Last Class: Low-Rank Approximation, Eigendecomposition, PCA

• For any symmetric square matrix $A$, we can write $A = V \Lambda V^T$ where columns of $V$ are orthonormal eigenvectors.

• Can approximate data lying close to in a $k$-dimensional subspace by projecting data points into that space.

• Can find the best $k$-dimensional subspace via eigendecomposition applied to $X^T X$ (PCA).

• Measuring error in terms of the eigenvalue spectrum.

This Class: SVD and Applications

• SVD and connection to eigenvalue value decomposition.

• Applications of low-rank approximation beyond compression.
The Singular Value Decomposition (SVD) generalizes the eigendecomposition to asymmetric (even rectangular) matrices.
The Singular Value Decomposition (SVD) generalizes the eigendecomposition to asymmetric (even rectangular) matrices. Any matrix $X \in \mathbb{R}^{n \times d}$ with $\text{rank}(X) = r$ can be written as $X = U \Sigma V^T$.

- $U$ has orthonormal columns $\vec{u}_1, \ldots, \vec{u}_r \in \mathbb{R}^n$ (left singular vectors).
- $V$ has orthonormal columns $\vec{v}_1, \ldots, \vec{v}_r \in \mathbb{R}^d$ (right singular vectors).
- $\Sigma$ is diagonal with elements $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ (singular values).
The Singular Value Decomposition (SVD) generalizes the eigendecomposition to asymmetric (even rectangular) matrices. Any matrix $X \in \mathbb{R}^{n \times d}$ with rank$(X) = r$ can be written as $X = U\Sigma V^T$.

- $U$ has orthonormal columns $\vec{u}_1, \ldots, \vec{u}_r \in \mathbb{R}^n$ (left singular vectors).
- $V$ has orthonormal columns $\vec{v}_1, \ldots, \vec{v}_r \in \mathbb{R}^d$ (right singular vectors).
- $\Sigma$ is diagonal with elements $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ (singular values).
The Singular Value Decomposition (SVD) generalizes the eigendecomposition to asymmetric (even rectangular) matrices. Any matrix $X \in \mathbb{R}^{n \times d}$ with $\text{rank}(X) = r$ can be written as $X = U \Sigma V^T$.

- $U$ has orthonormal columns $\vec{u}_1, \ldots, \vec{u}_r \in \mathbb{R}^n$ (left singular vectors).
- $V$ has orthonormal columns $\vec{v}_1, \ldots, \vec{v}_r \in \mathbb{R}^d$ (right singular vectors).
- $\Sigma$ is diagonal with elements $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ (singular values).

The ‘swiss army knife’ of modern linear algebra.
Writing $X \in \mathbb{R}^{n \times d}$ in its singular value decomposition $X = U\Sigma V^T$:

$$X^T X =$$

$X \in \mathbb{R}^{n \times d}$: data matrix, $U \in \mathbb{R}^{n \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{u}_1, \vec{u}_2, \ldots$ (left singular vectors), $V \in \mathbb{R}^{d \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{v}_1, \vec{v}_2, \ldots$ (right singular vectors), $\Sigma \in \mathbb{R}^{\text{rank}(X) \times \text{rank}(X)}$: positive diagonal matrix containing singular values of $X$. 
Writing \( X \in \mathbb{R}^{n \times d} \) in its singular value decomposition \( X = U \Sigma V^T \):

\[
X^T X = V \Sigma U^T U \Sigma V^T
\]

\( X \in \mathbb{R}^{n \times d} \): data matrix, \( U \in \mathbb{R}^{n \times \text{rank}(X)} \): matrix with orthonormal columns \( \vec{u}_1, \vec{u}_2, \ldots \) (left singular vectors), \( V \in \mathbb{R}^{d \times \text{rank}(X)} \): matrix with orthonormal columns \( \vec{v}_1, \vec{v}_2, \ldots \) (right singular vectors), \( \Sigma \in \mathbb{R}^{\text{rank}(X) \times \text{rank}(X)} \): positive diagonal matrix containing singular values of \( X \).
Writing $\mathbf{X} \in \mathbb{R}^{n \times d}$ in its singular value decomposition $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$:

\[
\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T
\]
Writing $X \in \mathbb{R}^{n \times d}$ in its singular value decomposition $X = U \Sigma V^T$:

$$X^T X = V \Sigma U^T U \Sigma V^T = V \Sigma^2 V^T$$ (the eigendecomposition)

**Explanation:**

- $X \in \mathbb{R}^{n \times d}$: data matrix,
- $U \in \mathbb{R}^{n \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{u}_1, \vec{u}_2, \ldots$ (left singular vectors),
- $V \in \mathbb{R}^{d \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{v}_1, \vec{v}_2, \ldots$ (right singular vectors),
- $\Sigma \in \mathbb{R}^{\text{rank}(X) \times \text{rank}(X)}$: positive diagonal matrix containing singular values of $X$. 
Writing \( \mathbf{X} \in \mathbb{R}^{n \times d} \) in its singular value decomposition \( \mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T \):

\[
\mathbf{X}^T \mathbf{X} = \mathbf{V}\Sigma\mathbf{U}^T \mathbf{U}\Sigma\mathbf{V}^T = \mathbf{V}\Sigma^2\mathbf{V}^T \quad \text{(the eigendecomposition)}
\]

Similarly: \( \mathbf{X}\mathbf{X}^T = \mathbf{U}\Sigma\mathbf{V}^T \mathbf{V}\Sigma\mathbf{U}^T = \mathbf{U}\Sigma^2\mathbf{U}^T \).
Writing $X \in \mathbb{R}^{n \times d}$ in its singular value decomposition $X = U\Sigma V^T$:

$$X^T X = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T \text{ (the eigendecomposition)}$$

Similarly: $XX^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T$.

The right and left singular vectors are the eigenvectors of the covariance matrix $X^T X$ and the gram matrix $XX^T$ respectively.

**X ∈ \mathbb{R}^{n \times d}**: data matrix, $U \in \mathbb{R}^{n \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{u}_1, \vec{u}_2, \ldots$ (left singular vectors), $V \in \mathbb{R}^{d \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{v}_1, \vec{v}_2, \ldots$ (right singular vectors), $\Sigma \in \mathbb{R}^{\text{rank}(X) \times \text{rank}(X)}$: positive diagonal matrix containing singular values of $X$. 
Writing $X \in \mathbb{R}^{n \times d}$ in its singular value decomposition $X = U\Sigma V^T$: 

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

(the eigendecomposition)

Similarly: $XX^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T$.

The right and left singular vectors are the eigenvectors of the covariance matrix $X^TX$ and the gram matrix $XX^T$ respectively.

So, letting $V_k \in \mathbb{R}^{d \times k}$ have columns equal to $\vec{v}_1, \ldots, \vec{v}_k$, we know that $XV_kV_k^T$ is the best rank-$k$ approximation to $X$ (given by PCA).

---

$X \in \mathbb{R}^{n \times d}$: data matrix, $U \in \mathbb{R}^{n \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{u}_1, \vec{u}_2, \ldots$ (left singular vectors), $V \in \mathbb{R}^{d \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{v}_1, \vec{v}_2, \ldots$ (right singular vectors), $\Sigma \in \mathbb{R}^{\text{rank}(X) \times \text{rank}(X)}$: positive diagonal matrix containing singular values of $X$. 
Writing $X \in \mathbb{R}^{n \times d}$ in its singular value decomposition $X = U \Sigma V^T$:

$$X^T X = V \Sigma U^T \Sigma V^T = V \Sigma^2 V^T \quad \text{(the eigendecomposition)}$$

Similarly: $XX^T = U \Sigma V^T \Sigma U^T = U \Sigma^2 U^T$.

The right and left singular vectors are the eigenvectors of the covariance matrix $X^T X$ and the gram matrix $XX^T$ respectively.

So, letting $V_k \in \mathbb{R}^{d \times k}$ have columns equal to $\vec{v}_1, \ldots, \vec{v}_k$, we know that $XV_k V_k^T$ is the best rank-$k$ approximation to $X$ (given by PCA).

What about $U_k U_k^T X$ where $U_k \in \mathbb{R}^{n \times k}$ has columns equal to $\vec{u}_1, \ldots, \vec{u}_k$?

---

$X \in \mathbb{R}^{n \times d}$: data matrix, $U \in \mathbb{R}^{n \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{u}_1, \vec{u}_2, \ldots$ (left singular vectors), $V \in \mathbb{R}^{d \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{v}_1, \vec{v}_2, \ldots$ (right singular vectors), $\Sigma \in \mathbb{R}^{\text{rank}(X) \times \text{rank}(X)}$: positive diagonal matrix containing singular values of $X$. 

---


Writing $X \in \mathbb{R}^{n \times d}$ in its singular value decomposition $X = U \Sigma V^T$:

$$X^T X = V \Sigma U^T U \Sigma V^T = V \Sigma^2 V^T$$ (the eigendecomposition)

Similarly: $XX^T = U \Sigma V^T V \Sigma U^T = U \Sigma^2 U^T$.

The right and left singular vectors are the eigenvectors of the covariance matrix $X^T X$ and the gram matrix $XX^T$ respectively.

So, letting $V_k \in \mathbb{R}^{d \times k}$ have columns equal to $\vec{v}_1, \ldots, \vec{v}_k$, we know that $X V_k V_k^T$ is the best rank-$k$ approximation to $X$ (given by PCA).

What about $U_k U_k^T X$ where $U_k \in \mathbb{R}^{n \times k}$ has columns equal to $\vec{u}_1, \ldots, \vec{u}_k$?

**Exercise:** $U_k U_k^T X = X V_k V_k^T = U_k \Sigma_k V_k^T$

---

$X \in \mathbb{R}^{n \times d}$: data matrix, $U \in \mathbb{R}^{n \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{u}_1, \vec{u}_2, \ldots$ (left singular vectors), $V \in \mathbb{R}^{d \times \text{rank}(X)}$: matrix with orthonormal columns $\vec{v}_1, \vec{v}_2, \ldots$ (right singular vectors), $\Sigma \in \mathbb{R}^{\text{rank}(X) \times \text{rank}(X)}$: positive diagonal matrix containing singular values of $X$. 
The best low-rank approximation to $X$, i.e.,

$$X_k = \arg \min_{\text{rank}-k \ B \in \mathbb{R}^{n \times d}} \| X - B \|_F$$

is given by $X_k = XV_k V_k^T = U_k U_k^T X = U_k \Sigma_k V_k^T$.
The best low-rank approximation to $X$, i.e.,

$$X_k = \arg \min_{\text{rank}-k \; B \in \mathbb{R}^{n \times d}} \| X - B \|_F$$

is given by $X_k = XV_kV_k^T = U_kU_k^TX = U_k\Sigma_kV_k^T$

Corresponds to projecting the rows (data points) onto the span of $V_k$ or the columns (features) onto the span of $U_k$
The best low-rank approximation to $X$, i.e.,

$$X_k = \arg\min_{\text{rank}-k\ B \in \mathbb{R}^{n \times d}} \|X - B\|_F$$

is given by $X_k = XV_kV_k^T = U_kU_k^TX = U_k\Sigma_kV_k^T$

Corresponds to projecting the rows (data points) onto the span of $V_k$ or the columns (features) onto the span of $U_k$
The best low-rank approximation to $X$, i.e.,

$$X_k = \arg \min_{\text{rank-k } B \in \mathbb{R}^{n \times d}} \|X - B\|_F$$

is given by $X_k = XV_kV_k^T = U_kU_k^TX = U_k\Sigma_kV_k^T$

Corresponds to projecting the rows (data points) onto the span of $V_k$ or the columns (features) onto the span of $U_k$
The best low-rank approximation to $\mathbf{X}$, i.e.,

$$
\mathbf{X}_k = \arg \min_{\text{rank}-k} \| \mathbf{X} - \mathbf{B} \|_F
$$

is given by $\mathbf{X}_k = \mathbf{X}\mathbf{V}_k\mathbf{V}_k^T = \mathbf{U}_k\mathbf{U}_k^T\mathbf{X} = \mathbf{U}_k\Sigma_k\mathbf{V}_k^T$

Corresponds to projecting the rows (data points) onto the span of $\mathbf{V}_k$ or the columns (features) onto the span of $\mathbf{U}_k$
BASIC IDEA TO PROVE EXISTENCE OF SVD

- Let \( \vec{v}_1, \vec{v}_2, \ldots, \in \mathbb{R}^d \) be orthonormal eigenvectors of \( X^T X \).
Basic idea to prove existence of SVD

- Let \( \vec{v}_1, \vec{v}_2, \ldots, \in \mathbb{R}^d \) be orthonormal eigenvectors of \( X^TX \).
- Let \( \sigma_i = \|X\vec{v}_i\|_2 \) and define unit vector \( \vec{u}_i = \frac{X\vec{v}_i}{\sigma_i} \).
• Let $\vec{v}_1, \vec{v}_2, \ldots \in \mathbb{R}^d$ be orthonormal eigenvectors of $\mathbf{X}^T \mathbf{X}$.

• Let $\sigma_i = \|\mathbf{X}\vec{v}_i\|_2$ and define unit vector $\vec{u}_i = \frac{\mathbf{X}\vec{v}_i}{\sigma_i}$.

• **Exercise:** Show $\vec{u}_1, \vec{u}_2, \ldots$ are orthonormal.
• Let \( \vec{v}_1, \vec{v}_2, \ldots, \in \mathbb{R}^d \) be orthonormal eigenvectors of \( X^T X \).
• Let \( \sigma_i = \|X\vec{v}_i\|_2 \) and define unit vector \( \vec{u}_i = \frac{X\vec{v}_i}{\sigma_i} \).
• **Exercise:** Show \( \vec{u}_1, \vec{u}_2, \ldots \) are orthonormal.
• This establishes that \( XV = U\Sigma \) and that \( V \) and \( U \) have the required properties.
BASIC IDEA TO PROVE EXISTENCE OF SVD

• Let \( \vec{v}_1, \vec{v}_2, \ldots, \in \mathbb{R}^d \) be orthonormal eigenvectors of \( X^T X \).

• Let \( \sigma_i = \|X\vec{v}_i\|_2 \) and define unit vector \( \vec{u}_i = \frac{X\vec{v}_i}{\sigma_i} \).

• Exercise: Show \( \vec{u}_1, \vec{u}_2, \ldots \) are orthonormal.

• This establishes that \( XV = U\Sigma \) and that \( V \) and \( U \) have the required properties.

• To see rest of the details, see https://math.mit.edu/classes/18.095/2016IAP/lec2/SVD_Notes.pdf
**Rest of Class:** Examples of how low-rank approximation is applied in a variety of data science applications.
Rest of Class: Examples of how low-rank approximation is applied in a variety of data science applications.

- Used for many reasons other than dimensionality reduction/data compression.
Consider a matrix $X \in \mathbb{R}^{n \times d}$ which we cannot fully observe but believe is close to rank-$k$ (i.e., well approximated by a rank $k$ matrix).
Consider a matrix $X \in \mathbb{R}^{n \times d}$ which we cannot fully observe but believe is close to rank-$k$ (i.e., well approximated by a rank $k$ matrix). Classic example: the Netflix prize problem.
Consider a matrix $X \in \mathbb{R}^{n \times d}$ which we cannot fully observe but believe is close to rank-$k$ (i.e., well approximated by a rank $k$ matrix). Classic example: the Netflix prize problem.

Solve: $Y = \arg \min_{\text{rank} - k B} \sum_{\text{observed } (j,k)} \left[ X_{j,k} - B_{j,k} \right]^2$
Consider a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ which we cannot fully observe but believe is close to rank-$k$ (i.e., well approximated by a rank $k$ matrix). Classic example: the Netflix prize problem.

**Solve:** $\mathbf{Y} = \arg\min_{\mathbf{B} \text{ rank } -k} \sum_{(j,k) \text{ observed}} [\mathbf{X}_{j,k} - \mathbf{B}_{j,k}]^2$

Under certain assumptions, can show that $\mathbf{Y}$ well approximates $\mathbf{X}$ on both the observed and (most importantly) unobserved entries.
Dimensionality reduction embeds $d$-dimensional vectors into $k \ll d$ dimensions. But what about when you want to embed objects other than vectors?
Dimensionality reduction embeds $d$-dimensional vectors into $k \ll d$ 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 $k \ll d$ 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

**Usual Approach:** Convert each item into a high-dimensional feature vector and then apply low-rank approximation.
Example: Latent Semantic Analysis

Term Document Matrix $X$

Low-Rank Approximation via SVD

$X \approx U_k \Sigma_k V_k^T$
EXAMPLE: LATENT SEMANTIC ANALYSIS

**Term Document Matrix X**

<table>
<thead>
<tr>
<th></th>
<th>car</th>
<th>loan</th>
<th>house</th>
<th>...</th>
<th>dog</th>
<th>cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc_1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>doc_2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>doc_n</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Low-Rank Approximation via SVD

\[ X \approx Y Z^T \]
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 $a$.
EXAMPLE: LATENT SEMANTIC ANALYSIS

• 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 $doc_i$ and $doc_j$ both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle \approx 1$ if $doc_i$ and $doc_j$ both don’t contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle \approx 0$

Since this applies for all words, documents with that involve a similar set of words tend to have high dot product with each other.
If $doc_i$ and $doc_j$ both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle \approx 1$ If $doc_i$ and $doc_j$ both don’t contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle \approx 0$

Since this applies for all words, documents with that involve a similar set of words tend to have high dot product with each other.

Another View: Column of $Y$ represent ‘topics’. $\vec{y}_i(j)$ 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.
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 $Z^T = \Sigma_k V_k^T$ where columns of $V_k$ are the top $k$ eigenvectors of $X^TX$. 
• 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 $Z^T = \Sigma_k V_K^T$ where columns of $V_k$ are the top $k$ eigenvectors of $X^T X$. 
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 $X^T X$: where $(X^T X)_{a,b}$ is the number of documents that both $word_a$ and $word_b$ appear in.

- Think about $X^T X$ as a similarity matrix (gram matrix, kernel matrix) with entry $(a, b)$ being the similarity between $word_a$ and $word_b$. 
EXAMPLE: WORD EMBEDDING

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 $word_b$.

- 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.
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$_b$.

• 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 $X^TX$ with these different metrics (sometimes appropriately transformed) leads to popular word embedding algorithms: word2vec, GloVe, fastText, etc.
Note: word2vec is typically described as a neural-network method, but it is really just a low-rank approximation of a specific similarity matrix. Neural word embedding can be seen as implicit matrix factorization, as Levy and Goldberg.
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.
GRAPH EMBEDDINGS
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.

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

$$\mathbf{A} \in \mathbb{R}^{n \times n} \text{ with } A_{i,j} = \text{ edge weight between nodes } i \text{ and } j$$
How do we compute an optimal low-rank approximation of $A$?
How do we compute an optimal low-rank approximation of \( \mathbf{A} \)?

- Project onto the top \( k \) eigenvectors of \( \mathbf{A}^T \mathbf{A} = \mathbf{A}^2 \). (Note 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 $\mathbf{A}^T \mathbf{A} = \mathbf{A}^2$. (Note these are just the eigenvectors of $\mathbf{A}$).
  1. $\mathbf{A} \approx \mathbf{A} \mathbf{V}_k \mathbf{V}_k^T$ where $\mathbf{V}_k$ is the matrix with the top $k$ eigenvectors as columns.
How do we compute an optimal low-rank approximation of $\mathbf{A}$?

- Project onto the top $k$ eigenvectors of $\mathbf{A}^T \mathbf{A} = \mathbf{A}^2$. (Note these are just the eigenvectors of $\mathbf{A}$).

1. $\mathbf{A} \approx \mathbf{A} \mathbf{V}_k \mathbf{V}_k^T$ where $\mathbf{V}_k$ is the matrix with the top $k$ eigenvectors as columns.
2. Rows of $\mathbf{A} \mathbf{V}_k$ are an embedding of the nodes into $\mathbb{R}^k$. 

• Similar vertices (close with regards to graph proximity) should have similar embeddings since $\| (\mathbf{A})_i - (\mathbf{A})_j \|_2 \approx \| (\mathbf{A} \mathbf{V}_k \mathbf{V}_k^T)_i - (\mathbf{A} \mathbf{V}_k \mathbf{V}_k^T)_j \|_2 = \| (\mathbf{A} \mathbf{V}_k)_i - (\mathbf{A} \mathbf{V}_k)_j \|_2$ where we showed the equality in Lecture 14.
How do we compute an optimal low-rank approximation of $A$?

- Project onto the top $k$ eigenvectors of $A^T A = A^2$. (Note these are just the eigenvectors of $A$).

  1. $A \approx AV_k V_k^T$ where $V_k$ is the matrix with the top $k$ eigenvectors as columns.

  2. Rows of $AV_k$ are an embedding of the nodes into $\mathbb{R}^k$.

- Similar vertices (close with regards to graph proximity) should have similar embeddings since

$$\| (A)_i - (A)_j \|_2 \approx \| (AV_k V_k^T)_i - (AV_k V_k^T)_j \|_2 = \| (AV_k)_i - (AV_k)_j \|_2$$
How do we compute an optimal low-rank approximation of $A$?

- Project onto the top $k$ eigenvectors of $A^T A = A^2$. (Note these are just the eigenvectors of $A$).
  1. $A \approx AV_k V_k^T$ where $V_k$ is the matrix with the top $k$ eigenvectors as columns.
  2. Rows of $AV_k$ are an embedding of the nodes into $\mathbb{R}^k$.

- Similar vertices (close with regards to graph proximity) should have similar embeddings since
  $$\| (A)_i - (A)_j \|_2 \approx \| (AV_k V_k^T)_i - (AV_k V_k^T)_j \|_2 = \| (AV_k)_i - (AV_k)_j \|_2$$
  where we showed the equality in Lecture 14.
SPECTRAL EMBEDDING

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 approximate distances in your original ‘non-linear space.’
SPECTRAL EMBEDDING

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 approximate distances in your original 'non-linear space.'
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 $\mathbb{R}^k$. 
**SPECTRAL EMBEDDING**

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 approximate distances in your original ‘non-linear space.’