Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Big Data

a.a. 2022/2023

Lezione12: Metodi Nearest Neighbor

Davide Tosi

Dipartimento di Scienze Teoriche e Applicate

davide.tosi@uninsubria.it

Come Misurare le Distanze

Dati due punti p e q in d dimensioni, una delle metriche più note per misurare la loro distanza è data dalla **Matrice Euclidea**:

$$d(p,q) = \sqrt{\sum_{i=1}^{d} |p_i - qi|^2}$$

Come Misurare le Distanze

Proprietà matematiche che devono soddisfare le **misure di distanza** per essere delle **metriche**:

- *Positivity*: $d(x, y) \geq 0 \ \ per\ tutti\ x\ e\ y$
- *Identity*: $d(x, y) = 0 \ \ sse\ x = y$
- *Symmetry*: $d(x, y) = d(y, x) \ \ per\ tutti\ x\ e\ y$
- *Triangle inequality*: $d(x, y) \leq d(x, z) + d(z, y) \ \ per\ tutti\ x, y\ e\ z$

Come Misurare le Distanze

Misure di Distanza sono sostanzialmente diverse dagli score di similarità (es. coeff. di correlazione) nella loro direzione di crescita

Le seguenti misure di similarità sono metriche di distanza?

- Coefficiente di correlazione
- Somiglianza del coseno (o prodotto scalare)
- Tempi di percorrenza in una rete diretta
- Biglietto aereo più economico

Come Misurare le Distanze

Misure di Distanza sono sostanzialmente diverse dagli score di similarità (es. coeff. di correlazione) nella loro direzione di crescita

Le seguenti misure di similarità sono metriche di distanza?

- Coefficiente di correlazione (da -1 a 1)
- Somiglianza del coseno (o prodotto scalare) (da -1 a 1)
- Tempi di percorrenza in una rete diretta (a volte non simmetriche)
- Biglietto aereo più economico (disuguaglianza del triangolo)

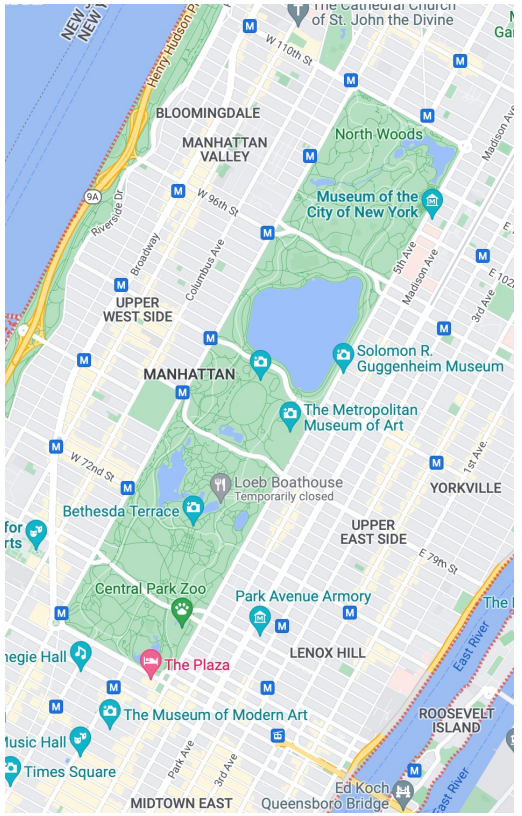
Come Misurare le Distanze

L_k Distance Norms è una generalizzazione della Distanza Euclidea:

$$d_k(p,q) = \sqrt[k]{\sum_{i=1}^{d} |p_i - qi|^k}$$

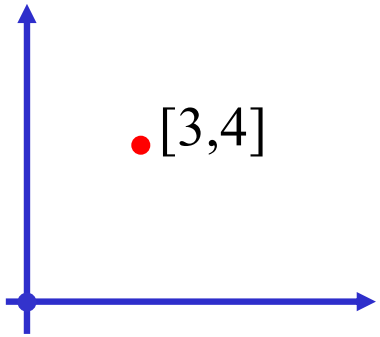
- Manhattan distance metric ($k = 1$)
- Euclidean distance metric ($k = 2$)
- Maximum component metric ($k = \infty$)

k regola il compromesso tra differenza dimensionale massima e totale.



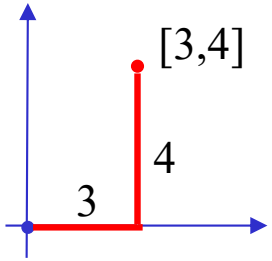
Come Misurare le Distanze

L_k Distance Norms è una generalizzazione della Distanza Euclidea:



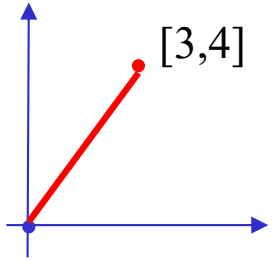
$$d_k(p,q) = \sqrt[k]{\sum_{i=1}^{d} |p_i - qi|^k}$$

- Manhattan distance metric ($k = 1$)



$$d = 4 + 3$$

- Euclidean distance metric ($k = 2$)



$$d = \sqrt[2]{4^2 + 3^2}$$

- Maximum component metric ($k = \infty$)

$$d = \max(3,4)$$

Come Misurare le Distanze

Esempio. L_k Distance Norms

Dati due punti P = [3,4,2] e Q = [1,2,6] calcoliamo le L_k distanze:

$L_1(P,Q) = |3-1|+|4-2|+|2-6| = 2+2+4 = 8$

$L_\infty(P,Q) = \max(|3-1| , |4-2| , |2-6|) = \max(2,2,4) = 4$

Come Misurare le Distanze

Esercizio. **L_k Distance Norms**

Aggiungiamo un punto C=[0,0,0]

Determinare quale tra i 2 punti P = [3,4,2] e Q = [1,2,6] è più vicino a C

Come Misurare le Distanze

Esercizio. L_k Distance Norms

Aggiungiamo un punto C=[0,0,0]

Determinare quale tra i 2 punti P = [3,4,2] e Q = [1,2,6] è più vicino a C

$L_1(C,P) = |3-0|+|4-0|+|2-0| = 3+4+2 = 9$

$L_1(C,Q) = |1-0|+|2-0|+|6-0| = 1+2+6 = 9$

$L_\infty(C,P) = \max(|3-0| , |4-0| , |2-0|) = \max(3,4,2) = 4$

$L_\infty(C,Q) = \max(|1-0| , |2-0| , |6-0|) = \max(1,2,6) = 6$

Come Misurare le Distanze

Siamo a nostro agio con la Distanza Euclidea perché viviamo in un mondo euclideo.

Crediamo nella verità del Teorema di Pitagora (i lati di un triangolo rettangolo obbediscono alla relazione che $a^2 + b^2 = c^2$).

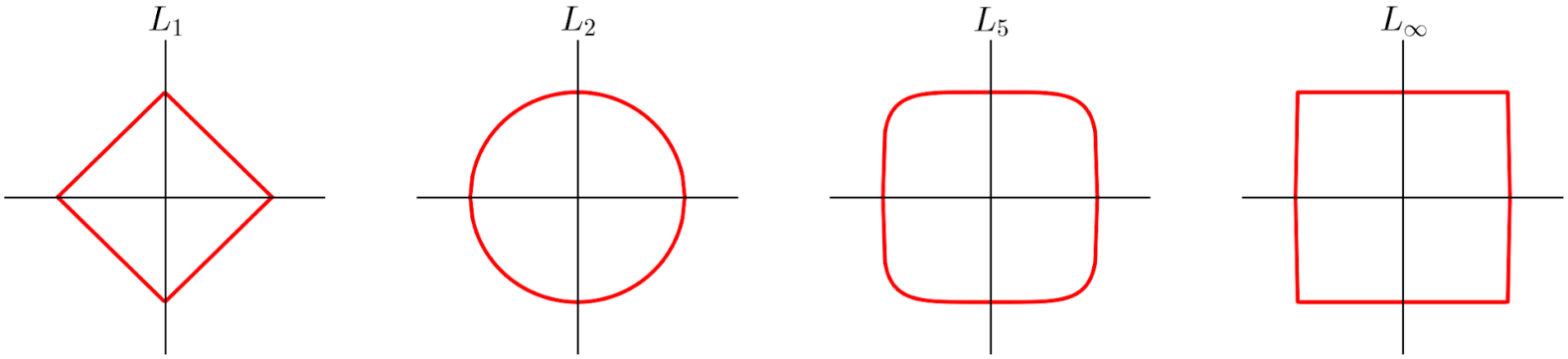
Ma nel mondo delle L_k distanze, il teorema di Pitagora diventa $a^k + b^k = c^k$

Allo stesso modo siamo a nostro agio con l'idea che i **cerchi siano rotondi**

(i.e., l'insieme di punti che si trovano a distanza *r* dal centro *c*)

Come Misurare le Distanze

La forma di un ''**L_k-cerchio**'' determina quali punti sono ugual vicini (**neighbor**) rispetto al centro *c*.



Selezionare il valore di k equivale a scegliere quale cerchio si adatta meglio al nostro modello di dominio. La distinzione diventa particolarmente importante negli spazi dimensionali superiori: ci interessano le deviazioni in tutte le dimensioni o principalmente le più grandi?

Lavorare in Dimensioni >3

Lavorare con set di dati di dimensioni superiori a 3 è complesso.

Attraverso **metodi di proiezione**, riduciamo la dimensionalità a livelli che possiamo comprendere. I metodi di proiezione (come Singular Value Decomposition SVD) comprimono o ignorano le dimensioni per ridurre la complessità della rappresentazione.

Se aumenta il numero di dimensioni nel nostro set di dati, allora ogni dimensione è una parte meno importante del tutto.

Diventa importante la scelta della metrica della distanza:

- se vogliamo premiare i punti per essere vicini su molte dimensioni, allora preferiamo una metrica L_1
- se invece le cose sono simili quando non ci sono singoli campi di grande dissomiglianza, allora L_∞

Lavorare in Dimensioni >3

Il **metodo di proiezione** SVD (Singular Value Decomposition)

SVD è utilizzato in una vasta gamma di applicazioni (l'elaborazione e classificazione d'immagini, compressione dati, ricerca di informazioni)

Esempio: Immaginiamo si avere n sensori distribuiti sul territorio italiano per la raccolta di dati sismici. Ogni sensore raccoglie dati multidimensionali nella forma: x, y, z, t, i (x, y, z sono le coordinate spaziali del punto di rilevamento; t il timestamp; i l'intensità sismica)

Applicando la SVD a questa matrice, possiamo per es. studiare la struttura spazio-temporale delle attività sismiche nell'area monitorata.

Riducendo la dimensionalità dei dati, selezionando solo le componenti più significative del dataset, facilitiamo la trasmissione o l'archiviazione del dataset.

Lavorare in Dimensioni >3

Il **metodo di proiezione** SVD (Singular Value Decomposition)

SVD è utilizzato in una vasta gamma di applicazioni (l'elaborazione e classificazione d'immagini, compressione dati, ricerca di informazioni)

Come: L'Algoritmo SVD decompone una matrice multidimensionale di dati in 3 matrici più piccole $M = U * S * V^T$

- U una matrice ortogonale di vettori singolari $m * m$
- S una matrice di valori singolari sulla diagonale $m * n$
- V una matrice ortogonale $n * n$

Questa decomposizione consente di identificare le parti più importanti dei dati e di rappresentarli in uno spazio di dimensioni più basse, rendendo più facile la loro analisi e la loro comprensione.

Lavorare in Dimensioni >3

Il **metodo di proiezione** SVD (Singular Value Decomposition)

SVD usa il metodo matematico EVD (**EigenValue Decomposition**) per la decomposizione della matrice dataset nelle 3 matrici, dove:

$$M = V * D * V^{-1} \; con\; V\; matrice\; ortogonale, D\; diagonale, V^{-1}\; inversa$$

Un po' di teoria…

Ogni matrice M simmetrica $n * n$ può essere decomposta nella somma dei sui n prodotti eigenvector U_i.

Ogni eigenvector U_i è una matrice $n * 1$ quindi il prodotto con la sua trasposta T è la matrice prodotto $n * n\; U_i U_i^T$

$$M = \sum_{i=1}^{n} \lambda_i U_i U_i^T$$

Lavorare in Dimensioni >3

Il **metodo di proiezione** SVD (Singular Value Decomposition)

Esempio. Data la Matrice $M = \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix}$ calcolarne la sua EVD

Step1. calcolo i valori propri di M: $\det(M - \lambda I) = 0 \rightarrow \begin{bmatrix} 3-\lambda & 2 \\ 1 & 4-\lambda \end{bmatrix}$

$(3-\lambda)(4-\lambda) - 2 * 1 \rightarrow \lambda^2 - 7\lambda + 10 = 0 \rightarrow \lambda_1 = 5, \lambda_2 = 2$

Step2. calcolo vettori propri di M: $Mv = \lambda v \ \forall\lambda$

Per $\lambda_1 = 5$: $\begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5x \\ 5y \end{bmatrix} \rightarrow \begin{cases} 3x + 2y = 5x \\ 1x + 4y = 5y \end{cases} \rightarrow V_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Per $\lambda_2 = 2$: $\begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix} \rightarrow \begin{cases} 3x + 2y = 2x \\ 1x + 4y = 2y \end{cases} \rightarrow V_2 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$

Step3. Matrice diagonale $\boldsymbol{D} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} \mathbf{5} & \mathbf{0} \\ \mathbf{0} & \mathbf{2} \end{bmatrix}$

Step4. Matrice ortogonale $\boldsymbol{V} = \begin{bmatrix} \mathbf{1} & \mathbf{-2} \\ \mathbf{1} & \mathbf{1} \end{bmatrix} \rightarrow \boldsymbol{V^{-1}} = \begin{bmatrix} \mathbf{1/3} & \mathbf{2/3} \\ \mathbf{-1/3} & \mathbf{1/3} \end{bmatrix}$ $essendo: V * V^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Lavorare in Dimensioni >3

Il **metodo di proiezione** SVD (Singular Value Decomposition)

Tutto questo per una matrice 2x2…

Per fortuna abbiamo la libreria Python **NumPy** per calcolare la decomposizione SVD e EVD di una matrice tramite le funzioni: `numpy.linalg.svd()` e `numpy.linalg.eig()`

Librerie Python equivalenti sono:

- **SciPy** con la funzione `scipy.linalg.svd()`
- **scikit-learn** con la funzione `TruncatedSVD()`
- **PyTorch** con la funzione `torch.svc()`

Egualitarismo Dimensionale

Tutte le **metriche di distanza L_k** pesano implicitamente ogni dimensione in egual modo.

Ma ricorda che le dimensioni contano: alcune caratteristiche possono essere più importanti di altre per la similarità.

Usiamo coeff. c per pesare in modo diverso ogni dimensione: $d_k(p,q) = \sqrt[k]{\sum_{i=1}^{d} c_i |p_i - q_i|^k}$

Questo approccio funziona **sse** siamo sicuri di quali e quanto le caratteristiche siano più o meno importanti.

Usiamo inoltre **Normalizzazione Z-score**: sostituire tutti i valori originali con Z-score prima di calcolare le distanze.

Classificazione Nearest Neighbor

La classificazione **Nearest Neighbor (NNC)** si basa sull'idea che un punto in un dataset sia classificato in base alla maggioranza dei punti classificati intorno ad esso.

L'Algoritmo NNC funziona come segue:

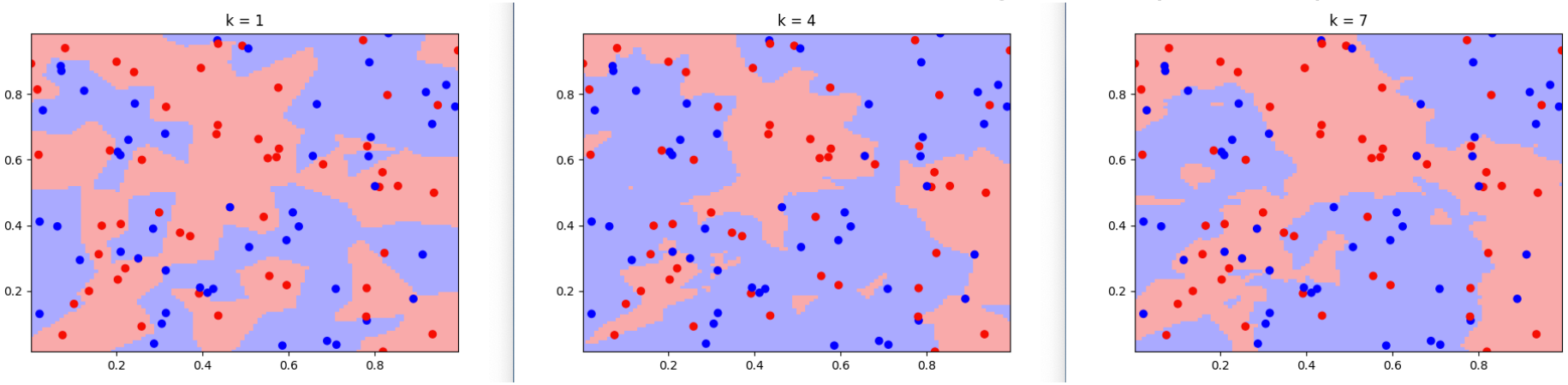
1. Il modello di classificazione viene addestrato su un set di dati di addestramento già etichettati
2. Quando si desidera classificare un nuovo punto, si calcola la distanza tra questo punto e tutti i punti del set di addestramento
3. Il punto del set di addestramento più vicino (o i k punti più vicini, se si utilizza una versione *k-NNC*) viene utilizzato per determinare la classe del nuovo punto

Classificazione Nearest Neighbor

Semplicità: il metodo NNC è semplice e ci permette di capire esattamente cosa sta succedendo (trovare bug o bias)

Interpretabilità: lo studio di un dato punto spiega esattamente perché il classificatore ha preso la decisione che ha preso
(I punti vicini erano etichettati in modo errato? La funzione di distanza non è riuscita a individuare gli elementi corretti?)

Non-linearità: i classificatori NNC hanno confini decisionali che sono lineari a tratti e complessi quanto vogliamo (**k-NCC**)



Classificazione Nearest Neighbor

Dati n punti in d-dimensioni, l'Algoritmo ha complessità **O(nd)** per trovare il NN.

Nell'ambito dei Big Data l'algoritmo NCC as-is è troppo costoso.

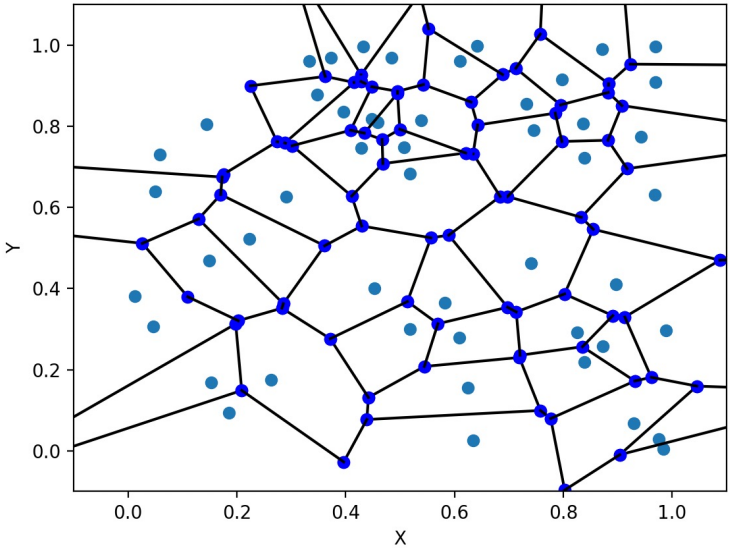
Si usano strutture dati più sofisticate:

- Diagrammi di Voronoi e Alberi KD
- Indici di Griglia
- Hashing sensibile alla località

Classificazione Nearest Neighbor Avanzata

Diagrammi di Voronoi

Dato un insieme di punti S, il **Diagramma di Voronoi** è la partizione del piano in regioni $V(p)\ \forall\ p \in S$ in modo tale che tutti i punti interni a $V(p)$ siano più vicini a p che a ogni altro punto in S.



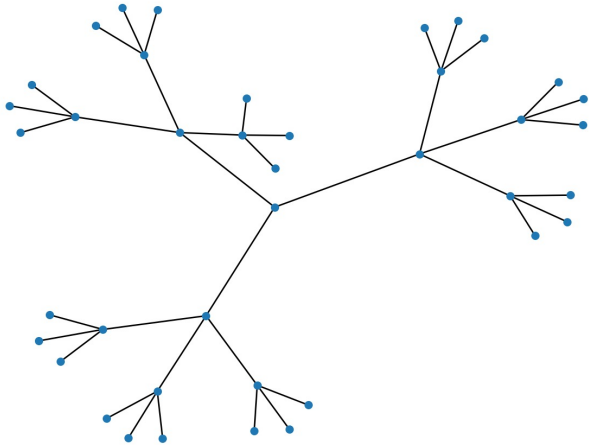
I confini sono definiti dalle bisettrici perpendicolari tra coppie di punti (*a, b*). Ogni bisettrice taglia lo spazio a metà (una per *a* e l'altra *b*).

La descrizione del diagramma di Voronoi di *n* punti in uno spazio *d*-dimensionale richiede $O(n^{d/2})$
→ funziona bene per d=2 o al massimo d=3

Classificazione Nearest Neighbor Avanzata

Diagrammi di Voronoi e Alberi KD

La ricerca dei NN in un diagramma di Voronoi avviene tramite l'uso di strutture dati efficienti come **Alberi KD** (K-dimensional Tree)



Albero binario di ricerca multi-dimensionale,
dove ogni nodo indicizza un punto in k-dimensioni.

Con k=2, il **livello** di un nodo $n \in N$ è definito ricorsivamente come:
$livello(n) = 0 \quad se\ n$ è $la\ radice\ dell'albero$
$livello(n) = livello(p_n) + 1 \quad se\ n\ non$ è $la\ radice\ e\ p$ è $il\ padre\ di\ n$

Classificazione Nearest Neighbor Avanzata

Diagrammi di Voronoi e Alberi KD

Sia n un nodo tale per cui livello(n) sia pari:
- ogni nodo figlio f_1 del sottoalbero sx di n ha che $f_1.xval < n.xval$
- ogni nodo figlio f_2 del sottoalbero dx di n ha che $f_2.xval \geq n.xval$

Sia n un nodo tale per cui livello(n) sia dispari:
- ogni nodo figlio f_1 del sottoalbero sx di n ha che $f_1.yval < n.yval$
- ogni nodo figlio f_2 del sottoalbero dx di n ha che $f_2.yval \geq n.yval$

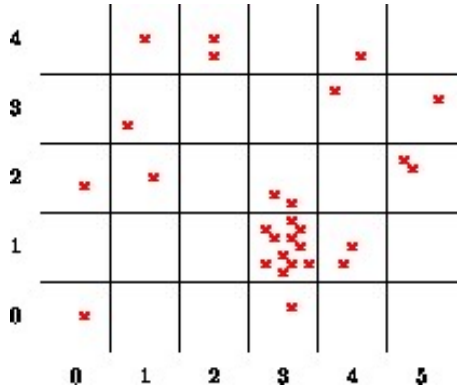
Il tempo di creazione di un Albero KD è: $O(NlogN)$

Il tempo di ricerca di un Punto nell'Albero KD è: $O(logN)$

Classificazione Nearest Neighbor Avanzata

Indici di Griglia

suddivide lo spazio in riquadri d-dimensionali, definendo una **griglia** regolare sullo spazio.
Ogni punto di addestramento viene associato alla cella della griglia a cui appartiene.
Ogni cella ha un indice.



Dato un punto di ricerca (x, y) e un insieme di punti p,
il NN è identificato:

1. trovando il riquadro in cui si trova il punto di ricerca (x, y)
2. esaminando solo i punti presenti nel riquadro identificato
3. tra questi punti, selezionando il NN al punto di ricerca utilizzando la distanza euclidea o qualsiasi altra metrica di distanza

Classificazione Nearest Neighbor Avanzata

Indici di Griglia

Problemi.

1. I punti di addestramento potrebbero non essere distribuiti uniformemente e molte celle potrebbero essere vuote
2. Non è detto che il vicino più prossimo di q viva effettivamente all'interno della stessa cella di q (in particolare se q si trova molto vicino al confine della cella). Dobbiamo cercare anche le celle vicine, per assicurarci di trovare il vicino più vicino in assoluto → costo ricerca cresce
3. Indicizzazione diventa complessa per dimensioni > 2

Classificazione Nearest Neighbor Avanzata

Hashing Sensibile alla Località (LSH)

è un metodo utilizzato per accelerare la ricerca di punti simili in un insieme di dati.

(Qui hashing funziona esattamente al contrario di come siete abituati a pensare. Es. crittografia).

LSH usa funzione hash $h(p)$ che prende un punto (o un vettore) come input e produce in output un hash tale che, con una certa probabilità:

$h(a) = h(b)$ $se\ a\ e\ b\ vicini$
$h(a) \neq h(b)$ $se\ distanti$

Classificazione Nearest Neighbor Avanzata

Hashing Sensibile alla Località (LSH)

LSH memorizza una tabella di hash dove i punti sono raggruppati in base al valore di hash unidimensionale.

Punti simili finiscano nello stesso bucket (o gruppo) di hash.

Per effettuare una ricerca NN con LSH:

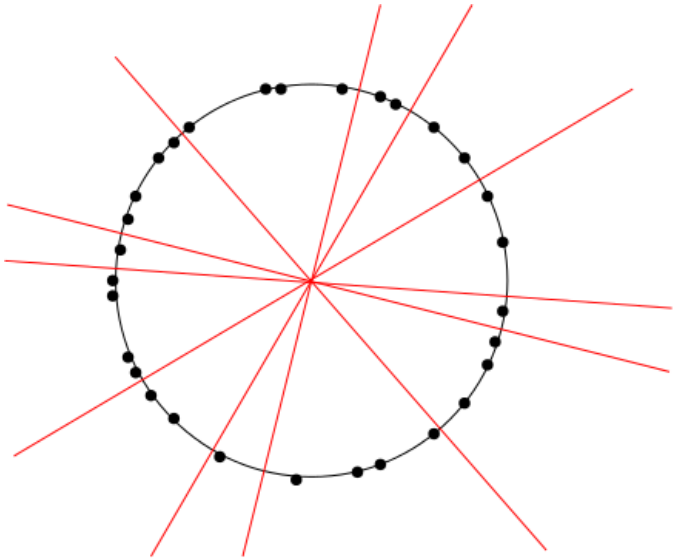
1. si effettua una ricerca nello spazio di hash per trovare i bucket che contengono il punto di ricerca
2. si esegue una ricerca tra i punti presenti in quei bucket per trovare il NN al punto di ricerca

Classificazione Nearest Neighbor Avanzata

Hashing Sensibile alla Località (LSH)

Diversi algo di hashing possono essere usati (es. **MinHash**).

L'Algo dei **Punti su una Sfera** sceglie piani casuali che tagliano l'origine del cerchio.



Se vicini l'uno all'altro, è probabile che due punti si trovino sullo stesso lato (sx o dx) di un dato piano casuale.

I pattern Sx/Dx per d piani casuali definiscono un codice hash LSH a d-bit.

Es. dati n punti, con 2 piani di proiezione casuali su sfera,
si creano 4 bucket b *{SxSx, SxDx, DxSx, DxDx}.*
Hashing associato può essere: *{00, 01, 10, 11}.*
Per ogni bucket dovremmo avere $n/2^b$ punti

Grafi, Reti e Distanze

Molti dataset hanno una naturale interpretazione come grafi/reti:

Social network: i vertici sono le persone, gli archi le amicizie

WWW: i vertici sono le pagine, gli archi gli hyperlink

Reti Produttore/Consumatore: gli archi sono le vendite

Genetica: i vertici sono i geni/proteine, gli archi le loro interazioni

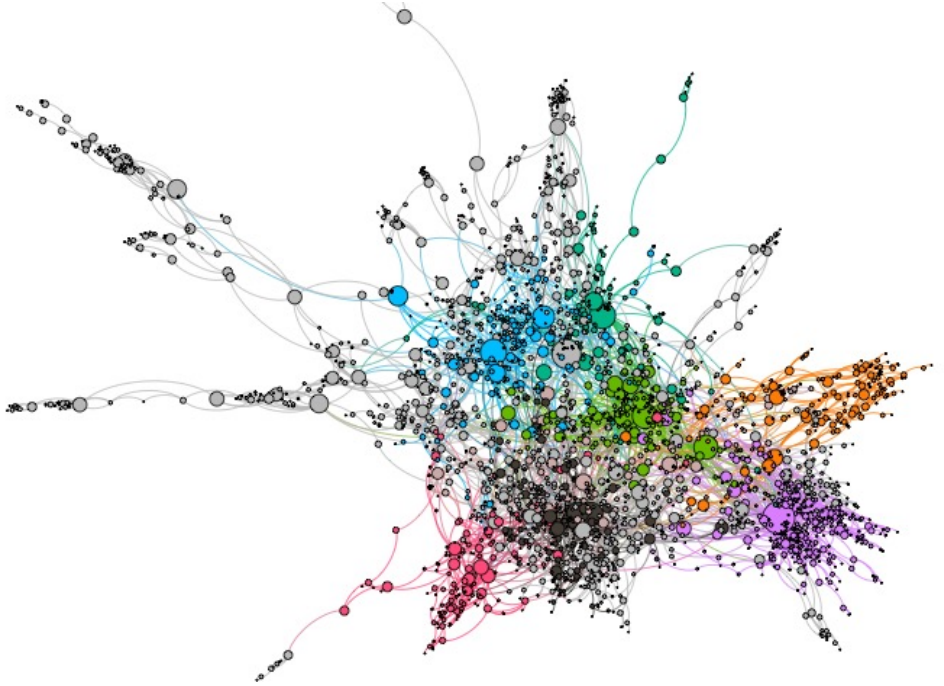
I grafi e i dataset sono oggetti strettamente correlati

- sono composti da entità discrete (punti o vertici) che rappresentano elementi in un insieme
- codificano nozioni importanti di distanza e relazioni, vicine-lontane o connesse-indipendenti
- i dataset possono essere rappresentati in modo significativo da grafi e i grafi da dataset

Grafi, Reti e Distanze

Ricorda quanto hai studiato in altri corsi: i classici algoritmi sui Grafi sono anche utili per l'Analisi Dati

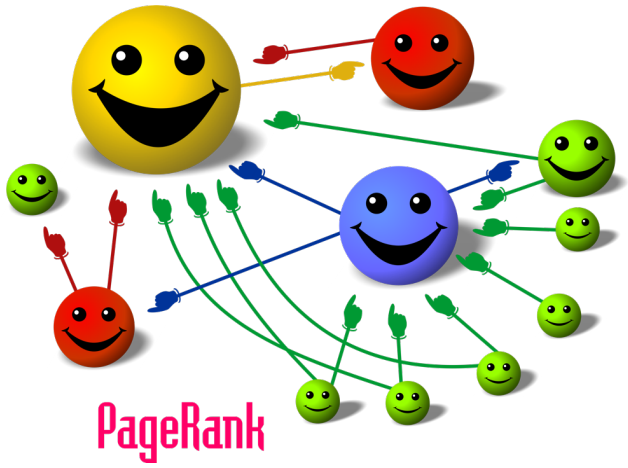
(shortest paths, connected components, spanning trees, cuts, flows, matchings, topological sorting)



Grafi, Reti e Distanze: PageRank Algo

PageRank(v,G) è la probabilità che un random walk in G termini al vertice v.

$$PR_j(v) = \sum_{(u,v)\in E} \frac{PR_{j\_1}(u)}{out - degree(u)}$$



Questa formula ricorsiva su j definisce un algoritmo iterativo, che converge rapidamente nella pratica.

Nella sua essenza, PageRank si basa sull'idea che se tutte le strade portano a Roma, allora Roma è una città importante

(PageRank valuta non solo il numero di archi che collegano un vertice ma anche il ''peso'' di questi archi)

Grafi, Reti e Distanze: PageRank Algo

PageRank(v,G) nella pratica:

- Modifica del grafo per rimuovere vertici/archi irrilevanti (spam)
- Consentire salti casuali tra vertici
- Pesare gli archi

PageRank Wikipedia.it (no filter)	
(1)	Gesù
1	Leonardo Da Vinci
2	Raffaello
3	Michelangelo
4	Caravaggio

Oggi PageRank è meno importante per **Google** di quanto comunemente si pensi.

Tramite analisi del Grafo di PageRank, possiamo determinare i **Nearest Neighbors**!