Dense Word Representation

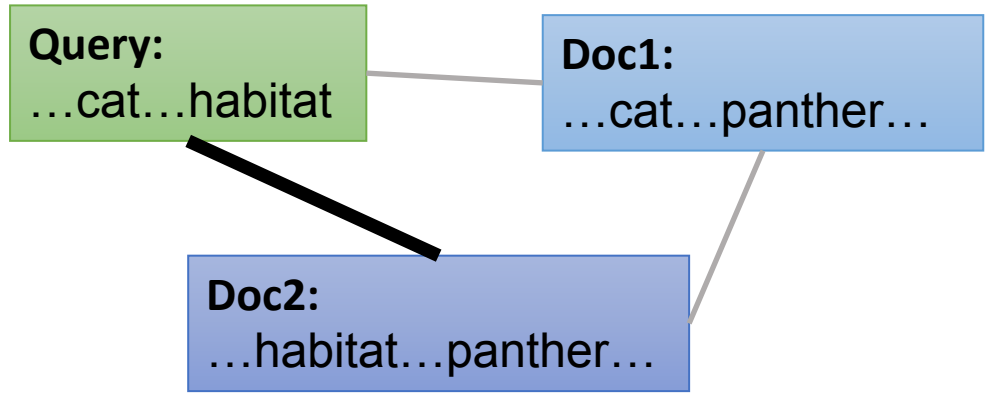
CS6101

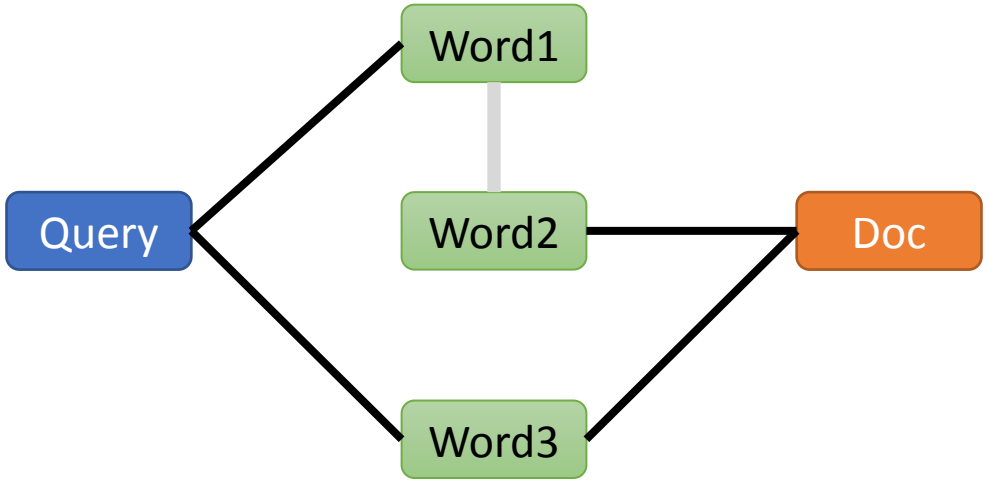
Learning word and text representations

- Historical perspective
- Supervised
  - For ranking (this module)
- Unsupervised (BC)
  - Distributional vectors, word clustering, matrix factorization
- Unsupervised (AD)
  - Word2vec, GloVe (next)
- Contextualized
  - ConvNet, RNN, Transformer (later)
- Pretrained then fine-tuned

Pseudo-relevance feedback (PRF)

- Synonymy and polysemy
- Need to match documents to queries without any shared word
- Practical approach: pseudo-relevance feedback (PRF)
  - Process query from user to get top hits
  - Assume these are relevant
  - Extract keywords from these documents
  - Pad query (perhaps with smaller weight)
  - Process padded query
  - Return merged result lists
- Why stop at two queries?
- How to set magic weights?





Distributional vectors

- Finite labeled data, feature sparsity, and out-of-vocabulary (OOV) words/features have always troubled POS and NER tagging and estimating n-gram statistics
- E.g. we saw car in the training sequences but never sedan, or Shanghai in test data but only other cities in training data
- Unlabeled corpus has enough clues that these are related
- A well-established partial fix is word cluster features
- First proposed by IBM researchers in 1992
- Given a word like tezgüino, collect all context windows (say) at most 11 words wide centered on it
- From these contexts collect a bag of other words, count them
- Possibly transform from raw counts to TFIDF

- A bottle of tezgüino is on the table.
- Everybody likes tezgüino.
- Tezgüino makes you drunk.
- We make tezgüino out of corn.

What is tezgüino?

Word clusters

- Represent as a sparse vector in a space as large as the corpus vocabulary
  - perhaps scale to unit length
- This is the distributional vector for tezgüino, sort of a mini-doc
  - bottle, drunk, corn
- Turns out the d.v.'s of similar/related words are similar
  - tezgüino vs tequilla vs toddy
- Can cluster these d.v.'s using standard clustering tools; see https://en.wikipedia.org/wiki/Brown_clustering
  - Best number of clusters may be application-dependent
- In many NLP tasks, one of the features for each token is its cluster ID

Progression of text representation

- Sparse vectors, TFIDF, BM25
- Distributional vectors
- Dense representation of DVs (via PCA, say)
- Latent semantic indexing (SVD)
- Random walks on word-doc bipartite graph
- word2vec and GloVe
- Weighted average of words in passage
- 1d convolutional network
- RNNs, GRUs, LSTMs
- Transformers

PRF as associative retrieval

- If $r(i)$, $s(i, j)$ large, may also include j
  - Even if $r(j)$ is small
- Query {cat}, i includes {cat, panther}, j contains {panther} and is actually more relevant than i
- $r(j) < r(i)$ because of limitation of sparse matching ("lexical gap" between 'cat' and 'panther')
- How can we discover (ideally without supervision) that 'cat' and 'panther' are related?
  - Perpendicular axes in any sparse vector space model

A graph-based formulation

- Nodes i with 'raw' scores $r(i)$
- Undirected edges (i, j) with weights $s(i, j)$
- 'Refined' scores $y(i)$
- Two part objective
  - Want small $\sum_i (y(i) - r(i))^2$ (keep refined scores close to raw scores)
  - Want small $\sum_{(i,j)} s(i, j)(y(i) - y(j))^2$ (if i, j are similar, refined scores should be close)
- Convex optimization; solve for and sort by $y(i)$
- Above we assume $r(i)$ and $s(i, j)$ are fixed inputs

"Graph Laplacian"; we will return to this later

Training graph formulation

- For each query, we are given a subset of relevant nodes $V_{\oplus}$ (remaining nodes $V_{\ominus}$)
- Want $y(g) > y(b)$ for all $g \in V_{\oplus}, b \in V_{\ominus}$
- Usual hinge loss $[y(b) + m - y(g)]_+$
- Three-part loss objective
- If $r(i)$ and $s(i, j)$ are output from neural networks, gradient of loss can be backpropagated into these networks
- Still does not explore word representation

Hardwired → trained 'IDF'

- If we are given sufficient $\{\langle q, D_{q\oplus}, D_{q\ominus}\rangle\}$, can we learn something like IDF?
  - Or better? IDF is not informed by query word distribution
- Represent query q and doc x using normalized TF vectors alone, called $\boldsymbol{q}, \boldsymbol{x} \in \mathbb{R}^W$
- Instead of defining similarity as $\boldsymbol{q} \cdot \boldsymbol{x} = \boldsymbol{q}^\top \boldsymbol{x}$, define as $\boldsymbol{q}^\top \boldsymbol{A}\, \boldsymbol{x}$
- Where $\boldsymbol{A}$ is a $W \times W$ non-negative diagonal matrix
  - E.g., define $\boldsymbol{A} = \exp(\boldsymbol{B})$ or $\boldsymbol{A} = \mathrm{ReLU}(\boldsymbol{B})$ with unconstrained $\boldsymbol{B}$
- $A[w, w]$ represents the importance (similar to IDF) of word w
- Can try RankSVM-style constraint/loss

$$\boldsymbol{q}^\top \boldsymbol{A}\, \boldsymbol{x}_{q\oplus} + \xi_q \geq \boldsymbol{q}^\top \boldsymbol{A}\, \boldsymbol{x}_{q\ominus} + 1$$

Does this work?

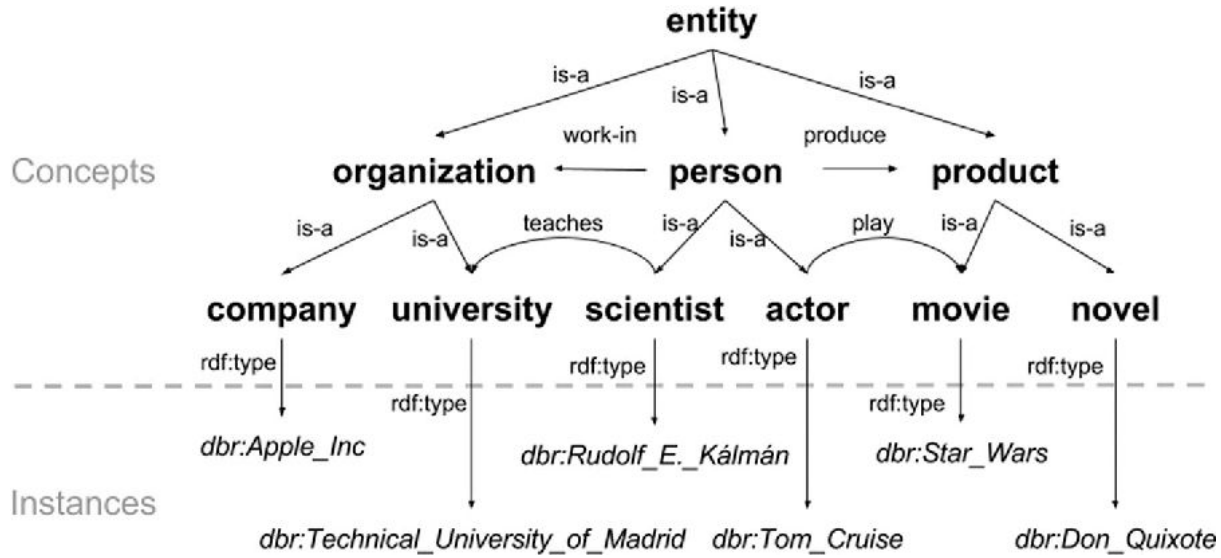
Programming assignment?

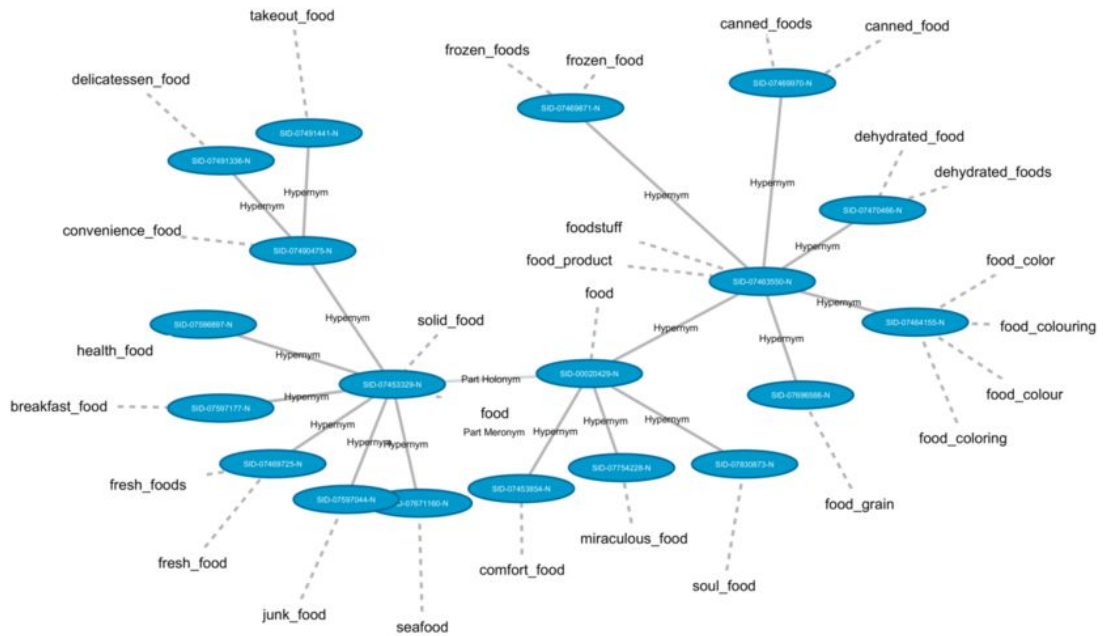
Structure of A

- Can we discover that ideally $A[\text{scold}, \text{rebuke}] > 0$?
  - Can improve recall
- We can relax $\boldsymbol{A}$ to a general matrix
  - Should $\boldsymbol{A}$ be symmetric? Diagonal-dominant? Sparse?
  - Do we have enough training data?
- Which $A[i, j]$ do we expect to be large?

Constraining A

- One way is to model
  $$A = \mathbb{I} + \lambda\ \mathrm{ReLU}(B^\top B)$$
  - where $B \in \mathbb{R}^{M \times W}$ (low-rank)
  - and λ is kept small
- Another way is to use lexical networks like wordnet
  - Allow $A[i, j] > 0$ only if words i, j are connected in wordnet
  - by some restricted types of edges (synonym, hypo/hypernym)

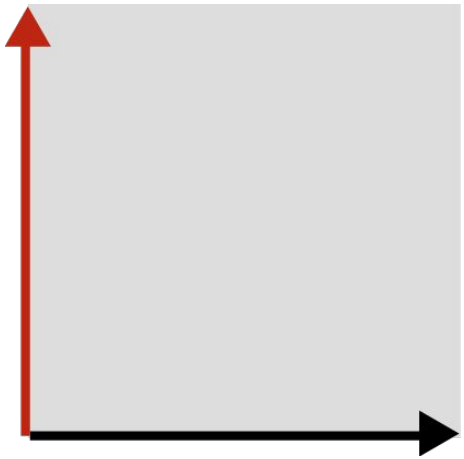




Finishing up

- In any case, similarity $s_B(q, x_i)$
- Now the loss is pairwise
  - Aka "triplet loss", the third element being q
- $\sum_q \sum_{g,b} \mathrm{ReLU}(\mathrm{margin} + s_B(q, b) - s_B(q, g))$
  - May need to tune margin here
- Minimize pairwise loss wrt B (or sparse A) to learn translation matrix (with possibly "IDF" on diagonals)
- Stare a bit at $B \in \mathbb{R}^{M \times W}$
  - Looks like M-dim embedding of each word
  - Fitted for end-task of retrieval

Linear transformations and SVD

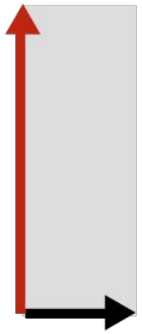


A square with orienting arrows

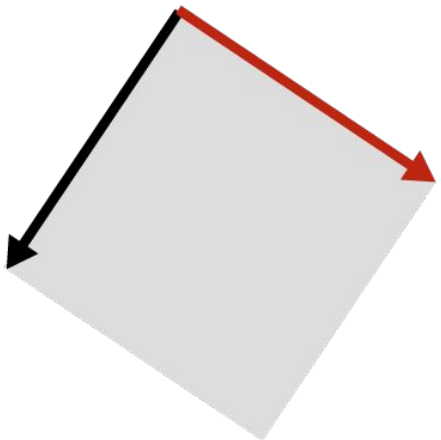
A: stretched on black axis
B: compressed on black axis
C: rotated
D: reflected
E: sheared

A

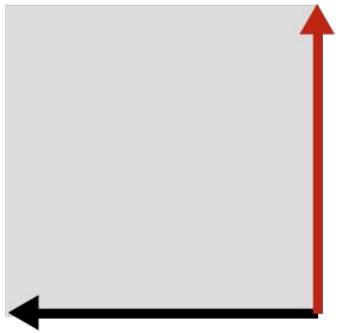
B



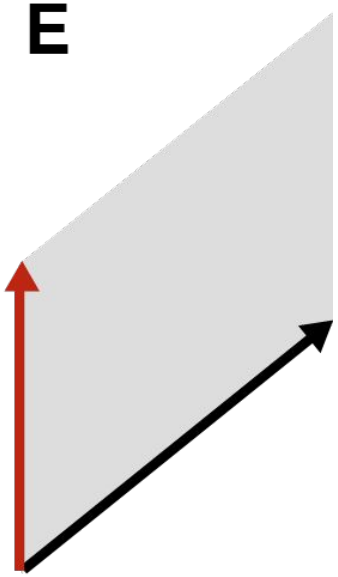
C



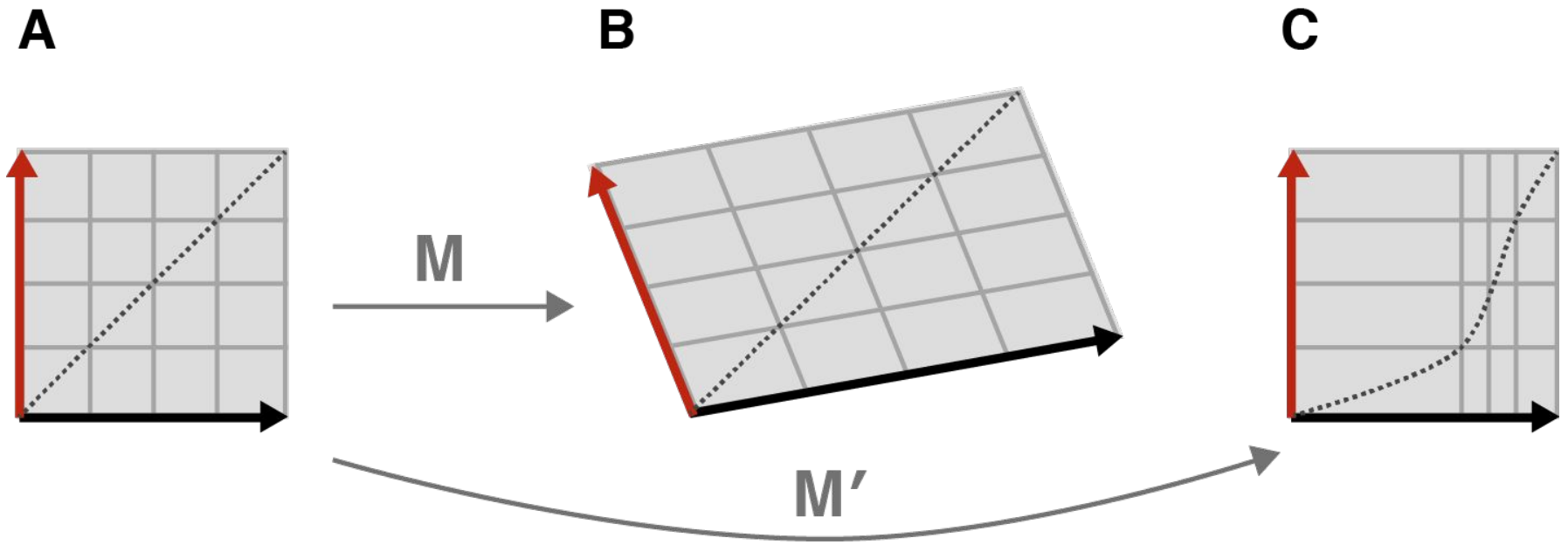
D



E



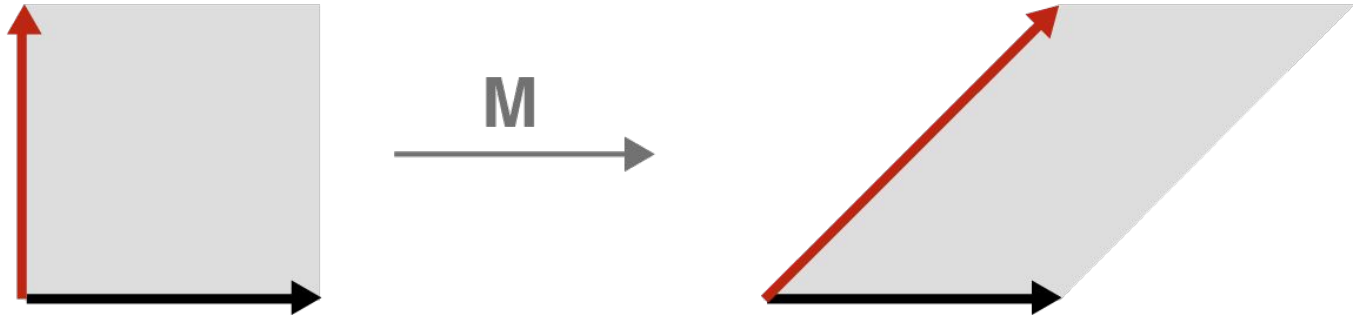
Let M be a linear transformation



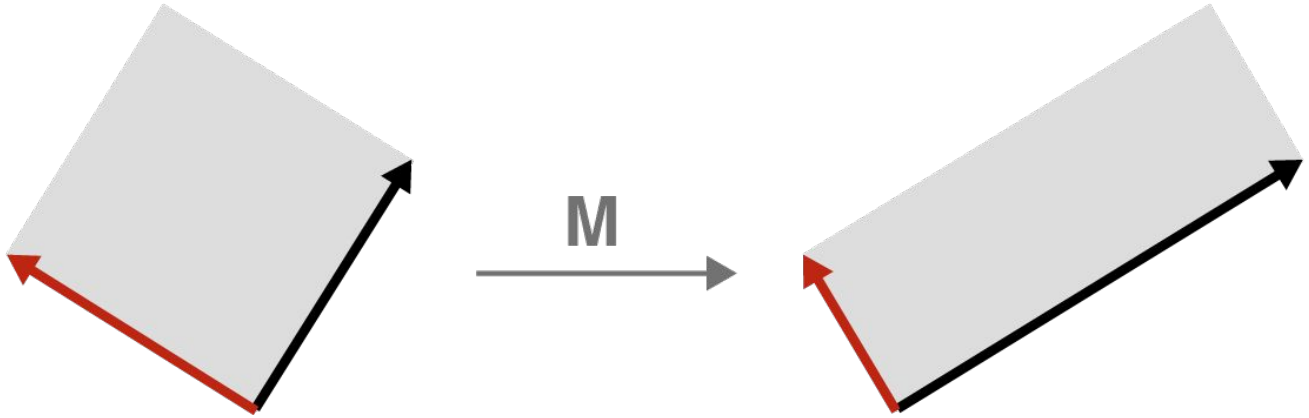
- A: original square
- B: under linear transformation M
  - Linear transformation transforms a straight line to a straight line
- C: under non-linear transformation M'

Rotate, then stretch/compress/reflect

A

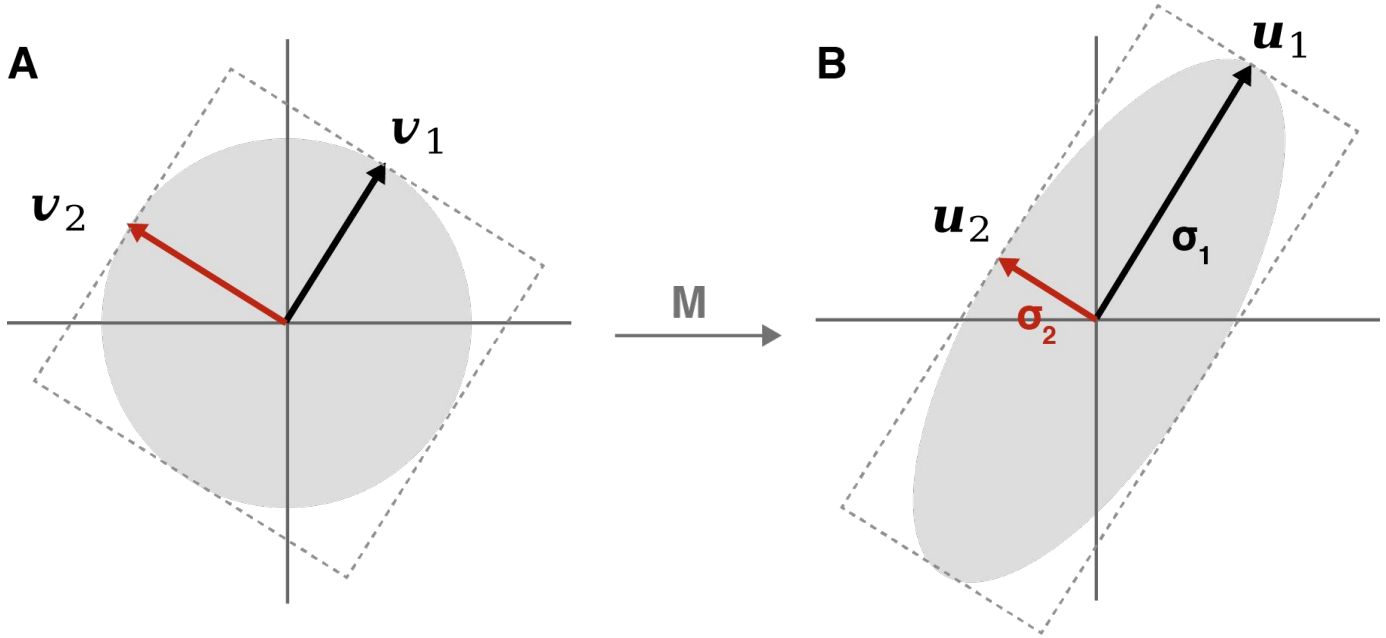


B



- Under M, original square is sheared
- But rotated square is only stretched or compressed

Circle to ellipse



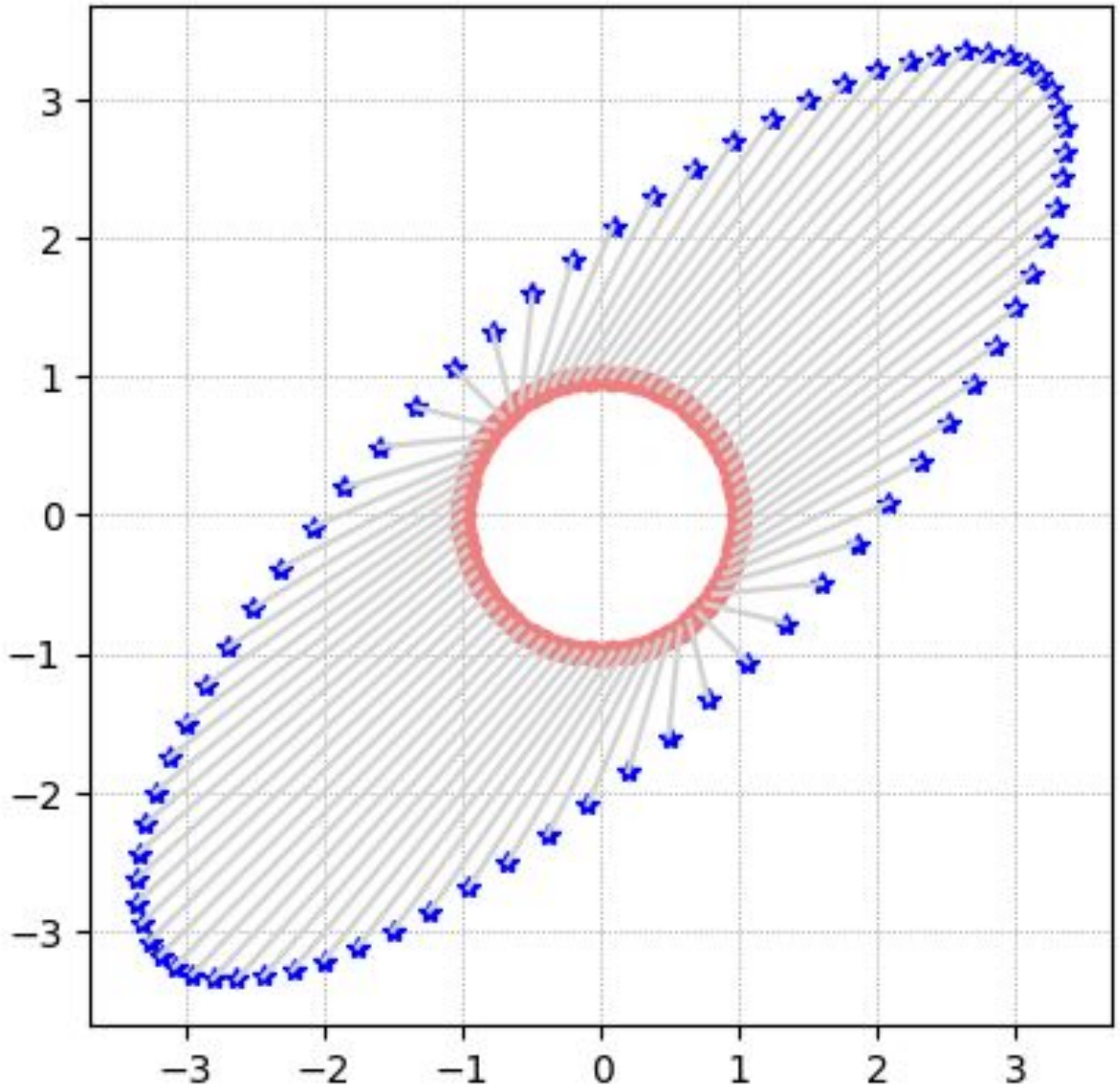
$$M\boldsymbol{v}_1 = \sigma_1\boldsymbol{u}_1$$
$$M\boldsymbol{v}_2 = \sigma_2\boldsymbol{u}_2$$

Where $\boldsymbol{u}_1, \boldsymbol{u}_2$ are unit vectors

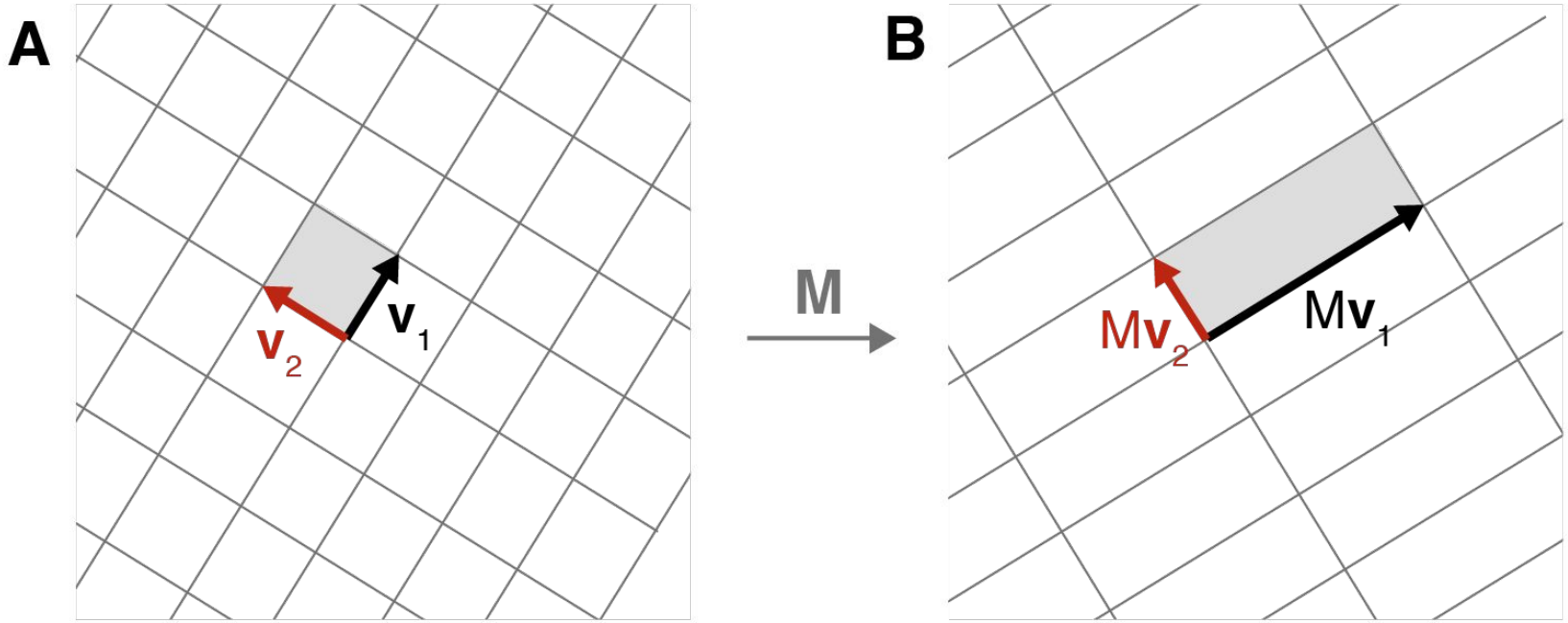
- Inscribe a circle inside the rotated square
- Under M, circle maps to ellipse
- Major and minor axes define stretch/shrink
  - Black and red directions do not change, only length

Effect of matrix transform on unit circle

- $M = \begin{bmatrix} 3 & 1.5 \\ 1.5 & 3 \end{bmatrix}$
- Circle mapped to ellipse
- Two radial lines (major and minor axes) get scaled but not turned
  - Eigenvectors of M
- Also explains why power iteration finds largest eigenvector



In other words



- Domain spanned by (unit) basis vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$
- Pre-rotate it through a specific angle, then apply linear transformation M
- Amounts to tranformation by a diagonal matrix

Transforming any vector x

- Given (unit) basis vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$
- Any vector $\boldsymbol{x}$ can be written as

$$\boldsymbol{x} = (\boldsymbol{x} \cdot \boldsymbol{v}_1)\boldsymbol{v}_1 + (\boldsymbol{x} \cdot \boldsymbol{v}_2)\boldsymbol{v}_2$$

- After transformation by M, we get

$$M\boldsymbol{x} = (\boldsymbol{x} \cdot \boldsymbol{v}_1)M\boldsymbol{v}_1 + (\boldsymbol{x} \cdot \boldsymbol{v}_2)M\boldsymbol{v}_2$$

- Which can be rewritten as

$$M\boldsymbol{x} = (\boldsymbol{x} \cdot \boldsymbol{v}_1)\sigma_1\boldsymbol{u}_1 + (\boldsymbol{x} \cdot \boldsymbol{v}_2)\sigma_2\boldsymbol{u}_2$$

- Rearrange to

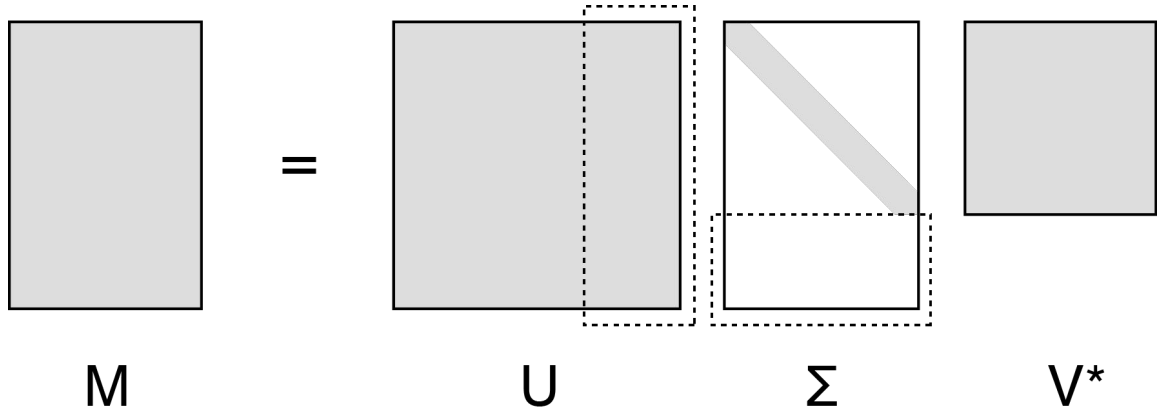
$$M\boldsymbol{x} = \boldsymbol{u}_1\sigma_1(\boldsymbol{x} \cdot \boldsymbol{v}_1) + \boldsymbol{u}_2\sigma_2(\boldsymbol{x} \cdot \boldsymbol{v}_2)$$

- Since this is true for all $\boldsymbol{x}$, we get

$$M = \boldsymbol{u}_1\sigma_1\boldsymbol{v}_1^\top + \boldsymbol{u}_2\sigma_2\boldsymbol{v}_2^\top$$

$$M\boldsymbol{v}_1 = \sigma_1\boldsymbol{u}_1$$
$$M\boldsymbol{v}_2 = \sigma_2\boldsymbol{u}_2$$

Generalizing to more dimensions

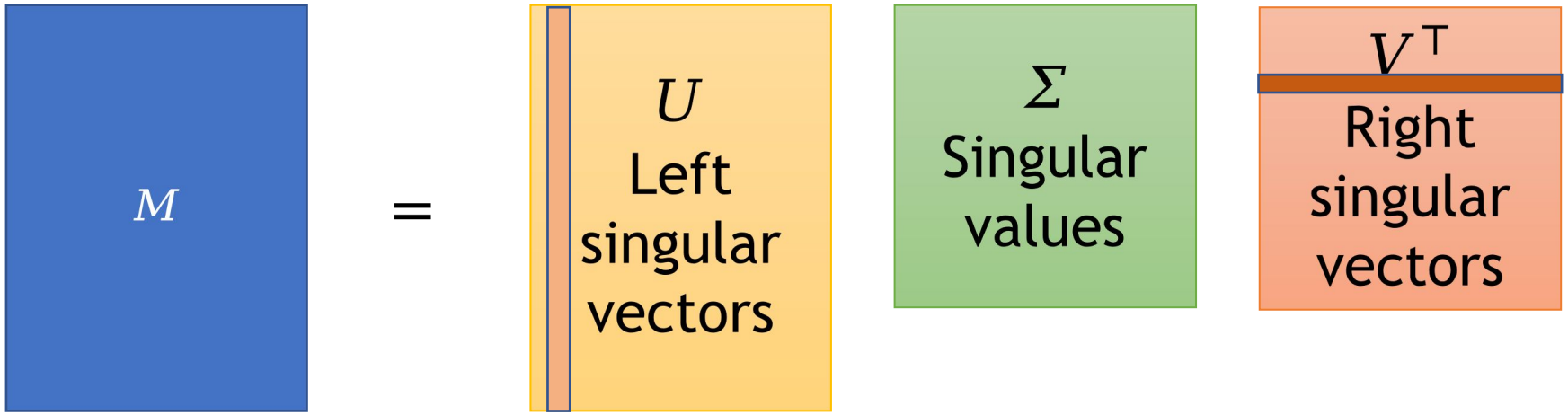


- In 2d, we got $M = [\boldsymbol{u}_1 \ \boldsymbol{u}_2] \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^\top \\ \boldsymbol{v}_2^\top \end{bmatrix}$
- Written compactly as $M = U \Sigma V^\top$
- $\Sigma = \text{diag}(\{\sigma_i\})$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$
- Columns of U are unit vectors, perpendicular to each other
- So are columns of V

Signal to noise

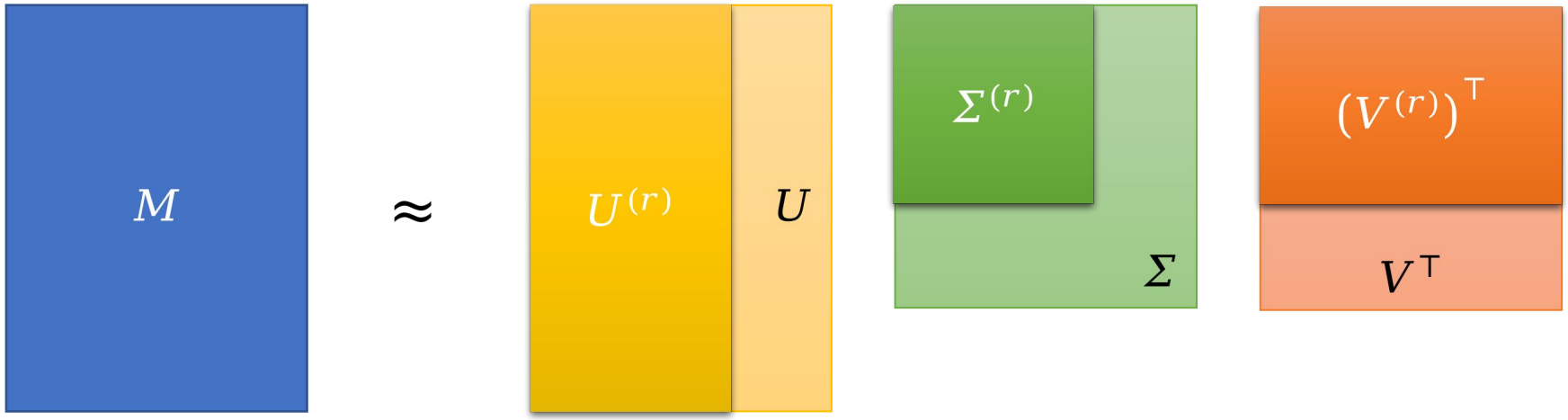
- Rank-revealing decomposition
  - If $\mathrm{rank}(M) = R$, then $\sigma_{R+1} = \sigma_{R+2} = \cdots = 0$
- Frobenius norm of a matrix $\|M\|_F^2 = \sum_{i,j} M_{i,j}^2$
  - "Total energy" in a matrix
- Trace of a square matrix $\mathrm{tr}(A) = \sum_i A_{i,i}$ is the sum of diagonal elements
- $\|M\|_F^2 = \mathrm{tr}(M M^\top) = \mathrm{tr}(U \Sigma V^\top V \Sigma^\top U^\top) = \mathrm{tr}(U \Sigma \Sigma^\top U^\top) = \mathrm{tr}(\Sigma \Sigma^\top U^\top U) = \mathrm{tr}(\Sigma \Sigma^\top) = \sum_i \sigma_i^2$
- "Total energy" of matrix = sum of squares of singular values

Low rank approximation



- Which matrix B with rank $r \leq R$ is the "closest approximation" to input matrix M?
- I.e., minimize $\|M - B\|_F^2$ s.t. $\text{rank}(B) = r$

Low rank approximation



- Which matrix B with rank $r \leq R$ is the "closest approximation" to input matrix M?
- I.e., minimize $\|M - B\|_F^2$ s.t. $\text{rank}(B) = r$
- Solution is $B = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^\top$
- With error $\|M - B\|_F^2 = \sum_{i=r+1}^{R} \sigma_i^2$ = the "lost power"
- What r is sufficient for doc-term matrices? What is approx rank?

Connection to eigen system

- $M = U \Sigma V^{\mathsf{T}}$
- $C = M M^{\mathsf{T}} = U \Sigma V^{\mathsf{T}} V \Sigma^{\mathsf{T}} U^{\mathsf{T}} = U \Sigma \mathbb{I} \Sigma^{\mathsf{T}} U^{\mathsf{T}} = U \Sigma^2 U^{\mathsf{T}}$
- $CU = U \Sigma^2 U^{\mathsf{T}} U = U \Sigma^2 \mathbb{I} = \Sigma^2 U$
- $C\boldsymbol{u}_1 = \sigma_1^2 \boldsymbol{u}_1, C\boldsymbol{u}_2 = \sigma_2^2 \boldsymbol{u}_2$
  - Left singular vectors of M are eigenvectors of $M M^{\mathsf{T}}$
  - Squares of singular values are eigenvalues
- Similar properties for right singular vectors and $M^{\mathsf{T}} M$

Mutually recursive similarity

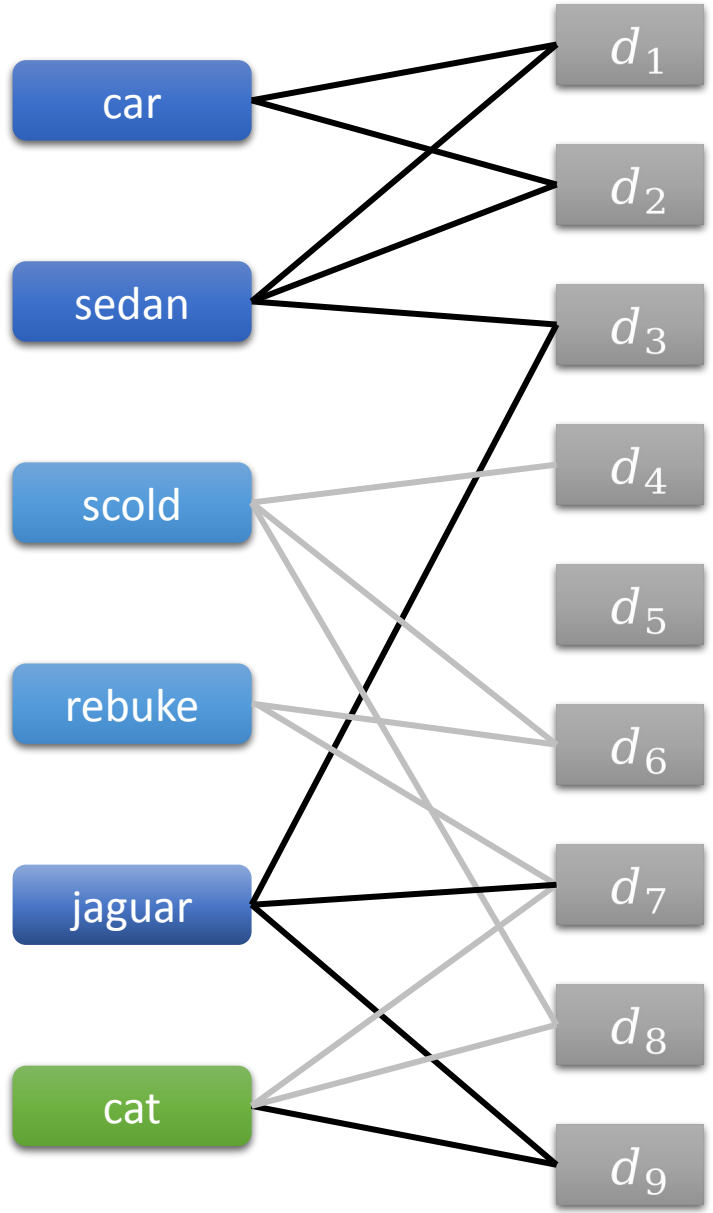
- Two words are similar if they appear in similar documents
- Two documents are similar if they contain similar words
- Turn this into a random walk problem
- Bipartite graph between words and docs
- Expressed via adjacency matrix $E \in \{0,1\}^{W \times D}$

Back to term-doc matrix

- Suppose M is the term (rows) x doc (cols) matrix
- What is $(MM^\top)[i, j]$?
  - Number of docs that contain both words i and j
  - The number of 2-hop paths between bipartite term-document graph — a form of similarity
- Closely related to principal component analysis (PCA)

Random walk on word-doc graph

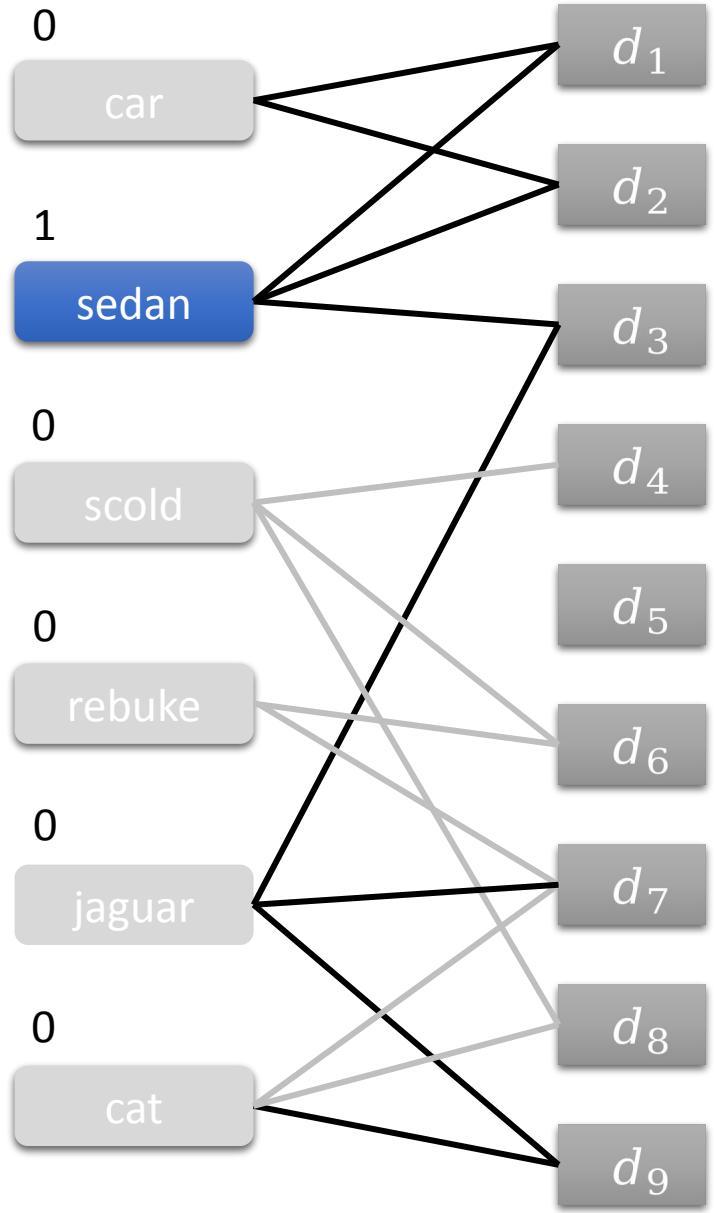
- Bipartite graph between words and docs
- Expressed via adjacency matrix $E \in \{0,1\}^{W \times D}$
- Two similar questions
  - If a random surfer starts from w_i, and resets to w_i w.p. α every step, what is the steady state probability of visiting w_j?
  - How many paths of length 2, 4, 6,… exist between words $w_i,\ w_j$?
- $EE^\top \in \mathbb{Z}_+^{W \times W}$; $(EE^\top)[i,j]$ gives the number of 2-hop paths between $w_i,\ w_j$
- $(EE^\top)^2[i,j]$ gives the number of 4-hop paths between $w_i,\ w_j$, and so on



Spreading activation

- $(EE^\top)^k[i, j]$ gives the number of $2k$-hop paths between w_i, w_j
- Start with a vector $\boldsymbol{u}^{(0)} \in \mathbb{R}^W$
  - Elements sampled from some distribution
  - Or the impulse at word w_0: set $\boldsymbol{u}^{(0)} = \delta_{w_0}$
- Repeat
  - $\boldsymbol{v}^{(k+1)} \leftarrow (EE^\top)\boldsymbol{u}^{(t)}$
  - $\boldsymbol{u}^{(k+1)} \leftarrow \boldsymbol{v}^{(k+1)} / \left\|\boldsymbol{v}^{(k+1)}\right\|_1$
- (If this converges) finds an eigenvector $\boldsymbol{u}$ such that $(EE^\top)\boldsymbol{u} = \lambda\boldsymbol{u}$

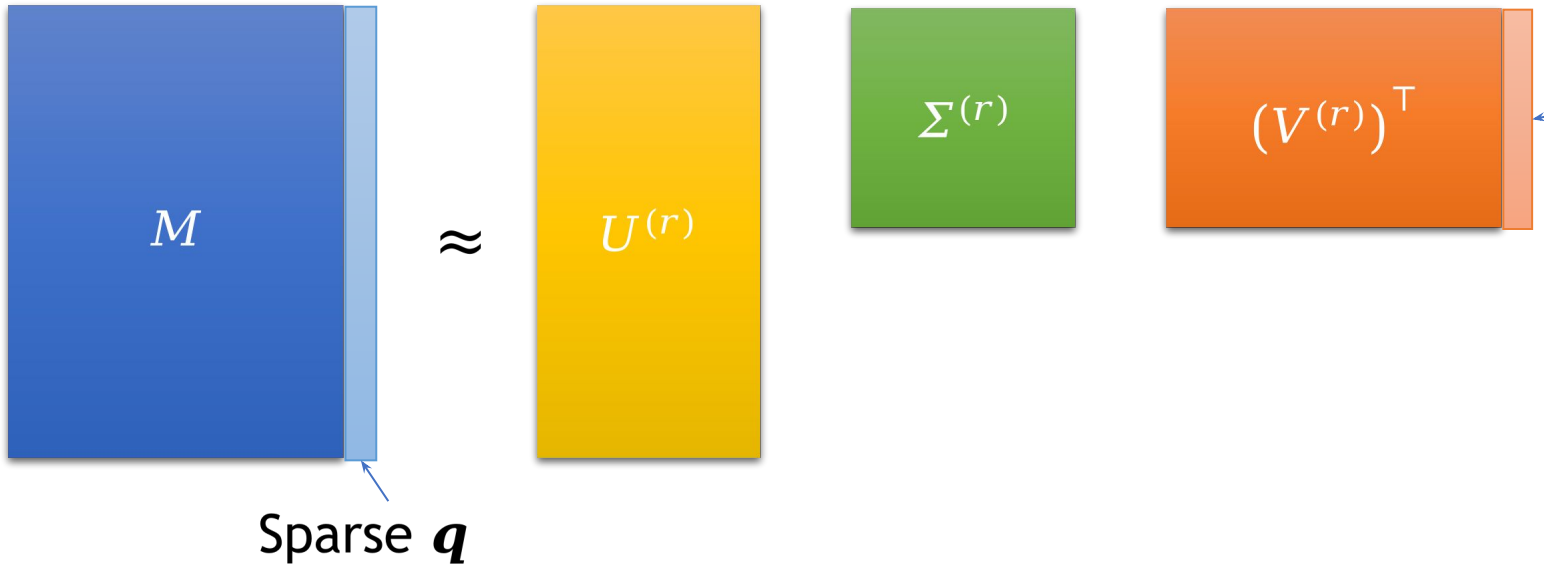
Depending on whether E is connected, $\boldsymbol{u}^{(\infty)}$ may or may not be sensitive to $\boldsymbol{u}^{(0)}$. More about this later.



Using SVD in search

- Sparse term-doc matrix M decomposed as $U \Sigma V^\top$
- Usually truncated to ≈ 300 dimensions
- Each row of U thus gives a 300-dim representation of a word
- Each row of V gives a 300-dim representation of a doc
- These “embeddings” are no longer sparse
- “Latent semantic indexing” or LSI, 1988
- Query is like a new sparse doc vector

LSI in search, continued

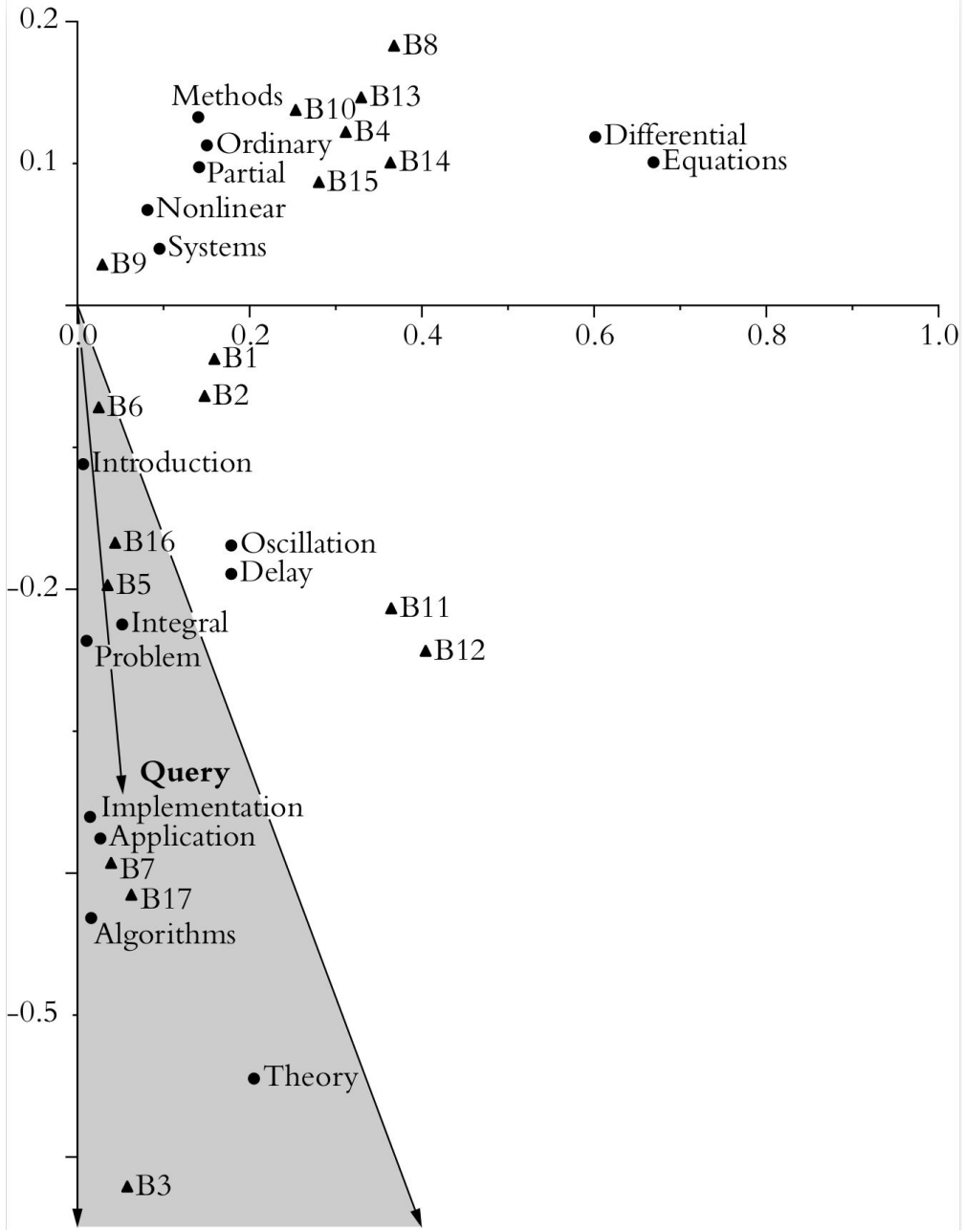


$$\boldsymbol{q} = U\,\Sigma\,\hat{\boldsymbol{q}}$$
$$U^\top \boldsymbol{q} = U^\top U\,\Sigma\,\hat{\boldsymbol{q}} = \Sigma\,\hat{\boldsymbol{q}}$$
$$\Sigma^{-1} U^\top \boldsymbol{q} = \hat{\boldsymbol{q}}$$

- Pad M with sparse query vector $\boldsymbol{q}$
- Assume perturbation small enough that U, Σ, V do not change much
- Project sparse query $\boldsymbol{q}$ to dense $\hat{\boldsymbol{q}}$ in doc space
- Find nearby doc vectors as $\max_j \hat{\boldsymbol{q}} \cdot \boldsymbol{v}_j$ (cannot use inverted index)

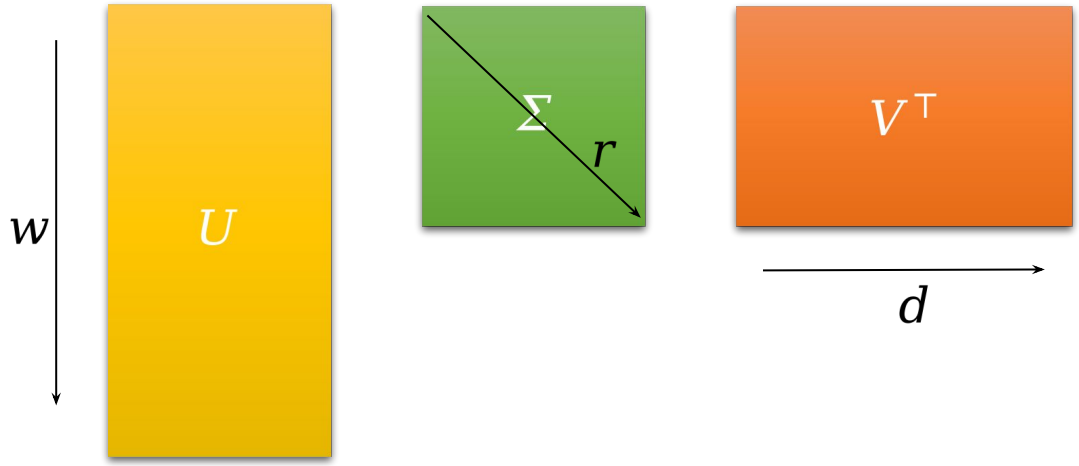
LSI example

Label	Titles
B1	A Course on Integral Equations
B2	Attractors for Semigroups and Evolution Equations
B3	Automatic Differentiation of Algorithms: Theory, Implementation, and Application
B4	Geometrical Aspects of Partial Differential Equations
B5	Ideals, Varieties, and Algorithms –An Introduction to Computational Algebraic Geometry and Commutative Algebra
B6	Introduction to Hamiltonian Dynamical Systems and the *N*-Body Problem
B7	Knapsack Problems: Algorithms and Computer Implementations
B8	Methods of Solving Singular Systems of Ordinary Differential Equations
B9	Nonlinear Systems
B10	Ordinary Differential Equations
B11	Oscillation Theory for Neutral Differential Equations with Delay
B12	Oscillation Theory of Delay Differential Equations
B13	Pseudodifferential Operators and Nonlinear Partial Differential Equations
B14	Sync Methods for Quadrature and Differential Equations
B15	Stability of Stochastic Differential Equations with Respect to Semi-Martingales
B16	The Boundary Integral Approach to Static and Dynamic Contact Problems
B17	The Double Mellin-Barnes Type Integrals and Their Applications to Convolution Theory



Non-negative matrix factorization

- Unconstrained SVD $M = U\Sigma V^\top$
- Rank-constrained approx $M \approx U^{(r)}\Sigma^{(r)}\left(V^{(r)}\right)^\top$
- A different way to constrain:
  - All elements of U, Σ, V must be non-negative
  - Each column of U and V sum to 1
  - Diagonal elements of Σ sum to 1
- Multinomial distribution over the joint (w, d) space
  - Both regarded as random variables



- For each r, $\sum_w U[w, r] = 1$
- Think $\sum_w p(w|r) = 1$
- For each r, $\sum_d V[d, r] = 1$
- Think $\sum_w p(d|r) = 1$
- $\sum_r \Sigma[r] = 1 \longrightarrow \sum_r p(r) = 1$

Dyadic or aspect model or PLSI

- Joint distribution $p(w, d)$ not product of marginals
  - I.e., w, d are strongly dependent
- Bottleneck through topics r
- $p(w, d) = \sum_r p(r)p(w, d|r)$ is always true
- We approximate $p(w, d|r) \approx p(w|r)p(d|r)$
  - "Word w and doc d are conditionally independent given topic r"
- Overall $p(w, d) \approx \sum_r p(r)\, p(w|r)p(d|r)$
  - (Looks structurally similar to SVD)

Fitting a dyadic model

- Model parameterized as $p(r) = \Sigma[r],\ p(w|r) = U[w, r],\ p(d|r) = V[d, r]$
- Total $WR + R + DR$ real parameters
- Event probability $p(w, d) \approx \sum_r p(r)\, p(w|r) p(d|r)$
- Let the term-doc matrix contain (w, d) counts $n[w, d]$
- Objective is $\max_{U,\Sigma,V} \sum_{w,d} n[w, d] \log p(w, d)$
- Subject to U, Σ, V satisfying probability-like constraints

EM optimization

- Given current ("M variables") $p(r), p(w|r), p(d|r)$, alternate E and M steps
- Estimate posterior (E)

$$p(r|w,d) = \frac{p(r)p(w,d|r)}{\sum_{r'} p(r')p(w,d|r')} = \frac{p(r)p(w|r)p(d|r)}{\sum_{r'} p(r')p(w|r')p(d|r')}$$

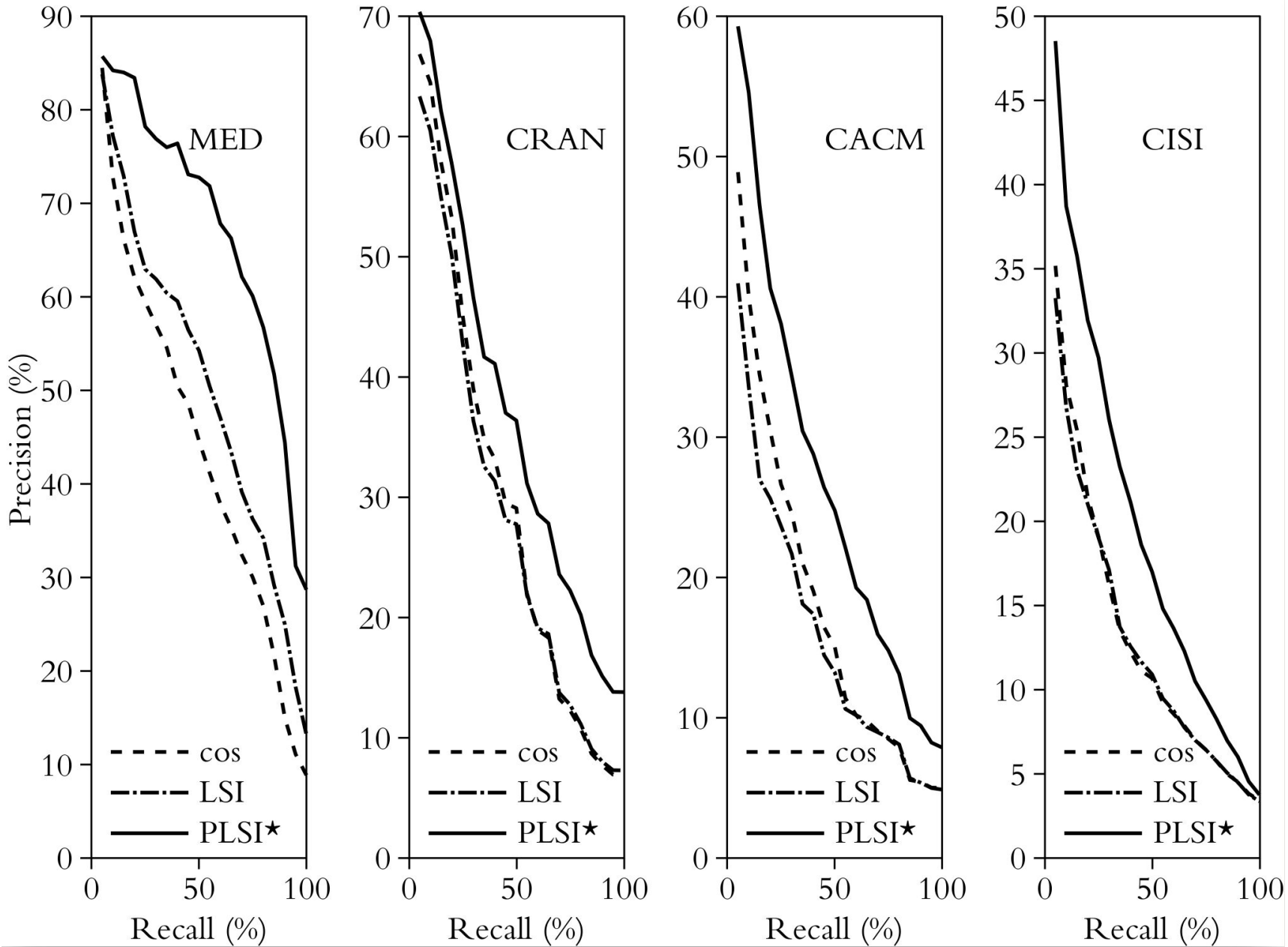
- Compute next "M variable" estimates

$$p(r) = \frac{\sum_{w,d} n(w,d)p(r|w,d)}{\sum_{r'}\sum_{w,d} n(w,d)p(r'|w,d)}$$

$$p(w|r) = \frac{\sum_{d} n(w,d)p(r|w,d)}{\sum_{w'}\sum_{d} n(w',d)p(r|w',d)}$$

$$p(d|r) = \frac{\sum_{w} n(w,d)p(r|w,d)}{\sum_{d'}\sum_{w} n(w,d')p(r|w,d')}$$

TFIDF vs LSI vs PLSI



Summary

- Lexical gap has been known for ages
- Pre-deep learning approaches
  - Pseudo relevance feedback
  - Word clustering
- Approaching word representation
  - Distributional vectors
  - LSI, PLSI
- Next: neural contextualized word rep.