CS 540 Introduction to Artificial Intelligence
Unsupervised Learning II

Yingyu Liang
University of Wisconsin-Madison

Oct 7, 2021

Based on slides by Fred Sala

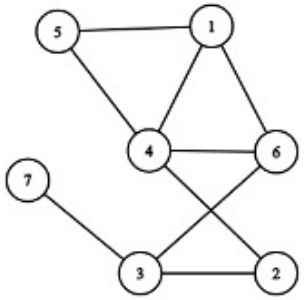
Outline

- Other Types of Clustering
  - Graph-based, cuts, spectral clustering
- Unsupervised Learning: Dim Reduction/Visualization
  - t-SNE, algorithm, example, vs. PCA
- Unsupervised Learning: Density Estimation
  - Kernel density estimation: high-level intro

Graph-Based Clustering

Graph-based/proximity-based

- Recall: Graph G = (V,E) has vertex set V, edge set E.
  - Edges can be weighted or unweighted
  - Encode **similarity**
- Don't need vectors here
  - Just edges (and maybe weights)

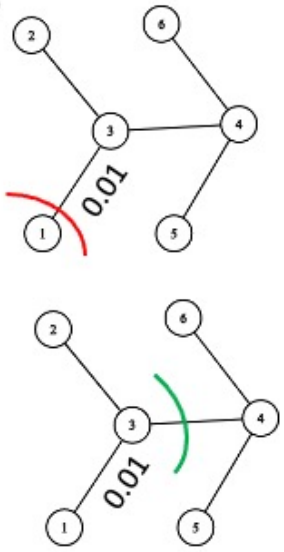


The weights on the edges are supposed to encode similarities, ie, larger weights mean more similarity between the two end points.

Graph-Based Clustering

Want: partition V into V_1 and V_2

- Implies a graph "cut"
- One idea: minimize the **weight** of the cut
  - Downside: might just cut of one node
  - Need: "**balanced**" cut



Consider 2-clustering: partition the vertices/nodes into two clusters.

The naïve idea of minimizing the weight of the cut (i.e., the sum of all the edges across the two clusters) has a drawback: unbalanced clusters. Typically, one cluster is very small (like only one node).

Partition-Based Clustering

Want: partition V into V_1 and V_2

- Just minimizing weight isn't good… want **balance!**
- **Approaches:**

$$\overline{\mathrm{Cut}}(V_1, V_2) = \frac{\mathrm{Cut}(V_1, V_2)}{|V_1|} + \frac{\mathrm{Cut}(V_1, V_2)}{|V_2|}$$

$$\mathrm{NCut}(V_1, V_2) = \frac{\mathrm{Cut}(V_1, V_2)}{\sum_{i \in V_1} d_i} + \frac{\mathrm{Cut}(V_1, V_2)}{\sum_{i \in V_2} d_i}$$

Sum of edge weights at vertex

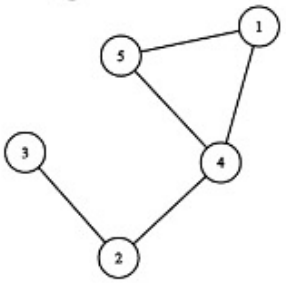
Normalize the weight of the cut Cut(V1,V2):

1. In \bar{Cut}, we normalize it by the number of nodes in each cluster.
2. In NCut, we normalize it by the sum of the degrees of the nodes in each cluster. (If the edges are weighted, then normalize it by the sum of the edge weights of the nodes in each cluster, i.e., d_i is the sum of the weights of the edges connecting to node i)

Partition-Based Clustering

How do we compute these?

- Hard problem → heuristics
  - Greedy algorithm
  - "Spectral" approaches
- Spectral clustering approach:
  - **Adjacency** matrix

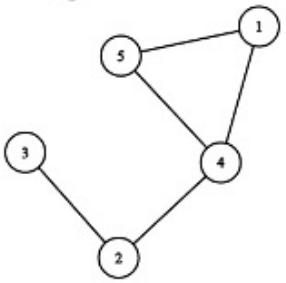


$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Those objectives are hard to optimize. Some greedy algorithms for those objectives eventually lead to the spectral approach. (We don't require to know how to derive the spectral approach.)

Partition-Based Clustering

- Spectral clustering approach:
  - **Adjacency** matrix
  - **Degree** matrix



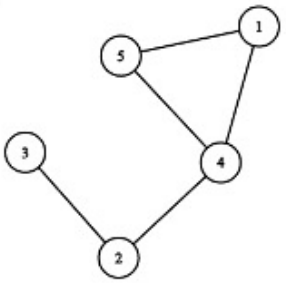
$$D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Spectral clustering:
The adjacency matrix: a matrix where the (i,j)-th entry is 1 iff node i is connected to node j. (If edges have weights, then the entry is the weight of that edge.)
The degree matrix: a diagonal matrix where the i-th diagonal entry is the degree of the node i. (If edges have weights, then the entry is the sum of the weights of the edges connected to node i.)

Spectral Clustering

- Spectral clustering approach:
  - 1. Compute **Laplacian L = D – A**

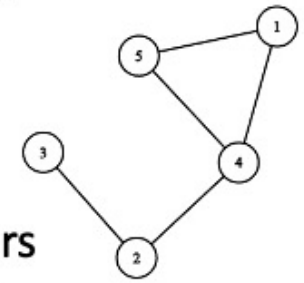
  (Important tool in graph theory)



$$L = \underbrace{\begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}}_{\text{Degree Matrix}} - \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Adjacency Matrix}} = \underbrace{\begin{bmatrix} 2 & 0 & 0 & -1 & -1 \\ 0 & 2 & -1 & -1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 3 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{bmatrix}}_{\text{Laplacian}}$$

Spectral Clustering

- Spectral clustering approach:
  - 1. Compute **Laplacian L = D – A**
  - 2. Compute *k* **smallest** eigenvectors
  - 3. Set *U* to be the *n* x *k* matrix with u_1, …, u_k as columns. Take the *n* rows formed as points
  - 4. Run k-means on the representations



In step 3: u_1 … u_k denote the k smallest eigenvectors of the Laplacian. The n rows are sometimes called the spectral embeddings of the nodes.

Spectral Clustering

- Compare/contrast to **PCA**:
  - Use an **eigendecomposition /** dimensionality reduction
    - But, run on Laplacian (not covariance); use smallest eigenvectors, not largest
- Intuition: Laplacian encodes structure information
  - "Lower" eigenvectors give partitioning information

Intuition: The Laplacian encodes the structure of the graph. In particular, the lower eigenvector can be viewed as a vector whose dimensions correspond to the nodes, and it will have similar values for similar nodes.

Extreme case: suppose the graph has two disconnected components, and each component is a complete graph (each node in the component is connected to all other nodes in the component). Then we have two lower eigenvectors:

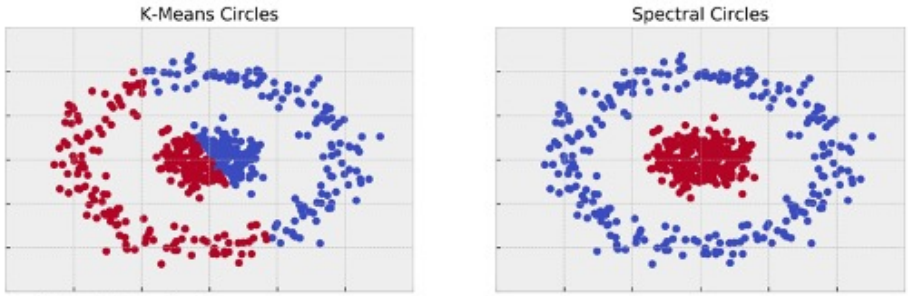
1. One is proportional to the indicator vector of the first component, i.e., a vector with value 1 on the dimensions corresponding to nodes in the first component and value 0 on the other dimensions.
2. The other is proportional to the indicator vector of the second component.

Then the spectral embeddings are indicator vectors of the components: all nodes in the first component correspond to a point [1, 0], and all nodes in the second component correspond to a point [0, 1]. Then 2-means on these n points leads to the desired clustering.

Spectral Clustering

Q: Why do this?

- 1. No need for points or distances as input
- 2. Can handle intuitive separation (k-means can't!)



Credit: William Fleshman

Break & Quiz

Q 1.1: We have two datasets: a social network dataset S_1 which shows which individuals are friends with each other along with image dataset S_2.

What kind of clustering can we do? Assume we do not make additional data transformations.

- A. k-means on both S_1 and S_2
- B. graph-based on S_1 and k-means on S_2
- C. k-means on S_1 and graph-based on S_2
- D. hierarchical on S_1 and graph-based on S_2

Break & Quiz

Q 1.1: We have two datasets: a social network dataset S_1 which shows which individuals are friends with each other along with image dataset S_2.

What kind of clustering can we do? Assume we do not make additional data transformations.

- A. k-means on both S_1 and S_2
- **B. graph-based on S_1 and k-means on S_2**
- C. k-means on S_1 and graph-based on S_2
- D. hierarchical on S_1 and graph-based on S_2

Break & Quiz

Q 1.1: We have two datasets: a social network dataset S_1 which shows which individuals are friends with each other along with image dataset S_2.

What kind of clustering can we do? Assume we do not make additional data transformations.

- A. k-means on both S_1 and S_2 **(No: can't do k-means on graph)**
- **B. graph-based on S_1 and k-means on S_2**
- C. k-means on S_1 and graph-based on S **(Same as A)**
- D. hierarchical on S_1 and graph-based on S_2 **(No: S_2 is not a graph)**

Break & Quiz

Q 1.2: The CIFAR-10 dataset contains 32x32 images labeled with one of 10 classes. What could we use it for?

(i) Supervised learning (ii) PCA (iii) k-means clustering

- A. Only (i)
- B. Only (ii) and (iii)
- C. Only (i) and (ii)
- D. All of them

Break & Quiz

Q 1.2: The CIFAR-10 dataset contains 32x32 images labeled with one of 10 classes. What could we use it for?

(i) Supervised learning (ii) PCA (iii) k-means clustering

- A. Only (i)
- B. Only (ii) and (iii)
- C. Only (i) and (ii)
- **D. All of them**

Break & Quiz

Q 1.2: The CIFAR-10 dataset contains 32x32 images labeled with one of 10 classes. What could we use it for?

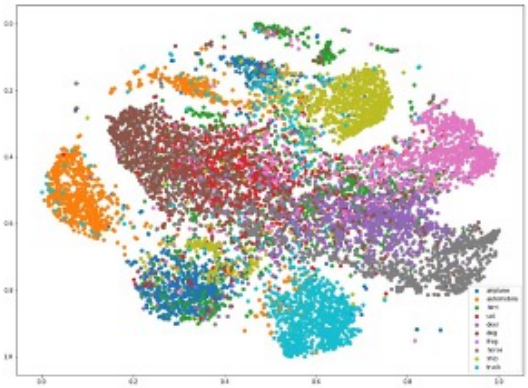
(i) Supervised learning (ii) PCA (iii) k-means clustering

- (i) **Yes: train an image classifier; have labels)**
- (ii) **Yes: run PCA on image vectors to reduce dimensionality**
- (iii) **Yes: can cluster image vectors with k-means**
- **D. All of them**

Unsupervised Learning Beyond Clustering

Data analysis, dimensionality reduction, etc

- Already talked about PCA
- Note: PCA can be used for visualization, but not specifically designed for it
- Some algorithms **specifically** for visualization



Philip Slingerland

We can use PCA to get 2-dim representation and then visualize them

Dimensionality Reduction & Visualization

Typical dataset: MNIST

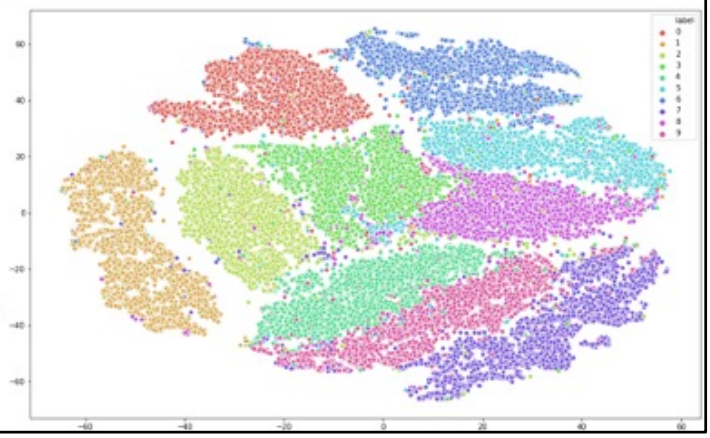
- Handwritten digits 0-9
  - 60,000 images (small by ML standards)
  - 28×28 pixel (784 dimensions)
  - Standard for image experiments

- Dimensionality reduction?

Visualization: **T-SNE**

Typical dataset: MNIST

- **T-SNE**: project data into just 2 dimensions
- Try to maintain structure



- MNIST Example
- **Input**: $x_1, x_2, \ldots, x_n$
- **Output**: 2D/3D $y_1, y_2, \ldots, y_n$

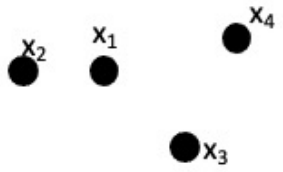
T-SNE stands for t-Distributed Stochastic Neighbor Embedding

It's designed for visualizing high dimensional data while preserving neighboring information

T-SNE Algorithm: Step 1

How does it work? Two steps

- **1.** Turn vectors into probability pairs
- **2**. Turn pairs back into **(lower-dim)** vectors



Step 1:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \quad p_{ij} = \frac{1}{2n}(p_{j|i} + p_{i|j})$$

Intuition: probability that x_i would pick x_j as its neighbor under a Gaussian probability

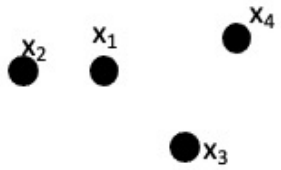
High level intuition: get low-dim vectors whose neighboring probability distributions are similar to those of the original vectors.

Step 1: compute the neighboring probability distributions of the original data vectors. Denote them as p_{ij}

T-SNE Algorithm: Step 2

How does it work? Two steps

- **1.** Turn vectors into probability pairs
- **2**. Turn pairs back into **(lower-dim)** vectors



Step 2: set $$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq \ell}(1 + \|y_k - y_\ell\|^2)^{-1}}$$

and minimize $$\sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$ ← KL Divergence between p and q

Step 2: compute the neighboring probability distributions of the lower-dim vectors. Denote them as q_{ij}. Note that q is using a different form from p.

Then find the set of lower-dim vectors that minimize the KL-divergence between p and q. Recall that KL-divergence is some dissimilarity metric between two distributions.

T-SNE Algorithm: Step 2

More on step 2:

- We have two distributions *p*, *q*. *p* is fixed
- *q* is a function of the y_i which we move around
- Move y_i around until the KL divergence is small
  - So we have a good representation!
- **Optimizing a loss function**---we'll see more in supervised learning.

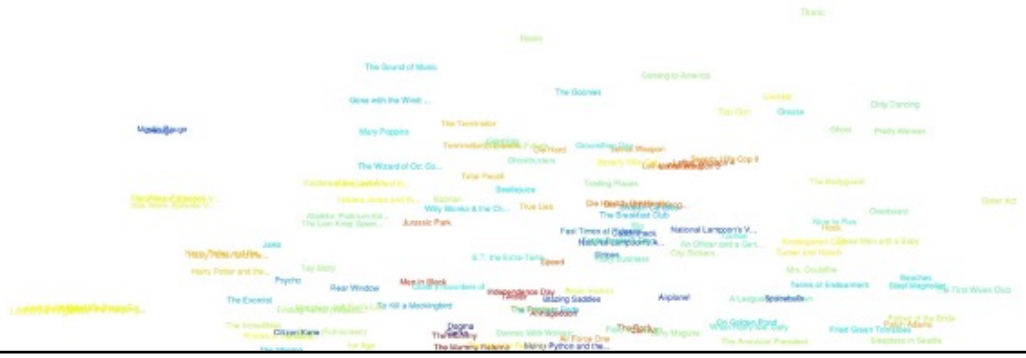
$$\sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

KL Divergence between p and q

Note that the only unknown variables in the KL-divergence are the set of lower-dim vectors. So we can view the KL-divergence as a quality measurement of the set of lower-dim vectors, and we would like to find the set with the smallest KL-divergence.

T-SNE Examples

- Examples: (from Laurens van der Maaten)
- **Movies**: https://lvdmaaten.github.io/tsne/examples/netflix_tsne.jpg



T-SNE Examples

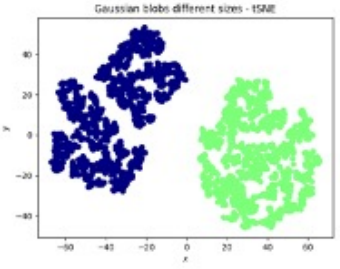
- Examples: (from Laurens van der Maaten)
- **NORB**:
  https://lvdmaaten.github.io/tsne/examples/norb_tsne.jpg

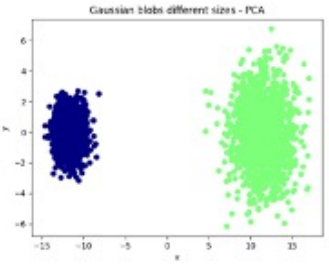
Visualization: **T-SNE**

t-SNE vs PCA?

- "Local" vs "Global"
- Lose information in t-SNE
  - not a bad thing necessarily
- Downstream use

Good resource/credit:
https://www.thekerneltrip.com/statistics/tsne-vs-pca/





T-SNE: try to preserve the neighboring information, which is local
PCA: try to preserve the variance, which is global.

Both can lose information.

Break & Quiz

Q 2.1: Can we do t-SNE on NLP (words) or graph datasets?

- A. Never
- B. Yes, after running PCA on them
- C. Yes, after mapping them into R^d (ie, embedding)
- D. Yes, after running hierarchical clustering on them

Break & Quiz

Q 2.1: Can we do t-SNE on NLP (words) or graph datasets?

- A. Never
- B. Yes, after running PCA on them
- **C. Yes, after mapping them into R^d (ie, embedding)**
- D. Yes, after running hierarchical clustering on them

Break & Quiz

Q 2.1: Can we do t-SNE on NLP (words) or graph datasets?

- A. Never **(No: too strong)**
- B. Yes, after running PCA on them **(No: can't run PCA on words or graphs directly. Need vectors)**
- **C. Yes, after mapping them into R^d (ie, embedding)**
- D. Yes, after running hierarchical clustering on them **(No: hierarchical clustering gives us a graph)**

Short Intro to Density Estimation

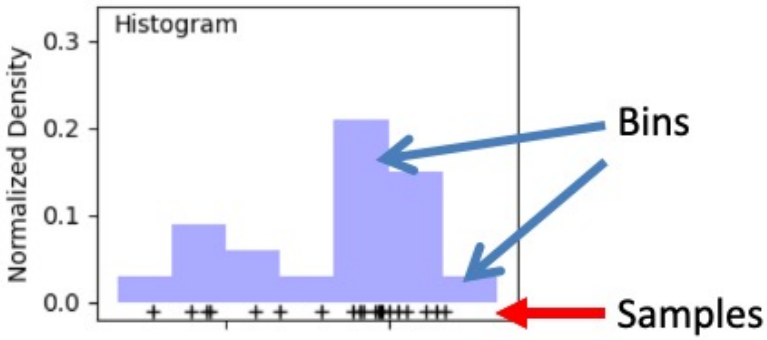
Goal: given samples $x_1, \ldots, x_n$ from some distribution P, estimate P.

- Compute statistics (mean, variance)
- Generate samples from P
- Run inference

Zach Monge

Simplest Idea: Histograms

Goal: given samples $x_1, \ldots, x_n$ from some distribution P, estimate P.



Define bins; count # of samples in each bin, normalize

If we know P is from certain family of distributions with parameters (e.g., Gaussians), then we can try to estimate the parameters.

If not, then we use nonparametric methods. The simplest one is histogram (essentially using frequency to estimate the probability).

Simplest Idea: Histograms

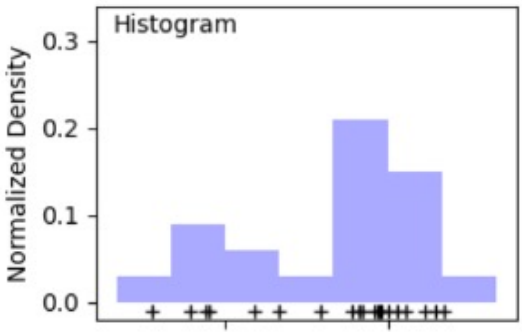
Goal: given samples $x_1, \ldots, x_n$ from some distribution P, estimate P.

Downsides:

i) High-dimensions: most bins empty

ii) Not continuous

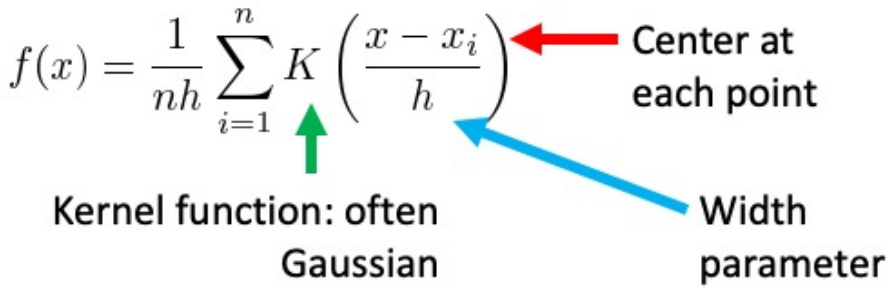
iii) How to choose bins?



Kernel Density Estimation

Goal: given samples $x_1, \ldots, x_n$ from some distribution P, estimate P.

Idea: represent density as combination of "kernels"



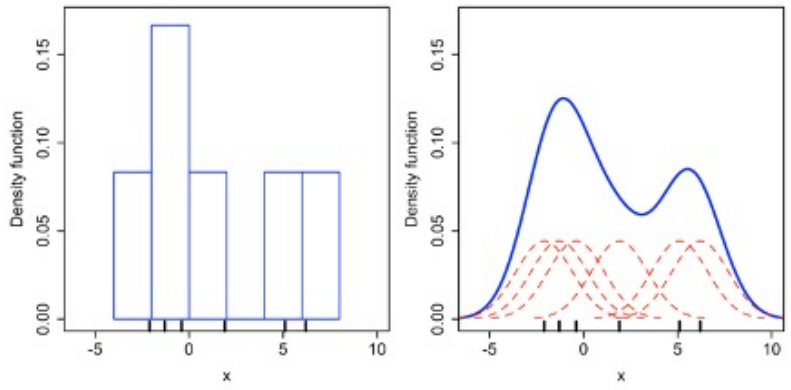
Over continuous space: we can estimate P using the combination of some special functions (kernel function).

K is chosen so that f is a density (ie. the integral of f over the whole input space is 1). Typical choice: the RBF kernel (Gaussian density function) K(x) = \frac{1}{\sqrt{2\pi}} exp(-x^2/2)

Kernel Density Estimation

Idea: represent density as combination of kernels

- "Smooth" out the histogram



In the histogram method, for a data point x_i, it puts all probability mass in the bin (the neighborhood of the data point) and puts 0 outside the bin.

Kernel puts a large fraction of probability mass in the neighborhood of the data point x_i, but also puts some far from the neighborhood. This gives a smooth version of the histogram method.