COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Fall 2021.

Lecture 18

- Problem Set 3 is due Monday at 11:59pm.
- No quiz due Monday.
- Problem Set 2 grades are posted.

SUMMARY

Last Class

$X = U\Sigma V^T$

- The Singular Value Decomposition (SVD) and its connection to eigendecomposition of X^TX and XX^T, and low-rank approximation.
- Low-rank matrix completion (predicting missing measurements using low-rank structure).

This Class: More applications of Low-Rank Approximation

- Entity embeddings.
- Low-rank approximation for non-linear dimensionality reduction.
- Eigendecomposition to partition graphs into clusters.

Dimensionality reduction embeds d-dimensional vectors into k dimensions. But what about when you want to embed objects other than vectors?

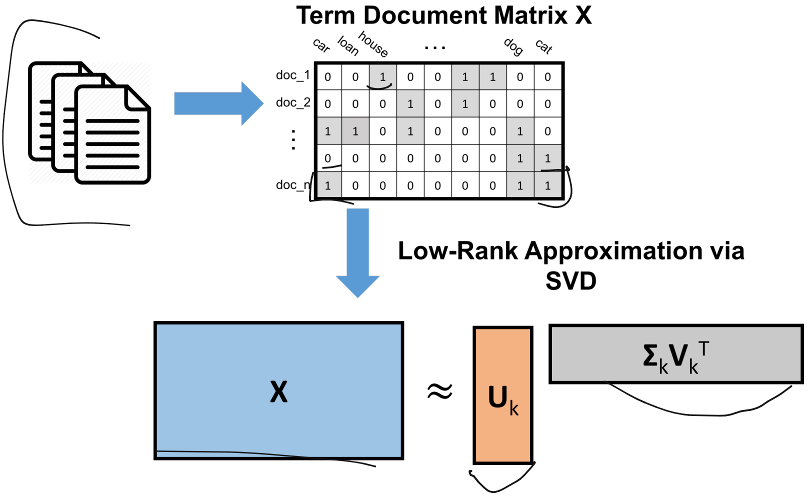
- Documents (for topic-based search and classification)
- Words (to identify synonyms, translations, etc.)
- Nodes in a social network

Dimensionality reduction embeds d-dimensional vectors into k dimensions. But what about when you want to embed objects other than vectors?

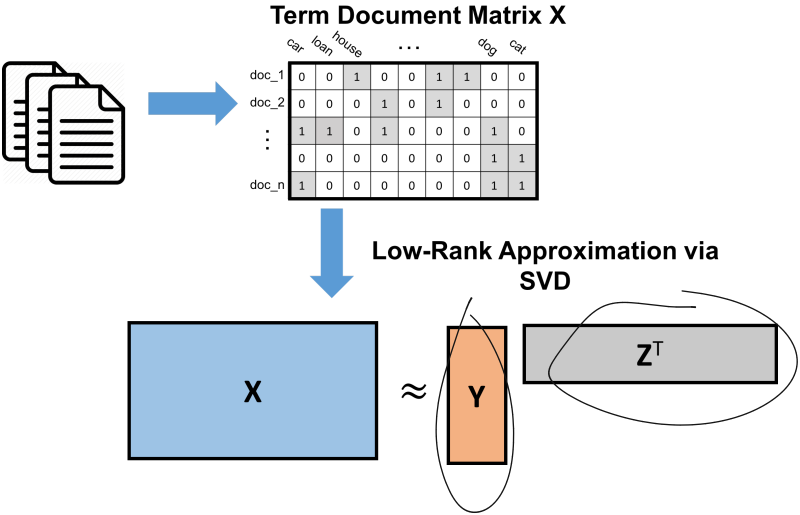
- Documents (for topic-based search and classification)
- Words (to identify synonyms, translations, etc.)
- Nodes in a social network

Classic Approach: Convert each item into a high-dimensional feature vector and then apply low-rank approximation.

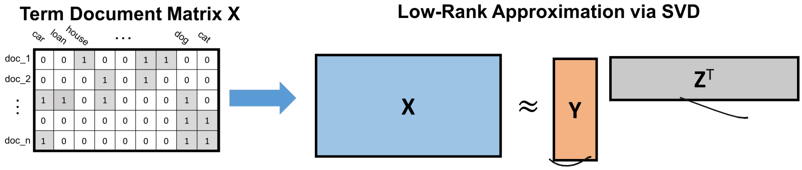
EXAMPLE: LATENT SEMANTIC ANALYSIS



EXAMPLE: LATENT SEMANTIC ANALYSIS



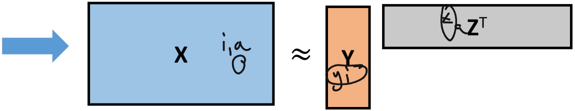
EXAMPLE: LATENT SEMANTIC ANALYSIS



Term Document Matrix X

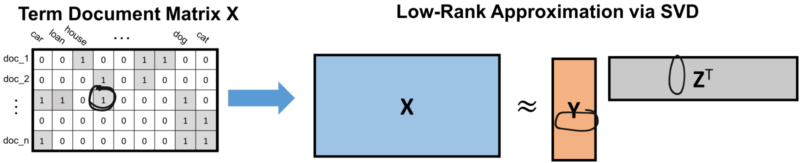
	car	loan	house	…				dog	cat
doc_1	0	0	1	0	0	1	1	0	0
doc_2	0	0	0	1	0	1	0	0	0
⋮	1	1	0	1	0	0	0	1	0
	0	0	0	0	0	0	0	1	1
doc_n	1	0	0	0	0	0	0	1	1

Low-Rank Approximation via SVD



- If the error $\|X - YZ^T\|_F$ is small, then on average,

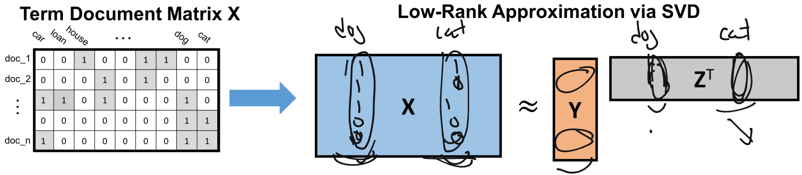
$$X_{i,a} \approx (YZ^T)_{i,a} = \langle \vec{y}_i, \vec{z}_a \rangle.$$



- If the error $\|X - YZ^T\|_F$ is small, then on average,

$$X_{i,a} \approx (YZ^T)_{i,a} = \langle \vec{y}_i, \vec{z}_a \rangle.$$

- I.e., $\langle \vec{y}_i, \vec{z}_a \rangle \approx 1$ when doc_i contains $word_a$.



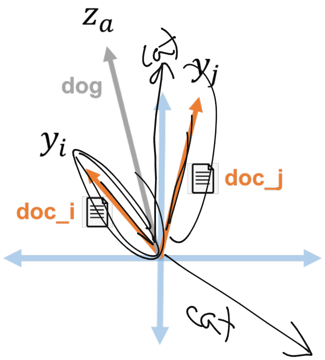
- If the error $\|X - YZ^T\|_F$ is small, then on average,

$$X_{i,a} \approx (YZ^T)_{i,a} = \langle \vec{y}_i, \vec{z}_a \rangle.$$

- I.e., $\langle \vec{y}_i, \vec{z}_a \rangle \approx 1$ when doc_i contains $word_a$.
- If doc_i and doc_j both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle \approx 1$.

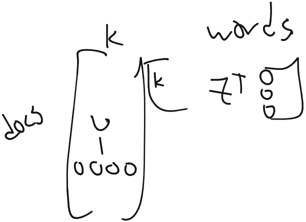
EXAMPLE: LATENT SEMANTIC ANALYSIS

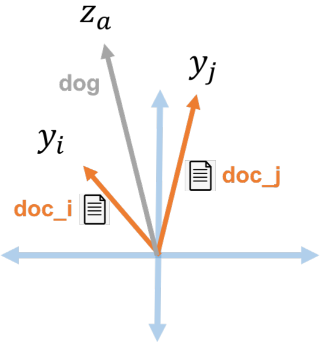
If doc_i and doc_j both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle \approx 1$



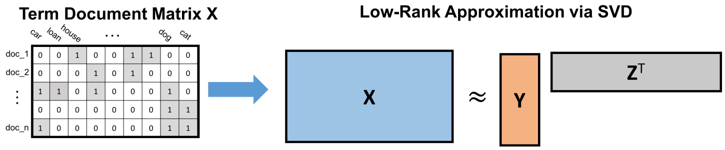
EXAMPLE: LATENT SEMANTIC ANALYSIS

If doc_i and doc_j both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle \approx 1$



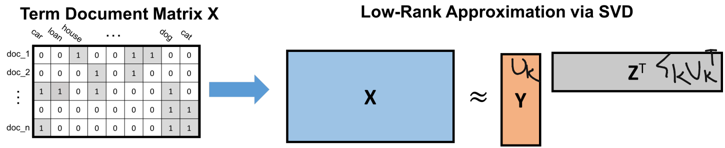


Another View: Each column of **Y** represents a 'topic'. $\vec{y}_i(j)$ indicates how much doc_i belongs to topic j. $\vec{z}_a(j)$ indicates how much $word_a$ associates with that topic.



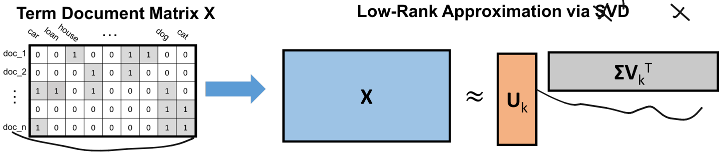
- Just like with documents, $\vec{z}_a$ and $\vec{z}_b$ will tend to have high dot product if $word_a$ and $word_b$ appear in many of the same documents.

EXAMPLE: LATENT SEMANTIC ANALYSIS



- Just like with documents, $\vec{z}_a$ and $\vec{z}_b$ will tend to have high dot product if $word_a$ and $word_b$ appear in many of the same documents.
- In an SVD decomposition we set $\mathbf{Z}^T = \mathbf{\Sigma}_k \mathbf{V}_K^T$.
- The columns of $\mathbf{V}_k$ are equivalently: the top k eigenvectors of $\mathbf{X}^T\mathbf{X}$.

$X = U\Sigma V^T \quad X^T X = V\Sigma U^T U \Sigma V^T = V\Sigma^2 V^T$



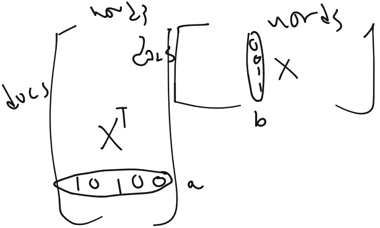
- Just like with documents, $\vec{z}_a$ and $\vec{z}_b$ will tend to have high dot product if $word_a$ and $word_b$ appear in many of the same documents.
- In an SVD decomposition we set $\mathbf{Z}^T = \mathbf{\Sigma}_k \mathbf{V}_K^T$.
- The columns of $\mathbf{V}_k$ are equivalently: the top k eigenvectors of $\mathbf{X}^T\mathbf{X}$.
- **Claim:** $\mathbf{Z}\mathbf{Z}^{\mathbf{T}}$ is the best rank-k approximation of $\mathbf{X}^T\mathbf{X}$. I.e., $\arg\min_{\text{rank}-k \ \mathbf{B}} \|\mathbf{X}^T\mathbf{X} - \mathbf{B}\|_F$

$[X^T][X] \quad V_k \Sigma_k^2 V_k^T$

$$ZZ^T \approx X^TX$$

LSA gives a way of embedding words into k-dimensional space.

- Embedding is via low-rank approximation of X^TX: where $(X^TX)_{a,b}$ is the number of documents that both $word_a$ and $word_b$ appear in.



LSA gives a way of embedding words into k-dimensional space.

- Embedding is via low-rank approximation of X^TX: where $(X^TX)_{a,b}$ is the number of documents that both $word_a$ and $word_b$ appear in.
- Think about X^TX as a similarity matrix (gram matrix, kernel matrix) with entry (a, b) being the similarity between $word_a$ and $word_b$.

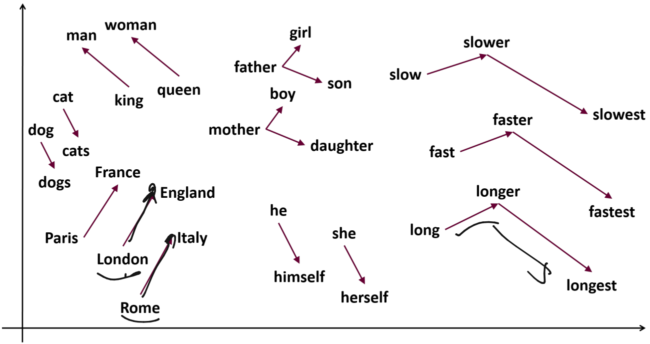
LSA gives a way of embedding words into k-dimensional space.

- Embedding is via low-rank approximation of $\mathbf{X}^T\mathbf{X}$: where $(\mathbf{X}^T\mathbf{X})_{a,b}$ is the number of documents that both $word_a$ and $word_b$ appear in.
- Think about $\mathbf{X}^T\mathbf{X}$ as a similarity matrix (gram matrix, kernel matrix) with entry (a, b) being the similarity between $word_a$ and $word_b$.
- Many ways to measure similarity: number of sentences both occur in, number of times both appear in the same window of w words, in similar positions of documents in different languages, etc.

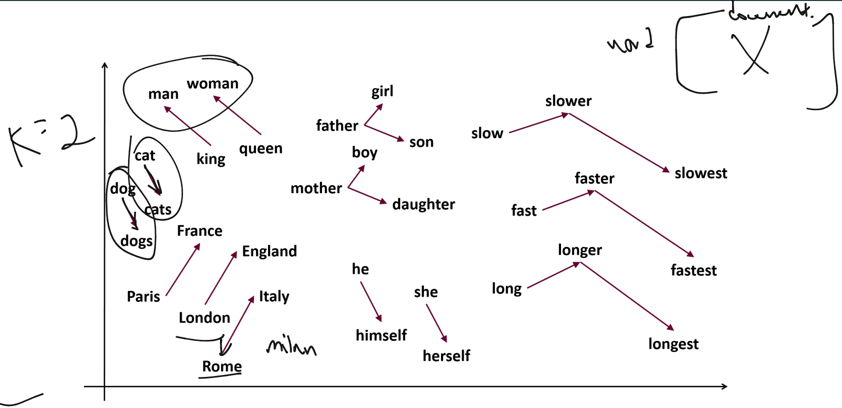
EXAMPLE: WORD EMBEDDING

LSA gives a way of embedding words into k-dimensional space.

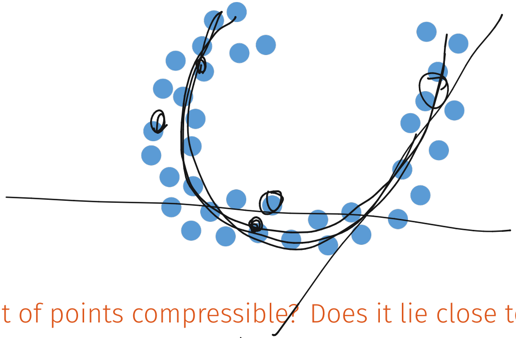
- Embedding is via low-rank approximation of $\mathbf{X}^T\mathbf{X}$: where $(\mathbf{X}^T\mathbf{X})_{a,b}$ is the number of documents that both $word_a$ and $word_b$ appear in.
- Think about $\mathbf{X}^T\mathbf{X}$ as a similarity matrix (gram matrix, kernel matrix) with entry (a, b) being the similarity between $word_a$ and $word_b$.
- Many ways to measure similarity: number of sentences both occur in, number of times both appear in the same window of w words, in similar positions of documents in different languages, etc.
- Replacing $\mathbf{X}^T\mathbf{X}$ with these different metrics (sometimes appropriately transformed) leads to popular word embedding algorithms: word2vec, GloVe, fastText, etc.


woman
man
girl
slower
father
son
slow
cat
king
queen
boy
slowest
faster
dog
mother
daughter
fast
cats
France
dogs
England
longer
fastest
he
Paris
Italy
she
long
London
himself
longest
herself
Rome

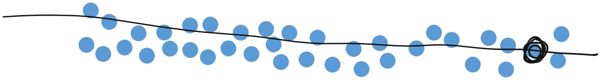
EXAMPLE: WORD EMBEDDING



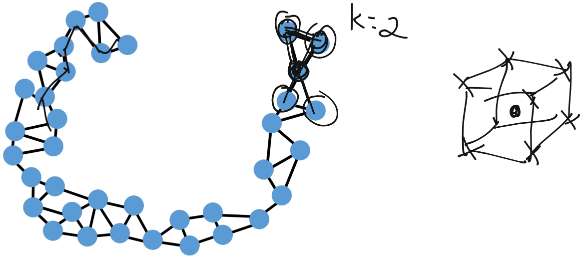
Note: word2vec is typically described as a neural-network method, but it is really just low-rank approximation of a specific similarity matrix. *Neural word embedding as implicit matrix factorization*, Levy and Goldberg.



Is this set of points compressible? Does it lie close to a low-dimensional subspace? (A 1-dimensional subspace of $\mathbb{R}^d$.)



Is this set of points compressible? Does it lie close to a low-dimensional subspace? (A 1-dimensional subspace of $\mathbb{R}^d$.)



Is this set of points compressible? Does it lie close to a low-dimensional subspace? (A 1-dimensional subspace of $\mathbb{R}^d$.)

A common way of automatically identifying this non-linear structure is to connect data points in a graph. E.g., a k-nearest neighbor graph.

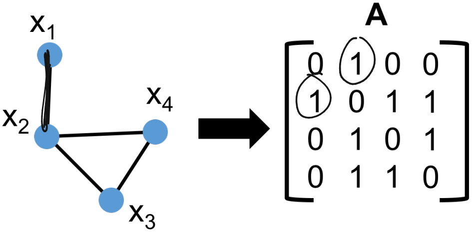
- Connect items to similar items, possibly with higher weight edges when they are more similar.

Once we have connected n data points $x_1, \ldots, x_n$ into a graph, we can represent that graph by its (weighted) adjacency matrix.

$$\mathbf{A} \in \mathbb{R}^{n \times n} \text{ with } \mathbf{A}_{i,j} = \text{ edge weight between nodes } i \text{ and } j$$

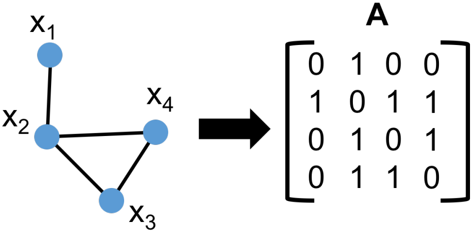
Once we have connected n data points $x_1, \ldots, x_n$ into a graph, we can represent that graph by its (weighted) adjacency matrix.

$$\mathbf{A} \in \mathbb{R}^{n \times n} \text{ with } \mathbf{A}_{i,j} = \text{ edge weight between nodes } i \text{ and } j$$



Once we have connected n data points $x_1, \ldots, x_n$ into a graph, we can represent that graph by its (weighted) adjacency matrix.

$$\mathbf{A} \in \mathbb{R}^{n\times n} \text{ with } \mathbf{A}_{i,j} = \text{ edge weight between nodes } i \text{ and } j$$



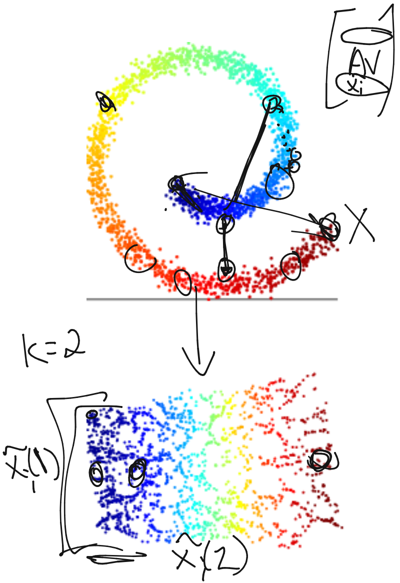
In LSA example, when $\mathbf{X}$ is the term-document matrix, $\mathbf{X}^T\mathbf{X}$ is like an adjacency matrix, where $word_a$ and $word_b$ are connected if they appear in at least 1 document together (edge weight is # documents they appear in together).

How do we compute an optimal low-rank approximation of $\mathbf{A}$?

- Project onto the top *k* eigenvectors of $\mathbf{A}^T\mathbf{A} = \mathbf{A}^2$. These are just the eigenvectors of $\mathbf{A}$.

How do we compute an optimal low-rank approximation of $\mathbf{A}$?

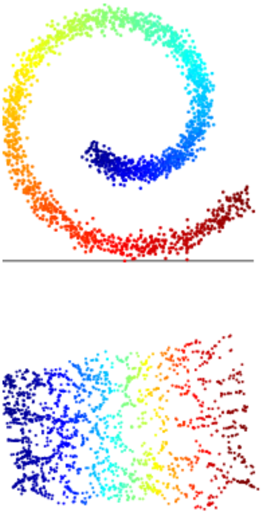
- Project onto the top k eigenvectors of $\mathbf{A}^T\mathbf{A} = \mathbf{A}^2$. These are just the eigenvectors of $\mathbf{A}$.
- $\mathbf{A} \approx \mathbf{A}\mathbf{V}\mathbf{V}^T$. The rows of $\mathbf{A}\mathbf{V}$ can be thought of as 'embeddings' for the vertices.
- Similar vertices (close with regards to graph proximity) should have similar embeddings.



Step 1: Produce a nearest neighbor graph based on your input data in $\mathbb{R}^d$.

Step 2: Apply low-rank approximation to the graph adjacency matrix to produce embeddings in $\mathbb{R}^k$.

Step 3: Work with the data in the embedded space. Where distances represent distances in your original 'non-linear space.'



What other methods do you know for embedding or representing data points with non-linear structure?