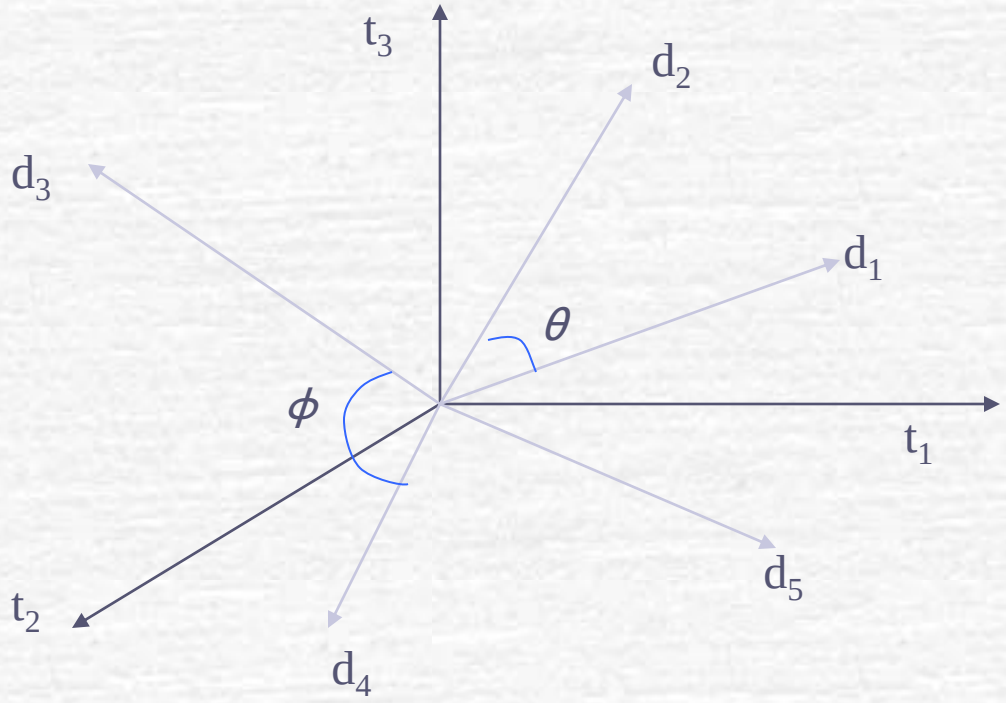
Information Retrieval

Riepilogo

Perché trasformare i documenti in vettori?

1. Per trattare una Query
2. Per cercare documenti simili: se il documento 1 è stato codificato in un vettore $\mathbf{d}_1$ è possibile identificare i documenti "simili" come vettori "vicini"

Intuitivamente



Assunzione: Documenti vicini nello spazio vettoriale "parlano" degli stessi temi

Il *vector space* model

La Query è trasformata in un vettore:

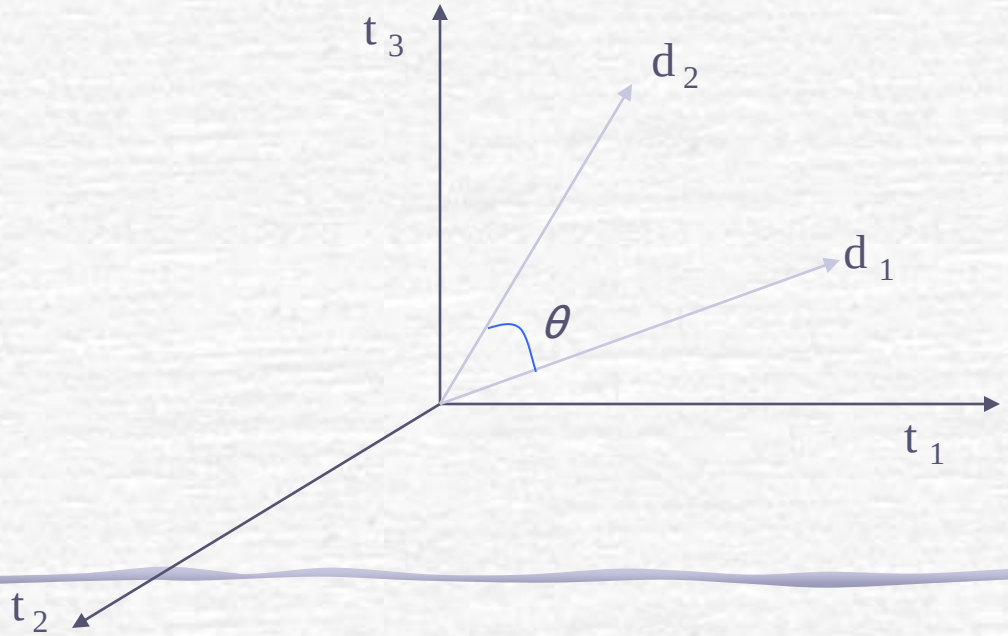
- la query è vista come un "piccolo" documento
- si individuano quindi i documenti, anch'essi codificati in vettori, ordinati per prossimità alla query

Prime considerazioni

- Se misuriamo la distanza fra d_1 e d_2 in termini di distanza euclidea si hanno problemi, poiché avremo che documenti lunghi sono "vicini" soltanto perché sono più lunghi degli altri e non perché trattano di argomenti simili
- Occorre, quindi, introdurre una qualche standardizzazione
- O, comunque, guardare gli angoli che si formano fra i vettori

Similarità dei Coseni

Naturalmente, bisogna tener presente che se ragioniamo sui coseni non è più rispettata la disuguaglianza triangolare e, quindi, non parliamo più di distanze, ma di misure di (dis)similarità



$$sim(d_j, d_k) = \frac{\mathbf{d}_j \cdot \mathbf{d}_k}{\|\mathbf{d}_j\| \|\mathbf{d}_k\|}$$

$$= \frac{\sum_{i=1}^{n} d_{i,j} d_{i,k}}{\sqrt{\sum_{i=1}^{n} d_{i,j}^2} \sqrt{\sum_{i=1}^{n} d_{i,k}^2}}$$

Coseno dell'angolo fra due vettori (normalizzato)

- Lunghezza di un documento vettore

$$\left\|\mathbf{d}_j\right\| = \sqrt{\sum_{i=1}^{n} d_{i,j}^2}$$

- Un vettore *normalizzato* ha lunghezza 1
- Questo vuol dire che tutti i vettori si trovano nella sfera di raggio unitaria
- I vettori normalizzati non hanno bisogno di essere ponderati, infatti:

$$\left\|\mathbf{d}_j\right\| = \sqrt{\sum_{i=1}^{n} d_{i,j}} = 1$$

Vettori normalizzati

- Per i vettori normalizzati, il coseno è semplicemente il loro prodotto scalare:

$$\cos(\mathbf{d}_j, \mathbf{d}_k) = \left|\mathbf{d}_j\right| \cdot \left|\mathbf{d}_k\right|$$

Esercizio

- *Ragionando in termini di similarità misurata dal coseno, ordinate in maniera decrescente i seguenti documenti:*
  - Due documenti che hanno in comune soltanto parole frequenti (articoli, congiunzioni, …)
  - Due documenti che non hanno in comune nessuna parola
  - Due documenti che hanno in comune alcune parole rare

Esercizio

- Distanza euclidea fra due vettori:

$$\left\| \mathbf{d}_j - \mathbf{d}_k \right\| = \sqrt{\sum_{i=1}^{n} \left(d_{i,j} - d_{i,k} \right)^2}$$

- Dimostrare che per vettori normalizzati la distanza euclidea fornisce lo stesso ordinamento della misura di similarità misurata dai coseni

Esempio

- *Ragione e sentimento*, *Orgoglio e pregiudizio* di Jane Austen e *Cime tempestose* di Charlotte Bronte

	SaS	OeP	CT
affection	115	58	20
jealous	10	7	11
gossip	2	0	6

	SaS	OeP	CT
affection	0,996	0,993	0,847
jealous	0,087	0,120	0,466
gossip	0,017	0,000	0,254

- cos(SAS, OeP) = 0,996 x 0,993 + 0,087 x 0,120 + 0,017 x 0,0 = 0,999
- cos(SAS, CT) = 0,996 x 0,847 + 0,087 x 0,466 + 0,017 x 0, 254 = 0,929

Altre considerazioni/digressioni

- Il *vector space model* è stato proposto prima della diffusione del Web. Si pensi alla differenza fra indicizzare un *corpus* statico, oppure una raccolta di documenti dinamica
- Inoltre, oggi ci sono persone (e società) che ad arte agiscono per influenzare l'ordinamento dei motori di ricerca

- Dato l'obiettivo di individuare un ordine di rilevanza nella soddisfazione di una query su una raccolta di documenti, sono state proposti numerosi metodi di ordinamento, sia dal punto di vista del *machine learning*, sia dal punto di vista statistico, in termini di classificazione/regressione
- Oggi il punto di vista si sta modificando, alla luce dei nuovi obiettivi che nascono per i motori di ricerca (**ordinamenti non inquinati**)

- Molte applicazioni sul Web sono basate su applicazioni di ordinamenti:
  - Text information retrieval
  - Ricerca di similarità fra immagini (QBIC)
  - Raccomandazioni di libri, film, corrispondenti (e-bay)
    - Collaborative filtering

Qual è il punto sull'uso del *vector space model*?

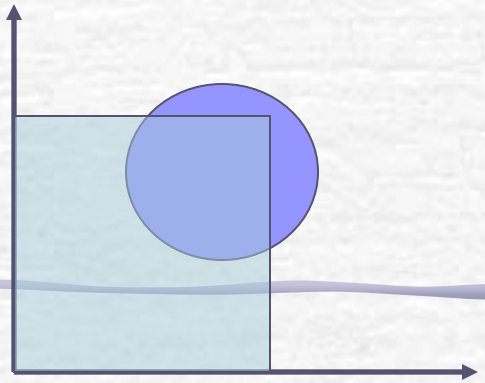
- Una query è vista come un documento (molto) corto
- La query diviene quindi un vettore nello stesso spazio dei documenti
- Possiamo quindi misurare la prossimità di tutti i documenti e la query
- Si ha così una misura naturale di punteggio/ordinamento
  - NOTA: la query, come i documenti è espressa nella codifica "bag of word"

Vettori e frasi

- Le frasi non entrano naturalmente in un mondo di *vector space*:
  - gli indici di posizione non riescono a cogliere le informazioni relative ai tf/idf di strutture non elementari
- E' possibile in fase di pre-trattamento identificare strutture complesse, quali le polirematiche, e decidere di trattarle insieme
- Attenzione: non ci si può aspettare che l'utente che formula la query abbia chiaro, o comunque ponga attenzione a questo tipo di problemi

Vettori e query booleana

- Vettori e query booleane non vanno molto d'accordo
- Nello spazio delle parole, la prossimità fra vettori è misurata in termini di sfere: ad esempio, si considerano tutti i documenti che abbiano un coseno di similarità ≥0,5 con la query
- Le query booleane, d'altra parte, selezionano sulla base di iper-rettangoli e le loro unioni/intersezioni



Vettori e query in linguaggio naturale

- Le query più adatte nel *vector space model* sono quelle espresse come *bag-of-words*, senza una sintassi specifica e, quindi, sostanzialmente in linguaggio naturale, filtrato per identificare le *content bearing words*

Ordinamento efficiente sui coseni

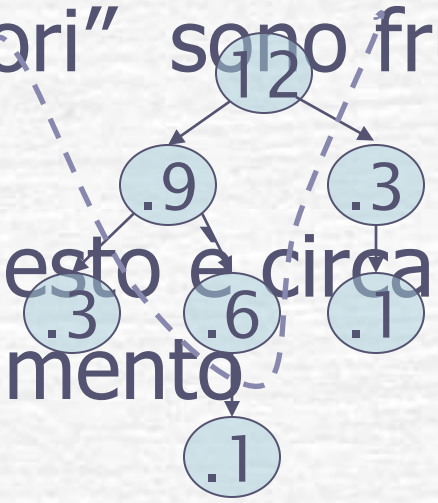
- Si tratta di risolvere un problema di individuazione dei "k-nearest neighbor" (vicini reciproci) di una query
- Trovare i k documenti nel corpus "più vicini" alla query $\Rightarrow$ trovare i k coseni query-documento più grandi
- Ordinarli in maniera efficiente:
  - Calcolare i singoli coseni im maniera efficiente
  - Scegliere i *k* coseni più grandi in maniera efficiente
    - E' possibile farlo senza calcolare tutti gli *n coseni*?

Calcolare i *k* coseni più grandi: selezione o ordinamento?

- L'obiettivo è cercare di evitare l'ordinamento di TUTTI i documenti nel corpus
- Il ricorso ad alberi binari può essere una soluzione

L'utilizzo di alberi binari

- Negli alberi binari il valore associato ai nodi è > del valore associato ai figli
- Occorrono *2n* operazioni per costruirli, quindi, *k* log *n* "vincitori" sono frutto di 2log *n* passi
- Per *n*=1M, *k*=100, questo è circa il 10% del costo di un ordinamento



Colli di bottiglia

- E' comunque necessario calcolare i coseni fra la query e tutti gli *n* documenti del *corpus*
- Si possono selezionare soltanto i coseni non-nulli
- Si possono ulteriormente selezionare i coseni non-nulli con parole rare
- Si possono trovare altri espedienti, ma, comunque

COSTO COMPUTAZIONALE ELEVATO

Candidati "Term-wise"

- Pre-trattamento: calcolare, per ogni "parola", i suoi *m* documenti "più vicini"
  - Trattare ogni parola come una query di un termine
  - Risultato: lista dei "favoriti" per ogni parola
- Ricerca:
  - Per ogni query a *t*-termini, si prende l'unione S delle loro liste di "favoriti"

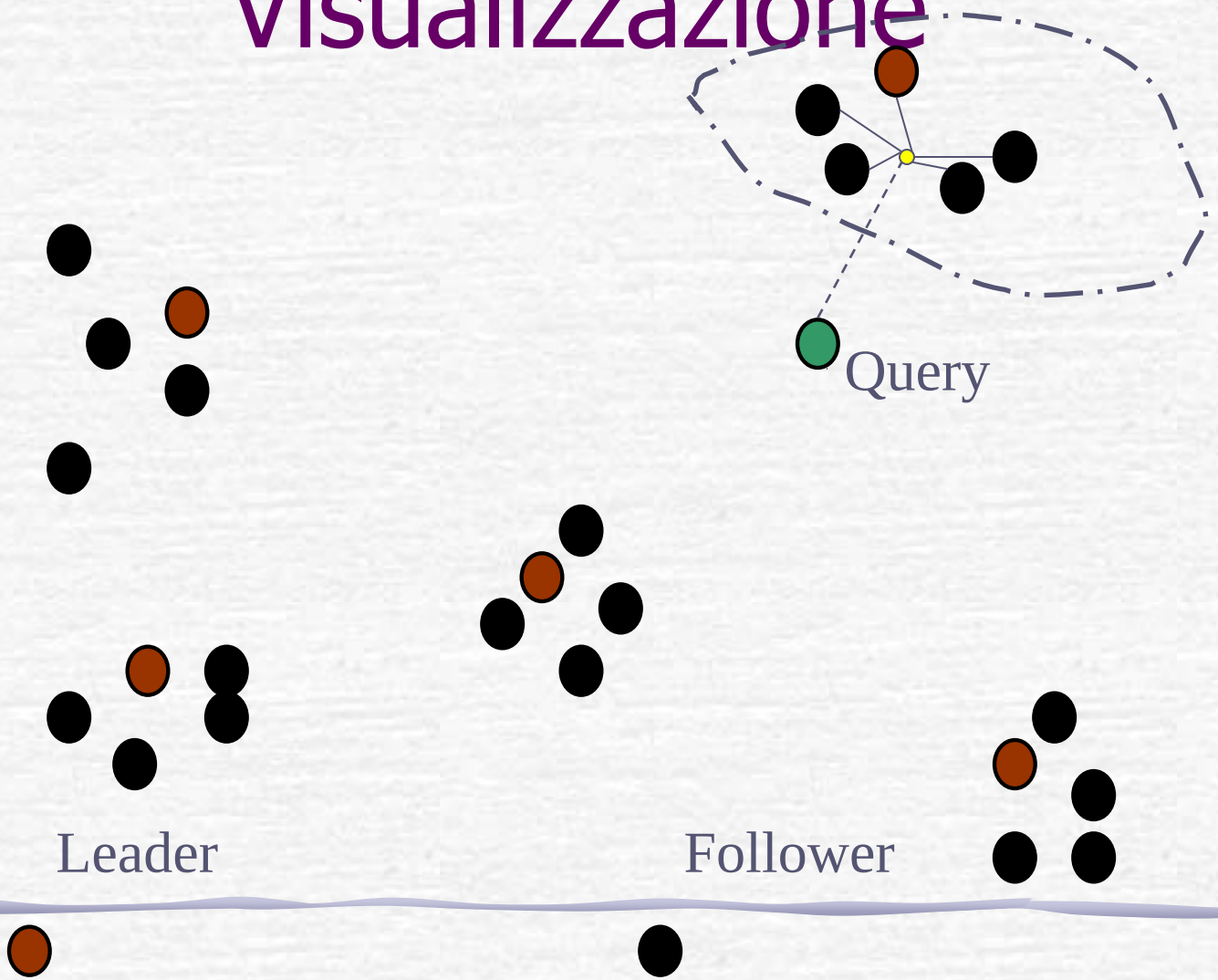
    $card(S) \leq mt$.
  - Si calcolano i coseni fra il vettore query e i documenti in *S* e si scelgono i *k* principali

Empiricamente si è visto che conviene prendere $m>k$

Pruning delle classi

- Fase di pre-trattamento:
  - Si prendono $\sqrt{n}$ documenti a caso, detti *leader*
  - Per ciascuno degli altri documenti (*follower*), si calcola il rispettivo (più vicino) *leader*
  - In media ogni *leader* avrà $\sqrt{n}$ *follower*
- Ricerca:
  - Data una query *Q*, si trova il suo *leader* più vicino *L*
  - Si individuano i *k* documenti più vicini soltanto fra i follower di *L*

Visualizzazione



Varianti

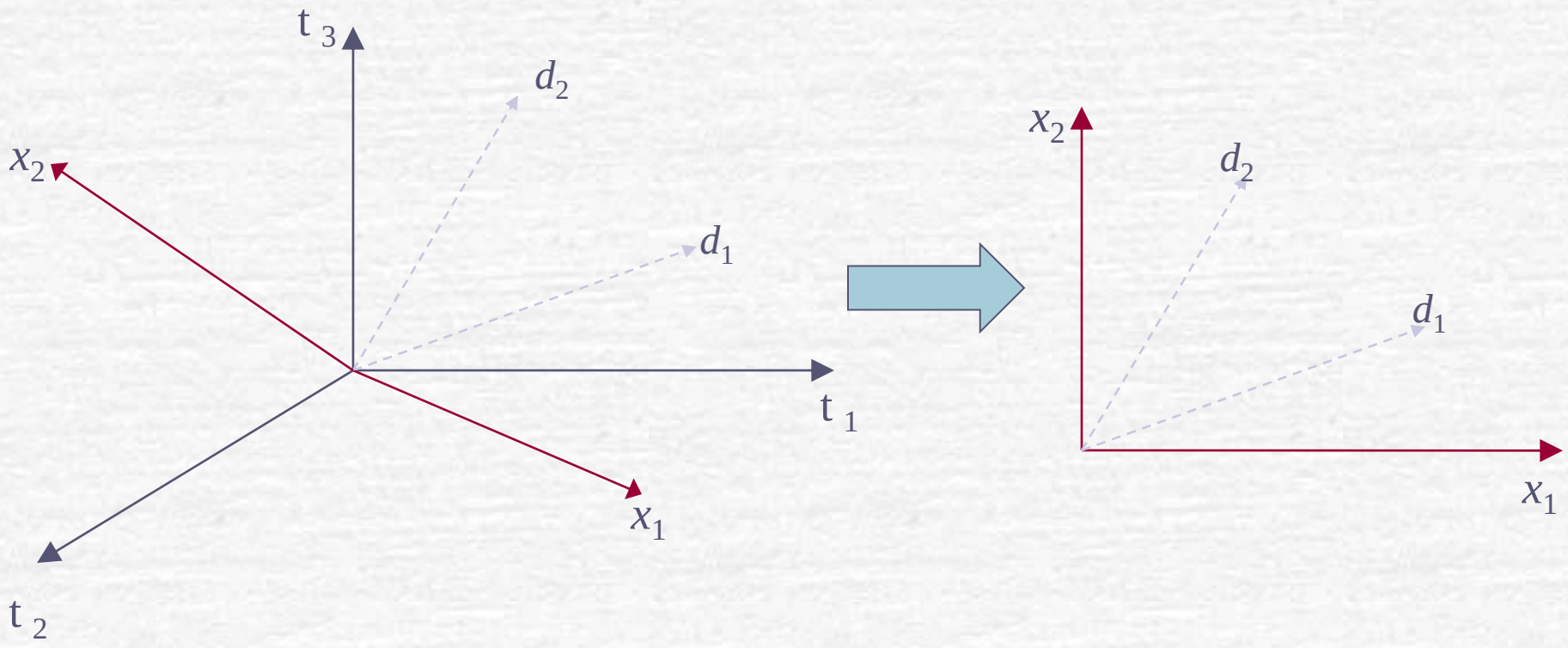
1. Considerare per ogni *follower* più di un *leader*
2. Trovare più *leader* vicini alla query e quindi considerare più *follower*
3. Procedure iterative per migliorare la struttura *leader - follower*

Riduzione della dimensionalità

- Non sarebbe meglio proiettare i nostri vettori in sottospazi di dimensioni inferiori (diciamo da 150000→100) , cercando di salvaguardare al meglio la struttura delle distanze?
- Questo renderebbe i calcoli dei coseni molto più veloce
- Due metodi:
  - "Latent semantic indexing"
  - Proiezione casuale

Proiezione casuale su $k << m$ assi

- Si sceglie a caso un asse x_1 nello spazio vettoriale
- Per $i = 2$ a k,
  - Si sceglie a caso un asse x_i ortogonale a $x_1, x_2, \ldots x_{i-1}$
- Si proiettano tutti i documenti vettore nel sottospazio generato da $\{x_1, x_2, \ldots, x_k\}$
- Sembra non giustificato, ma empiricamente si è riscontrato che funziona bene e che, in linea di massima le distanze non sono molto alterate



x_1 è un asse nello spazio tridimensionale generato da (t_1, t_2, t_3)
x_2 è scelto a caso, col vincolo di ortogonalità a x_1.

Algoritmo della proiezione casuale

- Si proiettano gli n vettori documenti da m-dimensioni a k-dimensioni:
  - Si parte dalla matrice $\mathbf{A}$ $m \times n$ (parole $\times$documenti)
  - Si costruisce casualmente il proiettore ortogonale ($k \times m$) $\mathbf{R}$
  - Si calcola la matrice delle nuove coordinate $\mathbf{W}=\mathbf{RA}$
- le colonne di $\mathbf{W}$ sono le coordinate dei documenti nello spazio a $k \ll m$ dimensioni

Latent semantic indexing (LSI)

- Un'altra tecnica per ridurre la dimensionalità
- La proiezione casuale è "data-*independent*"
- LSI è "data-*dependent*"
  - Elimina informazione "ridondante
  - Consente di identificare prossimità "semantiche"
    - *Ad esempio,* ***car*** e ***automobile***