Clustering Algorithms

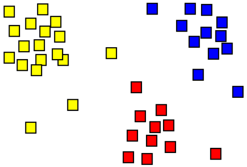
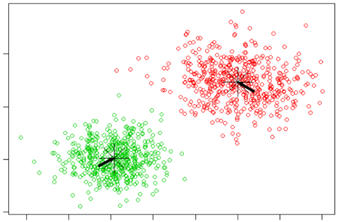
Alvaro del Val

Universidad Autónoma de Madrid

2020

Clustering

- Divide the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups.
  - Segregate groups with similar traits and assign them into clusters.
- A form of *unsupervised classification*

Clustering algorithms

- **Flat** clustering
  - Methods that produce a set of disjoint clusters
  - For example: k-means and isodata
  - *Hard* and *soft* clustering
- **Hierarchical** clustering
  - Methods that produce a hierarchy of clusters: some clusters contain the others
  - *Agglomerative* vs *divisive* methods

Clustering algorithms

- **Hard** clustering: each data point either belongs to a cluster completely or not.
- **Soft** clustering: instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned (e.g. GMM clustering or fuzzy c-means)

Clustering algorithms

- **Parametric** methods
  - We assume that data have been generated from a mixture of probability distributions (*components*) of a given shape, but with unknown parameters
  - The objective is to estimate the set of parameters that provide the best approximation of the mixture to observed data (*maximum likelihood*)
- **Non parametric** methods
  - There is no assumption about data distribution
  - We search for a partition of the input data into a set of *clusters* according to some similarity measure
  - Sometimes known as *unsupervised classification*

K-means algorithm

Algorithm:

1. Set the number of clusters k
2. Initialize the clusters
    - By assigning each data point to a randomly selected cluster (start in 3), or
    - By randomly choosing the cluster centers (start in 4), e.g. randomly select k data points as cluster centers.
3. *Move centroid:* compute the center of each cluster: mean value of all the data points within the cluster
4. *Cluster assignment*: reassign each point to the cluster with the closest center
5. If some point has been moved from one cluster to another, go back to step 3, otherwise stop

K-means algorithm: details

1. Let k be the number of clusters, and $z_i = j \Leftrightarrow \mathbf{x}_i \in C_j$
2. Initialize the algorithm by assigning each data point to a randomly selected cluster
3. Compute the center of each cluster C_j:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

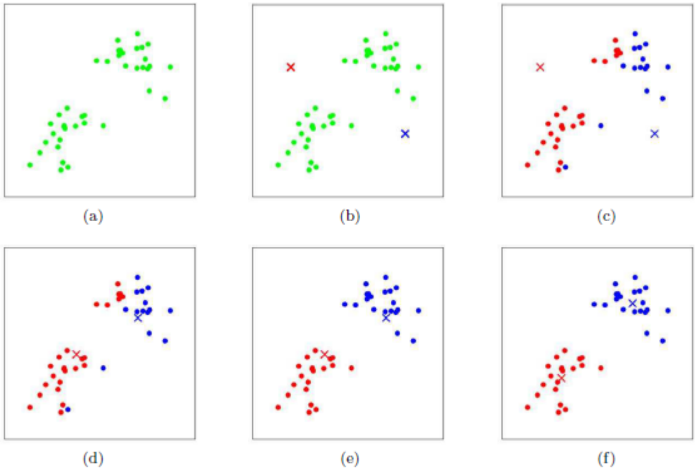
4. Reassign each point to the cluster with the closest center

$$z_i = j \Longleftrightarrow ||\mathbf{x}_i - \mu_j|| \leq ||\mathbf{x}_i - \mu_h|| \;\; \forall \;\; h = 1, 2, ..., k$$

$$\texttt{i.e.}\;\; z_i = \texttt{argmin}_j ||\mathbf{x}_i - \mu_j||^2$$

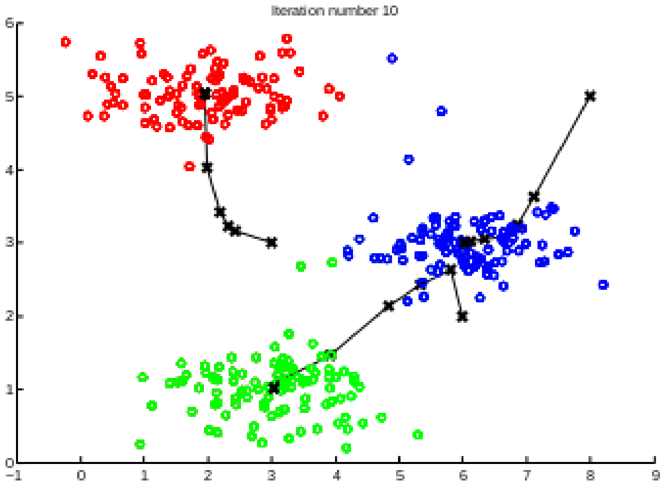
5. If some point has been moved from one cluster to another, go back to step 3, otherwise stop

K-means algorithm: examples



(a) Original dataset. (b) Random initial cluster centroids. (c-f) Two iterations of k-means, with cluster assignment and centroid moves. Images courtesy Michael Jordan.

K-means algorithm: examples



K-means algorithm: cost function

- Let:
  - z_i = index of cluster to which x_i is currently assigned
  - μ_k = cluster centroid for cluster k
  - μ_{z_i} = cluster centroid of x_i's current cluster
- The cost function to minimize is:

$$J(z_1, \ldots, z_k, \mu_1, \ldots, \mu_k) = \frac{1}{m} \sum_{i=1}^{m} ||x_i - \mu_{z_i}||^2$$

K-means algorithm: cost function

- That is, we will minimize the average of the distances of every training sample to its corresponding cluster centroid (the *distortion* of the training samples)
- In the cluster assignment step we are minimizing $J(\dots)$ with $z_1, \dots, z_m$, while holding $\mu_1, \dots, \mu_k$ fixed
- In the move centroid step, we minimize $J(\dots)$ with $\mu_1, \dots, \mu_k$
- Note: in K-means, the cost function cannot increase in any step. However, it can get stuck in local optima, so it is often advisable to rerun the algorithm multiple times with different random initializations.

K-means algorithm and GMM+EM

- The k-means algorithm is a particular case of the EM algorithm for a Gaussian mixture
- Setting $\Sigma_j = \sigma^2 \mathbf{I}$, $p(z_i = j) = 1/K$, and taking the limit $\sigma^2 \to 0$

$$\begin{aligned} p(z_i = j|\mathbf{x}_i;\theta) &= \frac{p(z_i = j)\exp\left(-(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j)/2\sigma^2\right)}{\sum_{j=1}^{k} p(z_i = j)\exp\left(-(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j)/2\sigma^2\right)} \\ &= \frac{p(z_i = j)\exp\left(-||\mathbf{x}_i - \mu_j||^2/2\sigma^2\right)}{\sum_{j=1}^{k} p(z_i = j)\exp\left(-||\mathbf{x}_i - \mu_j||^2/2\sigma^2\right)} \end{aligned}$$

$$\lim_{\sigma^2 \to 0} p(z_i = j|\mathbf{x}_i;\theta) = \begin{cases} 1 & \text{if } ||\mathbf{x}_i - \mu_j|| \leq ||\mathbf{x}_i - \mu_h|| \;\; \forall\, h \\ 0 & \text{otherwise} \end{cases}$$

Selection of the number of clusters k

- The value of k is set a priori
- The k-means algorithm finds a solution for any k
- The selection of the optimal number of clusters is a non trivial task (**cluster validation**)
    - Elbow method: plot cost against number of clusters, and choose k at the point where the curve begins to flatten out.
    - Choose according to some downstream purpose, e.g. market segmentation.

Similarity measures

- In general, in clustering we search for groups or **clusters** within the data such that, according to some *similarity measure*
  - Elements within a cluster are similar
  - Elements from two different clusters are different
- The clusters are found by optimizing a *criterion function* that is based on the similarity measure

Similarity measures (I)

- Euclidean distance (L_2)

$$||\mathbf{x} - \mathbf{y}||_2 = \sqrt{\sum_{i=1}^{d} |x_i - y_i|^2}$$

- Manhattan distance (L_1)

$$||\mathbf{x} - \mathbf{y}||_1 = \sum_{i=1}^{d} |x_i - y_i|$$

Similarity measures (II)

- Minkowski distance(L_k)

$$||\mathbf{x} - \mathbf{y}||_k = \sqrt[k]{\sum_{i=1}^{d} |x_i - y_i|^k}$$

- Dot product (for unitary vectors)

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} x_i y_i$$

Criterion function

- The most frequent criterion function is based on the mean square error (Euclidean distance)

$$J_{MSE} = \sum_{j=1}^{k} \sum_{i \in c_j} ||\mathbf{x}_i - \mu_j||^2$$

with

$$\mu_j = \frac{1}{|c_j|} \sum_{i \in c_j} \mathbf{x}_i$$

- This criterion function measures how well the data points $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ can be represented by the cluster centers $\{\mu_1, \mu_2, ..., \mu_k\}$

Optimization of the criterion function

- An exhaustive search that analyzes all possible partitions of the dataset into k clusters is not feasible

$$k^n \quad \text{different partitions for } n \text{ points}$$

- So in general we use iterative algorithms that produce suboptimal results, but are computationally feasible

GMM clustering

- We saw Gaussian Mixture Models as a (parametric) method for density estimation
- Model data by joint distribution $p(x_i, z_i) = p(x_i|z_i)p(z_i)$
  - each $z_i \sim$ Multinomial(ϕ) over k values, with $\phi_j = p(z_i = j)$,
  - $x_i|z_j \sim \mathbb{N}(\mu_j, \Sigma_j)$.
- z_i's are hidden or latent variables, can also be seen as clusters or classes.
- **Soft** clustering: we only get a probability that an item belongs to a cluster.
  - can assign item to cluster with largest probability

Expectation maximization (review)

- (Expectation step) For each i, j set:

$$w_{ij} = p(z_i = j \mid x_i; \phi, \mu, \Sigma)$$

- (Maximization step) Update the parameters:

$$\phi_j = \frac{1}{m} \sum_{i=1}^{m} w_{ij}$$

$$\mu_j = \frac{\sum_{i=1}^{m} w_{ij} x_i}{\sum_{i=1}^{m} w_{ij}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} w_{ij} (x_i - \mu_j)(x_i - \mu_j)^T}{\sum_{i=1}^{m} w_{ij}}$$

Expectation maximization (review)

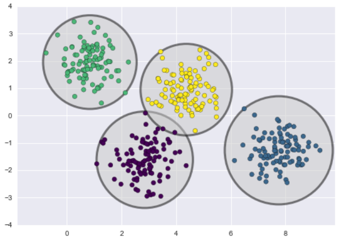
- (E-step) calculates posterior probability of the z_i's given the x_i's and the given setting of our parameters, using Bayes rule

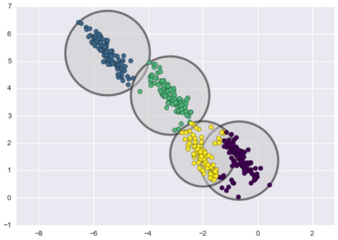
$$p(z_i = j \mid x_i; \phi, \mu, \Sigma) = \frac{p(x_i \mid z_i = j; \mu, \Sigma) p(z_i = j; \phi)}{\sum_{l=1}^{k} p(x_i | z_i = l; \mu, \Sigma) p(z_i = l; \phi)}$$

  - "soft guesses" (i.e. probabilities) for values of z_i's
  - provides lower bound on likelihood
- (M-step) optimizes that lower bound (maximum likelihood estimation)

GMM clustering vs k-means

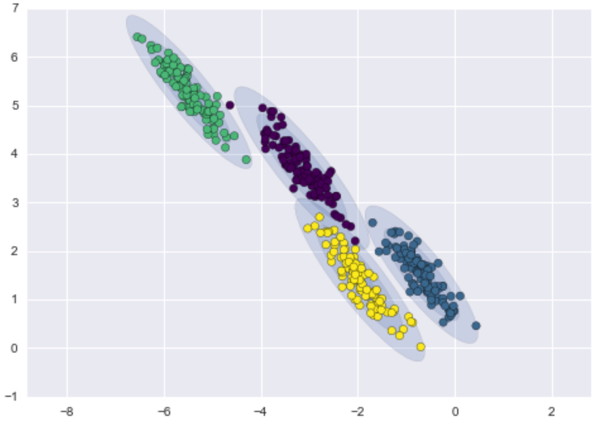
k-means ignores the variance. It can be seen as drawing a circle around the centroid, with radius given by the most distant point in cluster





GMM clustering vs k-means

In contrast, Gaussian Mixture Models can handle very oblong clusters

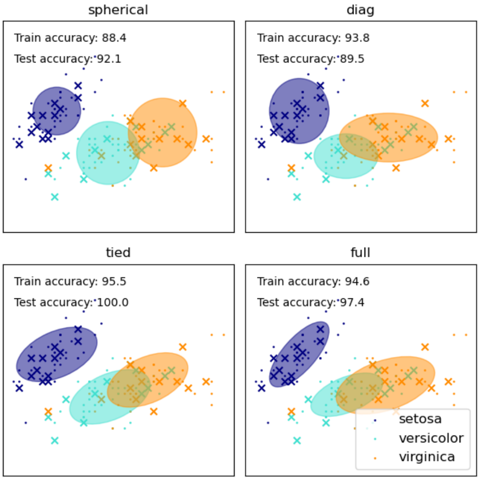


GMM clustering: constraints on covariance

It is also possible to place constraints on the form of the covariance in GMM

- Full covariance: Σ_k is arbitrary for each class (clusters are general ellipsoids, $O(Km^2)$ parameters)
- Shared Full covariance: Σ_k is arbitrary but same for each class (all clusters have same ellipsoid shape, $O(m^2)$ parameters)
- Diagonal (aka Naive Bayes): Σ_k is a diagonal matrix (clusters are axis-aligned ellipsoids, $O(Km)$ parameters)
- Shared Diagonal: Σ_k is a diagonal matrix but same for each class (all clusters have same axis-aligned ellipsoid shape, $O(m)$ parameters
- Spherical: Σ_k is $\sigma_k \mathbf{I}$ (clusters have spherical shape)
- Shared Spherical: Σ_k is $\sigma \mathrm{b} f I$ (clusters have spherical shape, all with the same radius)

GMM clustering: constraints on covariance



Choosing the number of GMM clusters

Let k be the number of parameters in a model, and $\hat{L}$ the maximum value of the likelihood function for the model

- AIC (Akaike information criterion): minimize

$$AIC = 2k - 2\ln(\hat{L})$$

- BIC (Bayesian Information Criterion): minimize

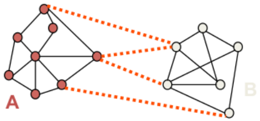
$$BIC = k\ln(m) - 2\ln(\hat{L})$$

Spectral clustering

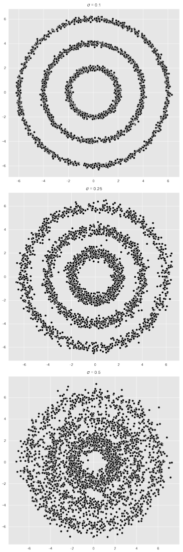
- Techniques bases on the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions.
- Solvable by linear algebra.
- Performs quite well on non-convex spaces

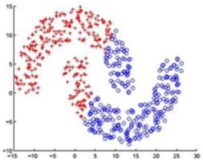
Spectral clustering: Intuition

- Given data points $x_1, \ldots, x_m$, compute similarities $W(i,j) = s(x_i, x_j)$
- Build similarity graph: vertices: data points, edges: similarities

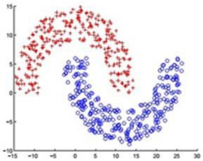


- Clustering: find a cut through the graph.
  - define a cut objective function
  - solve it
- Data is projected into a lower dimension by the use of spectral decomposition, where they are easily separable.

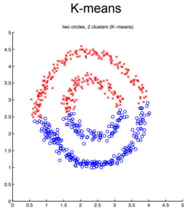




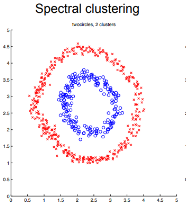
(a) K-means



(b) Spectral Clustering



(i)



(e)

Spectral clustering

Algorithm

1. Construct a similarity graph
2. Determine the Adjacency matrix W, Degree matrix D and the Laplacian matrix L
3. Compute the eigenvalues of the matrix L
4. Use the eigenvectors for the *k* smallest eigenvalues (other than the first one) to project the data, and run k-means.

Spectral clustering: Similarity graph

Similarity graph: graph nodes are the items in the data set, and edges are represented by a (weighted or unweighted) adjacency matrix W.

Various options for similarity $W(i,j)$:

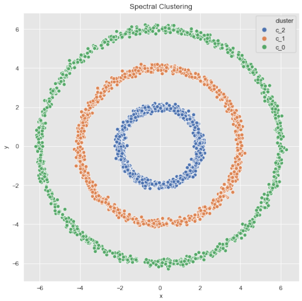
- Euclidean distance: $W(i,j) = ||x_i - x_j||$
- Gaussian kernel: $W(i,j) = exp(\frac{-||x_i - x_j||^2}{\sigma^2})$
- k-nearest neighbor: $W(i,j) = 1$ iff either i is a k-nearest neighbor of j or j is a k-nearest neighbor of i

Spectral clustering: Matrices

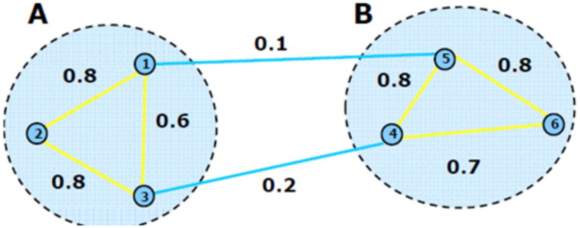
- Adjacency matrix W with W_{ij} either the weights $W(i,j)$, or unweighted edges depending on some criterion (k-nearest neighbor, or $W(i,j) \leq \epsilon$ for some ϵ)
- Degree matrix: diagonal matrix D with $D_{ii} = \sum_{j=1}^{n} W_{ij}$
- Laplacian matrix $L = D - W$.
    - Each row of L sums to 0, therefore $\lambda_0 = 0$ is an eigenvalue with the eigenvector $\mathbf{1}$ in the null space of L.
    - The multiplicity of the 0 eigenvalues corresponds to the number of connected components of the graph.

Spectral clustering: steps

- Choose the k smallest eigenvalues of L (ignoring the first one), with corresponding eigenvectors $u_1, \dots, u_k$.
- Let U be the $(n \times k)$ matrix containing $u_1, \dots, u_k$ as columns.
- For $i = 1, \dots, n$, let $y_i \in \mathbb{R}^k$ be the i-th row of U
- Cluster the points $y_i \in \mathbb{R}^k$ with the k-means algorithm into clusters $C_1, \dots, C_k$ (**dimensionality reduction**: $n \times n \to n \times k$)



Spectral clustering as min-cut



- $Vol(A) = \sum_{i \in A} W_{ij}$
- $Cut(A, B) = \sum_{i \in A, j \in B} W_{ij} = 0.3$
- Normalized cut $Ncut(A, B) = Cut(A, B)(\frac{1}{Vol(A)} + \frac{1}{Vol(B)})$: minimized when $Vol(A) = Vol(B)$, encourages balanced cut.
- Ratio cut $RCut(A, B) = Cut(A, B)(\frac{1}{|A|} + \frac{1}{|B|})$

Spectral clustering as min-cut

- Minimization of *Ncut* or *RCut* is NP-hard, the algorithms are based on a relaxation of the problem.
- For two clusters, the eigenvector with the second smallest eigenvalue provides a solution to the problem (need to choose splitting point for value of coordinate, often the sign of the coordinate determines cluster)
- For more clusters, the eigenvectors $u_1, \dots, u_k$ are solutions to one of the following cuts:

$$Rcut(A_1, \dots, A_k) = \sum_{i=1}^{k} \frac{Cut(A_i, \overline{A_1})}{|A_i|}$$

$$Ncut(A_1, \dots, A_k) = \sum_{i=1}^{k} \frac{Cut(A_i, \overline{A_1})}{Vol(A_i)}$$

Kernel k-means

- Enhance k-means clustering by means of a *kernel function* mapping the original input space to a higher-dimensional feature space: $\phi(x_i)$
- Allows to extract clusters that are non-linearly separable in input space

Kernel k-means

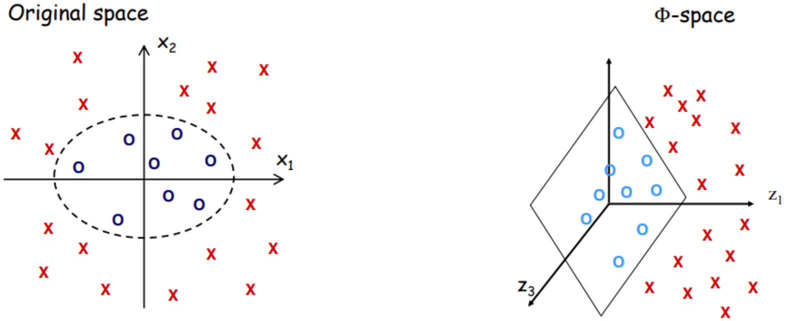
- Kernel trick: Kernel function $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ can often be computed without actually computing the projections.
- Given any algorithm that can be expressed solely in terms of dot products, this trick allows us to construct different non-linear versions of it.

Example: Polynomial kernel

For $n = 2$ and $d = 2$, the kernel $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d$ corresponds to the projection:

$$\phi : \mathbb{R}^2 \to \mathbb{R}^3,\ (x_1, x_2) \to \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\begin{aligned}\phi(\mathbf{x}) \cdot \phi(\mathbf{z}) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2) \\ &= (x_1z_1 + x_2z_2)^2 = (\mathbf{x} \cdot \mathbf{z})^2 = K(\mathbf{xz})\end{aligned}$$

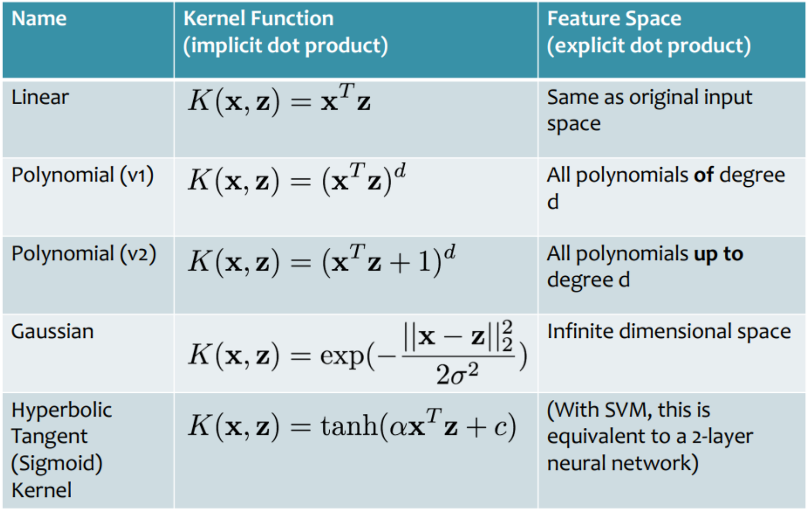


⇒ Visualization of linear separability in higher dimensions

Kernel matrix

- Kernel function as similarity measure between instances
- Kernel matrix K for similarity between x_i an x_j: $K_{(x_i,x_j)} = \phi(x_i) \cdot \phi(x_j)$.
    - Note: K is $(m \times m)$, instead of the smaller original $(m \times n)$ data set (m instances with n features)

Kernel examples



Name	Kernel Function (implicit dot product)	Feature Space (explicit dot product)
Linear	$K(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\mathbf{z}$	Same as original input space
Polynomial (v1)	$K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$	All polynomials **of** degree d
Polynomial (v2)	$K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z}+1)^d$	All polynomials **up to** degree d
Gaussian	$K(\mathbf{x},\mathbf{z}) = \exp(-\frac{\lVert\mathbf{x}-\mathbf{z}\rVert_2^2}{2\sigma^2})$	Infinite dimensional space
Hyperbolic Tangent (Sigmoid) Kernel	$K(\mathbf{x},\mathbf{z}) = \tanh(\alpha\mathbf{x}^T\mathbf{z}+c)$	(With SVM, this is equivalent to a 2-layer neural network)

Kernel k-means: example

- Gaussian kernel (also called radial basis function RBF) $K(x_i, x_j) = e^{-||x_i - x_j||^2/2\sigma^2}$, with $\sigma = 4$
- E.g. with the 2-dimensional data below, $||x_1 - x_2||^2 = (0-4)^2 + (0-4)^2 = 32$, so $K(x_1, x_2) = e^{-\frac{32}{2 \cdot 4^2}} = e^{-1}$

Input data set in $\mathbb{R}^2$			Kernel matrix in $\mathbb{R}^5$				
	x	y	$K(x_i, x_1)$	$K(x_i, x_2)$	$K(x_i, x_3)$	$K(x_i, x_4)$	$K(x_i, x_5)$
x_1	0	0	1	e^{-1}	e^{-1}	e^{-1}	e^{-1}
x_2	4	4	e^{-1}	1	e^{-2}	e^{-4}	e^{-2}
x_3	-4	4	e^{-1}	e^{-2}	1	e^{-2}	e^{-4}
x_4	-4	-4	e^{-1}	e^{-4}	e^{-2}	1	e^{-2}
x_5	4	-4	e^{-1}	e^{-2}	e^{-4}	e^{-2}	1

Kernel k-means

Algorithm: run k-means using kernel matrix

- SSE criterion for kernel k-means:

$$SSE(C) = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||\phi(x_i) - c_k||^2$$

- Cluster centroid:

$$c_k = \frac{\sum_{x_i \in C_k} \phi(x_i)}{|C_k|}$$

Kernel k-means

Algorithm: run k-means using kernel matrix

- Cluster assignment: compute $z_i = \texttt{argmin}_j ||\phi(x_i) - c_j||^2$, using:

$$||\phi(x_i) - c_k||^2 = \phi(x_i) \cdot \phi(x_i) - \frac{2\sum_{x_j \in C_k} \phi(x_i) \cdot \phi(x_j)}{|C_k|} + \frac{\sum_{x_j, x_l \in C_k} \phi(x_j) \cdot \phi(x_l)}{|C_k|}$$

  Using the kernel matrix, this can be simplified to (link):

$$z_i = \texttt{argmin}_j \frac{1}{|C_j|^2} \sum_{x_k, x_l \in C_j} K(x_k, x_l) - \frac{2}{|C_j|} \sum_{x_k \in C_j} K(x_i, x_k)$$

- Cluster centroid: not computed explicitly, just update clusters

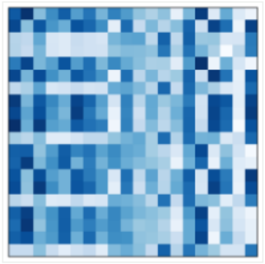
$$C_j = \{x_i \mid z_i = j\}$$

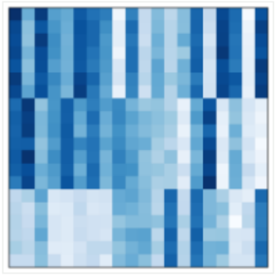
Kernel k-means and spectral clustering

- Weighted kernel k-means: introduce a weight function $w(x_i)$ for each data point.
- It can be shown (link) that both weighted kernel k-means and spectral clustering solve a trace maximization problem (trace: sum of eigenvalues): with appropiate weights, weighted kernel k-means solves the NCut minimization problem. And conversely, spectral clustering can be viewed as a form of weighted kernel k-means.

Biclustering

Given a set of m samples represented by an n-dimensional represented in an $m \times n$ matrix, the biclustering algorithm generates biclusters - a subset of rows which exhibit similar behavior across a subset of columns, or vice versa.

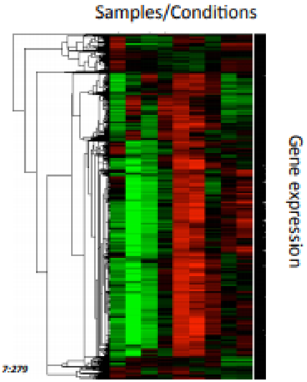




Biclustering for gene expression

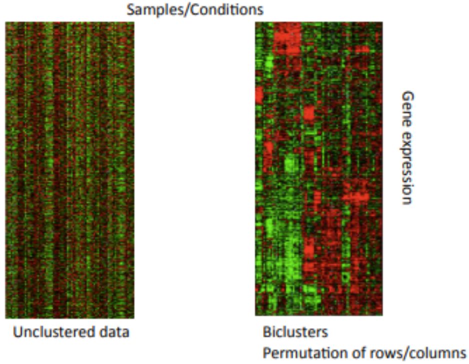
Subset of genes may behave similarly under only a **subset** of conditions.

- Only a subset of genes in cellular process of interest.
- Cellular process active in only a subset of conditions.



BMC Genomics 2006, 7:279

Biclustering for gene expression

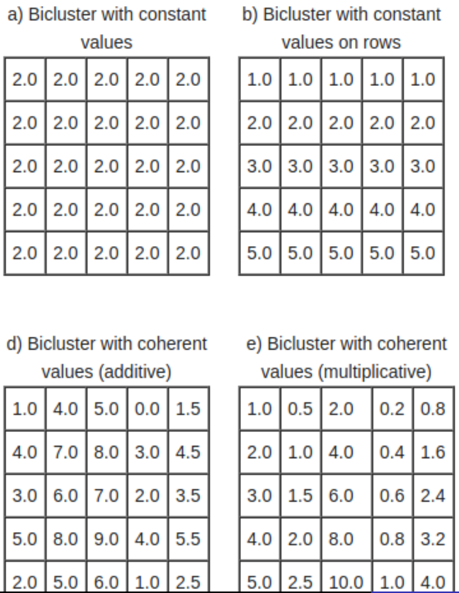


A bicluster corresponds to a submatrix

Types of biclustering

- Biclustering with constant values: $a_{ij} = \mu$ for all $i \in I$ and $j \in J$
- Biclustering with constant values in rows or columns: $a_{ij} = \mu + \alpha_i$ or $a_{ij} = \mu\alpha_i$ (rows)
- Biclustering with coherent values: $a_{ij} = \mu + \alpha_i + \beta_j$ or $a_{ij} = \mu' \times \alpha'_i \times \beta'_j$, where $\mu = log(\mu'), \alpha = log(\alpha'), \beta = log(\beta')$.

Types of biclustering



a) Bicluster with constant values

2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0
2.0	2.0	2.0	2.0	2.0

b) Bicluster with constant values on rows

1.0	1.0	1.0	1.0	1.0
2.0	2.0	2.0	2.0	2.0
3.0	3.0	3.0	3.0	3.0
4.0	4.0	4.0	4.0	4.0
5.0	5.0	5.0	5.0	5.0

c) Bicluster with constant values on columns

1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0
1.0	2.0	3.0	4.0	5.0

d) Bicluster with coherent values (additive)

1.0	4.0	5.0	0.0	1.5
4.0	7.0	8.0	3.0	4.5
3.0	6.0	7.0	2.0	3.5
5.0	8.0	9.0	4.0	5.5
2.0	5.0	6.0	1.0	2.5

e) Bicluster with coherent values (multiplicative)

1.0	0.5	2.0	0.2	0.8
2.0	1.0	4.0	0.4	1.6
3.0	1.5	6.0	0.6	2.4
4.0	2.0	8.0	0.8	3.2
5.0	2.5	10.0	1.0	4.0

Hierarchical clustering

- Sometimes we may be interested in a hierarchical representation of the data
    - Clusters and sub-clusters with a tree structure

Hierarchical clustering

- Sometimes we may be interested in a hierarchical representation of the data
    - Clusters and sub-clusters with a tree structure
- **Agglomerative clustering** (bottom-up)
    - We start from a set of *n* single-element clusters and we keep on merging clusters until only one remains

Hierarchical clustering

- Sometimes we may be interested in a hierarchical representation of the data
    - Clusters and sub-clusters with a tree structure
- **Agglomerative clustering** (bottom-up)
    - We start from a set of n single-element clusters and we keep on merging clusters until only one remains
- **Divisive clustering** (top-down)
    - We start from a single cluster containing the whole dataset, and we repeatedly perform cluster divisions until n single-element clusters have been generated

Divisive clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with a single cluster ($k = 1$) containing the whole dataset

Divisive clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with a single cluster ($k = 1$) containing the whole dataset
2. Find the *worst* cluster

Divisive clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with a single cluster ($k = 1$) containing the whole dataset
2. Find the *worst* cluster
3. Divide the *worst* cluster into two different clusters and update k

Divisive clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with a single cluster ($k = 1$) containing the whole dataset
2. Find the *worst* cluster
3. Divide the *worst* cluster into two different clusters and update k
4. If $k < n$ go back to 2

Divisive clustering. Practical concerns

- How to choose the *worst* cluster
  - The biggest one (highest number of poinst)?
  - The one with the highest variance?
  - The one with the highest squared error?

Divisive clustering. Practical concerns

- How to choose the *worst* cluster
  - The biggest one (highest number of poinst)?
  - The one with the highest variance?
  - The one with the highest squared error?
- How to divide a cluster
  - Mean or median with respect to one of the attributes?
  - Mean or median along the direction of highest variance?

Divisive clustering. Practical concerns

- How to choose the *worst* cluster
  - The biggest one (highest number of poinst)?
  - The one with the highest variance?
  - The one with the highest squared error?
- How to divide a cluster
  - Mean or median with respect to one of the attributes?
  - Mean or median along the direction of highest variance?
- In general, divisive clustering involves more complex operations than agglomerative clustering

Agglomerative clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with $k = n$ single-element clusters

Agglomerative clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with $k = n$ single-element clusters
2. Find the two closest clusters according to some *distance*

Agglomerative clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with $k = n$ single-element clusters
2. Find the two closest clusters according to some *distance*
3. Merge these two clusters and update k

Agglomerative clustering. Standard algorithm

n represents the number of points, k is the number of clusters

1. Start with $k = n$ single-element clusters
2. Find the two closest clusters according to some *distance*
3. Merge these two clusters and update k
4. If $k > 1$ go back to 2

How to measure the distance between two clusters

- Minimum distance between cluster points

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i,\ \mathbf{y} \in c_j$$

How to measure the distance between two clusters

- Minimum distance between cluster points

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

- Maximum distance between cluster points

$$d_{max}(c_i, c_j) = \max ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

How to measure the distance between two clusters

- Minimum distance between cluster points

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i,\ \mathbf{y} \in c_j$$

- Maximum distance between cluster points

$$d_{max}(c_i, c_j) = \max ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i,\ \mathbf{y} \in c_j$$

- Average distance

$$d_{avg}(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{\mathbf{x} \in c_i} \sum_{\mathbf{y} \in c_j} ||\mathbf{x} - \mathbf{y}||$$

How to measure the distance between two clusters

- Minimum distance between cluster points

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

- Maximum distance between cluster points

$$d_{max}(c_i, c_j) = \max ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

- Average distance

$$d_{avg}(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{\mathbf{x} \in c_i} \sum_{\mathbf{y} \in c_j} ||\mathbf{x} - \mathbf{y}||$$

- Distance between cluster means

$$d_{mean}(c_i, c_j) = ||\mu_i - \mu_j||$$

Agglomerative clustering with minimum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i,\ \mathbf{y} \in c_j$$

Agglomerative clustering with minimum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

- Nearest neighbours algorithm (*single-linkage*)

Agglomerative clustering with minimum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

- Nearest neighbours algorithm (*single-linkage*)
- The final result is a *minimum spanning tree*

Agglomerative clustering with minimum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{min}(c_i, c_j) = \min ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i,\ \mathbf{y} \in c_j$$

- Nearest neighbours algorithm (*single-linkage*)
- The final result is a *minimum spanning tree*
- Elongated classes

Agglomerative clustering with maximum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{max}(c_i, c_j) = \max ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i,\ \mathbf{y} \in c_j$$

Agglomerative clustering with maximum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{max}(c_i, c_j) = \max ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

- *Complete-linkage* algorithm

Agglomerative clustering with maximum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{max}(c_i, c_j) = \max ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i, \mathbf{y} \in c_j$$

- *Complete-linkage* algorithm
- Each cluster is a *complete graph*

Agglomerative clustering with maximum distance

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{max}(c_i, c_j) = \max ||\mathbf{x} - \mathbf{y}|| \text{ with } \mathbf{x} \in c_i,\ \mathbf{y} \in c_j$$

- *Complete-linkage* algorithm
- Each cluster is a *complete graph*
- Compact classes

Average distance and distance between cluster centers

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{avg}(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{\mathbf{x} \in c_i} \sum_{\mathbf{y} \in c_j} ||\mathbf{x} - \mathbf{y}||$$

  or

$$d_{mean}(c_i, c_j) = ||\mu_i - \mu_j||$$

Average distance and distance between cluster centers

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

$$d_{avg}(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{\mathbf{x} \in c_i} \sum_{\mathbf{y} \in c_j} ||\mathbf{x} - \mathbf{y}||$$

  or

$$d_{mean}(c_i, c_j) = ||\mu_i - \mu_j||$$

- Robustness against outliers

Average distance and distance between cluster centers

- In step 3 of the algorithm, we merge the clusters c_i and c_j that minimize

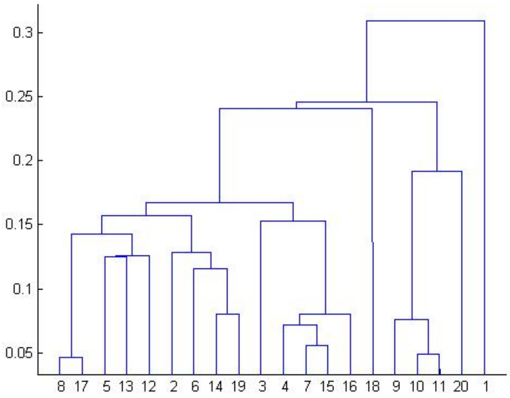
$$d_{avg}(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{\mathbf{x} \in c_i} \sum_{\mathbf{y} \in c_j} ||\mathbf{x} - \mathbf{y}||$$

  or

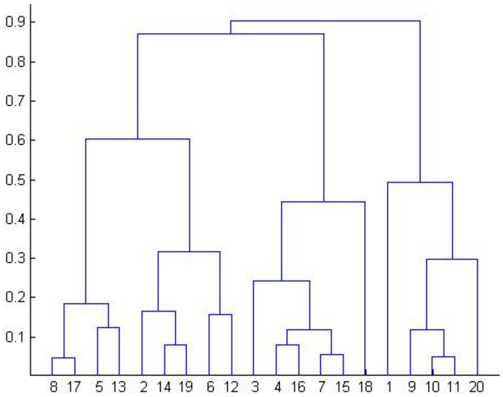
$$d_{mean}(c_i, c_j) = ||\mu_i - \mu_j||$$

- Robustness against outliers
- The distance between cluster centers is computationally very *cheap*

Dendrogram. Single linkage



Dendrogram. Complete linkage



Cluster validation

How to evaluate a clustering algorithm?

General goal:

- the objects in the same cluster are similar as much as possible
- the objects in different clusters are highly distinct

Types of cluster validation

- Internal cluster validation: use the internal information of the clustering process to evaluate the goodness of a clustering structure without reference to external information
- External cluster validation: compare the results of a cluster analysis to an externally known result, such as externally provided class labels (e.g.)
- Relative cluster validation: evaluate the clustering structure by varying different parameter values (e.g. the number of clusters) for the same algorithm

Internal cluster validation

- Cluster cohesion or compactness: how close are the objects within the same cluster, based on distance measures such as the cluster-wise average/median distances between observations in each cluster.
- Separation: how well-separated a cluster is from other clusters, with measures such as distances between cluster centers or the pairwise minimum distances between objects in different clusters
- Connectivity: to what extent items are placed in the same cluster as their nearest neighbors in the data space.

Internal cluster validation

- Measures of performance usually combine cohesion and separation in a single measure, e.g. $index = \frac{\alpha \times Separation}{\beta \times Compactness}$

Internal cluster validation: Silhouette

For each data point x_i, calculate:

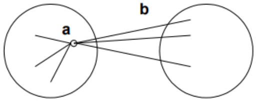
- a_i = average distance of x_i to the points in its cluster
- b_i = min (average distance of x_i to points in another cluster)

The silhouette coefficient for a point is then given by

$$s_i = 1 - a_i / b_i \quad \text{if } a < b$$

(or $s = b/a - 1$ in the less common case in which $a \geq b$)

- Usually between 0 and 1, the closer to 1 the better.



- Calculate average silhouette for a cluster or sets of clusters

Internal cluster validation: Silhouette

- Observations with a large s_i (almost 1) are very well clustered
- Small s_i (around 0) means that the observation lies between two clusters
- Observations with a negative s_i are probably placed in the wrong cluster

Dunn Index

- Intercluster separation: minimum of pairwise distance between points in one cluster and points in all other clusters (min.separation)
- Intracluster distance: maximal distance between points in the same cluster (max.diameter)
- Define the Dunn index

$$D = \frac{min.separation}{max.diameter}$$

- For compact and well-separated clusters, the cluster diameter should be small and the intercluster distance large, so we want to maximize this index.

External cluster validation

- Coherence with some other bodies of knowledge: e.g. is the clustering consistent with know biology about gene function?
- Downstream purposes: choose according to some downstream purpose, e.g market segmentation
- External validation with known class labels

External cluster validation

Assuming class labels are known, evaluate how well the clusters capture the known labels.

- Matching-based measures:
  - Purity
  - Maximum matching
  - F-measure
- Entropy-based measures
  - Conditional entropy (of the "true" class labeling with respect to a cluster)
  - Normalized mutual information
  - Variation of information
- Pairwise measures: based on true/false positives/negatives.
  - Jaccard, Rand, Fowlkes-Mallows.

See Zaki & Meira for details on all these measures.
For python, see clustering performance evaluation, sklearn.metrics and sklearn.metrics.cluster