Embeddings part I

Dimensionality Reduction

All code for this tutorial is available in:
https://github.com/shunita/structML/tree/main/Tutorial4v2

Embedding - Motivation

***Aim**: Translate DB data into numerical data*

Relational DB

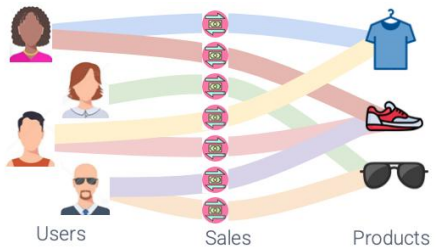
ID	Name
1	Bob
2	Alice

ID	Age	BMI	Smoking	Diabetes
1	23	25	current	0
2	67	31	former	1
3	50	29	never	0

Meaningless

Categorical

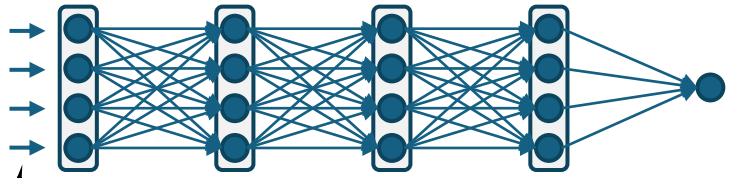
Graph representation



?

?

ML model

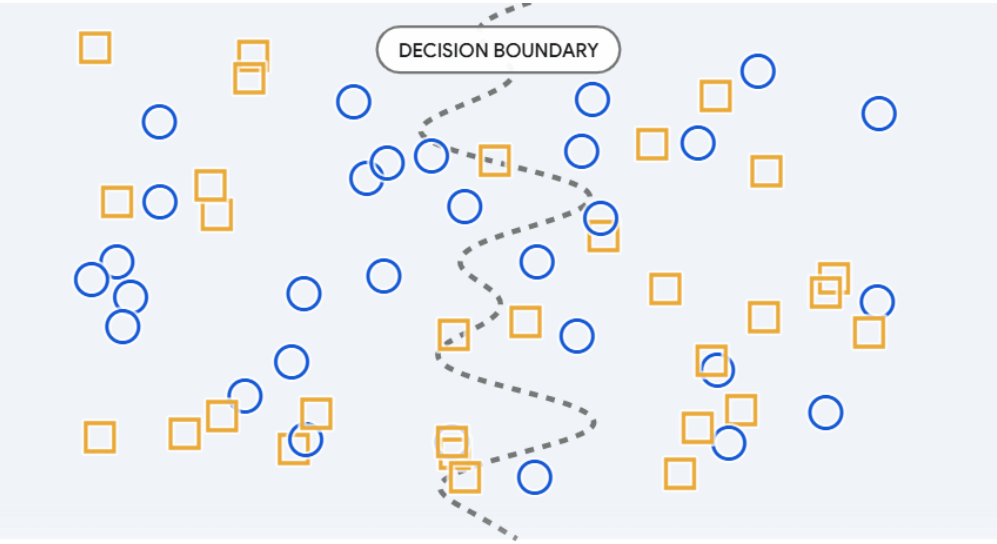


$\mathbb{R}^k$

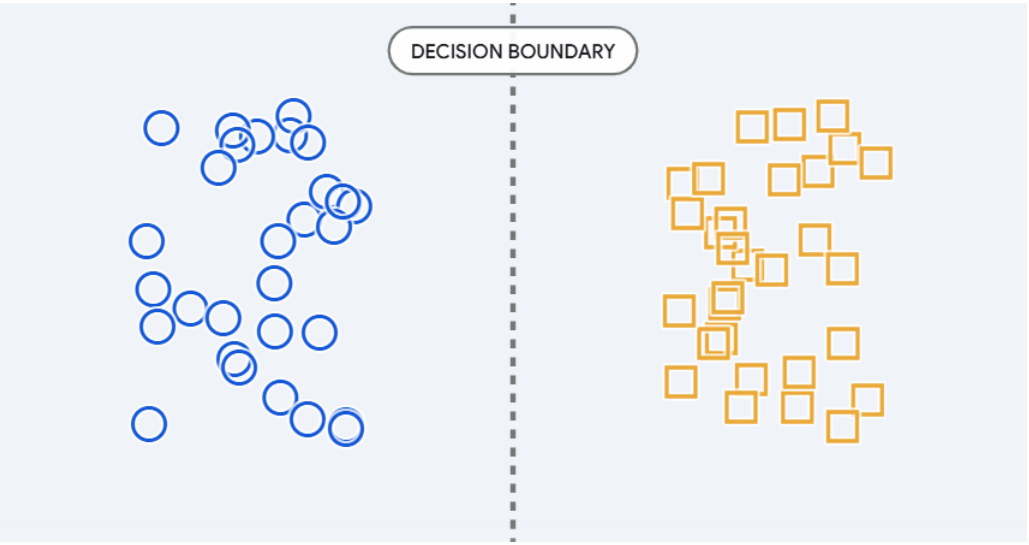
This is a **Geometric Euclidian Space.**
Does it matter how samples are distributed in it?

Embedding - Motivation

Arbitrary ***Encoding***



Meaningful ***Embedding***



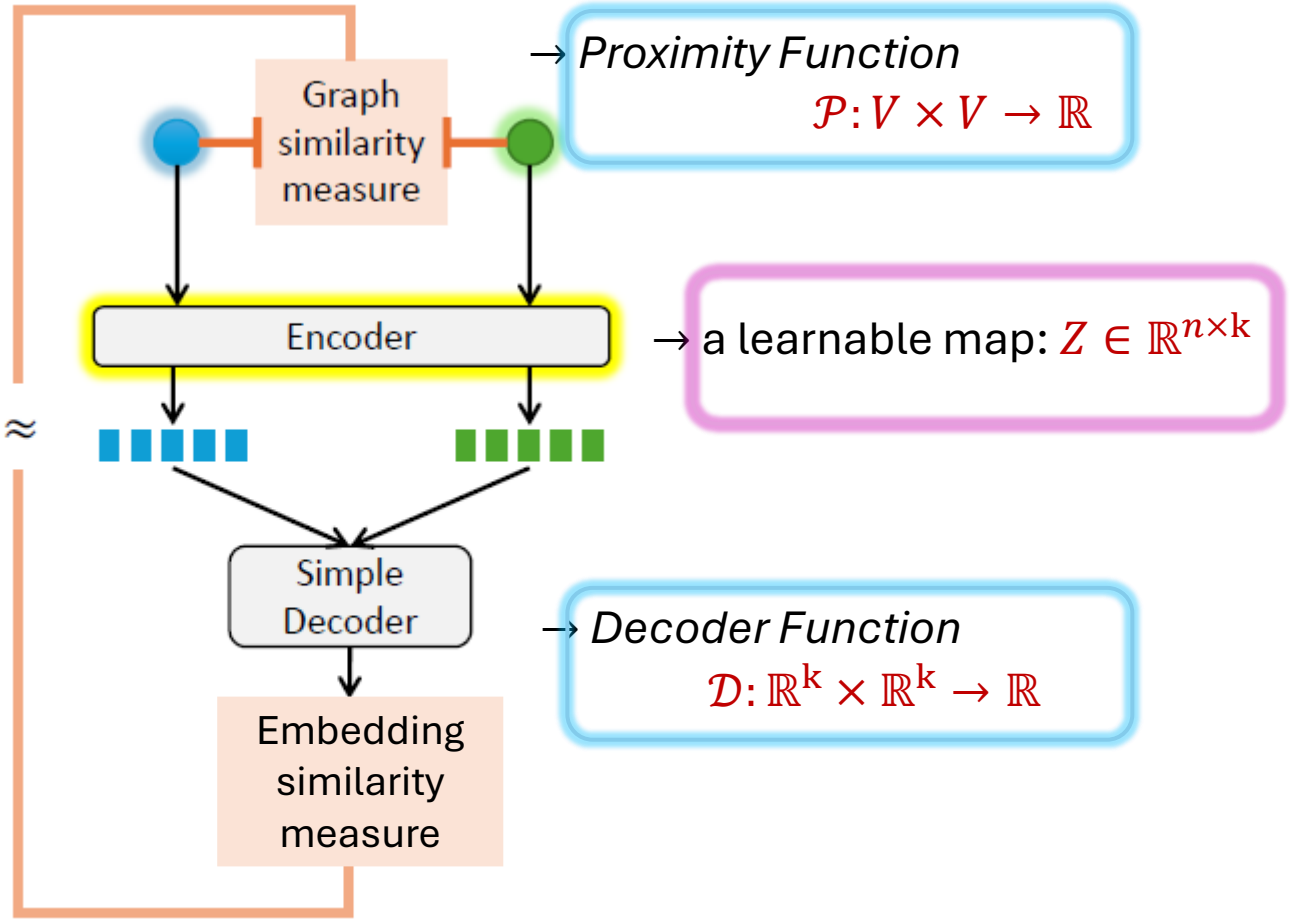
✅ Setting a meaningful embedding eases the model's ability to learn (approximate a function)
✅ Similar/connected tuples should have close embeddings
✅ Use unlabeled data to train embeddings – why?

Agenda

- Encoder-Decoder Framework (reminder)
- Truncated SVD - closed form solution (reminder)
- Different Encoder-Decoder instantiations:
  - Graph factorization
  - Laplacian Eigenmaps
  - Feature distance

Reminder: Embeddings via Encoder-Decoder Framework

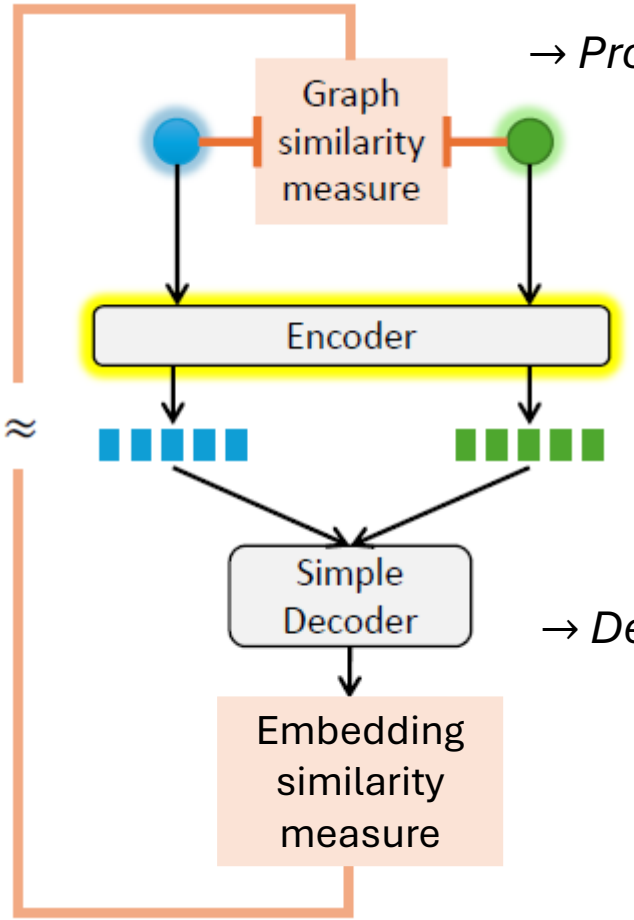
- *Tuple-level graph representation of the database*
- *Assumption: No features / Labels (homogenous)*
- *Fixed Embedding Dimension →* k



$$\text{cost}(\mathbf{Z}) := \sum_{u,v \in V} \text{Loss}_{\approx}\big(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z_u}, \mathbf{z_v})\big)$$

SVD as a closed form solution

Input: graph → $G = (V, E)$ $n := |V|$



→ *Proximity Function*

$\mathcal{P}: V \times V \to \mathbb{R}$

→ a learnable map: $Z \in \mathbb{R}^{n \times k}$

Loss Function ←

$\text{Loss}_{\approx}: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$\approx$

$$\text{cost}(\mathbf{Z}) := \sum_{u,v \in V} \text{Loss}_{\approx}\big(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z_u}, \mathbf{z_v})\big)$$

→ *Decoder Function*

$\mathcal{D}: \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$

$\mathcal{D}(z_u, z_v) = z_u \cdot z_v$ (dot product)

Dot Product Decoder - intuition

$$\mathcal{D}(\mathbf{z_u}, \mathbf{z_v}) = \mathbf{z_u} \cdot \mathbf{z_v}$$

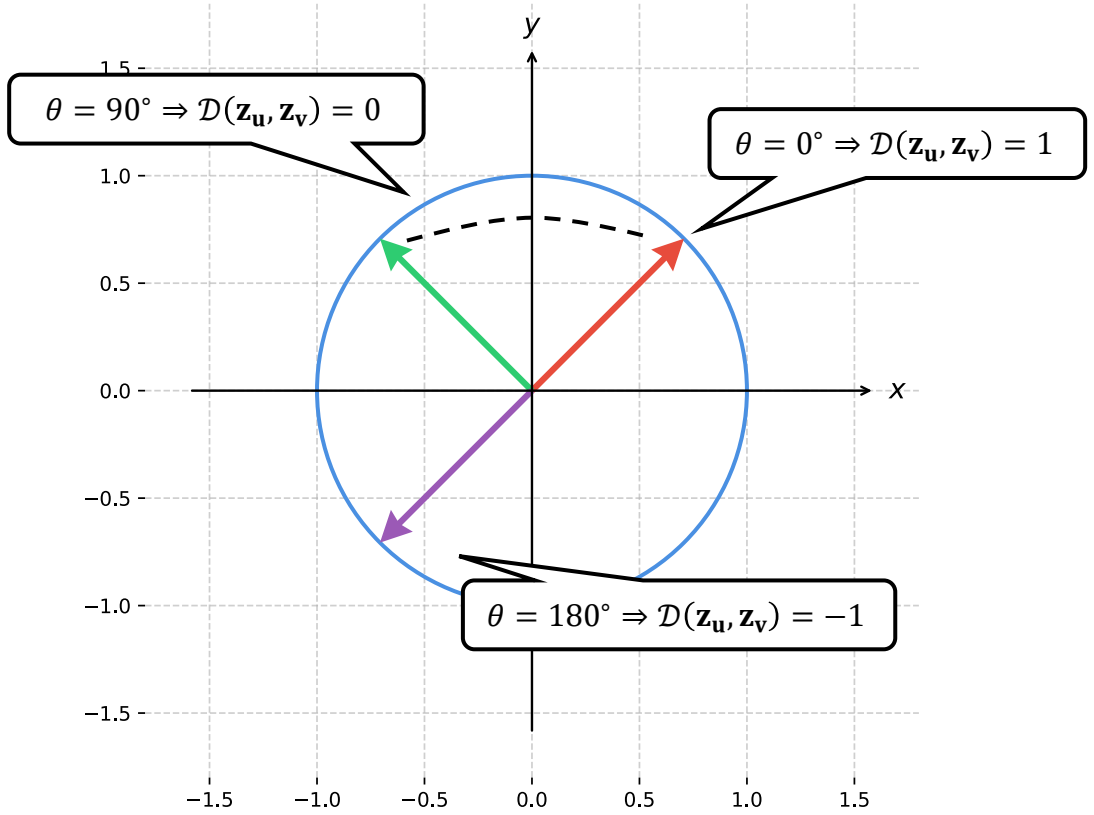
⇒ *Why does dot product measure "closeness"?*

cosine similarity

$$\cos(\theta) = \frac{z_u \cdot z_v}{|z_u||z_v|}$$

Suppose $|z_v| = 1$ for all $v \in V$

⇒ The closer the vectors – the higher the measure

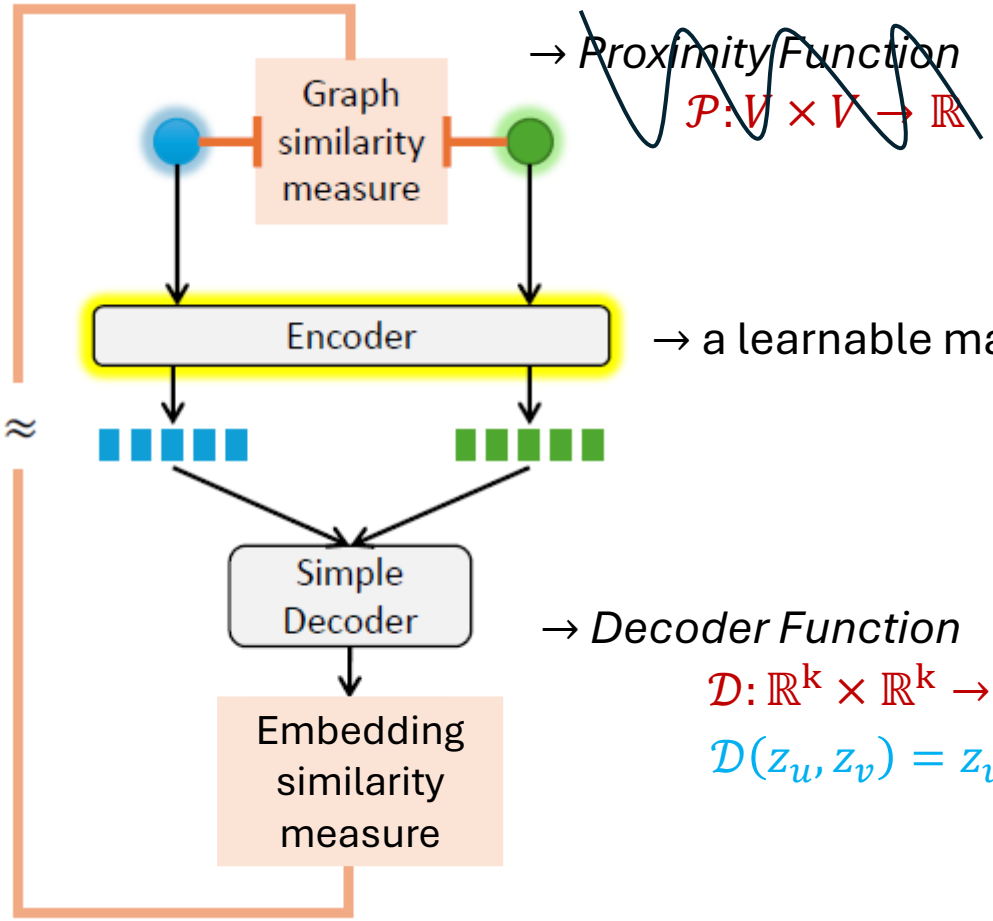


SVD as a closed form solution

Input: graph → $G = (V, E)$ $n := |V|$

→ *Proximity Function* → *Proximity matrix*

$\mathcal{P}: V \times V \to \mathbb{R}$ $\mathcal{P} \in \mathbb{R}^{|V| \times |V|}$



→ a learnable map: $Z \in \mathbb{R}^{n \times \mathrm{k}}$

Loss Function ←

$\mathrm{Loss}_{\approx}: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$\mathrm{Loss}_{\approx}(x, y) = (x - y)^2$

$$\mathrm{cost}(\mathbf{Z}) := \sum_{u,v \in V} \mathrm{Loss}_{\approx}\big(\mathcal{P}(u, v), \mathcal{D}(\mathbf{z_u}, \mathbf{z_v})\big)$$

$\mathrm{cost}(\mathbf{Z}) := \|Z^T \cdot Z - \mathcal{P}\|_2$ (Frobenius Norm)

→ *Decoder Function*

$\mathcal{D}: \mathbb{R}^{\mathrm{k}} \times \mathbb{R}^{\mathrm{k}} \to \mathbb{R}$

$\mathcal{D}(z_u, z_v) = z_u \cdot z_v$ (dot product)

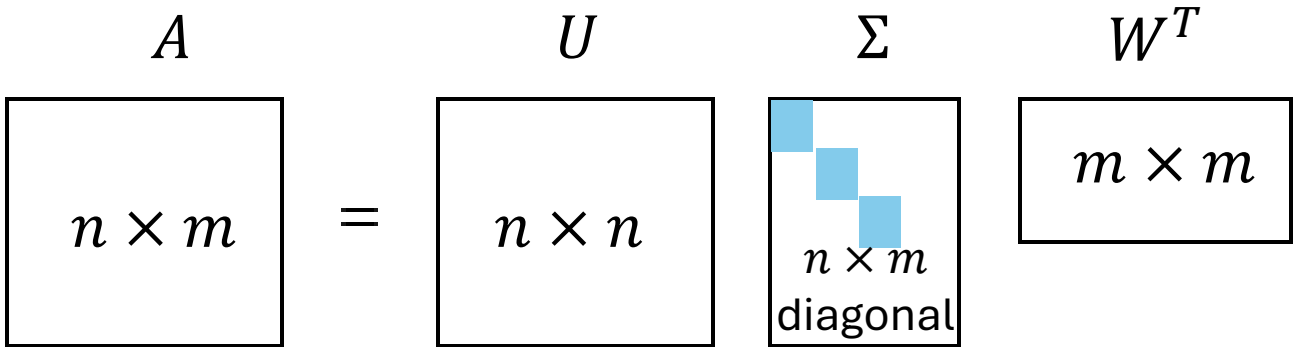
- We want Z where: $\mathcal{P} \approx Z^T \cdot Z$
- SVD will help us find an optimal Z
  - Without training!

Reminder: Singular Value Decomposition (SVD)

Eckart-Young-Mirsky theorem:

Truncated SVD yields the optimal rank-k approximation of A.

Full SVD



When A is symmetric (like $\mathcal{P}$),

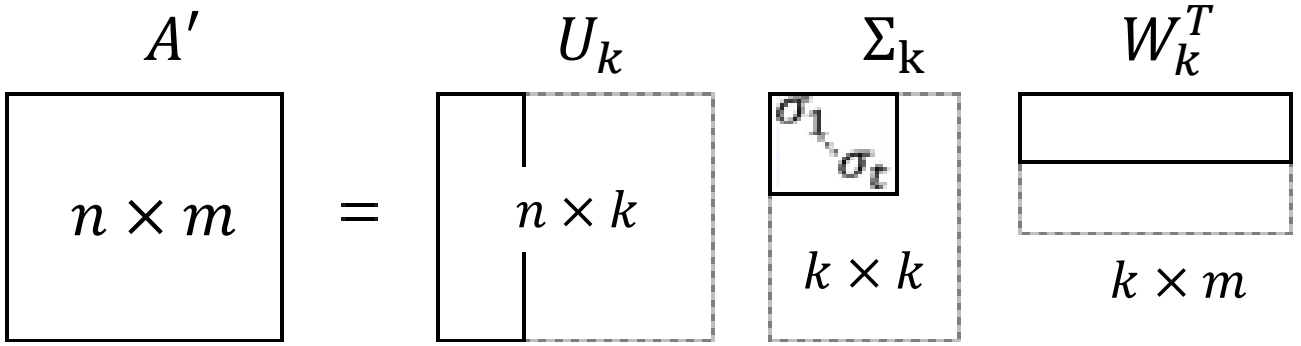
$$U = W$$

$$Z \cdot Z^T = \mathcal{P} = U_k \Sigma_k U_k$$

So the optimal embedding is:

$$Z = U_k \sqrt{\Sigma_k}$$

Truncated SVD



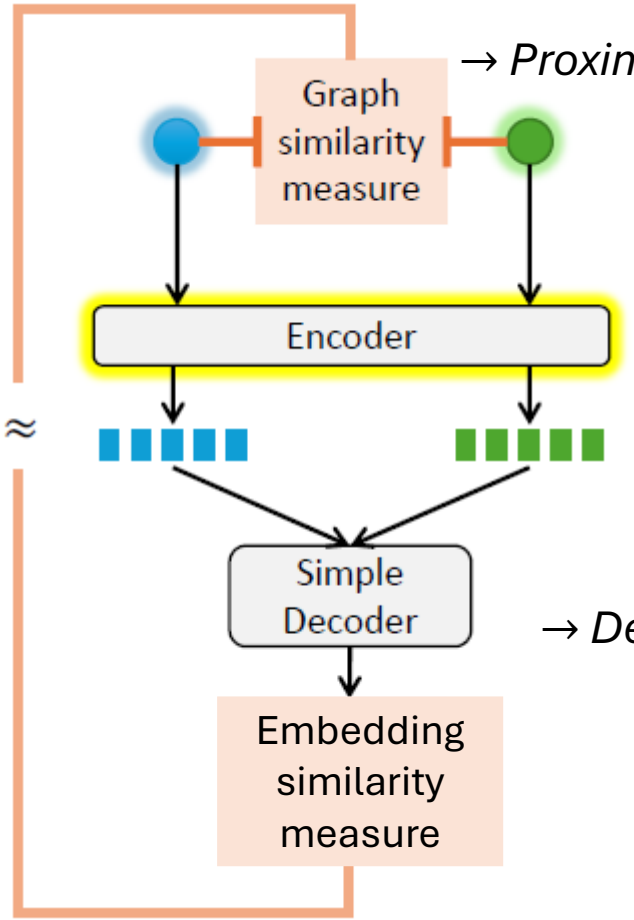
Agenda

- Encoder-Decoder Framework (reminder)
- Truncated SVD - closed form solution (reminder)
- Different Encoder-Decoder instantiations:
  - Graph factorization
  - Laplacian Eigenmaps
  - Feature distance

Graph Factorization

Caveat: only captures first-order - 1-hop neighbors.

Input: graph → $G = (V, E)$ $n := |V|$



→ *Proximity matrix*

$\mathcal{P} \in \mathbb{R}^{|V| \times |V|}$

$\mathcal{P}_{u,v} = A_{u,v} = \begin{cases} 1 & (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$

→ a learnable map: $Z \in \mathbb{R}^{n \times k}$

Loss Function ←

$\text{Loss}_{\approx}: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$\text{Loss}_{\approx}(x, y) = (x - y)^2$

$$\text{cost}(\mathbf{Z}) := \sum_{u,v \in V} \text{Loss}_{\approx}\left(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z_u}, \mathbf{z_v})\right)$$

$\text{cost}(\mathbf{Z}) := \|Z^T \cdot Z - \mathcal{P}\|_2$ (Frobenius Norm)

→ *Decoder Function*

$\mathcal{D}: \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$

$\mathcal{D}(z_u, z_v) = z_u \cdot z_v$ (dot product)

Code Example: Graph Factorization

Build a graph representation of the database:

```
G = nx.Graph()
G.add_nodes_from(…)
G.add_edges_from(…)
A = nx.adjacency_matrix(G, nodelist=sorted(G.nodes())).toarray()
```

Populating the graph – see previous tutorial

```
Adjacency Matrix Shape: (10, 10)
[[0 1 1 0 1 0 0 0 0 0]
 [1 0 1 1 0 0 0 0 0 0]
 [1 1 0 1 0 0 0 0 0 0]
 [0 1 1 0 1 0 0 0 0 0]
 [1 0 0 1 0 1 0 0 0 0]
 [0 0 0 0 1 0 1 1 0 1]
 [0 0 0 0 0 1 0 1 0 1]
 [0 0 0 0 0 1 1 0 1 0]
 [0 0 0 0 0 0 0 1 0 1]
 [0 0 0 0 0 1 1 0 1 0]]
```

customers

cID	Name	Interests
0	John	Tech
6	Paul	Music
4	George	Tech
…		

customer chats

cID1	cID2
0	1
0	2
6	9
…	..

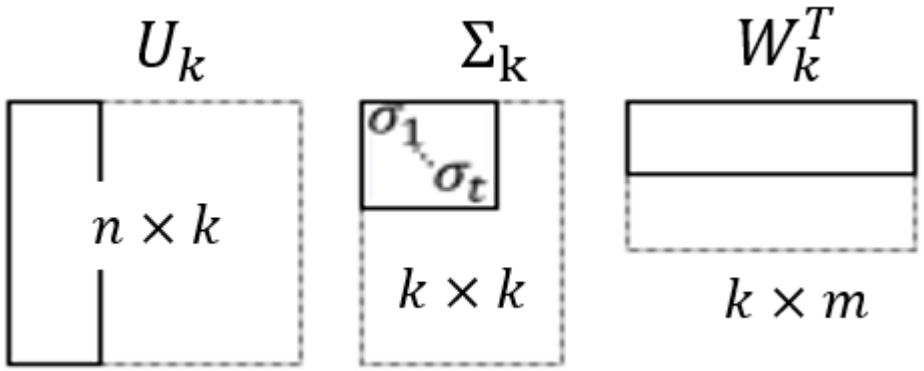
Code Example: Graph Factorization

Perform truncated SVD with embedding dimension k

```
from sklearn.decomposition import TruncatedSVD
k = 4  # Target embedding dimension
svd = TruncatedSVD(n_components=k, random_state=42)
U_k = svd.fit_transform(A)
Sigma_k = svd.singular_values_
Z = U_k * np.sqrt(Sigma_k)
```

Sanity check – what should be the output shapes?

```
U shape:     (10, 4)
Sigma shape: (4,)
Z shape:     (10, 4)
```



$$Z = U_k \sqrt{\Sigma_k}$$

customers

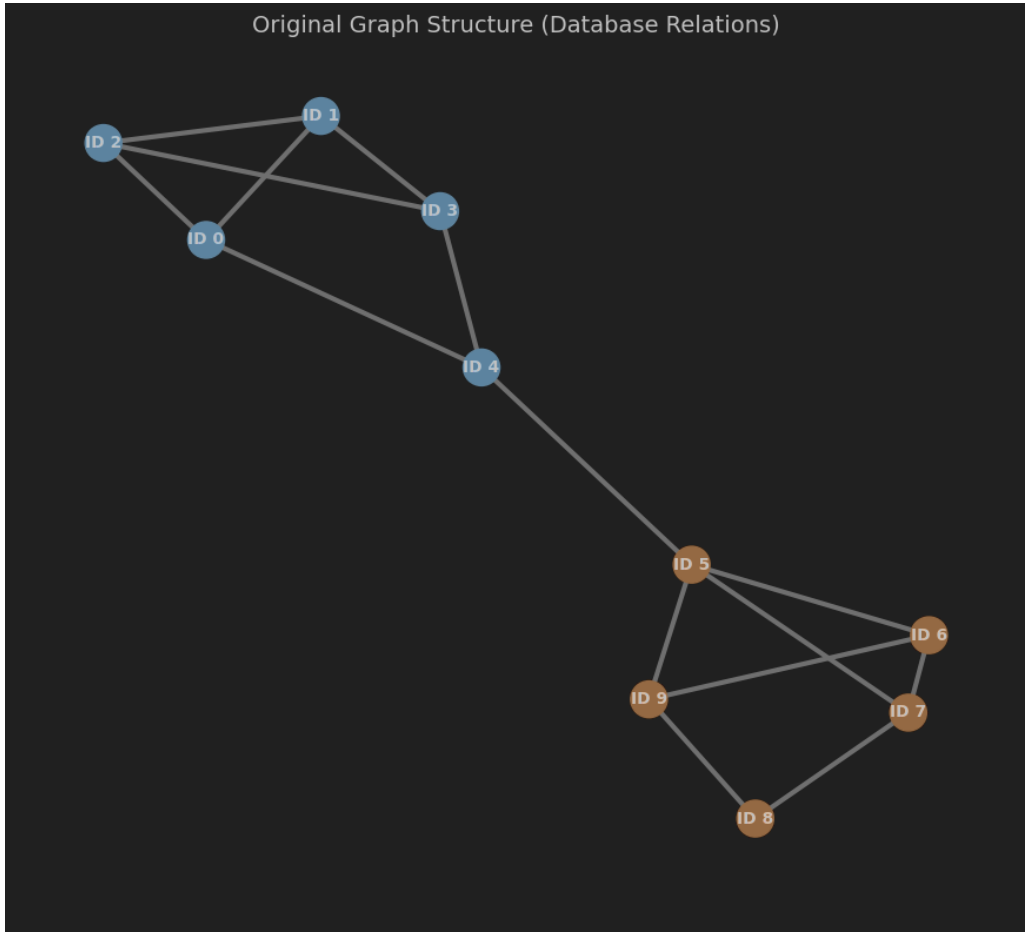
cID	Name	Interests
0	John	Tech
6	Paul	Music
4	George	Tech
…		

customer chats

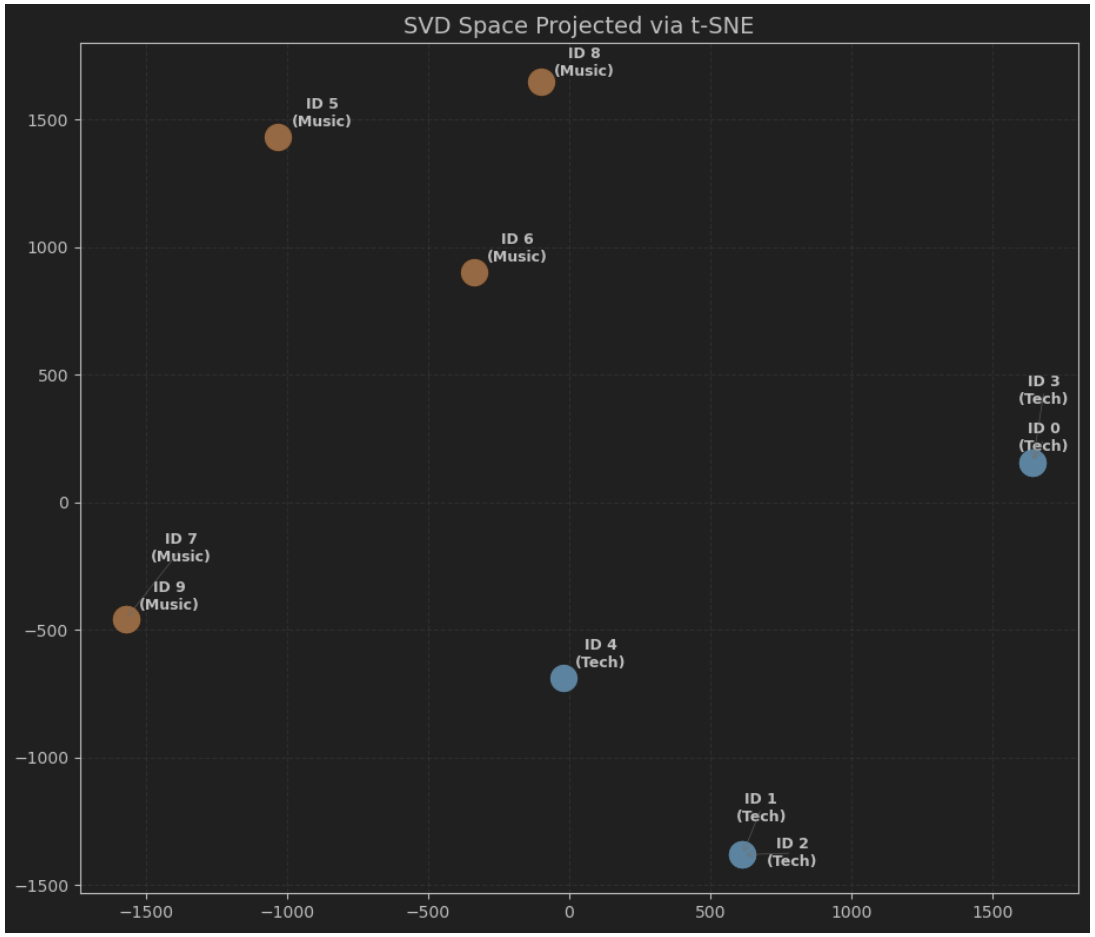
cID1	cID2
0	1
0	2
6	9
…	..

Code Example: Direct SVD on Adjacency Matrix

Visualize the original graph:



Visualize using t-sne: (reduction to 2 dimensions)



Preserves global graph structure: separation of highly connected components

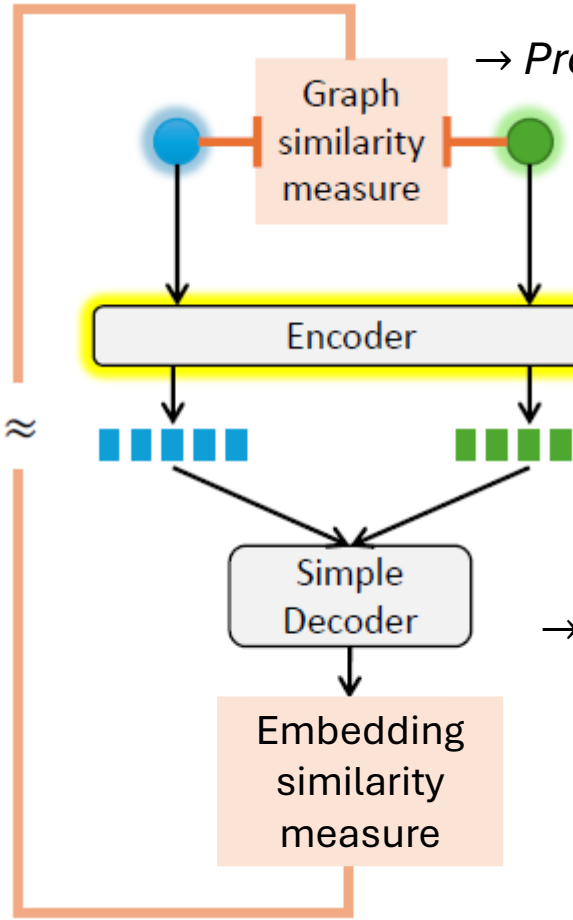
Agenda

- Encoder-Decoder Framework (reminder)
- Truncated SVD - closed form solution (reminder)
- Different Encoder-Decoder instantiations:
  - Graph Factorization
  - Laplacian Eigenmaps
  - Feature distance

Laplacian Eigenmaps

Preserves **local neighborhoods**:
u,v contribute to the loss only if they are connected.

Input: graph → $G = (V, E)$ $n := |V|$



→ *Proximity matrix*

$\mathcal{P} \in \mathbb{R}^{|V|\times|V|}$

$$\mathcal{P}_{u,v} = A_{u,v} = \begin{cases} 1 & (u,v) \in E \\ 0 & \text{otherwise} \end{cases}$$

→ a learnable map: $Z \in \mathbb{R}^{n\times k}$

Loss Function ←

$\text{Loss}_{\approx}: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$\text{Loss}_{\approx}(x, y) = x \cdot y$

$$\text{cost}(\mathbf{Z}) := \sum_{u,v\in V} \text{Loss}_{\approx}\big(\mathcal{P}(u,v), \mathcal{D}(\mathbf{z_u}, \mathbf{z_v})\big)$$

$$\text{cost}(\mathbf{Z}) := \sum_{u,v\in V} \mathcal{P}_{u,v} \|z_u - z_v\|^2$$

→ *Decoder Function*

$\mathcal{D}: \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$

$\mathcal{D}(z_u, z_v) = \|z_u - z_v\|^2$

(Euclidean distance)

Code Example: Laplacian Eigenmaps

Compute Laplacian eigenmaps embedding

```
from sklearn.manifold import SpectralEmbedding
laplacian_embedding = SpectralEmbedding(
    n_components=k, affinity='precomputed', random_state=42)
Z_laplacian = laplacian_embedding.fit_transform(A)

print("Shape of the derived Laplacian Embedding matrix Z:",
      Z_laplacian.shape)
```

Meaning: A is already the proximity matrix

```
Shape of the derived Laplacian Embedding matrix Z: (10, 4)
[[ 1.87741071e-01 -1.94402571e-02  5.71027833e-02  3.54921064e-01]
 [ 2.14260253e-01  1.62894471e-01 -1.31722402e-01 -5.67094933e-17]
 [ 2.14260253e-01  1.62894471e-01 -1.31722402e-01  5.85998421e-17]
 [ 1.87741071e-01 -1.94402571e-02  5.71027833e-02 -3.54921064e-01]
 [ 8.82289469e-02 -3.40589096e-01  2.71038500e-01  1.50091851e-16]
 [-1.32635652e-01 -2.20414975e-01 -7.81620643e-02  1.97405908e-16]
 [-1.88577730e-01 -5.88004489e-02 -3.20445709e-01 -3.87537907e-17]
 [-1.93208335e-01  8.78246641e-02  1.77741226e-02 -2.01736722e-01]
 [-2.10584488e-01  3.46077632e-01  4.00971429e-01 -7.95955075e-17]
 [-1.93208335e-01  8.78246641e-02  1.77741226e-02  2.01736722e-01]]
```

customers

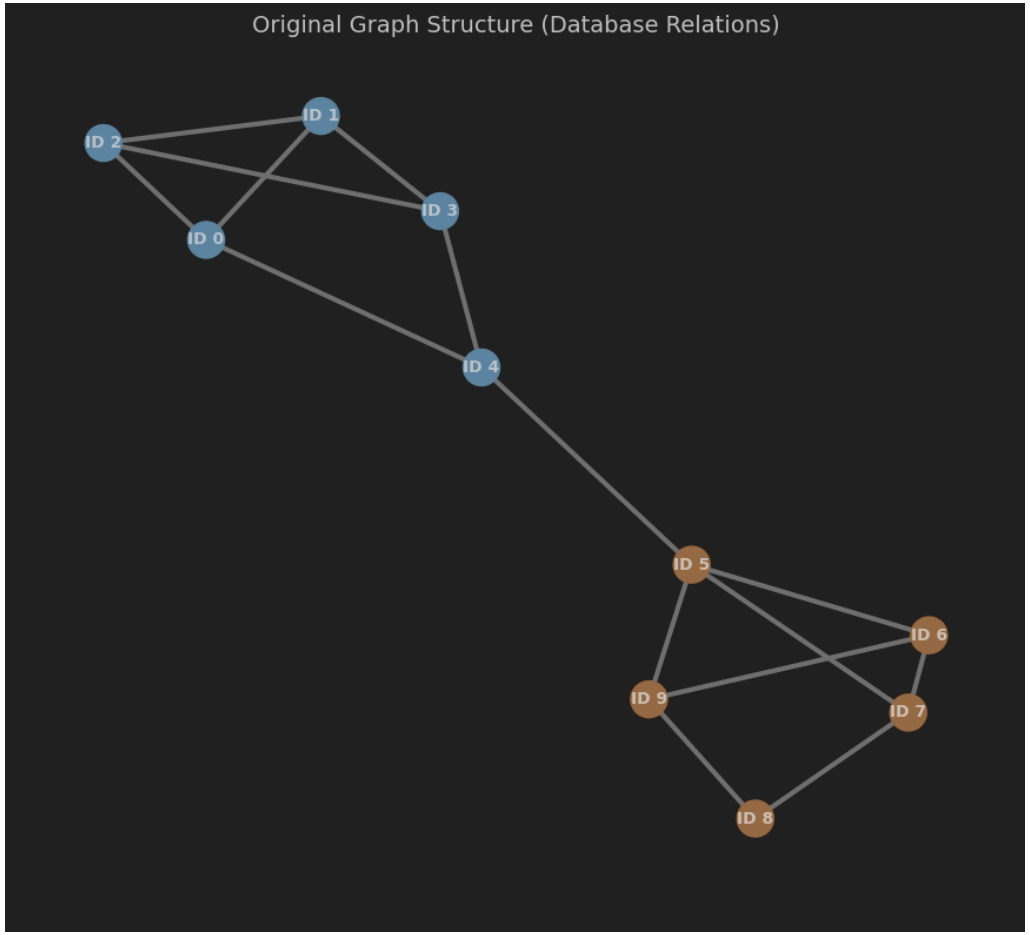
cID	Name	Interests
0	John	Tech
6	Paul	Music
4	George	Tech
…		

customer chats

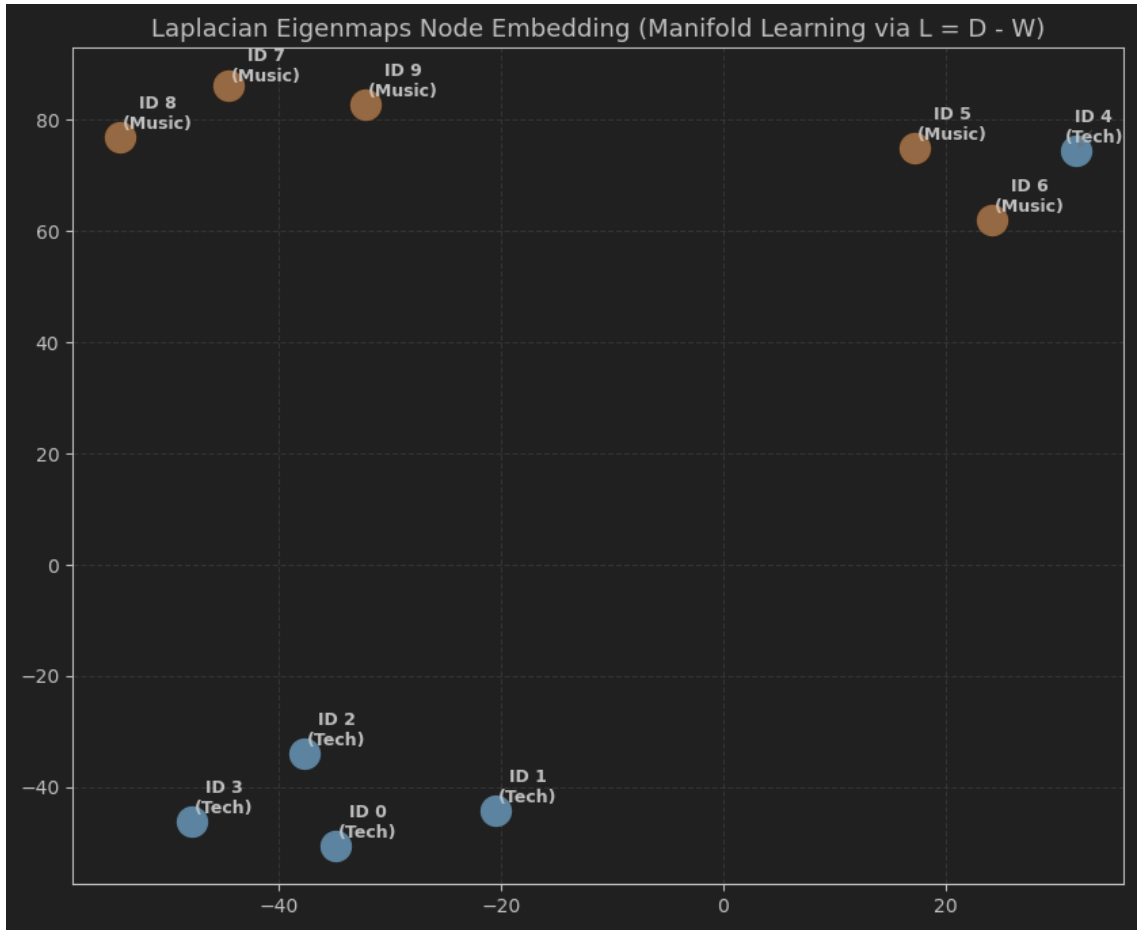
cID1	cID2
0	1
0	2
6	9
…	..

Code example: Laplacian Eigenmaps

Visualize the original graph:



Visualize using t-sne: (reduction to 2 dimensions)



Laplacian eigenmaps preserve local neighborhoods!

Graph Factorization vs. Laplacian Eigenmaps

Property	Graph Factorization	Laplacian Eigenmaps
Decoder + Loss (optimization target)	Dot product + Frobenius norm $\|Z \cdot Z^T - \mathcal{P}\|_2$	Euclidean dist. + localized neighborhood loss $\mathcal{P}_{u,v}\|z_u - z_v\|^2$
Meaning of $\mathcal{P}_{u,v} = 0$	**Unconnected:** A strong signal to force a low dot product $z_u \cdot z_v$.	**Ignored:** No connection = no penalty $\|z_u - z_v\|^2$ doesn't matter.
Meaning of optimization target	Try to reconstruct the entire proximity matrix	Try to preserve local (proximity) neighborhoods

Agenda

- Encoder-Decoder Framework (reminder)
- Truncated SVD - closed form solution (reminder)
- Different Encoder-Decoder instantiations:
  - Adjacency matrix – direct truncated SVD
  - Laplacian Eigenmaps
  - Feature similarity (kernel PCA)

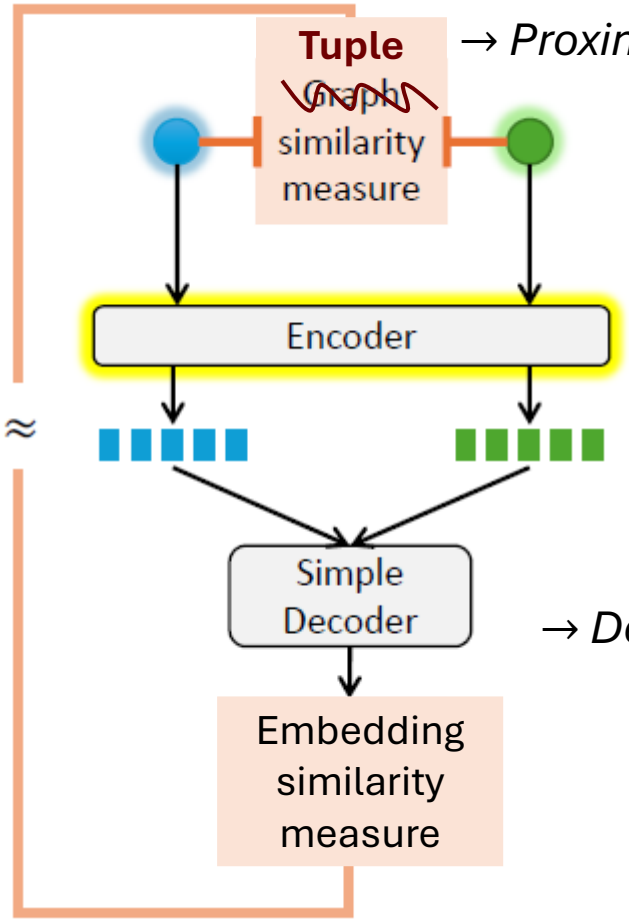
Kernel PCA: Truncated SVD with Feature Similarity

→ Consider the table $R(A_1, \ldots, A_m)$ as a matrix $X \in \mathbb{R}^{n \times m}$

→ What will Truncated SVD optimize?
Large similarity $z_i \cdot z_j$ for tuples with similar features, otherwise – large distance.

→ Caveat: must encode all required features to numeric.

Input: graph → $G = (V, E)$ $n := |V|$



→ *Proximity matrix*

$\mathcal{P} \in \mathbb{R}^{|V| \times |V|}$

$\mathcal{P}_{i,j} = e^{-\frac{\|x_i - x_j\|^2}{\sigma^2}}$

RBF kernel over distance between feature vectors

→ a learnable map: $Z \in \mathbb{R}^{n \times k}$

→ *Decoder Function*

$\mathcal{D}: \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$

$\mathcal{D}(z_u, z_v) = z_u \cdot z_v$ (dot product)

UNCHANGED

Loss Function ← ≈

UNCHANGED

$\text{Loss}_{\approx}: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

$\text{Loss}_{\approx}(x, y) = (x - y)^2$

$$\text{cost}(\mathbf{Z}) := \sum_{u,v \in V} \text{Loss}_{\approx}\left(\mathcal{P}(u, v), \mathcal{D}(\mathbf{z_u}, \mathbf{z_v})\right)$$

$\text{cost}(\mathbf{Z}) := \|Z^T \cdot Z - \mathcal{P}\|_2$ (Frobenius Norm)

Code Example: Kernel PCA

Customers + numeric features

cID	Name	Interests	Total spent ($)	Days active	Support tickets
0	John	Tech	1200	730	1
6	Paul	Music	150	30	0
4	George	Tech	500	200	12
…					

Construct a feature matrix

```
feature_cols = ['total_spend_usd', 'days_active', 'support_tickets']
X = customers_df[feature_cols].values
print(X)
```

```
[[1200  730    1]
 [1450  810    2]
 [1100  690    0]
 [1300  750    1]
 [ 150   30    0]
 [  80   15    1]
 [ 200   45    0]
 [ 500  200   12]
 [ 650  250   15]
 [ 480  180    9]]
```

Q: What could be the problem with this matrix?

A: Very different feature ranges may cause some features to dominate the proximity!

Code Example: Kernel PCA

Customers + numeric features

cID	Name	Interests	Total spent ($)	Days active	Support tickets
0	John	Tech	1200	730	1
6	Paul	Music	150	30	0
4	George	Tech	500	200	12
...					

Construct a feature matrix **and scale it**

```
feature_cols = ['total_spend_usd', 'days_active', 'support_tickets']
X = customers_df[feature_cols].values

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print(X_scaled)
```

Z-score normalization: $\frac{x - \mu}{\sigma}$

```
[[ 1.00678469  1.14033162 -0.57675042]
 [ 1.52150079  1.39373864 -0.3907019 ]
 [ 0.80089825  1.0136281  -0.76279895]
 [ 1.21267113  1.20368337 -0.57675042]
 [-1.15502293 -1.07697986 -0.76279895]
 [-1.29914344 -1.12449368 -0.57675042]
 [-1.05207971 -1.02946604 -0.76279895]
 [-0.43442039 -0.53848993  1.46978334]
 [-0.12559073 -0.38011054  2.02792891]
 [-0.47559768 -0.60184169  0.91163777]]
```

Code Example: Kernel PCA

Customers + numeric features

cID	Name	Interests	Total spent ($)	Days active	Support tickets
0	John	Tech	1200	730	1
6	Paul	Music	150	30	0
4	George	Tech	500	200	12
...					

Compute Kernel-PCA

```
from sklearn.decomposition import KernelPCA

computed_gamma = 1.0 / (X_scaled.shape[1] * X_scaled.var())
k_pca = KernelPCA(n_components=k, kernel='rbf',
                  gamma=computed_gamma, random_state=42)
Z_pca = k_pca.fit_transform(X_scaled)
```

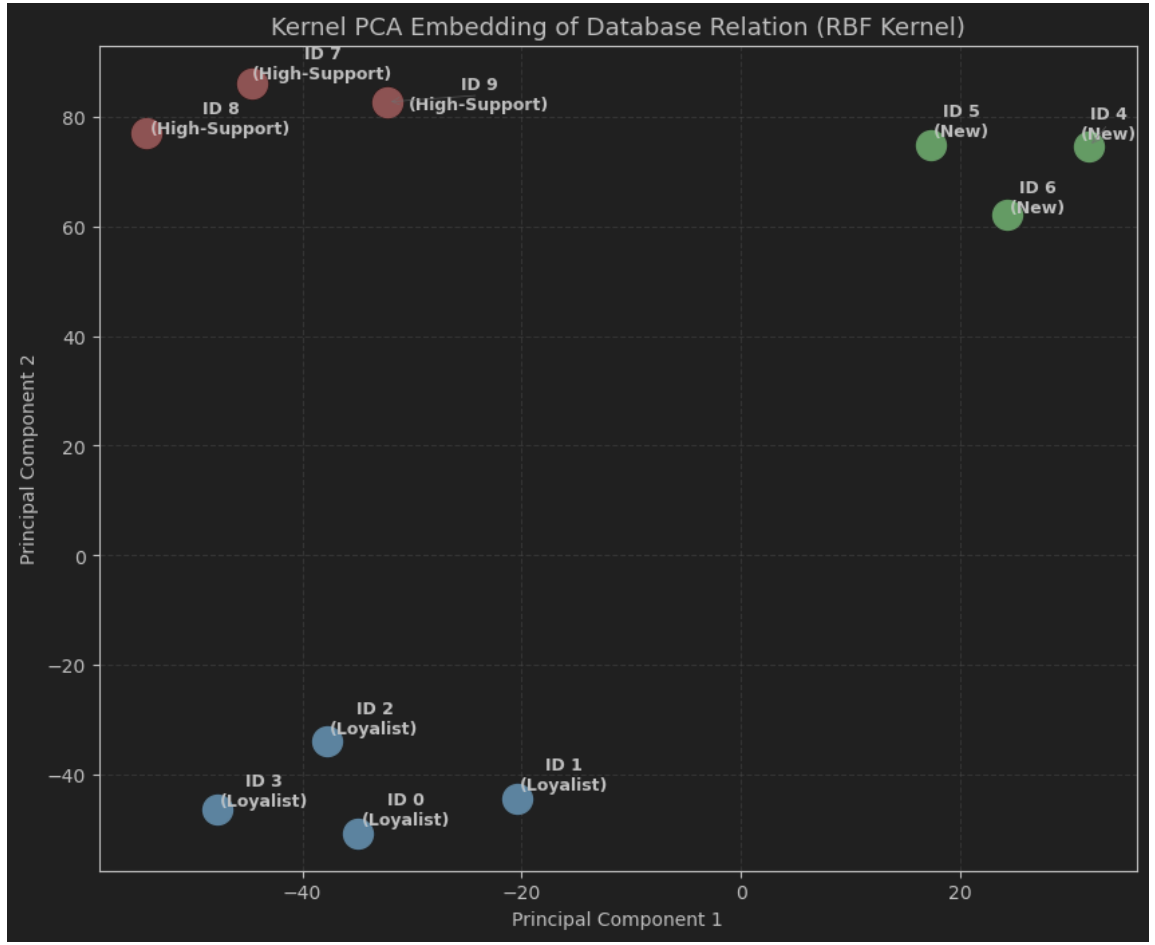
$$\mathcal{P}_{i,j} = e^{-\frac{\|x_i - x_j\|^2}{\sigma^2}}$$

$$\gamma = \frac{1}{\sigma^2}, \mathcal{P}_{i,j} = e^{-\gamma\|x_i - x_j\|^2}$$

```
Shape of the derived embedding matrix Z: (10, 4) (n x k)
[[ 0.72309919 -0.07473892 -0.03642598 -0.09855093]
 [ 0.69716065 -0.06473278  0.07867312  0.35517286]
 [ 0.67083933 -0.08434191 -0.07756254 -0.31048989]
 [ 0.73774781 -0.07548732  0.0042573   0.05371421]
 [-0.56353904 -0.56649649  0.05175138 -0.00780019]
 [-0.57451497 -0.53777502  0.01653178  0.02255076]
 [-0.55769329 -0.55561967  0.02923201 -0.02075482]
 [-0.3910118   0.71960279 -0.06752285  0.01908032]
 [-0.31039483  0.74090532  0.40730214 -0.07272798]
 [-0.43169304  0.498684   -0.40623634  0.05980568]]
```

Code Example: SVD the on feature matrix

Visualize using t-sne:



Customers + numeric features

cID	Name	Interests	Total spent ($)	Days active	Support tickets	Customer Type
0	John	Tech	1200	730	1	Loyal
6	Paul	Music	150	30	0	New
4	George	Tech	500	200	12	High-support
…						

Not shown to the embedding model

Summary – Dimensionality Reduction over DBs

Property	Kernel PCA	Graph Factorization	Laplacian Eigenmaps
Input matrix	Feature matrix	Adjacency matrix	Adjacency matrix
Proximity (u,v)	RBF kernel over Euclidean distance of feature vectors: $\mathcal{P}_{u,v} = e^{-\frac{\|x_u - x_v\|^2}{\sigma^2}}$	$\mathcal{P}_{u,v} = A_{u,v}$	$\mathcal{P}_{u,v} = A_{u,v}$
Decoder + Loss (optimization target)	Dot product + Frobenius norm $\|Z \cdot Z^T - \mathcal{P}_{uv}\|_2$	Dot product + Frobenius norm $\|Z \cdot Z^T - \mathcal{P}_{uv}\|_2$	Euclidean distance + localized neighborhood loss $A_{u,v}\|z_u - z_v\|^2$
Meaning of $\mathcal{P}_{u,v} = 0$	**Unsimilar:** A strong signal to force a low dot product $z_u \cdot z_v$.	**Unconnected:** A strong signal to force a low dot product $z_u \cdot z_v$.	**Ignored:** No connection = no penalty; $\|z_u - z_v\|^2$ doesn't matter.
Meaning of optimization target	Try to preserve feature similarity in higher dimensional space	Try to reconstruct the entire proximity matrix	Try to preserve local (proximity) neighborhoods

Many other encoder-decoder instantiations exist – see lecture + next tutorial!