Data Clustering (Contd)

CS771: Introduction to Machine Learning

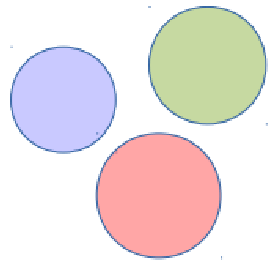
Piyush Rai

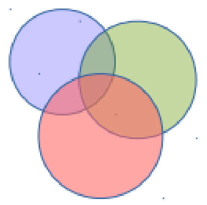
Plan

- K-means extensions
  - Soft clustering
  - Kernel K-means
- A few other popular clustering algorithms
  - Hierarchical Clustering
    - Agglomerative Clustering
    - Divisive Clustering
  - Graph Clustering
    - Spectral Clustering
  - Density-based clustering
    - DBSCAN
- Basic idea of probabilistic clustering methods, such as Gaussian mixture models (details when we talk about latent variable models)

K-means: Hard vs Soft Clustering

- K-means makes hard assignments of points to clusters
  - Hard assignment: A point either completely belongs to a cluster or doesn't belong at all


Hard-assignment okay


Hard-assignment tricky

A more principled extension of K-means for doing soft-clustering is via probabilistic mixture models such as the Gaussian Mixture Model

  - When clusters overlap, soft assignment is preferable (i.e., probability of being assigned to each cluster: say $K = 3$ and for some point $\boldsymbol{x}_n$, $p_1 = 0.7, p_2 = 0.2, p_3 = 0.1$)
- A heuristic to get soft assignments: Transform distances from clusters into prob.

$$\sum_{k=1}^{K} \gamma_{nk} = 1$$

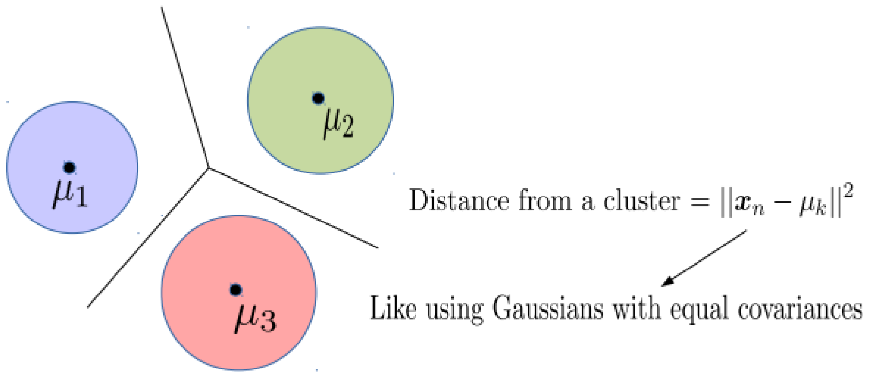
$$\gamma_{nk} = \frac{\exp(-||\boldsymbol{x}_n - \mu_k||^2)}{\sum_{\ell=1}^{K} \exp(-||\boldsymbol{x}_n - \mu_\ell||^2)} \quad \text{(prob. that } \boldsymbol{x}_n \text{ belongs to cluster } k\text{)}$$

- Cluster mean updates also change: $\mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma_{nk}}$ (all points contribute, fractionally)

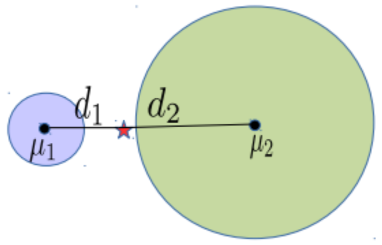
K-means: Decision Boundaries and Cluster Sizes/Shapes

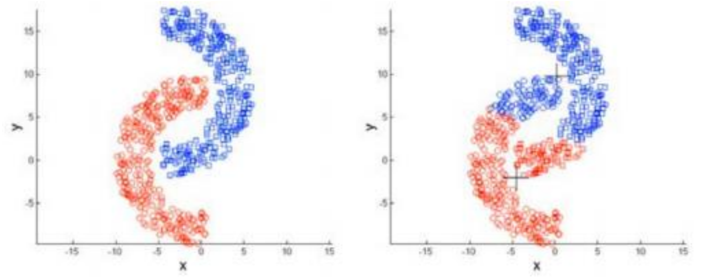
- K-mean assumes that the decision boundary between any two clusters is linear
- Reason: The K-means loss function implies assumes equal-sized, spherical clusters

Reason: Use of Euclidean distances


μ_1
μ_2
μ_3
Distance from a cluster = $||\boldsymbol{x}_n - \mu_k||^2$
Like using Gaussians with equal covariances

- May do badly if clusters are not roughly equi-sized and convex-shaped





Kernel K-means

Helps learn non-spherical clusters and nonlinear cluster boundaries

- Basic idea: Replace the Eucl. distances in K-means by the kernelized versions

$$||\phi(\boldsymbol{x}_n)-\phi(\boldsymbol{\mu}_k)||^2 = ||\phi(\boldsymbol{x}_n)||^2+||\phi(\boldsymbol{\mu}_k)||^2-2\phi(\boldsymbol{x}_n)^\top\phi(\boldsymbol{\mu}_k)$$
$$= k(\boldsymbol{x}_n,\boldsymbol{x}_n)+k(\boldsymbol{\mu}_k,\boldsymbol{\mu}_k)-2k(\boldsymbol{x}_n,\boldsymbol{\mu}_k)$$

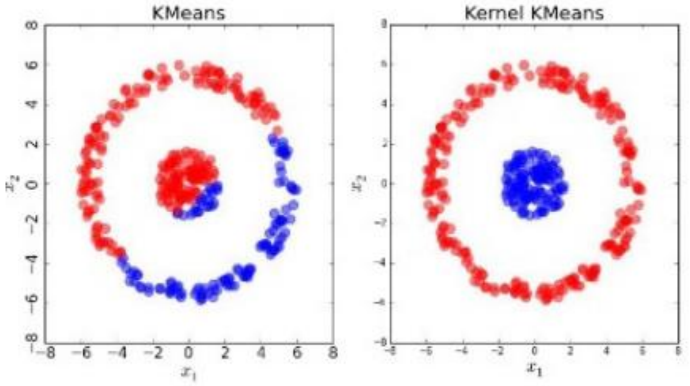
Kernelized distance between input $\boldsymbol{x}_n$ and mean of cluster k

- Here $k(.,.)$ denotes the kernel function and ϕ is its (implicit) feature map
- Note: $\phi(\mu_k)$ is the mean of ϕ mappings of the data points assigned to cluster k

Not the same as the ϕ mapping of the mean of the data points assigned to cluster k

$$\phi(\mu_k)=\frac{1}{|\mathcal{C}_k|}\sum_{n:z_n=k}\phi(\boldsymbol{x}_n)$$

Can also used landmarks or kernel random features idea to get new features and run standard k-means on those



Note: Apart from kernels, it is also possible to use other distance functions in K-means. Bregman Divergence* is such a family of distances (Euclidean and Mahalanobis are special cases)

*Clustering with Bregman Divergences (Banerjee et al, 2005)

Hierarchical Clustering

Similarity between two clusters (or two set of points) is needed in HC algos (e.g., this can be average pairwise similarity between the inputs in the two clusters)

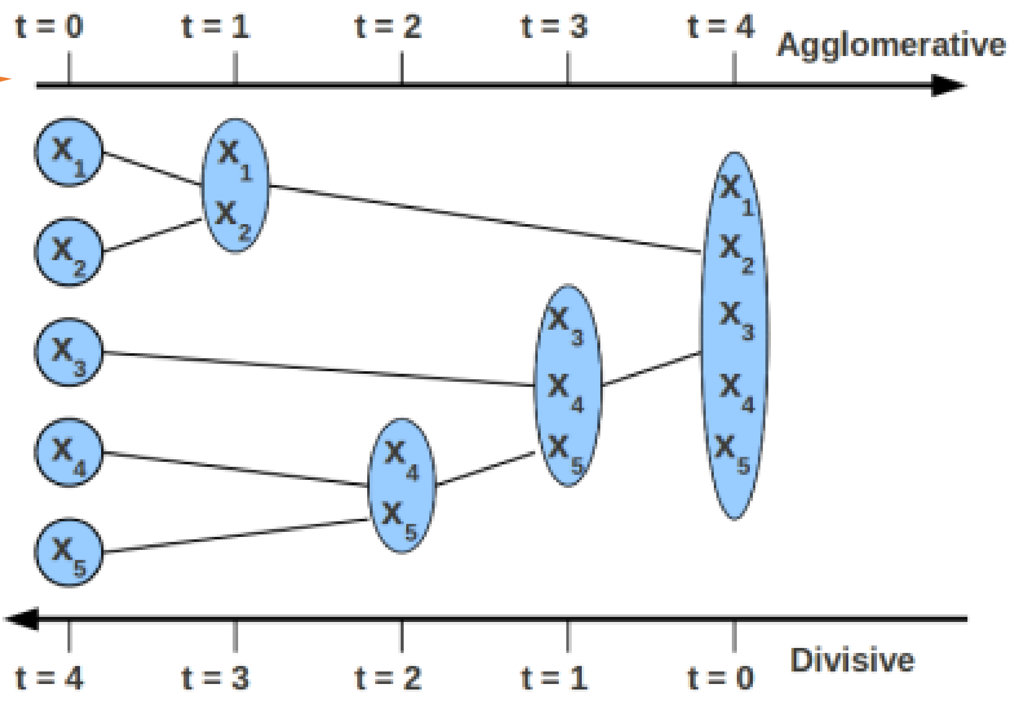
- Can be done in two ways: Agglomerative or Divisive

Agglomerative: Start with each point being in a singleton cluster

At each step, greedily merge two most "similar" sub-clusters

Stop when there is a single cluster containing all the points

Learns a dendrogram-like structure with inputs at the leaf nodes. Can then choose how many clusters we want



Keep recursing until the desired number of clusters found

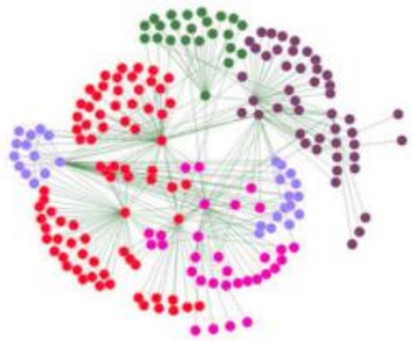
At each step, break a cluster into (at least) two smaller homogeneous sub-clusters

Divisive: Start with all points being in a single cluster

Tricky because no labels (unlike Decision Trees)

- Agglomerative is more popular and simpler than divisive (the latter usually needs complicated heuristics to decide cluster splitting).
- Neither uses any loss function

Graph Clustering



- Often the data is given in form of a graph, not feat. vec.
  - Usually in form of a pairwise similarity matrix $\boldsymbol{A}$ of size $N \times N$
  - $A_{nm} \geq 0$ is assumed to be the similarity between two nodes/inputs with indices n and m
- Examples: Social networks and various interaction networks
- Goal is to cluster the nodes/inputs into K clusters (flat partitioning)
- One scheme is to somehow get an <u>embedding</u> of the graph nodes to get <u>feature vector</u> for each node and run K-means or kernel K-means or any other clustering algo
- Another way is to perform direct graph clustering
- Spectral clustering is such a popular graph clustering algorithm

Various graph embedding algorithms exist (e.g., node2vec)

Spectral Clustering

Spectral clustering has a beautiful theory behind it (won't get into it in this course; may refer to a very nice tutorial article listed below, if interested)

- We are given the node-node similarity matrix $\boldsymbol{A}$ of size $N \times N$
- Compute the graph Laplacian $\boldsymbol{\mathcal{L}} = \boldsymbol{D} - \boldsymbol{A}$
  - $\boldsymbol{D}$ is a diagonal matrix s.t. $D_{nn} = \sum_{m=1}^{N} A_{nm}$ (sum of similarities of node n with all other nodes)
- Note: Often, we work with a normalized graph Laplacian $\boldsymbol{\mathcal{L}}^{norm} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$
- Given the graph Laplacian, solve this spectral decomposition problem

$$\boldsymbol{U} = \underset{\boldsymbol{U} \in \mathbb{R}^{N \times K}}{\operatorname{argmax}} \operatorname{trace}(\boldsymbol{U}^\top \boldsymbol{\mathcal{L}} \boldsymbol{U}) \quad \text{s.t.} \quad \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$$

Meaning U has orthonormal columns

- Now run K-means on the $\boldsymbol{U}$ matrix as the feature matrix of the N nodes
- Note: Spectral clustering* is also closely related to kernel K-means (but more general since $\boldsymbol{A}$ can represent any graph) and "normalized cuts" for graphs

*Kernel k-means, Spectral Clustering and Normalized Cuts (Dhillon et al, 2004)

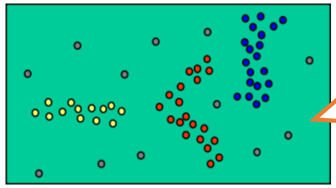
A Tutorial on Spectral Clustering (Ulrike von Luxburg, 2007)

Density based Clustering - DBSCAN

- DBSCAN: Density Based Spatial Clustering of Applications with Noise
  - Uses notion of <u>density</u> of points (not in the sense of probability density) around a point

DBSCAN treats densely connected points as a cluster, regardless of the shape of the cluster

- Has some very nice properties
  - Does not require specifying the number of clusters
  - Can learn arbitrary shaped clusters (since it only considers of density of points)
  - Robust against outliers (leaves them unclustered!), unlike other clust. algos like *K*-means



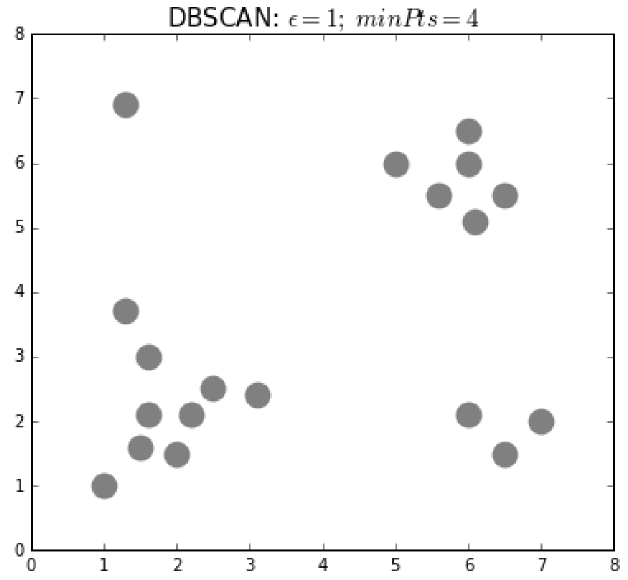
Grey points left unclustered since they are most likely outliers

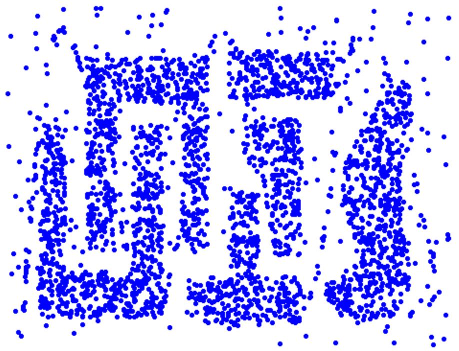
Accuracy of DBSCAN depends crucially on ϵ and minPoint hyperparams

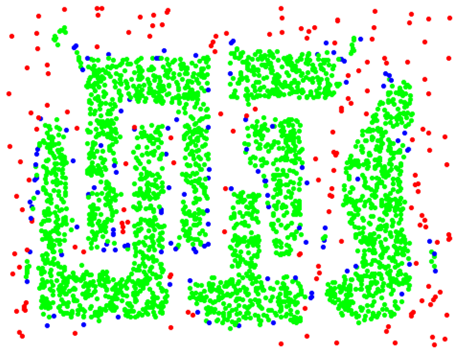
- Basic idea in DBSCAN is as follows
  - Want all points within a cluster to be at most ϵ distance apart from each other
  - Want at least minPoints points within ϵ distance of a point (such a point is called "core" point)
  - Points that don't have minPoints within ϵ distance are called "border" points
  - Points that are neither core nor border point are outliers

DBSCAN (Contd)

- The animation on the right shows DBSCAN in action
- The basic algorithm is as follows
  - A point is chosen at random
  - If more than minPoint neighbors $\leq \epsilon$ distance, then call it core point
  - Check if more points fall within ϵ distance of core/its neighbors
  - If yes, include them too in the same cluster
  - Once done with this cluster, pick another point randomly and repeat
- An example of clustering obtained by DBSCAN







Green points are core points, blue points are border points, red points are outliers

DBSCAN is mostly a heuristic based algorithm. No loss function unlike K-means

Animation credit: https://dashee87.github.io/

Going the Probabilistic Way..

- Assume a generative model for inputs and $\boldsymbol{\Theta}$ denotes all the unknown params
- Clustering then boils down to computing posterior cluster probability $p(\boldsymbol{z}_n|\boldsymbol{x}_n,\Theta)$ where $z_n \in \{1,2.,\ldots,K\}$ denote the cluster assignment of $\boldsymbol{x}_n$

$$p(z_n = k|x_n,\Theta) = \frac{p(z_n = k|\Theta)p(x_n|z_n = k,\Theta)}{p(x_n|\Theta)}$$ (from Bayes rule)

- Assuming prior $p(\boldsymbol{z}_n|\Theta)$ to be multinoulli with prob. vector $\boldsymbol{\pi} = [\pi_1,\pi_2,\ldots,\pi_K]$ and each of the class-conditional $p(\boldsymbol{x}_n|\boldsymbol{z}_n = k,\Theta)$ to be a Gaussian $\mathcal{N}(\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$

$$p(\boldsymbol{z}_n = k|\boldsymbol{x}_n,\Theta) \propto \pi_k \times \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$$ (Here $\Theta = \{\pi_k,\mu_k,\Sigma_k\}_{k=1}^K$)

Posterior prob. Of cluster assignment also depends on prior probability (fraction of points in that cluster if using MLE)

Different clusters can have different covariances (hence different shapes)

- We know how to estimate $\boldsymbol{\Theta}$ if $\boldsymbol{z}_n$ were known (recall generative classification)
- But since we don't know $\boldsymbol{z}_n$, need to estimate both (and ALT-OPT can be used)

Just like in K-means

Going the Probabilistic Way..

- At a high-level, a probabilistic clustering algorithm would look somewhat like this

Sketch of a Probabilistic Clustering Algorithm

1. Initialize the model parameters Θ somehow

   Akin to initializing the cluster means in K-means

2. Given the current Θ, estimate $\mathbf{Z}$ (cluster assignments) in a soft/hard way

   Akin to computing cluster assignments in K-means

$$p(\boldsymbol{z}_n = k|\boldsymbol{x}_n, \Theta) = \gamma_{nk} \quad = \quad \frac{p(\boldsymbol{z}_n = k|\Theta)p(\boldsymbol{x}_n|\boldsymbol{z}_n = k, \Theta)}{p(\boldsymbol{x}_n|\Theta)}, \quad k = 1, \ldots, K$$

$$\text{OR} \quad \hat{\boldsymbol{z}}_n \quad = \quad \arg\max_{k\in\{1,\ldots,K\}} \gamma_{nk}$$

   Akin to updating cluster means in K-means

3. Use $\{\hat{\boldsymbol{z}}_n\}_{n=1}^N$ (hard cluster labels) or $\{\gamma_{nk}\}_{n,k=1}^{N,K}$ (soft labels) to update Θ via MLE/MAP (similar to how we do for gen. classification where the labels are known)
4. Note: The soft-label based Θ updates slightly more involved (wait until we see EM)
5. Go to step 2 if not converged yet.

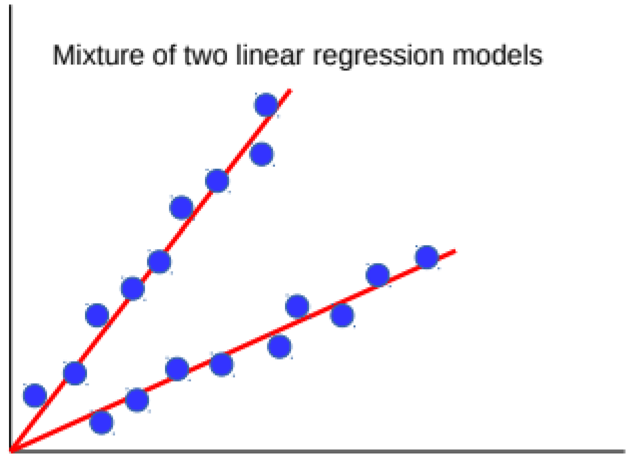
- The above algorithm is an instance of a more general Expectation Maximization (EM) algorithm for latent variable models (we will see this soon)

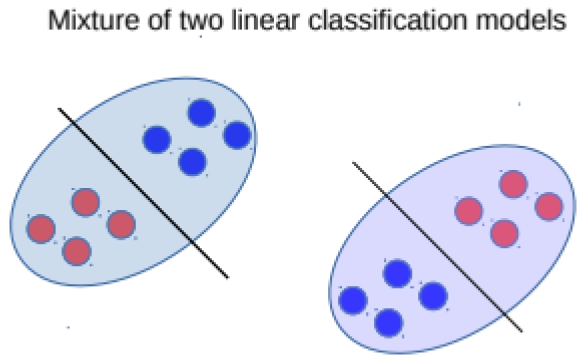
Clustering vs Classification

- Any clustering model (prob/non-prob) typically learns two type of quantities
  - Parameters Θ of the clustering model (e.g., cluster means in K-means)
  - Cluster assignments $\boldsymbol{Z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_N\}$ for the points
- If cluster assignments $\boldsymbol{Z}$ were known, learning the parameters Θ is just like learning the parameters of a classifn model (typically generative classification) using labeled data
- Thus helps to think of clustering as (generative) classification with unknown labels
- Therefore many clustering problems are typically solved in the following fashion
  1. Initialize Θ somehow
  2. Predict Z given current estimate of Θ
  3. Use the predicted Z to improve the estimate of Θ (like learning a generative classification model)
  4. Go to step 2 if not converged yet

Clustering can help supervised learning, too

- Often “difficult” sup. learning problems can be seen as mixture of simpler models
- Example: Nonlinear regression or nonlinear classification as mixture of linear models





- Don’t know which point should be modeled by which linear model ⇒ Clustering
- Can therefore solve such problems as follows
  - Initialize each linear model somehow (maybe randomly)
  - Cluster the data by assigning each point to its “closest” linear model (one that gives lower error)
  - (Re-)Learn a linear model for each cluster’s data. Go to step 2 if not converged.

Such an approach is also an example of divide and conquer and is also known as “mixture of experts” (will see it more formally when we discuss latent variable models)

Coming up next

- Latent Variable Models
  - Mixture models using latent variables
  - Expectation Maximization algorithm