogni area ha la sua terminologia, in questo corso viene ovviamente visto tramite



Una **definizione**:

Il clustering rappresenta l'organizzazione di un insieme di patterns (entità/oggetti) in gruppi sulla base delle similarità.

Pattern che appartengono allo stesso gruppo sono tutti simili tra loro e gruppi diversi invece sono differenti tra loro.

I gruppi si chiamano **clusters**.

È fondamentale il fatto che il processo sia non supervisionato, non è quindi data alcuna informazione a priori sui gruppi.

Applicazione

Ha tantissime applicazioni, soprattutto per analisi esplorative, è fondamentale quando c'è poca informazione a priori ed è utile per inferire relazioni tra gli oggetti del problema, in modo da avere un'idea.

Nel 1850 John Snow, un fisico di Londra, durante un'epidemia di colera ha disegnato su una mappa la posizione dei casi e delle morti per poter trovare delle informazioni utili: i morti erano raggruppati accanto a tombini d'acqua. Questo è il primo clustering noto.

Il clustering ha quindi permesso di:

- i. Capire il problema
- ii. Sugerirne una soluzione

- Possiamo dividere le immagini in zone simili, identificando gli oggetti all'interno delle immagini.

Il concetto di clustering è intimamente legato al concetto di similarità.

- Inferire le relazioni genealogiche tra gli organismi (filogenesi)
- Analisi di dati di espressione per trovare geni coregolati
- Analisi di Social Networks

Perché il clustering è un problema così difficile?

Primo problema: Il concetto di cluster è molto vago: potrei avere clusterizzazioni per diverse features, colore, forma, categoria. A seconda di come definisco la misura di similarità posso avere clustering diversi.

Il secondo problema: occorre derivare il modello senza alcuna informazione a priori.

Terzo problema: non posso fare validazione quantitativa in un clustering. Non c'è una "ground truth" !

Un tipico sistema di clustering - Carrellata

Data samples → Pattern Representation → Definition of similarity → Clustering Algorithm Design → Clusters OR Cluster Validation → Results Interpretation (devo avere un modo per ridurre il numero di opzioni altrimenti il medico non sa cosa rispondermi)

— Comunicazione e diversità di approccio: c'è difficoltà nell'interazione con un mondo non informatico

Definire il concetto di similarità

È fondamentale! Deve essere la più veritiera possibile, la maggior parte della

La distanza tra due sequenze ad esempio viene legata al numero di differenze "sostituzioni" (mismatch) che ci sono tra le due sequenze, eventualmente pesate. Esempio: misura di Jukes-Cantor.

Definire un metodo di clustering

La scelta dell'algoritmo appropriato dipende da diversi

Non è facile farne una divisione, quella più accettata è:

- **Metodi partizionali**

Il risultato è una singola partizione del dataset. Tipicamente il numero di cluster è dato in ingresso. Le K-means sono molto usate

- **Metodi gerarchici**

Ho una diversa granularità, a seconda della granularità posso avere risultati diversi. Le partizioni sono innestate. L'output è un dendrogramma.

Validazione del clustering

Devo vedere se i cluster ottenuti sono ottimali.

- **BOOTSTRAP** - Bootstrap consensus tree
 - Misuro la credibilità e la robustezza dei clustering: ad esempio avendo le sequenze allineate con la filogenesi. Costruisco l'albero dei replicati con diversi metodi di clustering. Lì dove ho la maggior percentuale, il cluster è più credibile.
- Approcci differenti portano a differenti cluster

Similarità

Coefficiente di similarità indica la forza della relazione tra due oggetti. Maggiore è la somiglianza tra questi oggetti, più alto è il coefficiente di similarità. Esistono una serie di proprietà (della misura di similarità, proprietà molto sensate) che se soddisfatte mi fanno avere una **metrica**.

Ci sono moltissime misure però per cui non valgono queste proprietà: misure non metriche, che si trovano spesso lì dove c'è una **struttura**: sono misure che hanno un enorme senso della **realtà**. (esempio del percorso con i sensi unici).

La distanza tra A e B è diversa dalla distanza tra B e A: la misura è **estremamente sensata**, ma se devo costruire un algoritmo che lavora con questa misura ho moltissimi problemi. Se rilasso le proprietà della metrica ho dei problemi nel disegnare degli algoritmi di clustering. È difficile creare uno spazio vettoriale in cui questo genere di proprietà valga.

Ci sono diversi modi per passare da similarità a distanza: la similarità è $sim(x_i, x_j) = \frac{1}{d(x_i, x_j)}$ oppure $sim(x_i, x_j)$. Bisogna avere dei metodi ad hoc per usare queste misure al meglio.

Come scelgo la misura della similarità?

Questa scelta è molto più impotente qui

Occorre cercare di inglobare la maggior quantità possibile di informazione a priori:

- contesto applicativo
- tipo di pattern (vettore, sequenza, dati mancanti)
- dimensionalità del pattern
- scala
- cardinalità dell'insieme
- requisiti (velocità vs precisione): e.g. retrieval by content

Tipi di misure di similarità

Vettori numerici

- Distanza euclidea

- **Distanza di Manhattan**

La distanza lungo la prima feature più la distanza lungo la seconda feature
Si chiama così per la struttura di Manhattan: movimenti solo orizzontali e verticali.

- **Maximum distance (distanza sup)**

Tengo la massima tra le due distanze (verticale e orizzontale). Due punti per essere vicini devono esserlo per ogni feature perché la distanza si calcola prendendo il caso peggiore

- **Distanza di Mahalanobis**

Tiene conto della scalatura degli assi. L'idea è che tengo conto nel calcolo della distanza, della varianza: di quanto una feature varia in una determinata direzione: uso la matrice di covarianza per "pesare" le distanze

- **Distanza di Minkowsky**

Generalizzazione della distanza euclidea e di Manhattan

- **Similarità Coseno**

Similarità non distanza! Posso misurare la distanza tra due vettori tramite il prodotto scalare (lunghezza della proiezione di un vettore sull'altro)

Valori categorici

Ogni feature non può prendere qualsiasi valore, ma ha un determinato valore che può assumere.

- **Simple matching dissimilarity measure**: conta dove due sequenze sono diverse

- **Distanza di Hamming** (per binari)

- **Similarità di Jaccard** (sempre binari)

Misura del grado di overlap fra due insiemi A e B

- **Distanza Geodesica**

È un concetto, una famiglia di distanza che tiene conto del supporto. È la distanza migliore, perché più realistica.

Misure non vettoriali

Bisogna definire la specifica distanza. NON siamo più in uno spazio vettoriale. Vengono caratterizzate dal tipo di dato che hanno in ingresso

- **Edit Distance**

La distanza tra due oggetti misura come costo della trasformazione di un oggetto in un altro.
Consideriamo: sostituzioni, cancellazioni e inserzioni.

Quando definisco l'Edit Distance, vado a contare quante operazioni devo fare da una stringa ad esempio, per essere trasformata in un'altra: il numero minimo delle operazioni è il costo.

Alcune inserzioni, costano molto più di altre.

Lezione 8

Blast

Basic Local Alignment Search Tool

Algoritmo che confronta sequenze biologiche, con buone approssimazioni e buoni tempi di esecuzione.

È computazionalmente efficiente -- la ricerca effettuata con algoritmi di programmazione dinamica è assolutamente inefficiente, vista la mole di dati presente oggi -- BLAST è 50 volte più veloce.

Idee:

- non cerca l'allineamento ottimale
- non effettua tutti

Passo 1. Rimuovere le regioni di bassa complessità della sequenza query

Passo 2. Creare una lista delle "word" di N lettere della sequenza query

Passo 3. Cercare in tutte le sequenze del database le word formatesi nel passo precedente, con un buon match: score di allineamento sopra una certa soglia.

Il match viene fatto senza gap: cerco la tripletta esatta

Passo 4. Utilizzo ogni "hit" come "seme" per ampliare la similarità. Cerco di estendere la coppia di similarità a destra e sinistra fino a quando lo score non diminuisce.

Il risultato si chiama HSP (High Scoring Segment Pair). BLAST mi fornisce anche la significatività statistica del match che ho trovato.

Clustering Algorithm Design

Come si disegna un algoritmo di clustering?

Tipicamente si usano due passaggi: la prima è definire il criterio di clustering, ovvero una misura della "bontà" di un dato clustering. Un clustering è un partizionamento in cui gli oggetti nei gruppi sono simili, e quelli in gruppi diversi sono dissimili. Devo riuscire a quantificare quanto è buono un dato clustering.

Esempio:

Definisco che il cluster è buono se la distanza tra due centroidi è molto alta.

Dato il criterio di ottimalità voglio trovare la partizione per cui il criterio è massimizzato.

Il numero possibile di clustering è enorme, quindi il secondo passo è molto difficile.

Criterio di clustering

Sum-of-squared errors (SSE)

Idea: gli oggetti che appartengono al cluster devono essere vicini al suo centroide. Un buon cluster è un cluster in cui tutti gli oggetti di un determinato gruppo sono vicini alla media.

Non c'è un criterio che va bene per tutti i problemi: ad esempio SSE va bene per cluster molto compatti.

Una volta definito il criterio quindi devo cercare di ottimizzarlo. Lo spazio delle soluzioni è enorme quindi spesso si usano **algoritmi iterativi**.

Scelta classica:

- Trovare una ragionevole partizione iniziale
- Ripetere: spostare oggetti da un gruppo ad un altro in modo che la funzione obiettivo J migliori

Algoritmi di clustering

Si dividono in:

- **Clustering Partizionali**

Il risultato è una singola partizione dei dati (insieme di cluster disgiunti la cui unione ritorna il data set iniziale). Per dataset molto grandi si usa questo.

Vantaggi del Clustering Partizionale:

Ottimo riassunto dei dati, identifica i gruppi naturali presenti nel dataset. Ho una rappresentazione compatta del mio dataset.

Svantaggi:

Tipicamente richiede che i dati siano rappresentati in maniera partizionale, devo scegliere il numero di cluster e tipicamente estrae solo cluster complessi.

- **Clustering Gerarchico:**

Il risultato è una serie di partizioni innestate. Il clustering Gerarchico dà molte più informazioni, non partono da rappresentazioni vettoriali ma da distanze: posso usare questi stessi algoritmi per altro (grafi, sequenze).

Inoltre non è necessario dare in ingresso il numero di cluster, perché ho tutte le possibili partizioni a diversi livelli.

Vantaggi del Clustering Gerarchico:

riesce ad evidenziare le relazioni tra i vari pattern, richiede una matrice di prossimità, quindi non per forza vettoriali e non richiede in input il numero di cluster

Svantaggi:

È improponibile per dataset grandi. Molti sono greedy (subottimali).

La visualizzazione di 300 oggetti già è difficile.

Sommario degli algoritmi che vedremo:

- BSAS (Basic Sequential Algorithm Scheme)
- K-Means (e sue varianti)
- Algoritmi gerarchici agglomerativi (Single Link, Complete Link)
- Mixture di Gaussian (cenni)

BSAS

Basic Sequential Algorithm Scheme

L'algoritmo è sequenziale, quindi processa i pattern uno alla volta.

Supponendo di avere 4 pattern, con un ordine randomico.

Inizialmente creo un primo cluster e ci metto dentro x_1 ; considero ora il secondo punto: x_1 e x_2 sono abbastanza simili? Se no, creo un nuovo cluster.

Con x_3 cerco di capire se posso aggiungerlo a uno dei due cluster (la sua distanza è sufficientemente bassa), suppongo di sì e aggiungo x_3 nel secondo cluster.



Genericamente:

1. **Creo il primo cluster con il primo oggetto**
2. **Dal secondo oggetto in poi devo capire se posso inserirlo in un cluster già esistente: se la distanza tra gli oggetti è minore di una certa soglia, allora lo inserisco in un cluster.**

Mi servono due cose:

- **La distanza tra i punti**

- **La soglia di distanza**

Ci sono molti modi per definire la distanza tra un punto e un cluster.

Posso prendere tutte le distanze e farne la media, posso prendere la distanza più piccola, posso prendere la distanza dalla media, ecc.

Supponendo che la distanza sia sotto la soglia per tutti i cluster esistenti, il punto viene inserito in quello a distanza minore tra tutti.

Ordinamenti diversi danno risultati diversi (e questo è uno svantaggio del metodo), un vantaggio è che il numero di cluster non va settato, ma viene calcolato automaticamente. Un altro svantaggio è che il numero di cluster dipende dalla soglia: troppo grande - tutti in un unico cluster, troppo piccola - troppi cluster.

Notazioni

- x_i : vettore di punti, $\{x_1, \dots, x_N\}$ dataset da clusterizzare
- C_j : j-esimo cluster
- m : numero di cluster trovati ad un determinato istante

Parametri da definire:

- $d(x, C)$: distanza tra un punto e un insieme (un cluster)
- Max : distanza massima
- Min : distanza minima
- $Average$: distanza media
- *center - based*: distanza dal "rappresentante"
- Θ : soglia di dissimilarità

m = 1 C_m = x₁

Vantaggi:

Funziona anche per dati non vettoriali, è un approccio molto semplice e intuitivo

Svantaggi:

L'ordine è cruciale e così la scelta della soglia. Per settarla automaticamente creo un grafico con la soglia sull'asse delle x e il numero di cluster ottenuto. Siccome il sistema è dipendente dall'ordine, devo fare una media (20/30 volte). Spostandomi con un'altra soglia

Arriverò ad un certo punto per cui ho un plateau: ho molte soglie per cui il numero di cluster non cambia, prendo il valore centrale del plateau come valore soglia.

K-means

Algoritmo importantissimo, uno dei più famosi e conosciuti. È un algoritmo center-based per cui ogni cluster è rappresentato dalla sua media e ottimizza una funzione di errore.

Come funziona?

In ingresso dico quanti cluster ci sono.

Ogni cluster è rappresentato da una media (da stima iniziale). Partiamo da un'inizializzazione casuale delle medie (assumiamolo per poi tornarci). Per ogni oggetto calcolo la distanza che ha dalla prima media e dalla seconda media. Poi assegno ad ogni punto al cluster la cui media è più vicina. Ricalcolo con diverse medie, se ottengo sempre le stesse medie

Cambio prospettiva: faccio una stima iniziale delle etichette.

Notazioni

- $X = \{x_1, \dots, x_N\}$ N oggetti da clusterizzare
- K numero di clusters (in ingresso)
- $y^{(t)}$: il cluster alla t-esima iterazione

$$y^{(t)} = \{y_1^{(t)}, y_2^{(t)}, \dots, y_N^{(t)}\}$$

$y_1^{(t)} = 1 \rightarrow x_1 \in \text{cluster 1}$ alla t-esima iterazione

$y_1^{(t)} = 2 \rightarrow x_1 \in \text{cluster 2}$

$$\begin{array}{cc} x & y^{(3)} \\ \left[\begin{array}{cc} x_1 & 1 \\ x_2 & 1 \\ x_3 & 2 \end{array} \right] \end{array}$$

- $d(\cdot, \cdot)$: distanza euclidea tra punti
- $RANDSAMPLING(S)$: estrae casualmente un elemento da S
- $|S|$ cardinalità di S

L' algoritmo:

sotto si trovano funzioni e condizioni di stop

```
%INIZIALIZZAZIONE for i = 1 to N: y_i(0) = RANDSAMPLING({1,2,3,...,K}); end  
t = 0; RIPETI t = t + 1; %nuova iterazione %ricalclo medie passo  
precedente for j = 1 to K Cj = {x_i t.c. y_i(t-1) == j} %calcolo u_j come  
sotto end %riassegno gli oggetti nei cluster for i = 1 to N %per ogni punto  
calcolo la sua distanza dalle medie for j = 1 to K d_ij = d(x_i, u_j) end  
y_i = arg(min(d_ij)) % sempre calcolato sotto end UNTIL condizioni di stop
```

Calcolo di u_j :

$$u_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

Calcolo di $\arg \min$

$$y_1^{(t)} = \arg(\min_{j \in \{1, \dots, K\}} d_{ij}))$$

Condizioni di stop:

$$\forall i \quad y_i^{(t)} == y_i^{(t-1)}$$

Vantaggi:

- List

Svantaggi:

- Il numero di cluster deve essere fissato a priori
- L'inizializzazione è cruciale: una cattiva inizializzazione porta ad un clustering pessimo
 - Questo perché la funzione che utilizzo ha molte mode, molti minimi

Varianti:

ISODATA: tecnica che permette lo splitting e il merging dei cluster risultanti, così facendo ho dei controlli sul numero di cluster

PAM: il k-means calcola la media, ogni cluster è rappresentato dalla media, ma la media è un valore fittizio (è influenzata da valori estremi ed è un dato inventato: il profilo di un punto che non esiste). PAM usa il medoide, ovvero il punto del cluster più vicino alla media.

Come inizializzo K-means?

1. Genero casualmente il clustering iniziale.
2. Invece di generare le etichette (che mi provoca delle medie molto centrali), inizializzo le medie scegliendo casualmente i punti.
3. K-means ++ seleziona le medie estraendole, successivamente alla prima, in modo che i punti più lontani dalle medie esistenti siano più probabili.

Lezione 9

Varianti del K-Mean

PAM(Partitioning around the medoid)

Invece di usare la media, uso il medoide: è il punto vero vicino alla media, quindi a differenza della media è un punto vero.

Distance PAM

Sostituisce il concetto di media con quello di "concetto più centrale" quindi non è necessario che i dati siano vettoriali.

Prima strada: random partition.

Ho i miei punti e faccio un partizionamento casuale, mettendo casualmente le etichette ai miei punti.

Random Select Centroids

Scelgo due punti casuali come due medie, e faccio un giro di K-Means

K-means ++

Il modo più furbo: l'idea è di non scegliere i punti a caso, ma inizializzare le medie come il Random Points Initialization. Il primo punto è scelto in maniera casuale, il secondo viene scelto dando maggiore probabilità di essere estratti a quei punti che sono lontani da quello già scelto. Spalmo quindi meglio i miei centroidi.

K-Means Generalizzato

Posso avere un descrittore più ricco della media (un classificatore, una gaussiana, un SVM). Creo quindi un K-Means che nel primo passo stima il modello. Per il resto funziona allo stesso modo del K-Means, ma se scelgo bene il modello posso superare molti dei limiti del K-Means.

Clustering Gerarchico Agglomerativo

Generano una serie di partizione innestate.

Cosa vuol dire Agglomerativo?

Agglomero le mie partizioni, nel primo livello ogni punto è in un cluster, nel secondo metto assieme i due punti più simili.

es.

$\{x_1\} \{x_2\} \{x_3\} \{x_4\}$

$\{x_1, x_3\} \{x_2\} \{x_4\}$

$\{x_1, x_3\} \{x_2, x_4\}$

Possiamo calcolare tutte le possibili distanze tra tutti i possibili punti per calcolare la distanza, ci sono diverse possibilità: l'importante è avere **una misura della distanza**.

Algoritmo

Ne esistono due formulazioni, qui si vede quella basata su matrici.

Notazioni:

- O : insieme degli oggetti da clusterizzare, non è necessario siano vettori possono essere oggetti di qualsiasi tipo

$$O = \{o_1, \dots, o_N\}$$

- $d^o(o_i, o_j)$: distanza tra due oggetti o_i e o_j
- $d^c(C_r, C_s)$: distanza tra cluster, in particolare tra i due cluster C_r e C_s , si calcola a partire dalle distanze tra gli oggetti.
 - $d^c(C_r, C_s)$ è la minima $d^o(o_i, o_j)$, $o_i \in C_r, o_j \in C_s$, è la coppia di punti più vicini
 - $d^c(C_r, C_s)$ è la massima $d^o(o_i, o_j)$, $o_i \in C_r, o_j \in C_s$, è la coppia di punti più lontani
 - A seconda di quello che uso ho due varianti:
Single-Link per la minima e **Complete-Link** per la massima
Se faccio la media ho un **UPGMA**
- $C(O)$: partizione livello O ogni oggetto in un cluster
- $C(m)$: partizione livello m che si ottiene mettendo insieme i cluster più simili della partizione $m - 1$

Idea di Base:

Dalla matrice delle distanze supponiamo di aver visto che i due cluster più simili siano il secondo e il quinto, dalla matrice: la rielaboro cancellando il secondo e il quinto cluster (seconda e quinta colonna) e aggiungo una riga in cui calcolo la distanza tra il nuovo cluster e tutti gli altri. La matrice ad ogni livello mi tiene la distanza tra le possibili coppie di cluster e quando decido di unirne uno, viene aggiornata di conseguenza.

Algoritmo formale

Primo passo. Inizializzazione

Definizione di C_0 (il clustering di primo livello)

$$C(0) = \{C_1, \dots, C_N\}$$
$$C_i = \{o_i\}$$

La matrice D invece deve contenere le distanze tra tutte le coppie

$$D^0(i, j) = d^0(o_i, o_j)$$

Secondo passo. Ripeti

$$m = m + 1$$

% trovare i cluster più simili

$$(r, s) = \arg \min_{i, j \ i \neq j} D^{m-1}(i, j)$$

Terzo passo. Trovare $C(m)$

$$C(m) = C(m-1)C(s) \setminus C_r \setminus C_s \cup \{C_r, C_s\}$$

Quarto passo. Creare la matrice $D^{(m)}$

$$D^{(m)} = D^{(m-1)}$$
$$D^{(m)}([r, s], [r, s]) = [\]$$

cancello le righe r e s e le colonne r e s

Quinto passo. Aggiungo riga e colonna

$$\forall j \neq r, s$$
$$CALCOLARE \ d^c(C_{rs}, C_j)$$

Variante **single-link**:

$$d^c(C_{rs}, C_j) = \min[d^c(C_r, C_j), d^c(C_s, C_j)] =$$
$$\min[D^{(m-1)}(r, j), D^{(m-1)}(s, j)]$$

Variante **complete-link**:

$$d^c(C_{rs}, C_j) = \max[d^c(C_r, C_j), d^c(C_s, C_j)] =$$
$$\max[D^{(m-1)}(r, j), D^{(m-1)}(s, j)]$$

Sesto passo. Chiusura del ciclo

UNTIL ho un cluster solo

Differenze tra Single-Link e Complete-Link

Prendere il minimo o il massimo ha un effetto forte sui risultati, quindi c'è un'importante differenza tra **Single-Link** e **Complete-Link**.

Concettualmente sono scelte molto diverse:

- se prendo il minimo vuol dire che due cluster per essere simili basta che abbiano due punti vicini, è molto meno conservativo
- se prendo il massimo la distanza tra i cluster aumenta, e quindi per essere simili, devono essere simili tutte le coppie di punti che appartengono ai due cluster, è molto più conservativo

I risultati possono essere molto diversi.

Ci sono altri metodi però per unire i cluster.

UPGMA

La distanza tra cluster è definita dalla media

Metodo di Ward

Nel momento in cui definisco la similarità, ho più modi per farlo.

Misture di Gaussiane

È una tecnica che appartiene alla classe delle **Model Base Clustering**

Pensando al K-Means ogni cluster è rappresentato dalla media, in questo caso il cluster è rappresentato dal modello: trovo il fitting ottimale.

Avendo un insieme di punti, posso assumere che l'insieme sia generato dalla commistione di due o più gaussiane, voglio capire qual è il miglior modo di rappresentare l'insieme con due o più gaussiane.

Quanto più i punti rientrano nella campana della gaussiana, meglio il punto è "spiegato" dalla gaussiana, usando una tecnica di stima (es. maximum likelihood).

Come risolvo questo problema?

Uso un modello probabilistico (uso le tecniche viste quando si parlava di stime della media). Devo semplicemente massimizzare i parametri rispetto al mio caso. È difficile (non è come trovare il migliore parametro): uso però un algoritmo, Expectation-Maximisation. Vado a ristimare i parametri delle varie gaussiane.

Ho un'inizializzazione casuale dei due modelli, andando a ristimare i modelli, uso l'assegnamento iniziale (non proprio un assegnamento, ma una gradazione) per stimare il modo in cui devo modificare le gaussiane per massimizzare la stima. (comportamento simile al K-means). Uso una forma di membership nella stima dei modelli.

La validazione del clustering

Validare un clustering è difficile, perché se quello che vogliamo fare è riuscire a prendere gli oggetti e in maniera non supervisionata trovare i gruppi all'interno del dataset, quindi ci sono moltissime soluzioni, tutte valide.

Per farlo ci sono 3 strade:

1. Interpretazione

più qualitativa

Usiamo informazioni del contesto applicativo. Supponendo di avere un insieme di soggetti per clusterizzare i sottotipi di sclerosi multipla a partire dalle reti di connettività. Riesco a capire se l'algoritmo di clustering ha fatto un buon lavoro? Vado direttamente dal medico e lui mi potrà dire se è corretto o meno.

2. Validazione Quantitativa

Ottenuto un raggruppamento cerco di capire se ha senso:

- **Validazione Interna**

Qual è la definizione di buon clustering? Raggruppamento di soggetti, dove se appartengono allo stesso cluster sono molto vicini, altrimenti sono lontani (simili/dissimili).

Scelgo quindi un criterio di bontà e uso quello per capire. Abbiamo criteri diversi a seconda dei dati.

Uso quindi criteri interni per la validazione: uso solo i dati disponibili.

- **Validazione Esterna**

Prendo un problema identico di cui ho già la soluzione e confronto il clustering da me ottenuto con la soluzione, se riesco a ritrovarla ho fatto bene.

Uso criteri esterni come indici di validità.

Indici:

- Rand
- Jaccard
- Fowlkes
- Mallows

Costruisco una matrice di partizionamento: Ho una funzione indicatrice $I_U(i,j)$ che vale 1 se gli oggetti i e j sono nello stesso cluster secondo il clustering 1. Avrò ovviamente due funzioni indicatrici, quella del clustering vero e quella del mio clustering.

Posso quindi calcolare la matrice di contingenza:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- a mi dice il numero di coppie che sono messe nello stesso cluster in tutte e due le partizioni
- b mi indica il numero di coppie di oggetti che sono messi nello stesso cluster da U ma non da V e c esattamente il contrario
- d indica il numero di coppie di oggetti messi in cluster diversi sia da U che da V

I criteri interni sono molto difficili da stimare, devono misurare il fitting tra una partizione data e il dataset. Il problema fondamentale è stimare il numero di cluster e la baseline.

Indice di Davies-Bouldin

Viene usato spesso per stimare il numero ottimale di cluster.

Setto $K=2$, calcolo il risultato con K-Means, calcolo un'indice della bontà. Rifaccio il K-Means con vari K , a seconda dell'indice, scelgo quello in cui è massimizzato. Non risponde alla domanda "è buono?", trova il miglior cluster disponibile, non è detto però che sia buono.

Come funziona questo indice?

Un clustering è buono nel momento in cui gli oggetti all'interno di uno stesso cluster sono compatti e i cluster tra loro sono molto separati. Voglio quindi dare una misura di compattezza e separazione.

DEFINIZIONI

- $\{x_1, \dots, x_N\}$ punti da clusterizzare
- C_1, \dots, C_K : partizione da valutare

Si possono calcolare il centroide, la variazione intracluster e la variazione intercluster.

Per ogni coppia di cluster si calcola:

$$R_{jh} = \frac{e_j + e_h}{dm(j, h)}$$

Per ogni cluster si calcola

$$R_j = \max_{j \neq h} R_{jh}$$

L'**indice** viene determinato come:

$$DB(\{C_1, \dots, C_K\}) = \frac{1}{K} \sum_{j=1}^K R_j$$

Più piccolo è l'indice meglio è! Vuol dire che i cluster sono compatti e molto distanti.

Clustering Tendency

Gli algoritmi di clustering producono sempre un output, a prescindere dal dataset. Identificare senza effettuare il clustering, se i dati hanno una predisposizione ad aggregarsi in gruppi naturali.

Esempio: Scan Test

Divido il mio spazio delle feature in blocchi e vado a contare quanti punti per blocco. Se ottengo dappertutto più o meno lo stesso valore, non ci sono diversità e non ha senso fare clustering. Non è facilissimo da applicare perché devo definire le regioni (grandi, piccole, sufficientemente diverse, ecc.)

Lezione 10

Reti Neurali

💡 Le reti neurali si definiscono come un sistema artificiale di elaborazione delle informazioni che si pone come obiettivo l'emulazione del sistema nervoso animale.

Rappresentano un tentativo di replicare il modo in cui l'uomo ragiona ed elabora le informazioni. Dal punto di vista di un sistema di calcolo un sistema nervoso animale presenta delle caratteristiche vantaggiose: è piccolo, elabora calcoli abbastanza velocemente, è flessibile (impara) ed è robusto e resistente ai guasti (i neuroni muoiono regolarmente ma non si hanno grossi peggioramenti nelle prestazioni).

Negli anni '60 nascono gli algoritmi che per primi cercano di emulare il cervello.

Per prima cosa si cerca di capire: **come funziona?**

E poi si cerca di replicarlo.

I due ingredienti fondamentali: il sistema nervoso è un insieme di unità fondamentali di calcolo (un neurone)

Come si implementa un neurone e come si implementa un insieme di neuroni)?

Il cervello ci permette di lavorare anche con informazioni **"non buone"**:

- approssimata
- incompleta
- affetta da errore

Lo schema di base

Lo schema di base implementato è una struttura complessa, formata da tante unità di calcolo chiamate neuroni.

Questo è consolidato, ma molti altri meccanismi di funzionamento sono ancora ignoti.

Voglio comunque replicare con un sistema di calcolo, un neurone. Questo aspetto è molto particolare perché per tanti problemi noi prendiamo soluzioni che sono molto lontane dal regno animale.

Com'è fatto un neurone?

- C'è un **nucleo** che in qualche modo fa i conti.
- Ci sono una serie di **dendriti** che forniscono l'input da altri neuroni e sono collegati (pesati) tra loro tramite **sinapsi**.
- Ci sono degli **output**

Dal punto di vista matematico è una box a cui arrivano gli input e da cui escono degli output.

Gli input possono essere sensori dal mondo esterno (es. retina) oppure input da altri neuroni, ogni input è "pesato" dalla sinapsi.

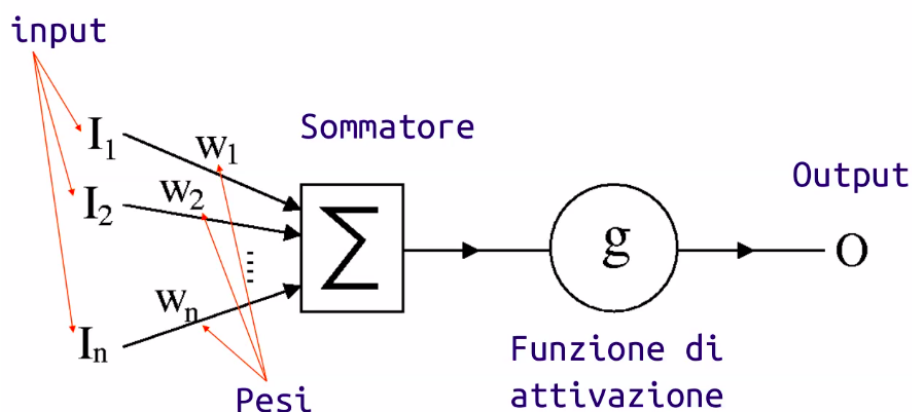
Per ogni dendrito dal punto di vista biologico all'interno del nucleo si fa una "somma" dei pesi, se la somma supera **una certa soglia** il neurone si attiva. Se il neurone si attiva oppure no, dipende quindi dalle sinapsi.

Ogni neurone possiede anche un sistema per calcolare la soglia di attivazione al momento successivo.

Diverse reti neurali differiscono per:

- tipo di neuroni
- tipo di connessioni tra neuroni

Il modello matematico di calcolo



Input I_i

- possono essere gli input del problema
- possono essere output di altri neuroni

Pesi w_i

Una misura di quanto conta l'input in ingresso

Sommatore \sum

Fa la somma pesata

Funzione di attivazione g

Definisce l'output sulla base della somma pesata

$$O = g\left(\sum_{i=1}^n w_i I_i\right)$$

Il perceptrone

Nel caso più semplice la funzione di attivazione è una **funzione scalino** (devo superare una certa soglia), chiamata **funzione soglia** θ .

$$\theta : R \rightarrow \{0, 1\}$$

$$\theta(x) = \begin{cases} 0, & \text{se } x < 0 \\ 1, & \text{se } x \geq 0 \end{cases}$$

Questo è il modello più biologicamente plausibile.

Se uso questo sistema il neurone si chiama perceptrone.

Guardando com'è definita la funzione, posso anche scrivere: $\sum_{i=1}^n w_i I_i - t$, dove t è la **soglia**, chiamata **funzione di Heavside**, posso fare ancora un altro passaggio, inglobando la soglia come un ulteriore parametro: inserisco un peso che vale $-t$ (la soglia cambiata di segno) che sia già automaticamente quindi sottratta dalla sommatoria.

È un modello molto vecchio: è stato introdotto nel '62.

Lo scalino di attivazione è molto brusco, quindi si può ovviare a questo problema usando altre funzioni di attivazione

Altre funzioni soglia

- **Sigmoide**

Sale in maniera più morbida

Meno biologicamente plausibile, ma il modello matematico funziona lo stesso e mi dà un output continuo, a differenza del perceptrone, ottenendo così una granularità (risoluzione) migliore

Il senso è di non avere il salto brusco, ma una transizione dolce dallo zero all'uno.

- **Funzione logistica**

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

- **Funzione tangente iperbolica**

$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Mettere insieme più neuroni

Mettere insieme più neuroni comporta complicare le cose, la complessità diventa enorme anche dal punto di vista del numero di parametri.

es. lettore di immagini - ogni pixel è un input, avendo 10 mila input ho 10 mila parametri, perché ogni input ha un peso su un arco.

La cosa più difficile da gestire però è il fatto che l'output di determinati neuroni diventa l'input di altri, e il comportamento non è stazionario, ma si diversifica ed evolve nel tempo.

| Il modo più semplice di ragionare questa cosa è per **livelli**.

Ragionamento per livelli

Creo livelli di neuroni che hanno come input l'output del livello precedente - **reti feed forward** (costruita in modo che l'informazione si propaghi sempre dall'input verso l'output e vada solo avanti e mai intralivello).

I livelli compresi tra l'input e l'output vengono chiamati "**Livelli nascosti**".

Nel caso in cui le unità principali siano i perceptron, queste reti si chiamano **MLP (Multilayer Perceptron, Percettrone Multilivello)**

In questo tipo di reti non sono possibili autocollegamenti o connessioni con i neuroni del proprio livello. Ogni neurone ha quindi la funzione di propagare il segnale attraverso la rete, flusso di informazione dagli ingressi verso le uscite.

Conseguenza:

la rete risponde sempre nello stesso modo se sollecitata dagli stessi output.

Come si costruiscono le reti neurali?

L'intelligenza delle reti neurali si trova nelle sinapsi, quindi è settare le sinapsi che cambia il modo in cui essa elabora.

Supponiamo di avere due ingressi I_1 e I_2 che entrano in un perceptrone, che mi dice che se la somma pesata degli ingressi $-t$ è maggiore di 0, allora l'output è 1.

In pratica **divido lo spazio in due regioni**, una regione per cui per qualsiasi input è zero e una in cui qualsiasi input dà 1. Se ci riflettiamo invece t è l'**equazione di una retta** (come per le SVM): $ax + by + c = 0$. Il confine tra le due regioni è dove $\sum_{i=1}^n w_i I_i - t = 0$, quindi:

Questo primo neurone quindi separa lo spazio in due regioni, cosa succede se metto assieme più neuroni?

Se ho un altro neurone, con altri coefficienti e la sua retta t , anche questo separa lo spazio in due regioni. La regione in cui tutti e tre i neuroni si accendono è la regione dello spazio caratterizzata da un neurone con tutti gli input a 1 e $-t$ uguale al numero degli input, così che il neurone si accenda solo se tutti i neuroni a esso afferenti sono accesi. Questa struttura si chiama **Neurone AND** (fa l'and di tutti gli ingressi). Sono riuscito a creare una rete neurale in grado di approssimare una regione dello spazio.

Facendo un ragionamento al contrario, supponiamo di voler approssimare una regione dello spazio (ad esempio in cui si trova una classe del mio problema), man mano che uso più neuroni, più raffino la stima di quella regione dello spazio. Quando faccio ragionamenti di questo tipo ragiono sulla **capacità espressiva della rete neurale**.

Addestramento della Rete Neurale

Una volta definita la topologia (es. decidere il numero di livelli, il tipo di neuroni ecc.)

Come per tutti i classificatori abbiamo un **training set**: quindi ho dei valori d'esempio, da cui voglio ottenere dei valori di uscita.

Si usa una **funzione di errore** da minimizzare:

io vorrei minimizzare la differenza tra l'applicazione della mia rete con l'output che ottengo il valore vero.

Se da determinati ingressi dovrei ottenere 1, ma invece ottengo 0.5, otterrò un errore di 0.5. Io voglio minimizzare questo errore, devo quindi settare i pesi in modo che vada bene per tutti gli esempi inseriti.

Discesa lungo il gradiente

💡 Avendo la funzione di errore, voglio trovare il minimo, quindi usando la discesa lungo il gradiente, avanzando mi sposto verso la direzione opposta a quella del massimo gradiente, trovando il minimo della funzione.

$$w^{(t+1)} = w^{(t)} - \eta \frac{\sigma E(W)}{\sigma W} \Big|_{W=W^t}$$

Ad ogni iterazione mi sposto. C'è un parametro importante che è **Eta (η)** che è la **lunghezza del passo** con cui faccio salti di iterazione in iterazione: è importante perché se faccio un passo troppo piccolo impiego troppo tempo, altrimenti se fosse troppo grande potrei saltare il mio minimo.

Quando fermare l'addestramento?

Solitamente si fanno un determinato numero di salti e poi ci si ferma, la rete diventa "troppo brava" quindi sta imparando troppo a memoria - *overtraining*.

La rete neurale ovviamente deve essere in grado di **Generalizzare**.

Per farlo mi serve un **Validation Set**, con cui capire se la rete sta diventando troppo specifica. Mi serve sapere quando devo fermare il mio addestramento.

Creo un grafico, sulla x il numero di iterazioni (nella comunità di reti neurali si chiamano **epoche**) - le reti neurali hanno una comunità molto specifica. Con il training set ho un andamento che scende, perché man mano la rete impara sempre meglio, guardo cosa succede con il Validation Set e quindi quando comincia a risalire, allora mi fermo e trovo la mia soluzione, il mio numero di epoche.

Il calcolo della derivata che mi serve per spostarmi dal massimo al minimo gradiente, è quadratico rispetto al numero di pesi (parametri), se ho una rete molto grande computazionalmente è molto pesante.

L'algoritmo che va per la maggiore è lineare - si chiama **Back Propagation**.

Ottimizziamo il calcolo della derivata (qui non è spiegato come)

Ha due fasi:

- Forward

Presento un esempio alla rete e propago per ottenere un'uscita e calcolare la funzione di errore.

- Backward

L'errore viene propagato indietro alla rete, aggiustando i pesi man mano che torna indietro

Una prospettiva storica

Le reti neurali sono estremamente di moda (con il nome di *Deep Learning* o *Convolutional Neural Networks*) - perché? Le reti neurali sono quel meccanismo che porta l'intelligenza artificiale nella vita di tutti i giorni (Alexa, Siri, ecc.) - si autoaddestrano.

Da un punto di vista storico cerchiamo di capire questo fenomeno.

1. C'è stata una prima fase, prima degli anni 90, all'introduzione del modello matematico in cui c'è stato un entusiasmo enorme rispetto le reti neurali.
2. Tra gli anni 90 e 2000 c'è stato un calo sostanziale dato dal fatto che la rete neurale è una BLACK BOX:
 - approssima benissimo le regioni, ma se vado a vedere i pesi, non ho alcuna intuizione su quello che succede: non c'è alcun modo di capire cosa succede all'interno
 - ovviamente ciò non piace, dato che le SVM arrivate in quel periodo, sono comprensibili e competono a livello di prestazioni con le reti neurali, per 20 anni si sono dimenticate le reti neurali
3. Successivamente, dal 2010 in poi, le reti neurali hanno avuto un boom, soprattutto le Convolutional Neural Network, arrivando a quella che viene definita la "deep learning revolution"

Qual è il motivo?

Visione del prof:

- **curse of dimensionality**

all'aumentare del numero delle features (dopo un certo livello) crea dei problemi al classificatore, che si incasina e rende peggio.

Ne esiste un'altra versione, data dalla complessità del classificatore (es. numero di parametri) - quando ho un classificatore troppo semplice ho un errore molto alto, ma se prendo un classificatore troppo complesso il classificatore peggiora - **errore di generalizzazione**. Al crescere della complessità, ho un miglioramento delle prestazioni fino a un certo punto, e poi peggiora. [Più oggetti ho nel Training Set, più questo problema migliora, quindi con oggetti infiniti, non ho problemi di curse of dimensionality.](#)

La complessità della rete neurale è data dal numero delle sue connessioni, ma non soffre della curse of dimensionality allo stesso modo degli altri classificatori - in realtà anni dopo si è scoperto che il problema era che nell'addestramento delle reti neurali, l'addestramento si ferma sempre dopo poche operazioni e ottenendo buoni risultati per la limitatezza delle macchine utilizzate!

Con il miglioramento delle macchine a disposizione, anche nelle reti neurali si è vista la curse of dimensionality.

Nel 2010 si è affrontato il problema: addestro una rete enorme con milioni di dati!

La rivoluzione consiste nell'aumentare enormemente sia la dimensione del training set che la dimensione della rete, questo è stato possibile in parte grazie ai Big Data (la quantità di dati che abbiamo a disposizione al giorno d'oggi è pressoché infinita) e dalle GPU (che permettono computazioni parallele estremamente veloci)

La “Deep Learning Revolution”

Approccio tipico: pre-addestrare reti specifiche per un determinato problema usando dataset enormi.

Le grandi aziende hanno investito tantissimo sul Deep Learning e le Reti Neurali.

Costruiscono reti neurali General Purpose e poi faccio tante iterazioni sul mio problema specifico quante ce ne sono bisogno.

Le Reti Neurali, classificatori alla portata di tutti, possono essere usate in maniera impropria: se non ho le competenze di base, posso ottenere risultati non veritieri.

Hidden Markov Models

Modelli di Markov a Stati Nascosti

Sono modelli probabilistici (torniamo nel mondo della probabilità) per modellare dati sequenziali (temporali e non).

Principali proprietà:

- Hanno la capacità intrinseca di modellare i dati sequenziali e l'evoluzione sequenziali
- Esiste un algoritmo di training veloce ed efficiente (facili da addestrare)
- Chiaramente interpretabili dal punto di vista della teoria di Bayes
- Se usati in modo appropriato, funzionano molto bene in moltissime applicazioni.

Questi modelli sono nati negli anni '60 da Baum et al., come modelli per il riconoscimento del parlato. Dagli anni '90 in poi sono stati usati ampiamente, in particolare con dati sequenziali.

Usati in contesti in cui la sequenzialità non era così evidente (es. riconoscere oggetti dalla silhouette o il riconoscimento delle facce - sembra essere molto importante l'ordine (es. ordine invertito riconosce meno): si può fare replicando i movimenti saccolari che usiamo per

osservare le facce come esseri umani, facendo una serie di movimenti e guardando diversi punti).

Bioinformatica

L'evento base: la sequenza di amminoacidi e nucleotidi è una sequenza, quindi gli HMM sono molto utilizzati, oppure per modellare le famiglie di proteine.

Modelli di Markov

Il modello di Markov è un modello probabilistico, che modella un sistema con determinate caratteristiche, facendo alcune assunzioni.

Assunzione 1.

Il sistema che stiamo modellando **evolve in passi discreti** (es. del semaforo: adesso, tra 1 secondo, tra 2 secondi: ci sono istanti di tempo prefissati)

Assunzione 2.

Il sistema **ad ogni istante è in un possibile stato** (con N stati, si trova in uno di questi N : il semaforo ha 3 stati, s_1 , s_2 e s_3 che corrispondono al semaforo rosso, giallo e verde).

Lo stato al tempo t si indica come q_t .

Assunzione 3

Lo stato in cui il sistema si trova al **tempo t** , **dipende solo dallo stato a cui si trovava nel tempo $t - 1$** .

- **Markovianità del primo ordine**
- Il sistema non ha "memoria"

Assunzione 4

La transizione tra gli stati è descritta in modo probabilistico.

Come passo da q_3 a q_4 ?

Abbiamo due insiemi distinti di probabilità:

1. Probabilità iniziale

Probabilità di essere in un certo stato all'inizio dell'evoluzione è

$$\pi_j = p(q_1 = S_j), \quad \forall j = 1, \dots, N$$

2. Probabilità di transizione

La transizione degli stati è data dalla probabilità:

$$a_{ij} = p(q_t = j | q_{t-1} = i), \quad \forall i, j = 1, \dots, N$$

Quindi mi basta specificare la probabilità di passare da uno stato all'altro, perché poi questo vale per ogni stato, a prescindere da t , dato che la probabilità dipende solo dallo stato.

Come possiamo disegnare la possibilità del semaforo?

Come una matrice 3x3, dove s_1 è rosso, s_2 è giallo e s_3 è verde

	s1	s2	s3
s1		0	alta
s2	alta		0
s3	0	media	

La probabilità è caratteristica di un singolo semaforo. Settando in maniera diversa le probabilità ottengo sistemi diversi (ad esempio questo rimane in rosso molto poco)

Assunzione 5

Quando il sistema entra in uno stato viene emesso un simbolo.

- Il simbolo rappresenta un'osservazione (quello che posso osservare in un sistema)
- Il simbolo cambia a seconda dello stato in cui mi trovo.

Ad esempio nel caso del semaforo il simbolo è la lampada accesa in quel momento.



La proprietà di Markov ha più ordini:

- una di **ordine due dipende dai due stati precedenti**
- una di ordine tre, dai tre precedenti

Le matrici si possono tranquillamente imparare, se ho un'osservazione del sistema, posso tranquillamente costruirla.

Riassumendo

Per definire i modelli di Markov:

- Insieme degli stati

$$\{S_1, \dots, S_N\}$$

- Probabilità iniziali

$$\pi = \{\pi_1, \dots, \pi_N\}$$

- Probabilità di transizione

$$A = \begin{pmatrix} a_{11} & \dots & a_{1N} \\ \dots & \dots & \dots \\ a_{N1} & \dots & a_{NN} \end{pmatrix}$$

- Osservazioni

$$V = \{v_1, \dots, v_N\}$$

La caratteristica principale è che gli stati sono osservabili, questo spesso è limitante, perché l'osservazione che faccio a volte non mi serve perché guardando il simbolo emesso capisco lo stato in cui mi trovo. Alle volte voglio modellare un modello in cui questo non è possibile.

Posso osservare le conseguenze dell'evoluzione di un sistema ma non è un'osservazione certa.

