COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Andrew McGregor

Lecture 17

Last Class:

Last Class:

- SVD: Can write any matrix $\mathbf{X}$ with rank r as $\mathbf{U\Sigma V}^T$ where $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is diagonal and columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal.

Last Class:

- SVD: Can write any matrix $\mathbf{X}$ with rank r as $\mathbf{U\Sigma V}^T$ where $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is diagonal and columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal.
- Low-Rank Matrix Completion: Predict missing entries, e.g., movie scores in the context of the Netflix prize.

Today: Latent Semantic Analysis.

Last Class:

- SVD: Can write any matrix $\mathbf{X}$ with rank r as $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ where $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is diagonal and columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal.
- Low-Rank Matrix Completion: Predict missing entries, e.g., movie scores in the context of the Netflix prize.

Today: Latent Semantic Analysis.

- SVD Can reveal relationships between words and topics of documents.

Last Class:

- SVD: Can write any matrix $\mathbf{X}$ with rank r as $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ where $\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is diagonal and columns of $\mathbf{U}$ and $\mathbf{V}$ are orthonormal.
- Low-Rank Matrix Completion: Predict missing entries, e.g., movie scores in the context of the Netflix prize.

Today: Latent Semantic Analysis.

- SVD Can reveal relationships between words and topics of documents.

Today: Spectral Graph Theory & Spectral Clustering.

- Low-rank approximation on graph adjacency matrix for non-linear dimensionality reduction.
- Eigendecomposition to partition graphs into clusters.

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

ENTITY EMBEDDINGS

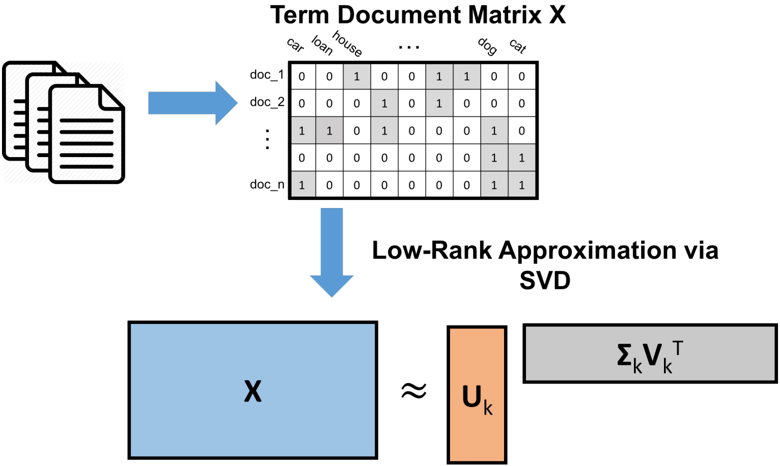
Dimensionality reduction embeds d-dimensional vectors into $k \ll d$ dimensions. But what about when you want to embed objects other than vectors?

- 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.



	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

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

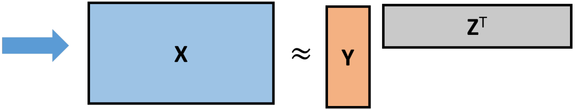
$$\mathbf{X} \approx \mathbf{Y}\,\mathbf{Z}^T$$

EXAMPLE: LATENT SEMANTIC ANALYSIS

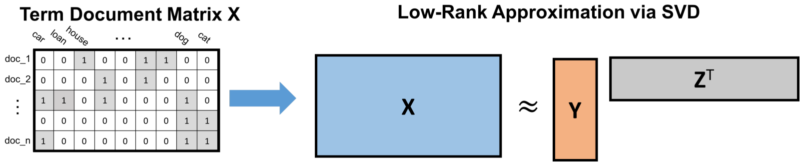
Term Document Matrix X

	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



EXAMPLE: LATENT SEMANTIC ANALYSIS



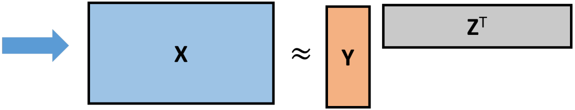
- If the error $\|\mathbf{X} - \mathbf{Y}\mathbf{Z}^T\|_F$ is small, then on average,

$$\mathbf{X}_{i,a} \approx (\mathbf{Y}\mathbf{Z}^T)_{i,a} = \langle \vec{y}_i, \vec{z}_a \rangle.$$

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



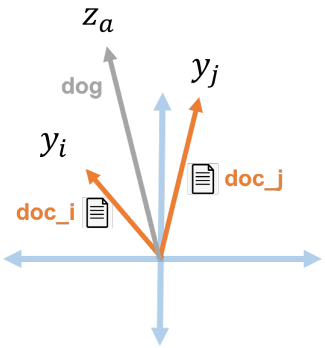
- If the error $\|\mathbf{X} - \mathbf{YZ}^T\|_F$ is small, then on average,

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

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

EXAMPLE: LATENT SEMANTIC ANALYSIS

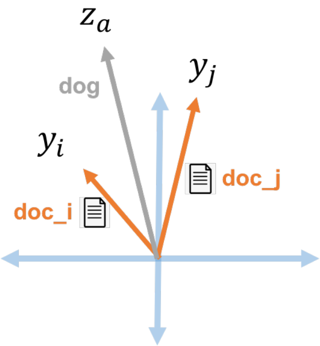
If doc_i and doc_j both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx 1$ and $\langle \vec{y}_j, \vec{z}_a \rangle \approx 1$ and so $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle$. Similarly if both don't contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle$



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

EXAMPLE: LATENT SEMANTIC ANALYSIS

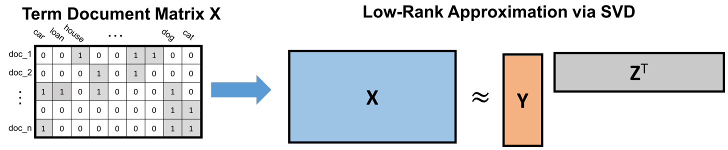
If doc_i and doc_j both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx 1$ and $\langle \vec{y}_j, \vec{z}_a \rangle \approx 1$ and so $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle$. Similarly if both don't contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle$



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

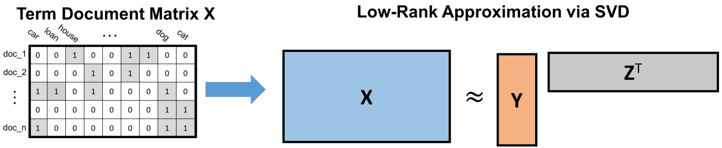
Another View: Column of $\mathbf{Y}$ represent 'topics'. $\vec{y}_i(j)$ indicates how much doc_i belongs to topic j. $\vec{z}_a(j)$ indicates how much $word_a$ associates with topic j.

EXAMPLE: LATENT SEMANTIC ANALYSIS

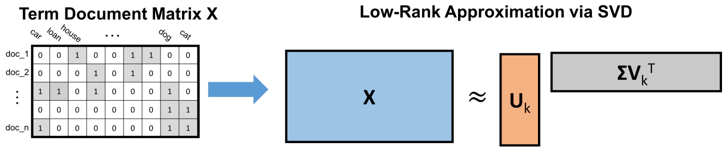


- Just like with documents, $\vec{z}_a$ and $\vec{z}_b$ will tend to have high dot product if $word_a$ and $word_b$ appear in many of the same documents.

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 $\mathbf{Z}^T = \mathbf{\Sigma}_k \mathbf{V}_K^T$ where columns of $\mathbf{V}_k$ are the top k eigenvectors of $\mathbf{X}^T\mathbf{X}$.



- 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 $\mathbf{Z}^T = \mathbf{\Sigma}_k \mathbf{V}_K^T$ where columns of $\mathbf{V}_k$ are the top k eigenvectors of $\mathbf{X}^T\mathbf{X}$.

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.

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 $word_b$.

EXAMPLE: WORD EMBEDDING

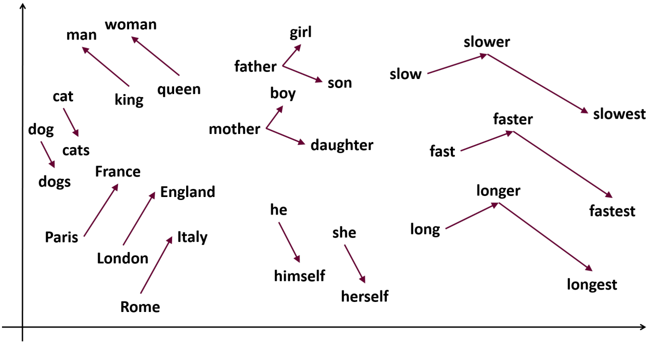
LSA gives a way of embedding words into k-dimensional space.

- 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.

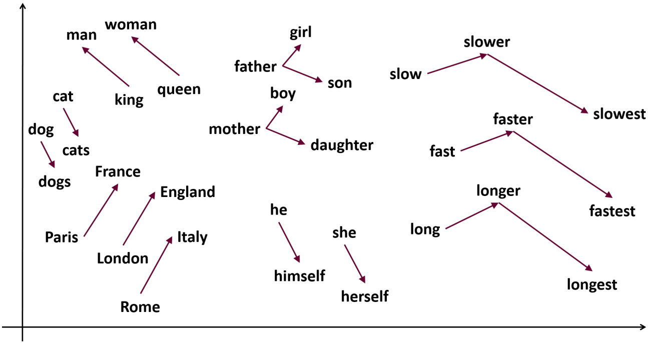
LSA gives a way of embedding words into k-dimensional space.

- 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.

EXAMPLE: WORD EMBEDDING

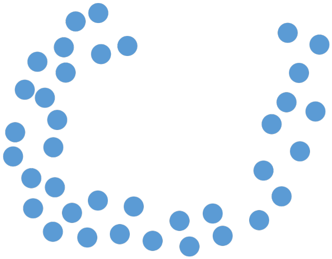


EXAMPLE: WORD EMBEDDING

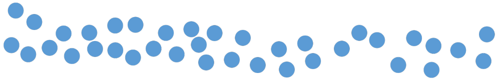


Note: word2vec is typically described as a neural-network method, but it is really just low-rank approximation of a specific similarity matrix. *Neural word embedding as implicit matrix factorization*, Levy and Goldberg.

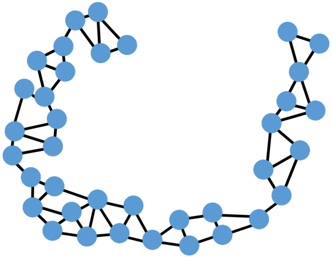
GRAPH EMBEDDINGS



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.

LINEAR ALGEBRAIC REPRESENTATION OF A GRAPH

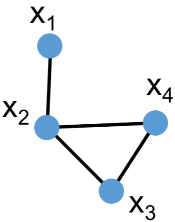
Once we have connected n data points $x_1, \ldots, x_n$ into a graph, we can represent that graph by its (weighted) adjacency matrix.

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

LINEAR ALGEBRAIC REPRESENTATION OF A GRAPH

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.

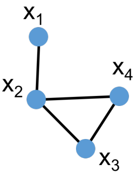
$$\mathbf{A} \in \mathbb{R}^{n \times n} \text{ with } \mathbf{A}_{i,j} = \text{ edge weight between nodes } i \text{ and } j$$



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

THE LAPLACIAN VIEW

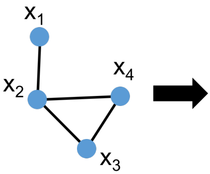
For a graph with adjacency matrix **A** and degree matrix **D**, $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the graph Laplacian.



$$\overset{\mathbf{D}}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}} - \overset{\mathbf{A}}{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}} = \overset{\mathbf{L}}{\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}}$$

THE LAPLACIAN VIEW

For a graph with adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$, $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the graph Laplacian.



$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}}_{\mathbf{D}} - \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}}_{\mathbf{L}}$$

For any vector $\vec{v}$, its 'smoothness' over the graph is given by:

$$\sum_{(i,j)\in E} (\vec{v}(i) - \vec{v}(j))^2 = \vec{v}^T \mathbf{L} \vec{v}.$$

REWRITING LAPLACIAN

Lemma:

$$\sum_{(i,j)\in E} (\vec{v}(i) - \vec{v}(j))^2 = \vec{v}^T \mathbf{L} \vec{v}$$

Lemma:

$$\sum_{(i,j)\in E} (\vec{v}(i) - \vec{v}(j))^2 = \vec{v}^T \mathbf{L} \vec{v}$$

Proof:

REWRITING LAPLACIAN

Lemma:

$$\sum_{(i,j)\in E} (\vec{v}(i) - \vec{v}(j))^2 = \vec{v}^T \mathbf{L} \vec{v}$$

Proof:

- Let L_e be the Laplacian for graph just containing edge e.

REWRITING LAPLACIAN

Lemma:

$$\sum_{(i,j)\in E} (\vec{v}(i) - \vec{v}(j))^2 = \vec{v}^T \mathbf{L} \vec{v}$$

Proof:

- Let L_e be the Laplacian for graph just containing edge e.
- By linearity,

$$\vec{v}^T \mathbf{L} \vec{v} = \sum_{e\in E} \vec{v}^T \mathbf{L}_e \vec{v}$$

REWRITING LAPLACIAN

Lemma:

$$\sum_{(i,j)\in E} (\vec{v}(i) - \vec{v}(j))^2 = \vec{v}^T \mathbf{L} \vec{v}$$

Proof:

- Let L_e be the Laplacian for graph just containing edge e.
- By linearity,

$$\vec{v}^T \mathbf{L} \vec{v} = \sum_{e\in E} \vec{v}^T \mathbf{L}_e \vec{v}$$

- If $e = (i, j)$, then $\vec{v}^T \mathbf{L}_e \vec{v} = (v(i) - v(j))^2$

ADJACENCY MATRIX EIGENVECTORS

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

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.

ADJACENCY MATRIX EIGENVECTORS

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 $\mathbb{R}^k$.

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

$$\|(\mathbf{A})_i - (\mathbf{A})_j\|_2 \approx \|(\mathbf{A}\mathbf{V}_k\mathbf{V}_k^T)_i - (\mathbf{A}\mathbf{V}_k\mathbf{V}_k^T)_j\|_2 = \|(\mathbf{A}\mathbf{V}_k)_i - (\mathbf{A}\mathbf{V}_k)_j\|_2$$

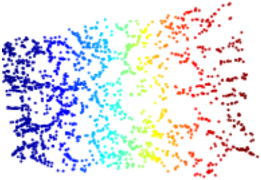
ADJACENCY MATRIX EIGENVECTORS

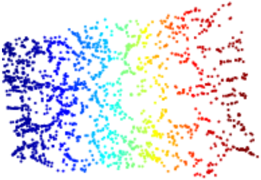
How do we compute an optimal low-rank approximation of $\mathbf{A}$?

- 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

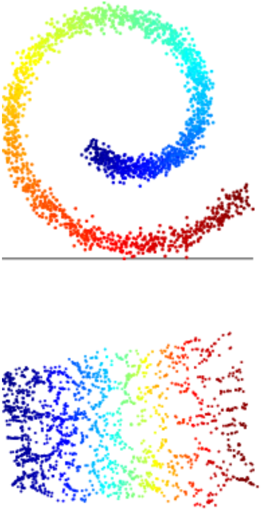
$$\|(\mathbf{A})_i - (\mathbf{A})_j\|_2 \approx \|(\mathbf{A}\mathbf{V}_k\mathbf{V}_k^T)_i - (\mathbf{A}\mathbf{V}_k\mathbf{V}_k^T)_j\|_2 = \|(\mathbf{A}\mathbf{V}_k)_i - (\mathbf{A}\mathbf{V}_k)_j\|_2$$

  where we showed the equality in Lecture 14.





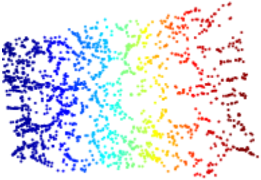
Step 1: Produce a nearest neighbor graph based on your input data in $\mathbb{R}^d$.



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.’