Comp790-166: Computational Biology

Lecture 2

January 12, 2022

Today

- Linear algebra background/review (content from SLMT)
  - Matrix Rank and Low Rank Approximations
  - SVD and singular values
- Graphs
  - Notation, Basic Properties
  - Paths and Diffusion
  - Graph Laplacian
- Building Graphs from Data
  - k-d trees and random projection trees
  - LargeVis

Notation

I will be using the following notation throughout these notes.

- $\mathbf{X}$ is a matrix (bold, uppercase)
- $\mathbf{x}_i$ is column i of $\mathbf{X}$, so a column vector (bold, lowercase)
- X_{ij} is the ijth entry of $\mathbf{X}$

People are always talking about matrix rank

- Rank relates to how much unique 'signal' there is the dataset
  - To predict movie preferences, we probably only need to look at a few meaningful genres.
- The rank of a matrix refers to the number of linearly independent rows (columns).
  - This means, for example, that you can't add together columns to produce another column
- The **rank** of the matrix is the number of linearly independent rows (columns)
- A matrix is said to be **full rank** if its rank is $\min\{\text{number of rows}, \text{number of columns}\}$

Example

Is this matrix full rank?

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 1 \\ -2 & -3 & 1 \\ 3 & 3 & 0 \end{bmatrix}$$

This matrix has rank 2 because you can subtract the second column from the first to get the third column.

Example 2

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

Why do we really care about this?

- If you find out that a matrix is not full rank, it can serve as a proxy for estimating ‘information content‘ or unique signal in the data, and allow us to filter useless noise out for improved computational efficiency
- Many methods were designed to exploit low-rank structure of data and are applied in tasks, such as,
  - imputing missing data
  - Denoising data
  - Feature Extraction
- If a suitable low-rank approximation, $(\mathbf{X}_k)$, of $\mathbf{X}$ can be identified, then $\mathbf{X}_k$ can be studied instead of $\mathbf{X}$.

Rank 1 Matrix

Definition: A rank 1 matrix can be expressed as the outer product of two vectors, given as $\mathbf{u}\mathbf{v}^T$, where $\mathbf{u}$ and $\mathbf{v}$ are column vectors. Similarly, a matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ is of rank at most k if it can be written as the sum of k rank-1 matrices,

$$\mathbf{M} = \sum_{j=1}^{k} \mathbf{u}_j \mathbf{v}_j^T \tag{1}$$

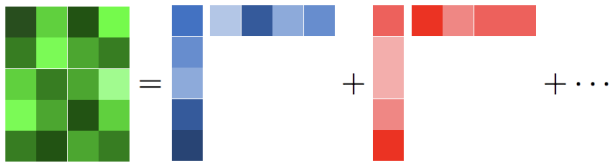


Figure 2.1: Decomposition of a matrix $M \in \mathbb{R}^{5\times4}$ as sum of the rank-1 components. Note that each component is the product of a column vector u_j and a row vector $v_j^\top$.

Figure: From SLMT (Janzamin *et al.*) A matrix can often be determined by a small number of *factors*, where each factor is a rank 1 matrix.

A Rank-2 Matrix Toy Example

$$\begin{array}{r} \\ \text{Alice} \\ \text{Bob} \\ \text{Carol} \\ \text{Dave} \\ \text{Eve} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Math.} & \text{Classics} & \text{Physics} & \text{Music} \end{array} \\ \begin{pmatrix} 19 & 26 & 17 & 21 \\ 8 & 17 & 9 & 12 \\ 7 & 12 & 7 & 9 \\ 15 & 29 & 16 & 21 \\ 31 & 40 & 27 & 33 \end{pmatrix} \end{array} = \begin{array}{c} \text{Verbal} \\ \begin{pmatrix} 4 \\ 3 \\ 2 \\ 5 \\ 6 \end{pmatrix} \end{array} \begin{pmatrix} 1 \\ 5 \\ 2 \\ 3 \end{pmatrix}^\top + \begin{array}{c} \text{Quantitative} \\ \begin{pmatrix} 3 \\ 1 \\ 1 \\ 2 \\ 5 \end{pmatrix} \end{array} \begin{pmatrix} 5 \\ 2 \\ 3 \\ 3 \end{pmatrix}^\top$$

Figure 2.2: Score Matrix M is an example for the scores of students (indexing the rows) in different tests on distinct subjects (indexing the columns). A corresponding low rank decomposition is also provided where the rank is two in this example.

Figure: (From SLMT (Janzamin *et al.*) Write the student $\times$ subject score matrix based on the quantitative and verbal abilities of students and required scores.

Score(student,test)= $\mathit{student}_{\text{verbal ability}} \times \mathit{test}_{\text{verbal}} + \mathit{student}_{\text{quant ability}} \times \mathit{test}_{\text{quant}}$

Relating this to our definition of matrix rank

- Let $\mathbf{u}_{\text{verbal}}$ and $\mathbf{u}_{\text{quant}} \in \mathbb{R}^n$, be the vectors describing verbal and quantitative strengths for each student, respectively
- Let $\mathbf{v}_{\text{verbal}}$ and $\mathbf{v}_{\text{quant}} \in \mathbb{R}^m$ be the vectors describing the requirements for such skills in the verbal and quantitative settings for these tests

So,

$$\mathbf{M} = \mathbf{u}_{\text{verbal}}\mathbf{v}_{\text{verbal}}^T + \mathbf{u}_{\text{quant}}\mathbf{v}_{\text{quant}}^T \tag{2}$$

is a rank 2 matrix.

Matrices in Real Life are Not Exactly Low Rank

Some reasons why a matrix could deviate from low rank structure:

- Noisy entries (common in biology, entries could even be missing)
- The low rank components (factors) may not interact linearly
- Several prominent factors and many small factors.

However, a matrix can still be *approximately* low rank and we can use singular value decomposition (SVD) to find the low rank matrix that is closest, or best approximates the observed matrix.

- If you find yourself fascinated by low rank structure of data matrices, check this out `https://arxiv.org/abs/1705.07474`.

Matrix Norms, for Quantifying when Matrices are Close

Frobenius Norm is the simplest example and is just L_2 norm,

$$||\mathbf{M}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} M_{ij}^2} \tag{3}$$

We seek the matrix $\mathbf{N}$ that makes $||\mathbf{M} - \mathbf{N}||_F$ as small as possible for some particular rank k.

Singular Value Decomposition (SVD)

The SVD of a matrix, $\mathbf{M}$ with $\mathbf{M} \in \mathbb{R}^{n \times m}$ is defined as,

$$\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T = \sum_{j=1}^{\min\{m,n\}} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \tag{4}$$

- $\mathbf{U} = [\mathbf{u_1}|\mathbf{u_2}|....|\mathbf{u_n}] \in \mathbb{R}^{n \times n}$ (left singular vectors),
  $\mathbf{V} = [\mathbf{v_1}|\mathbf{v_2}|....|\mathbf{v_m}] \in \mathbb{R}^{m \times m}$ (right singular vectors).
- $\mathbf{U}$ and $\mathbf{V}$ are orthonormal ($\mathbf{U}^T\mathbf{U} = \mathbb{I}, \mathbf{V}^T\mathbf{V} = \mathbb{I}$).
- $\mathbf{D}$ is a diagonal matrix with diagonal entries
  $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_{\min\{m,n\}} \geq 0$ (σ_is are singular values)

Optimization Point of View of the SVD

We can find the top singular value, σ_1 and the top singular vectors, $\mathbf{u}_1$ and $\mathbf{v}_a$ as,

$$\sigma_1 = \max_{||\mathbf{u}|| \leq \mathbf{1}, ||\mathbf{v}|| \leq \mathbf{1}} \mathbf{u}^T \mathbf{M} \mathbf{v} \tag{5}$$

$$\mathbf{u}_1, \mathbf{v}_1 = \operatorname{argmax}_{||\mathbf{u}|| \leq \mathbf{1}, ||\mathbf{v}|| \leq \mathbf{1}} \mathbf{u}^T \mathbf{M} \mathbf{v} \tag{6}$$

The rest of the singular vectors can be found by maximizing the same form, while constraining singular values to be orthogonal to the previous ones.

$$\sigma_j = \max_{\|\mathbf{u}\| \leq 1, \|\mathbf{v}\| \leq 1, \forall i < j : \mathbf{u} \perp \mathbf{u}_i, \mathbf{v} \perp \mathbf{v}_i} \mathbf{u}^\top \mathbf{M} \mathbf{v},$$

$$\mathbf{u}_j, \mathbf{v}_j = \arg\max_{\|\mathbf{u}\| \leq 1, \|\mathbf{v}\| \leq 1, \forall i < j : \mathbf{u} \perp \mathbf{u}_i, \mathbf{v} \perp \mathbf{v}_i} \mathbf{u}^\top \mathbf{M} \mathbf{v}.$$

In practice...

numpy.linalg.svd

linalg.svd(*a, full_matrices=True, compute_uv=True, hermitian=False*) [source]

Singular Value Decomposition.

When *a* is a 2D array, it is factorized as `u @ np.diag(s) @ vh = (u * s) @ vh`, where *u* and *vh* are 2D unitary arrays and *s* is a 1D array of *a*'s singular values. When *a* is higher-dimensional, SVD is applied in stacked mode as explained below.

Truncated SVD

Here is how you can construct a rank k approximation for your matrix, $\mathbf{M} \in \mathbb{R}^{n \times m}$. Assuming that $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ is the SVD of $\mathbf{M}$, then you can compute the rank k truncated SVD of $\mathbf{M}$ as,

$$\mathbf{M}_{(k)} = \mathbf{U}_{(k)}\mathbf{D}_{(k)}\mathbf{V}_{(k)}^T \tag{7}$$

- entries of $\mathbf{D}$ are sorted in descending order on the diagonal and $\mathbf{D}_{(k)}$ is the first $k \times k$ submatrix of $\mathbf{D}$
- $\mathbf{U}_{(k)} \in \mathbb{R}^{n \times k}$ is the first k columns of $\mathbf{U} \in \mathbb{R}^{n \times n}$
- $\mathbf{V}_{(k)} \in \mathbb{R}^{m \times k}$ is the first k columns of $\mathbf{V} \in \mathbb{R}^{m \times m}$

What Can we Do with That?

You can construct rank k approximations of $\mathbf{M}$ for a bunch of k. You can also sequentially compute $||\mathbf{M} - \mathbf{M}_k||$. At some point, these two matrices will become quite close. If you don't need that high of a k for that to be the case, then the matrix doesn't have as much unique 'signal'.

An Application of SVD in Something You Use All of the Time

The Moore Penrose Pseudo Inverse can approximate the inverse of a rank k matrix, $\mathbf{M}$. Suppose $\mathbf{M} = \mathbf{UDV}^T$ is the rank-k truncated SVD. Then the pseudo inverse of $\mathbf{M}$ is defined as,

$$\mathbf{M}^{-1} = \mathbf{VD}^{-1}\mathbf{U}^T \tag{8}$$

A very exciting application of linear algebra $\rightarrow$ graphs.

Graphs

A graph is a collection of nodes joined by edges. Or generally, a data structure for capturing relational information.

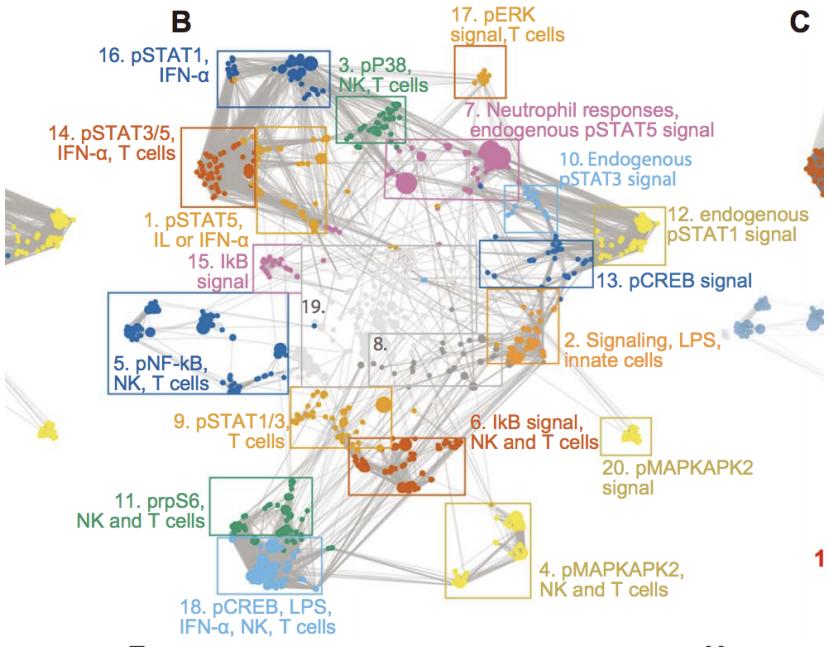


Figure: Figure from Aghapeeour *et al.* Science Immunology. 2018. A graph of function and frequency features immune cell-types (nodes), with edges representing strong correlation across patients.

Mathematical Representations of Graphs

Consider an undirected, unweighted network with N nodes. Then its adjacency matrix ($\mathbf{A}$) is defined as,

$$A_{ij} = \begin{cases} 1 & \text{there is an edge between nodes } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

In practical situations, you will want to use a sparse representation of your adjacency matrix or define an edgelist also known a $|E| \times 2$ matrix of 'source' (column 1) and 'target' (column 2) node indices.

Encoding the Strength of Connection Between Nodes

Sometimes we have a notion about how similar pairs of nodes are. For example, correlation, or another similarity measure. This extra information can be helpful in our analysis and interpretation of graph structure. The adjacency matrix of an undirected, weighted graph with N nodes is similarly defined as,

$$A_{ij} = \begin{cases} w_{ij} & \text{there is an edge between nodes } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

Here w_{ij} represents the strength of the connection between nodes i and j.

- The adjacency matrix can be used to quickly gain intuition about properties of the graph, which we will see as a coming attraction.

Example

Consider the adjacency matrix for an undirected graph with $N = 3$ nodes,

$$\mathbf{A} = \begin{bmatrix} 0 & 2 & 1 \\ 2 & 0 & 0.5 \\ 1 & 0.5 & 0 \end{bmatrix}$$

Here, nodes 1 and 2 have twice as strong of a connection as nodes 1 and 3. This graph is also **complete graph** in that there exists an edge connecting every pair of nodes.

The Simplest Node Ranking Strategy: Degree

Let **A** be the adjacency matrix for a graph with N nodes. The **degree** of a node of node i (k_i) counts the number of nodes that node i is connected to or,

$$k_i = \sum_{j=1}^{N} A_{ij} \tag{9}$$

Every edge has two ends. So, for m edges in a graph, there are $2 \times m$ ends of edges. Therefore, the total sum of edges in the graph can be written as,

$$2m = \sum_{i=1}^{N} k_i \tag{10}$$

Paths in a Graph

- **Path:** A path in a graph is a sequence of nodes such that every consecutive pair of nodes in the sequence is connected by an edge. This is also known as a route across the graph.
- **Path Length:** The path length between nodes i and j is the number of edges that need to be traversed to reach node j from node i.
- **Geodesic Path:** The geodesic path or shortest path is the shortest network distance between a pair of nodes.
  - There are a lot of beautiful shortest path algorithms for doing this efficiently that we will not cover here.

Example

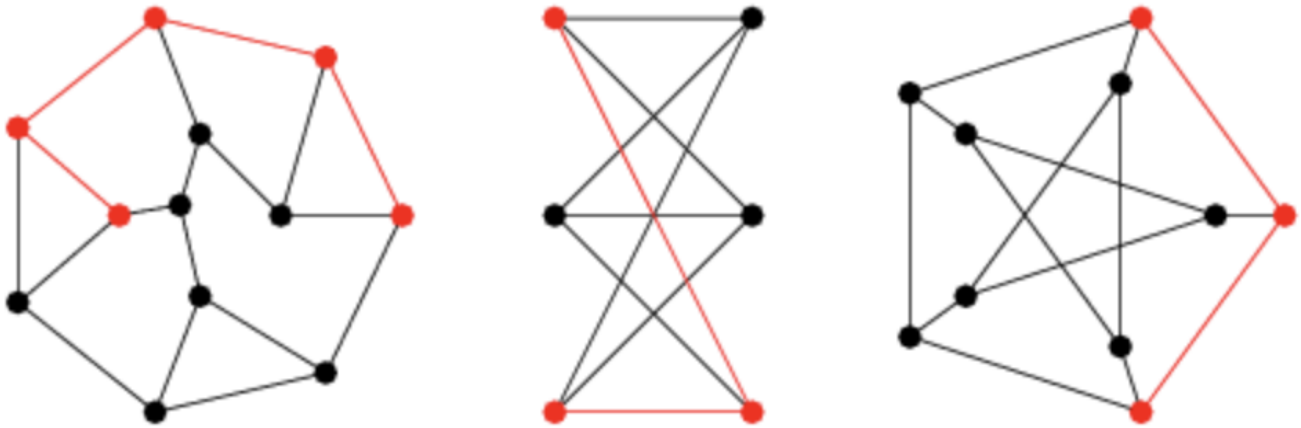


Figure: There could be multiple paths between a pair of nodes, but we are interested in the path that is the shortest.

With linear algebra and graph fundamentals established, we will move to discussion about quantifying structural properties of graphs.

Setup for Graph Diffusion

Diffusion offers the chance to study how some quantity (e.g. signal) relates to connectivity in the graph. Here we assume that some quantity can only flow to adjacent nodes.

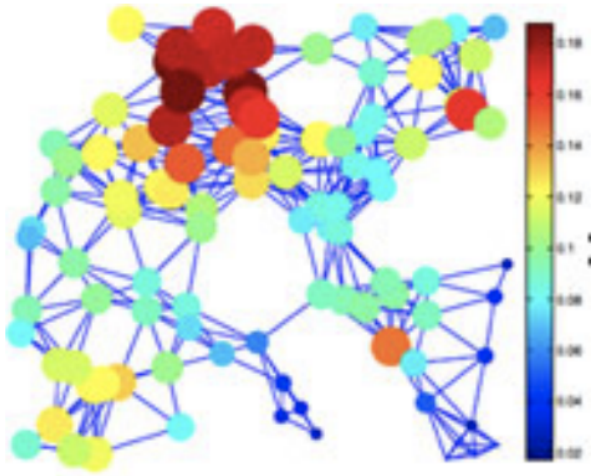


Figure: from `https://web.media.mit.edu/~xdong/paper/tsipn17.pdf`

Setup for Graph Diffusion

Understanding how long this quantity takes to reach a different part of the graph, for example, helps to understand how localized the signal is. So, if the signal is very localized and specific to a particular graph region, it will take longer to flow to another part of the graph.

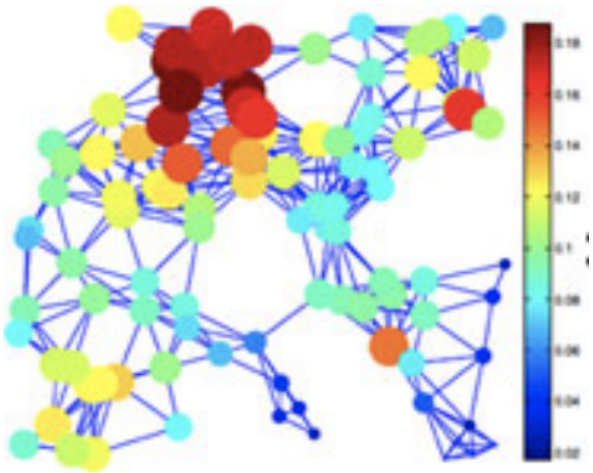


Figure: from `https://web.media.mit.edu/~xdong/paper/tsipn17.pdf`

Diffusion on Graphs

Suppose we have some quantity that will spread or *diffuse* across a graph, given the graph's connectivity. Suppose there is a quantity of ψ_i at node i. Suppose that this quantity flows along the graph from node j to an adjacent node i at a rate of $C(\psi_j - \psi_i)$. The rate at which ψ_i is changing is given as,

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C \sum_j A_{ij}(\psi_j - \psi_i) \tag{11}$$

Here, A_{ij} enforces the idea that the only terms in a sum are pairs of nodes connected by an edge.

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C \sum_j A_{ij}(\psi_j - \psi_i) \tag{12}$$

Splitting up terms, we can rewrite this as,

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C \sum_j A_{ij}\psi_j - C\psi_i \sum_j A_{ij} = C \sum_j A_{ij}\psi_j - C\psi_i k_i \tag{13}$$

- Remember, $k_i = \sum_j A_{ij}$ is simply the degree of node i

Diffusion continued

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j A_{ij}\psi_j - C\psi_i k_i \tag{14}$$

further simplifies to,

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j (A_{ij} - \delta_{ij}k_j)\psi_j. \tag{15}$$

Here, k_i is the degree of node i and $\delta_{ij} = 1$ if $i = j$ and is 0 otherwise.

- That was just some massaging so that everything can be written as a $\sum_j$ to make our next step easier.

Massaging Explained

In case you were wondering how we reworked $C\psi_i k_i$ to be written in terms of $\sum_j$

- k_i is the degree of node i, so, $k_i = \sum_j A_{ij}$
- We only need to count it for node i, so δ_{ij} is our indicator for indexing over j that ensures that we only add a non-zero value when $i = j$

Diffusion and Connection with the Graph Laplacian

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j (A_{ij} - \delta_{ij}k_j)\psi_j. \tag{16}$$

.

- This simplifies because $C\psi_i k_i = C\sum_j \delta_{ij}\psi_j k_j$

This can further can be expressed in matrix form as,

$$\frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} = C(\mathbf{A} - \mathbf{D})\boldsymbol{\psi}. \tag{17}$$

Here, $\mathbf{D}$ is an $N \times N$ diagonal matrix where k_i is the ith diagonal element and represents the degree of node i.

- The **Graph Laplacian** is the matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, and satisfies $\frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} + C\mathbf{L}\boldsymbol{\psi} = 0$ (aka the diffusion equation).

Diffusion on a Bunny Using its Graph Representation

A small amount of signal spreads locally according to specified edges and takes many time steps to spread across the bunny.

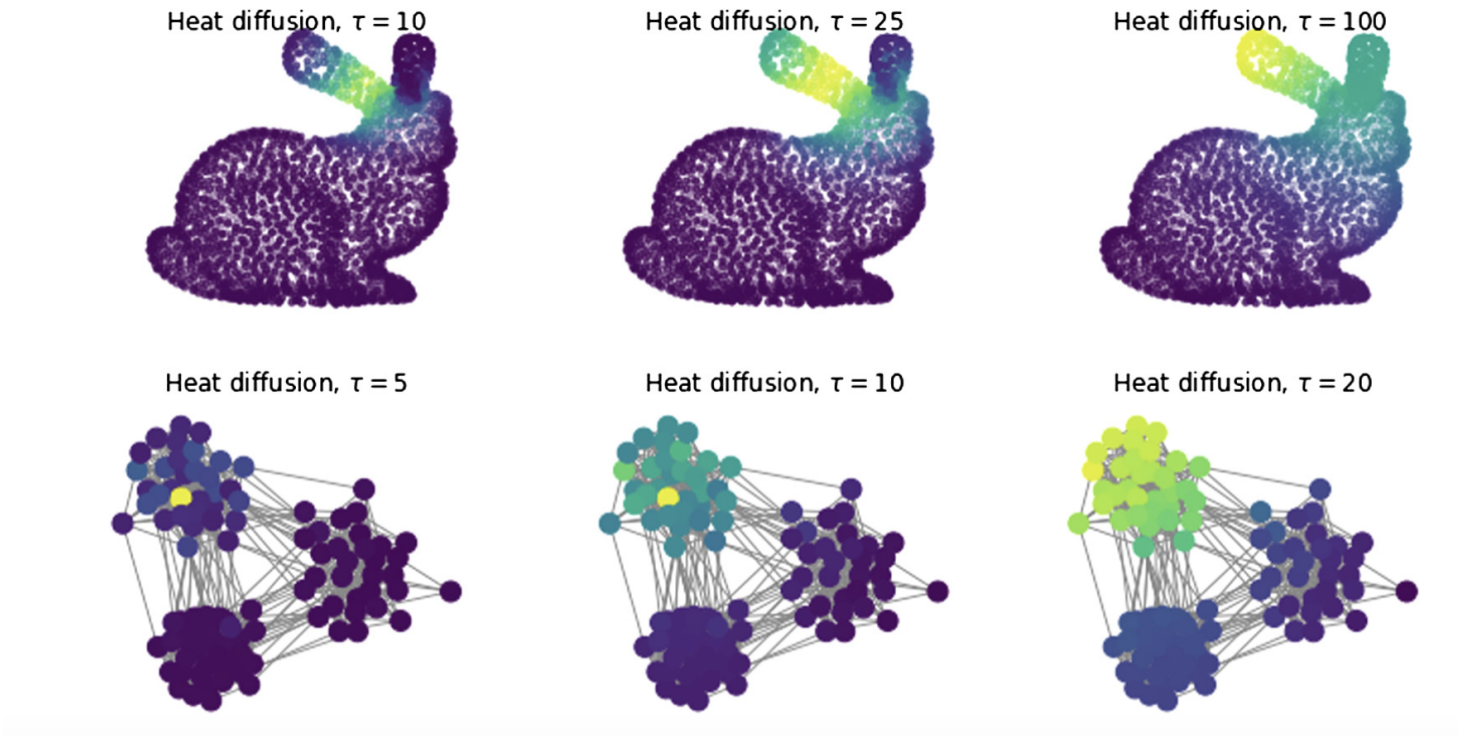


Figure: from `https://www.sciencedirect.com/science/article/pii/S1631070519301094`

Welcome Graph Laplacian

- Besides describing diffusion, the Graph Laplacian has many other nice properties.
- Notably, properties about **connectivity patterns** in the graph and can be useful for **graph partitioning**.
- We are going to be using the graph Laplacian for many tasks!

Laplacian Matrix in More Detail

The elements of the Laplacian matrix are defined as follows for an undirected, unweighted matrix.

- Remember: $\mathbf{L} = \mathbf{D} - \mathbf{A}$

$$L_{ij} = \begin{cases} k_i & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and there is an edge } (i, j) \\ 0 & \text{otherwise} \end{cases}$$

The Graph Laplacian Has Many Nice Linear Algebraic Properties

- In a network with c components, the number of zero eigenvalues is equal to the number of components, (c).

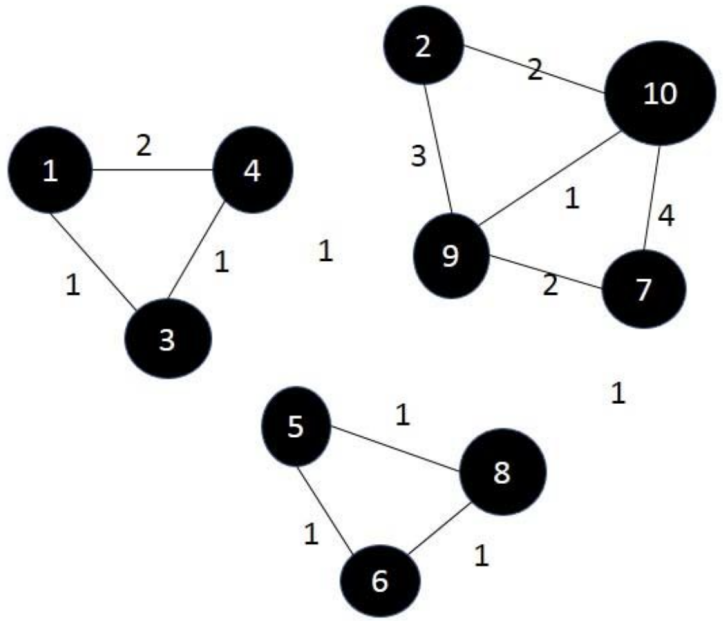


Figure: Each group of nodes makes a component.

Building graphs from data

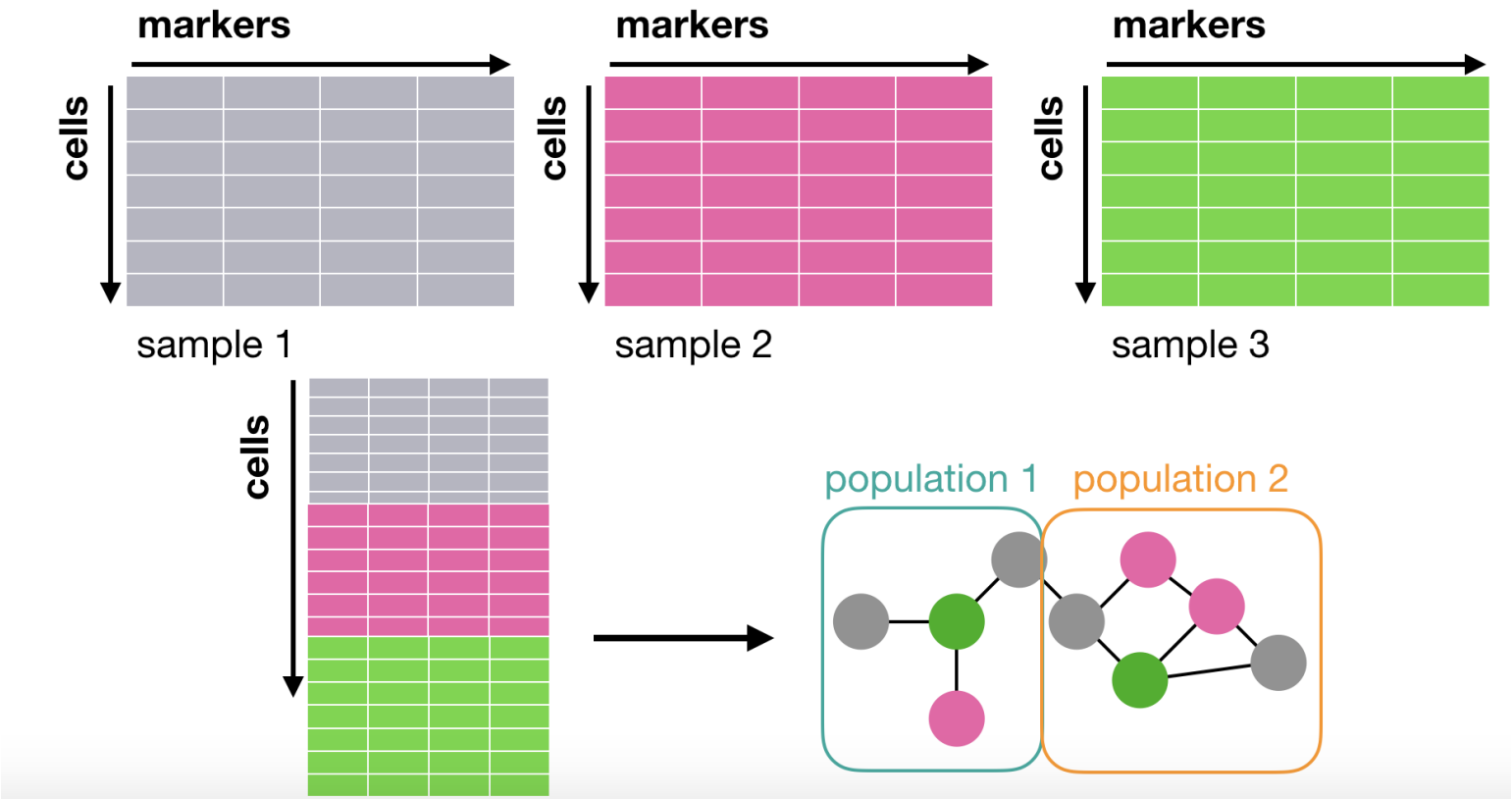


Figure: The task is to compute distances between all pairs of cells, and connect each cell with their nearest neighbors

k-Nearest Neighbor Graphs

- Suppose our dataset, $\mathcal{X}$ is defined as $\mathcal{X} = \{\mathbf{x}_i\}_{i=1\ldots N}, \mathbf{x}_i \in \mathbb{R}^d$.
- The general idea is to connect each node with its k nearest neighbors
  - Compute all pairwise similarities between all pairs of nodes
  - For a node, i, find the k nodes that are closest and draw edges.
- Computing all pairwise distances quickly becomes expensive ($O(N^2 d)$).

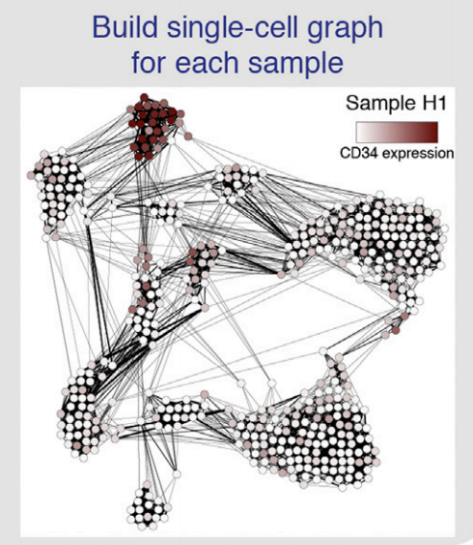


Figure: from Levine *et al.* Cell. 2015. Each cell is connected to its k-nearest neighbors

A Common way to Calculate Edge Weights in a Graph

For better or for worse, people commonly assume a Gaussian distribution on pairwise similarities between nodes. So, a weight between nodes i and j is often computed as,

$$W_{ij} = \exp\left(\frac{-||x_i - x_j||^2}{2\sigma^2}\right) \tag{18}$$

Randomized Trees for Nearest Neighbor Search

People have come up with clever ways to compute k-NN graphs much faster. A popular approach is k-d trees and more recently random projection trees.

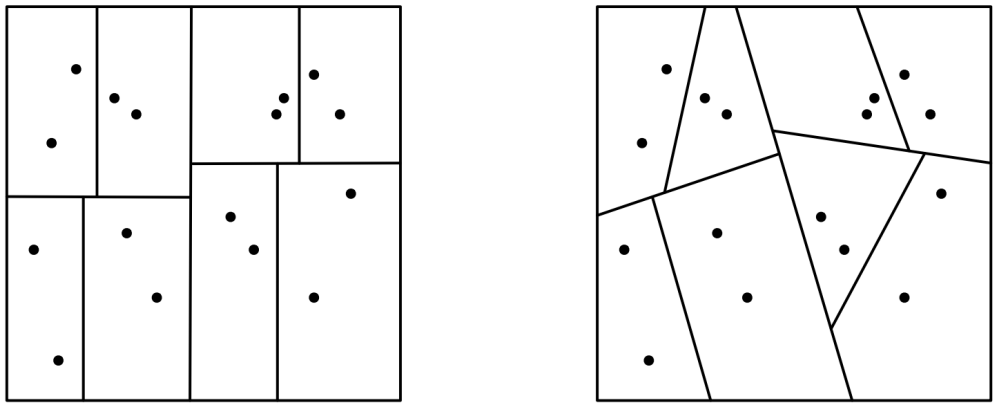


Figure: from Dasgupta *et al.* JMLR. 2013. $k-d$ tree (left), random projection tree (right).

k-d tree

This is a partition of $\mathbb{R}^d$ hyper-rectangular cells based on the datapoints.

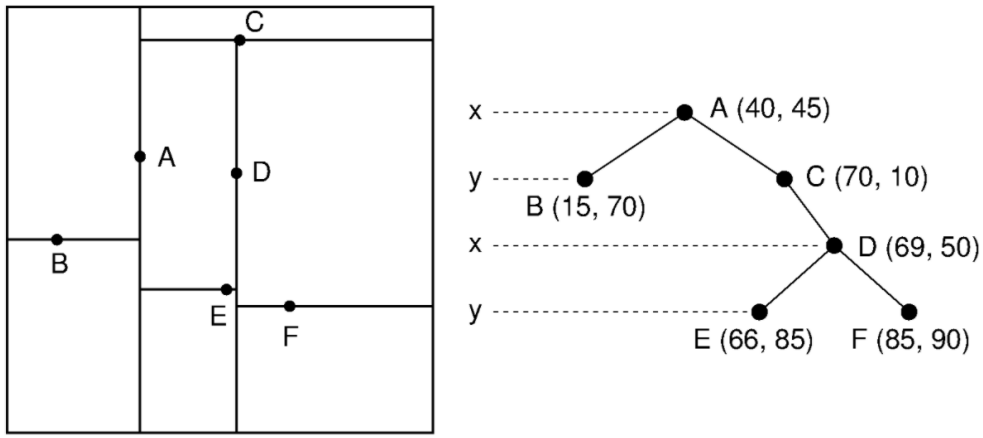


Figure: Alternate between splitting according to the ‘x‘ and ‘y‘ dimensions. First split according to ‘A‘. Based on partition by A, you can split horizontally according to C and B.

Random Projection Trees

Instead of doing axis parallel splits, split directions and split points are randomly selected.

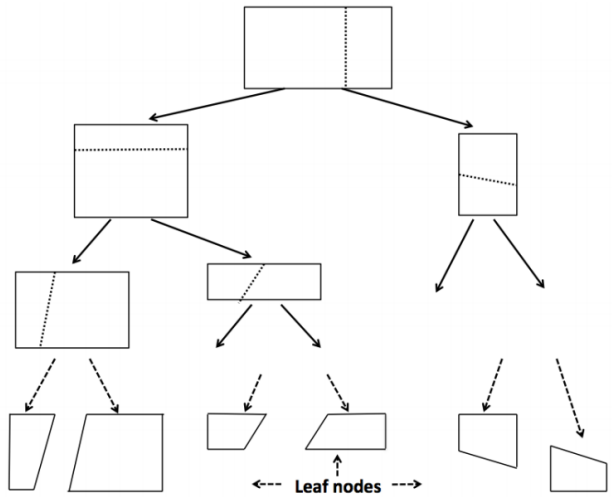
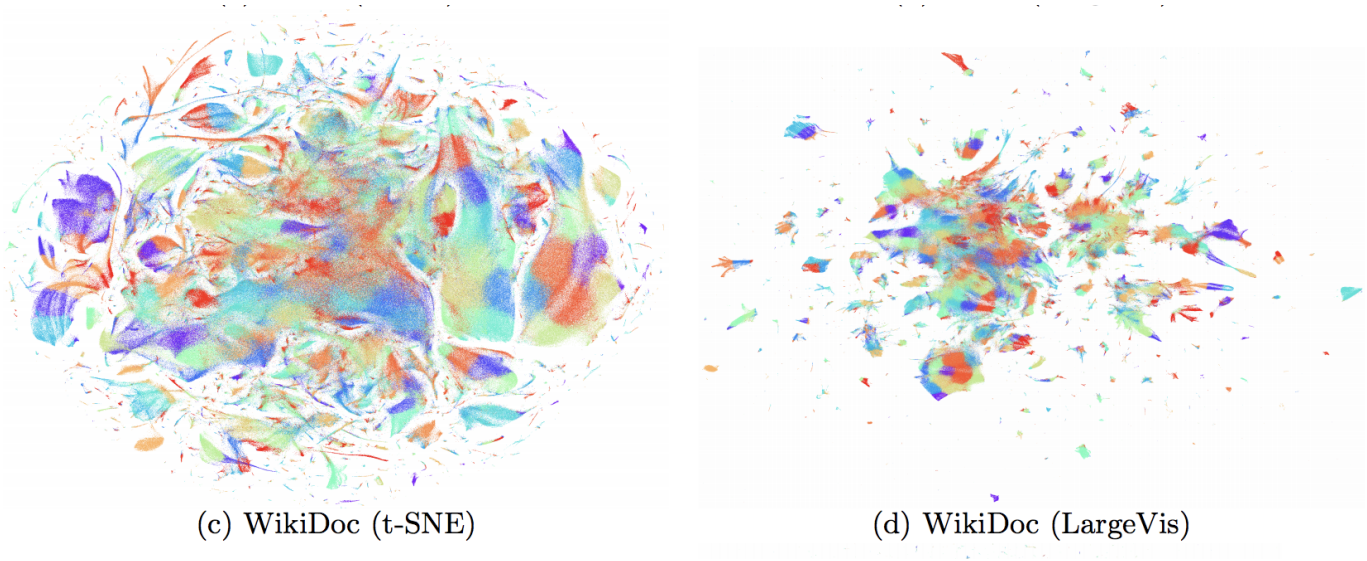


Figure: Random projection tree. (see `https://cseweb.ucsd.edu/~dasgupta/papers/exactnn-colt.pdf` for a discussion on randomized partition trees.)

One Application of k-NN Graph is Visualization (LargeVis)

Using random projection trees, LagreVis creates an embedding for a set of points with a probabilistic model.



(c) WikiDoc (t-SNE)

(d) WikiDoc (LargeVis)

Figure: from Tang *et al.* The Web Conference (WWW). 2016. tSNE and 'string behavior' from being designed to preserve super local similarities.

A Probabilistic Model for Graph Embedding

For some nodes, i and j, let $\mathbf{y}_i$ and $\mathbf{y}_j$ be their embedding coordinates such that $\mathbf{y}_i, \mathbf{y}_j \in \mathbb{R}^d$ (for visualization purposes, $d = 2$). We can model the probability that an edge exists between nodes i and j as a function ($f(\cdot)$) of their learned embedding coordinates, or,

$$P(e_{ij} = 1) = f(||\mathbf{y}_i - \mathbf{y}_j||) \tag{19}$$

- An important criterion to be satisfied is that $f(\cdot)$ must ensure a high probability when i and j are close, and low otherwise.
- For example: $f(x) = \frac{1}{1+ax^2}$

Writing Down the Likelihood of the Observed Graph

The authors define a similarity measure between pairs of nodes where there is an edge, based on the original data, $\mathbf{x}_i$ and $\mathbf{x}_j$. Here, γ being a constant fixed weight for negative edges.
The likelihood of the graph can be written as,

$$O = \prod_{(i,j)\in E} P(e_{ij} = 1) \prod_{(i,j)\in \tilde{E}} (1 - P(e_{ij} = 1))^{\gamma}$$
$$\propto \sum_{(i,j)\in E} \log P(e_{ij} = 1) + \sum_{(i,j)\in \tilde{E}} \gamma \log(1 - P(e_{ij} = 1))$$

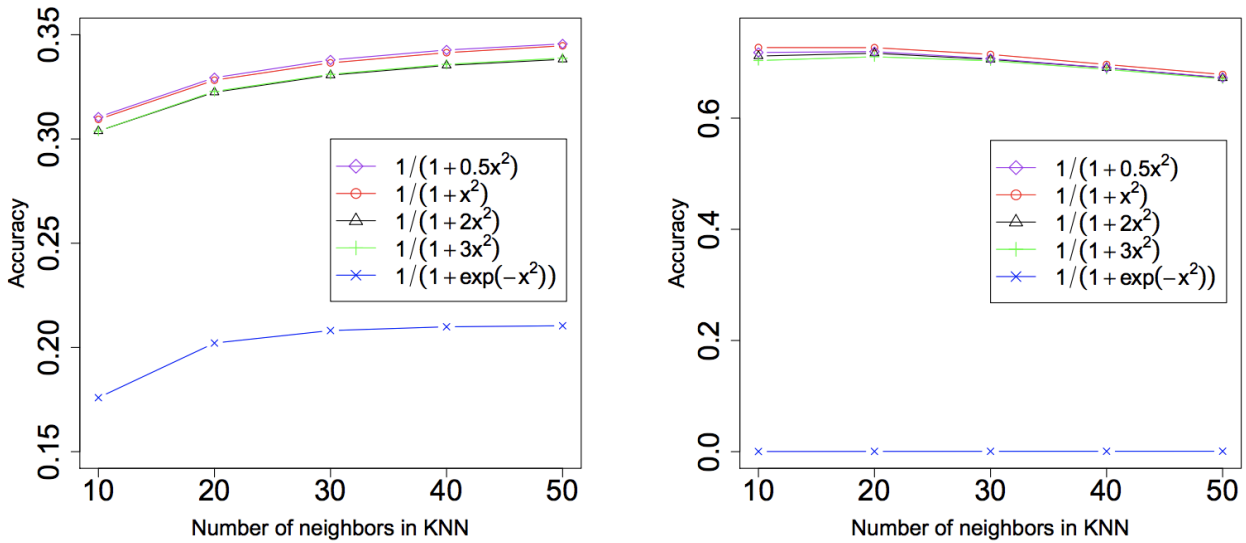
$\tilde{E}$ represents pairs of nodes with no edge.

Brief Overview of Optimization of Parameters

- Since these graphs are very sparse, there are a lot of 'negative edges' or edges $\in \tilde{E}$. So the authors used a smart negative sampling approach to only consider a subset of negative node pairs
  - For each node, i, consider sample some 'negative' pair nodes.
  - A node, j from the negative pair set to pair with i as a negative example is sampled with probability $\propto d_j^{0.75}$. Here d_j is the degree of node j.
  - Optimize parameters with asynchronous stochastic gradient descent (`https://papers.nips.cc/paper/2011/file/218a0aefd1d1a4be65601cc6ddc1520e-Paper.pdf`)

Choice of $f(\cdot)$

The authors tested how choice of $f(\cdot)$ affected the quality of their embedding. Looks like there is one bad choice, but overall, does not make a huge difference.



(a) WikiDoc (b) LiveJournal

Figure: from Tang *et al.* The Web Conference (WWW). 2016.

More Practical Reasons to Use LargeVis: Scales better than tSNE

Nothing would ever be as fast as PCA. But, if you need to understand more local types of similarities between data points, perhaps it's worth the wait....

Table 2: Comparison of running time (hours) in graph visualization between the t-SNE and LargeVis.

Algorithm	20NG	MNIST	WikiWord	WikiDoc	LiveJournal	CSAuthor	DBLPPaper
t-SNE	0.12	0.41	9.82	45.01	70.35	28.33	18.73
LargeVis	0.14	0.23	2.01	5.60	9.26	4.24	3.19
Speedup Rate	0	0.7	3.9	7	6.6	5.7	4.9

Figure: from Tang *et al.* The Web Conference (WWW). 2016. Comparison in run-time between tSNE and LargeVis.

Recap

- Matrix Rank and SVD
- Graphs: Properties, Laplacian
- LargeVis: An example of using a graph-based representation of data that was computed efficiently for visualization.

Next time:
Graph partitioning and ranking nodes (in a way that is more sophisticated than just node degree....)