Unsupervised Learning

Javier Béjar

URL - 2024 Spring Term

CS - MAI

Unsupervised Learning

- Learning can be done in a supervised or unsupervised way
- There is a strong bias in the machine learning community towards supervised learning
- But a lot of concepts are learned unsupervisedly
- The discovery of new concepts is always unsupervised

- We assume that data is embedded in a N-dimensional space with a similarity/dissimilarity function
- Similarity defines how examples are related to each other
- **Bias**:
  - Examples are more related to the nearest examples than to the farthest
  - Patterns are compact groups that are maximally separated from each other
- **Areas**: Statistics, machine learning, graph theory, fuzzy theory, physics

- Discovery goals:
  - **Summarization:** To obtain representations that describe an unlabelled dataset
  - **Understanding:** To discover the concepts inside the data
- Difficult tasks because discovery is biased by context
  - Different answers could be valid depending on the discovery goal or the domain
  - There are few criteria to validate the results
- **Representation of the clusters**: Unstructured (partitions) or relational (hierarchies)

Hierarchical algorithms

- Examples are organized as a binary tree
- Based on the relationship among examples defined by similarity/dissimilarity functions
- No explicit division into groups, has to be chosen a posteriori

Partitional algorithms

- Only a partition of the dataset is obtained
- Based on the optimization of a criteria (assumptions about the characteristics of the cluster model)

Hierarchical Algorithms

- **Based on graph theory**
  - The examples form a fully connected graph
  - Similarity defines the length of the edges
  - Clustering is decided using a connectivity criterion
- **Based on matrix algebra**
  - A distance matrix is calculated from the examples
  - Clustering is computed using the distance matrix
  - The distance matrix is updated after each iteration (different updating criteria)

- Graphs
  - Single Linkage, Complete Linkage, MST
  - Divisive, Agglomerative
- Matrices
  - Johnson algorithm
  - Different update criteria (S-L, C-L, Centroid, minimum variance)

Computational cost

From $O(n\_inst^3 \times n\_dims)$ to $O(n\_inst^2 \times n\_dims)$

Algorithm: Agglomerative graph algorithm

Compute distance/similarity matrix

repeat

  Find the pair of examples with the smallest distance

  Add an edge to the graph corresponding to this pair

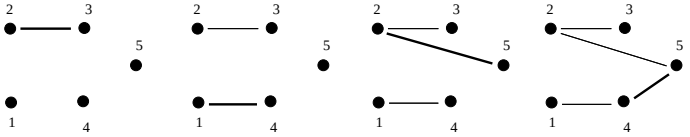
  if *Agglomeration criteria holds* **then**

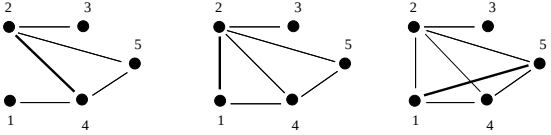
    Merge the clusters the pair belongs to

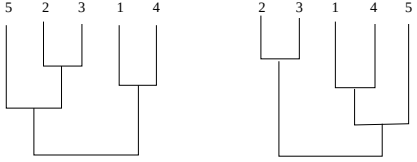
until *Only one cluster exists*

- Single linkage = New edge is between two disconnected sub-graphs
- Complete linkage = New edge creates a clique with all the nodes of both sub-graphs

	2	3	4	5
1	6	8	2	7
2		1	5	3
3			10	9
4				4







Single Link

Complete Link

Algorithm: Agglomerative Johnson algorithm

Compute Distance/similarity matrix

repeat

> Find pair of groups/examples with the smallest distance
> Merge the pair of groups/examples
> Delete the rows and columns corresponding to the pair
> Add a new row and column with the new distances for the new group

until *Matrix has one element*

- Single linkage = Distance between the closest examples
- Complete linkage = Distance between the farthest examples
- Average linkage = Distance between group centroids

	2	3	4	5
1	6	8	2	7
2		1	5	3
3			10	9
4				4

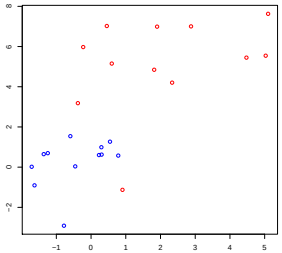
	2,3	4	5
1	7	2	7
2,3		7.5	6
4			4

	1,4	5
2,3	7.25	6
1,4		5.5

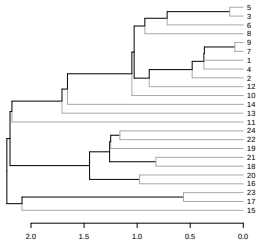
	1,4,5
2,3	6.725

- A partition of the data has to be decided a posteriori
- Some undesirable and strange behaviours could appear (chaining, inversions, breaking large clusters)
- Some algorithms have problems when different sized and convex shaped clusters appear in the data
- Dendrograms are not a practical representation for large amounts of data
- Computational cost is too high for large datasets
  - Time is $O(n^2)$ in the best case, $O(n^3)$ in general

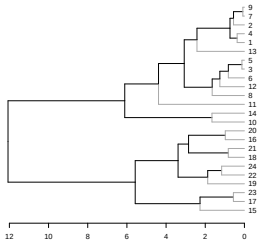
Data



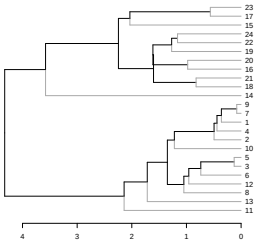
Single Link



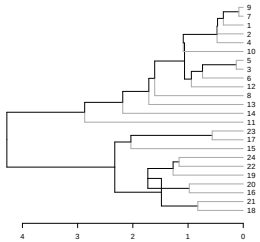
Complete Link



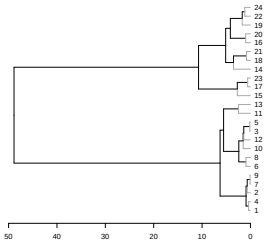
Median



Centroid



Ward



This Python Notebook shows examples of using different hierarchical clustering algorithms

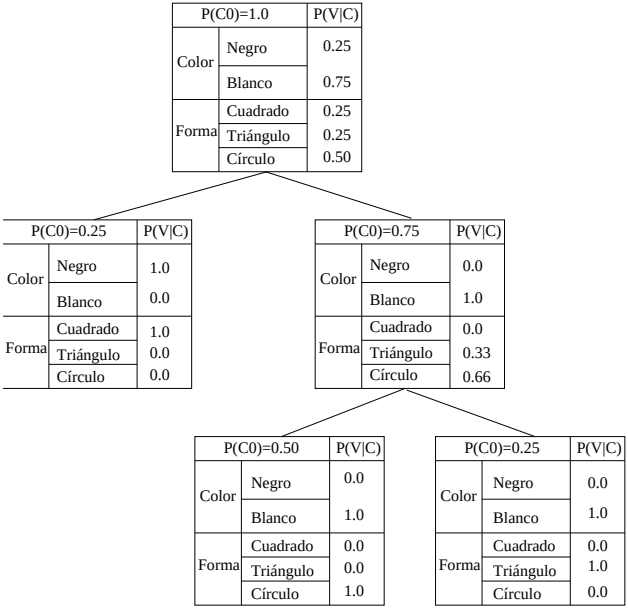
- Hierarchical Clustering Algorithms Notebook (click here to open the notebook in colab)

If you download the notebook you will be able to use it locally (run jupyter notebook to open the notebooks)

- Learning has an incremental nature (experience is acquired from continuous observation, not at once)
- Concepts are learned at the same time as their relationships (polythetic hierarchies of concepts)
- Learning is a search in the space of hierarchies
- An objective function measures the utility of the structure
- The updating of the structure is performed by a set of conceptual operators
- The result depends on the order of the examples

JH Gennari, P Langley, D Fisher, **Models of incremental concept formation**, Artificial intelligence, 1989

- Based on ideas from cognitive psychology
  - Learning is incremental
  - Concepts are organized in a hierarchy
  - Concepts are organized around a prototype and described probabilistically
  - Hierarchical concept representation is modified via cognitive operators
- Builds a hierarchy top/down
- Four conceptual operators
- Heuristic measure to find the *basic level* (Category utility)

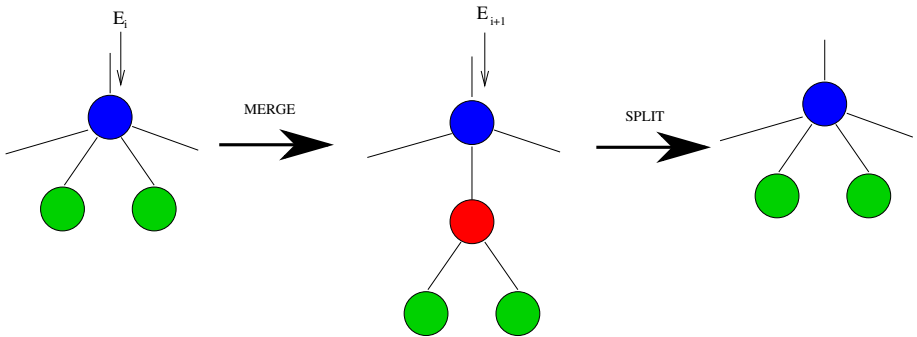

P(C0)=1.0
P(V|C)
Color
Negro
0.25
Blanco
0.75
Forma
Cuadrado
0.25
Triángulo
0.25
Círculo
0.50
P(C0)=0.25
P(V|C)
Color
Negro
1.0
Blanco
0.0
Forma
Cuadrado
1.0
Triángulo
0.0
Círculo
0.0
P(C0)=0.75
P(V|C)
Color
Negro
0.0
Blanco
1.0
Forma
Cuadrado
0.0
Triángulo
0.33
Círculo
0.66
P(C0)=0.50
P(V|C)
Color
Negro
0.0
Blanco
1.0
Forma
Cuadrado
0.0
Triángulo
0.0
Círculo
1.0
P(C0)=0.25
P(V|C)
Color
Negro
0.0
Blanco
1.0
Forma
Cuadrado
0.0
Triángulo
1.0
Círculo
0.0

- Category utility balances:
  - Intra class similarity: $P(A_i = V_{ij}|C_k)$
  - Inter class similarity: $P(C_k|A_i = V_{ij})$
- It measures the difference between a partition of the data and no partition at all
- For qualitative attributes and k categories $\{C_1, \ldots C_k\}$ is defined as:

$$\frac{\sum_{k=1}^{K} P(C_k) \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij}|C_k)^2 - \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij})^2}{K}$$

(see the full derivation on the paper)

- **Incorporate:** Put the example inside an existing class
- **New class:** Create a new class at this level
- **Merge:** Two concepts are merged, and the example is incorporated inside the new class
- **Divide:** A concept is substituted by its children



Procedure: Depth-first limited search COBWEB (x: Example, H: Hierarchy)

Update the father with the new example

if *we are in a leaf* **then**

    Create a new level with this example

else

    Compute **CU** of incorporating the example to each class
    Save the two best **CU**
    Compute **CU** of merging the best two classes
    Compute **CU** of splitting the best class
    Compute **CU** of creating a new class with the example
    Recursive call with the best choice

Partitional algorithms

To find the optimal partition of N objects in K groups is NP-hard, we need **approximated algorithms**

- Model/prototype based algorithms (K-means, Gaussian Mixture Models, Fuzzy K-means, Leader algorithm...)
- Density based algorithms (DBSCAN, DENCLUE...)
- Grid based algorithms (STING, CLIQUE...)
- Graph theory based algorithms (Spectral Clustering...)
- Other approaches:
  - Affinity Clustering
  - Unsupervised Neural networks
  - SVM clustering

Model/Prototype Clustering

- Our model is a set of k hyperspherical clusters
- An iterative algorithm assigns each example to one of K groups (K is a parameter)
- **Optimization criteria**: Minimize the distance of each example to the centroid of the cluster (squared error)

$$Distortion = \sum_{k=1}^{K} \sum_{i \in C_k} \| x_i - \mu_k \|^2$$

- Optimization by a Hill Climbing/gradient descent search algorithm
- The algorithm converges to a local minima

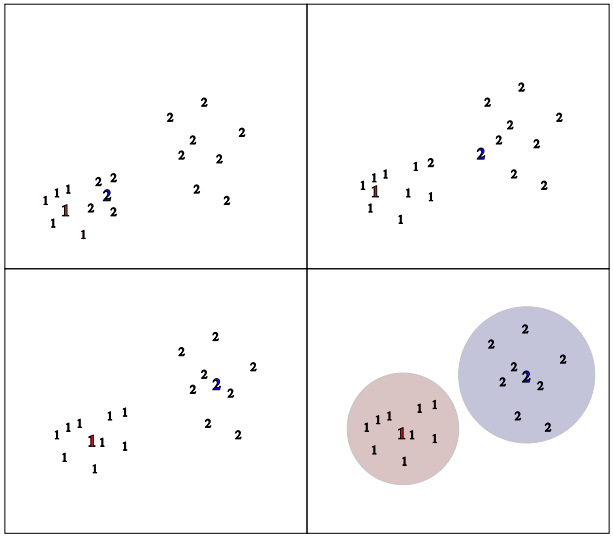
Algorithm: K-means (X: Examples, k:integer)

Generate k initial prototypes (e.g. k random examples)

repeat

> Reassign the examples to their nearest prototype
>
> Recalculate prototypes (centroids)

until *no change in assignments or a number of iterations passes*



- The algorithm is sensitive to initialization (to run from several random initializations is a common practice)
- Sensitive to clusters with different sizes/densities and outliers
- To find the value of k is not a trivial problem
- No guarantee about the quality of the solution
- A solution is found even when not hyper-spherical clusters exist
- The spatial complexity makes it not suitable for large datasets

- K-means++ modifies the initialization strategy
- It tries to maximize distance among initial centers
- **Algorithm:**
  1. Choose one center uniformly from among all the data
  2. For each data point x, compute $d(x, c)$, the distance between x and the nearest center already chosen
  3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $d(x, c)^2$
  4. Repeat Steps 2 and 3 until k centers have been chosen
  5. Proceed with the standard K-means algorithm

- Bisecting K-means iteratively splits one of the current clusters into two until obtaining the desired number of clusters
- Pros:
  - Reduces the effect of initialization
  - A hierarchy is obtained
  - It can be used to determine K
- Con: Different criteria could be used to decide which cluster to split (the largest, the one with the largest variance...)

- **Algorithm:**
  1. Choose a number of partitions
  2. Apply K-means to the dataset with k=2
  3. Evaluate the quality of the current partition
  4. Pick the cluster to be split using a quality criterion
  5. Apply K-means to the cluster with k=2
  6. If the number of clusters is less than desired, repeat from step 3

- Minimizes initialization dependence exploring all clusterings that can be generated using the examples as initialization points
- For generating a partition with K clusters explores all the alternative partitions from 1 to K clusters.
- Pro: Reduces the initialization problem/obtains all partitions from 2 to K
- Con: Computational cost (runs K-means $K \times N$ times)

- **Algorithm:**
  - Compute the centroid of the partition with 1 cluster
  - For C from 2 to K:
    - for each example x, compute K-means initialized with the $C-1$ centroids from the previous iteration and an additional one with x as the C-th centroid
  - Return as solution the partition with K clusters with the best objective function value

- Kernel K-means:
  - Distances are computed using a kernel
  - Pro: Clusters that are non-linearly separable can be discovered (non-convex)
  - Con: Centroids are in the feature space, no interpretation in the original space (image problem)
- Fast K-means (Elkin variant)
  - Use of the triangular inequality to reduce the number of distance computations for assigning examples
- K-Harmonic means
  - Uses the Harmonic mean of the squared distances instead of the distortion as objective function
  - Pro: Less sensitive to initialization

- K-means assumes a centroid can be computed
- In some problems a centroid makes no sense (nominal attributes, structured data)
- One or more examples for each cluster are maintained as a representative of the cluster (medoid)
- The distance from each example to the medoid of their cluster is used as optimization criteria
- Pro: It is not sensitive to outliers
- Con: For one representative the cost per iteration is $O(n^2)$, for more it is NP-hard

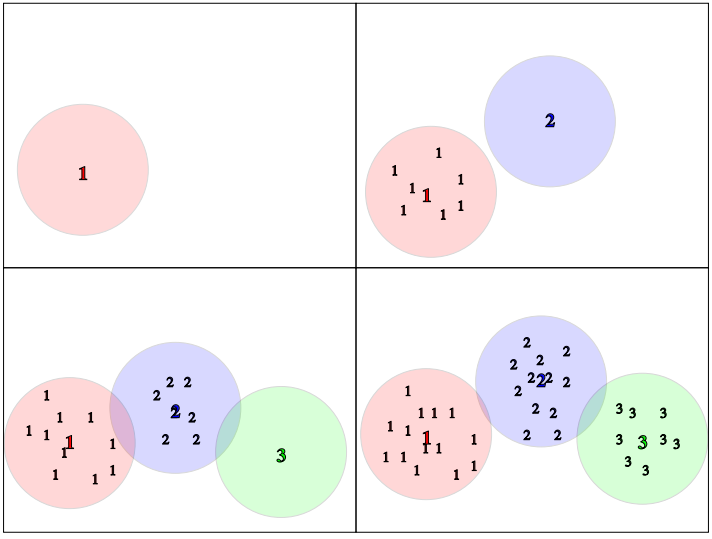
Partitioning Around Medoids (PAM):

1. Randomly select k of the n data points as the medoids
2. Associate each data point to the closest medoids
3. For each medoid m
   - For each non-medoid o: Swap m and o and compute the cost
4. Keep the best solution
5. If medoids change, repeat from step 2

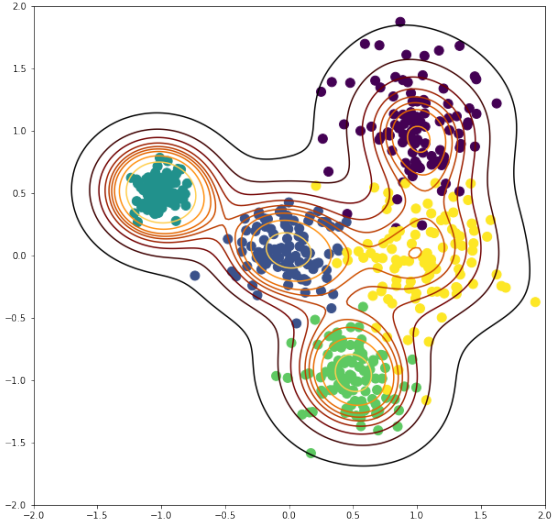
- The previous algorithms need all the data from the beginning
- An incremental strategy is needed when data comes as a stream (Leader Algorithm):
  - A distance/similarity threshold (D) determines the extent of a cluster
  - Inside the threshold: Incremental updating of the model (prototype)
  - Outside the threshold: A new cluster is created
- The threshold D determines the granularity of the clusters
- The clusters are dependent on the order of the examples

Algorithm: Leader Algorithm (X: Examples, D:double)

Generate a prototype with the first example
while *there are examples* **do**
    e= current example
    d= distance of e to the the nearest prototype
    if $d \leq D$ **then**
        Add the example to the cluster
        Recompute the prototype
    else
        Create a new cluster with this example


1
2
3

- We can use a generative approach to clustering
- We assume that data is a mixture of K probability distributions
- The goal is to disentangle the distributions, assigning groups of the data to the distribution that generates them
- Generating the groups we obtain an estimation of the parameters of the distributions
- This give us a generative model that we can sample from


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

- We assume that data are drawn from a mixture of probability distributions (usually Gaussian)
- Search the space of parameters of the distributions to obtain the mixture that explains better the data (parameter estimation)
- The model of the data is:

$$P(x|\theta) = \sum_{k=1}^{K} w_k P(x|\theta_k)$$

with K the number of clusters and $\sum_{k=1}^{K} w_k = 1$, π_k represents the contribution of the distribution k to the mixture

- Each example has a probability to belong to a cluster, Soft partitions

- We can estimate the parameters of the mixed distribution using maximum likelihood
- For the Gaussian case:

$$p(x|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

  Being μ_k the vectors of means and Σ_k the covariance matrices of each Gaussian distribution
- Computing the log likelihood of this distribution and equating the derivative to zero will give us the estimators for the μ_k and Σ_k parameters

- The actual computations depend on the assumptions that we make about the attributes of the data and their distribution
- These assumptions result on different number of parameters to estimate and computational cost
  - Attributes are independent, so the covariance matrices are diagonal and different, but all the attributes of a component share the same variance ($O(d)$ parameters, hyper spheres parallel to coordinate axis)
  - Attributes are independent, so the covariance matrices are diagonal, but with different variances ($O(d)$ parameters, ellipsoids parallel to coordinate axis)
  - The same covariance matrices for all the components ($O(d^2)$ parameters, identical hyper ellipsoids non-parallel to coordinate axis)
  - Full covariance matrix for all the components ($O(d^2)$ parameters, hyper ellipsoids non-parallel to coordinate axis)

- There is not a closed form for the computation of the parameters, so an iterative algorithm has to be used, this is known as Expectation-Maximization (EM)
- The goal is to estimate the parameters of the distribution that describes each cluster (e.g. μ and Σ)
- EM is a general algorithm that is used for the task of parameter estimation by Maximum Likelihood (it is not specific of GMM)
- The algorithm maximizes the likelihood of the distribution with respect to the data
- It performs iteratively two steps:
  - **Expectation:** We calculate a function that assigns to all the examples a degree of membership to all the K probability distributions
  - **Maximization:** We re-estimate the parameters of the distributions to maximize the memberships

- For the case of A independent attributes:

$$p(x|\mu_k, \Sigma_k) = \prod_{j=1}^{A} \mathcal{N}(x|\mu_{kj}, \sigma_{kj})$$

- The model to fit is

$$p(x|\mu, \sigma) = \sum_{k=1}^{K} \pi_k \prod_{j=1}^{A} \mathcal{N}(x|\mu_{kj}, \sigma_{kj})$$

  we have a diagonal covariance matrix

- We can derive the estimators of the parameters using Maximum Likelihood from the log-likelihood of $p(x|\mu, \sigma)$ ($\hat{\mu}$, $\hat{\sigma}$ and $\hat{\pi}$)

- The expectation step computes the weights for each example and component (responsibilites)

$$\hat{\gamma}_{ik} = w_k P(x_i|\mu_k, \sigma_k)$$

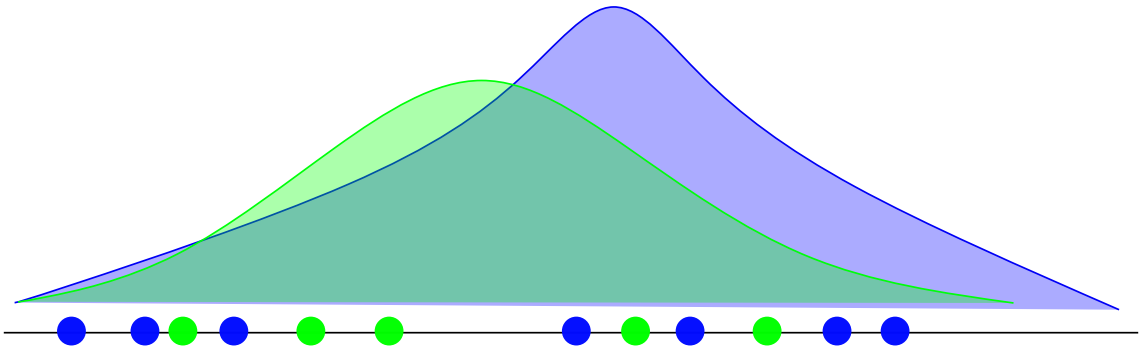
- This represents the probability that an example x_i is generated by component C_k

- The maximization steps recomputes μ, σ, and w for each component proportionally to the weights computed in the expectation step

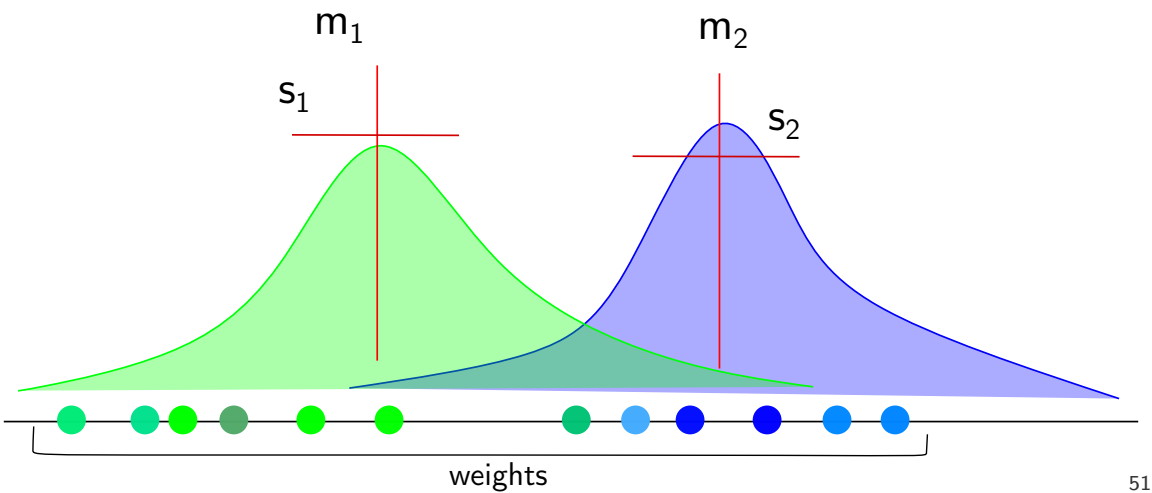
$$
\begin{aligned}
\hat{\mu}_k &= \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} x_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}} \\
\hat{\sigma}_k &= \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} (x_k - \hat{\mu}_i)^2}{\sum_{i=1}^{N} \hat{\gamma}_{ik}} \\
\hat{w}_k &= \frac{1}{N} \sum_{k=1}^{N} \hat{\gamma}_{ik}
\end{aligned}
$$

- K initial distributions are generated $\mathcal{N}(\mu_k, \sigma_k)$, defining their parameters μ_k, and σ_k, K-means can be used as initial estimate
- Repeat until convergence (no log-likelihood improvement):
  1. **Expectation:** Compute the membership of each example to each mixture component
     - Each instance will have a weight ($\hat{\gamma}_{ik}$, responsibility) computed from on the components computed by the previous iteration that indicate how likely the example is from a component
  2. **Maximization:** Recompute the parameters estimators using the weights from the previous step to obtain the new $\hat{\mu}$, $\hat{\sigma}$, and $\hat{\pi}$ for each distribution. Recompute the log-likelihood.
- Finally, each example has a probability to each component that corresponds to the responsibility (soft assignment)

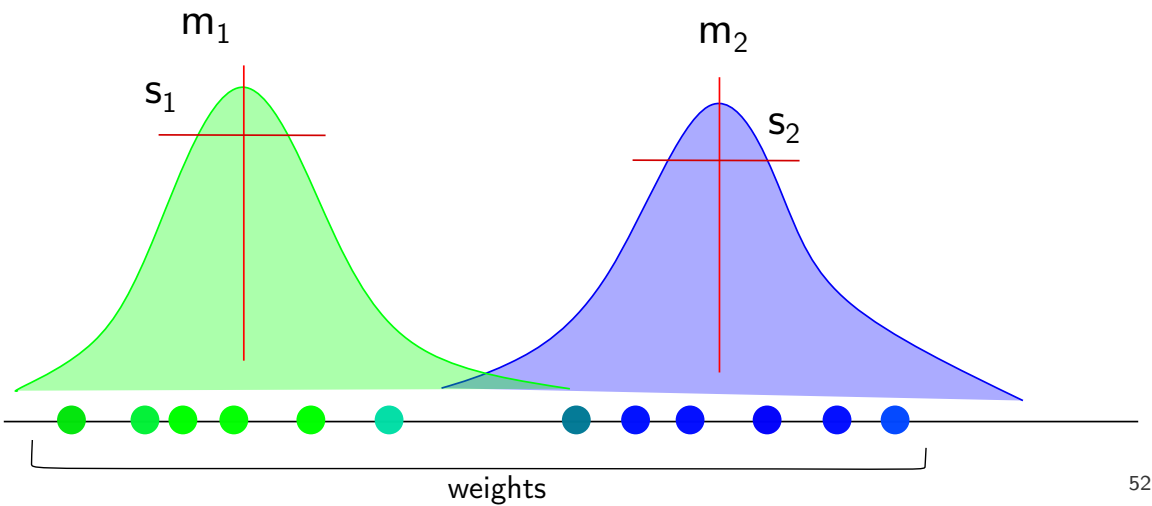
Initial Assignment



Expectation + Maximization = new parameters

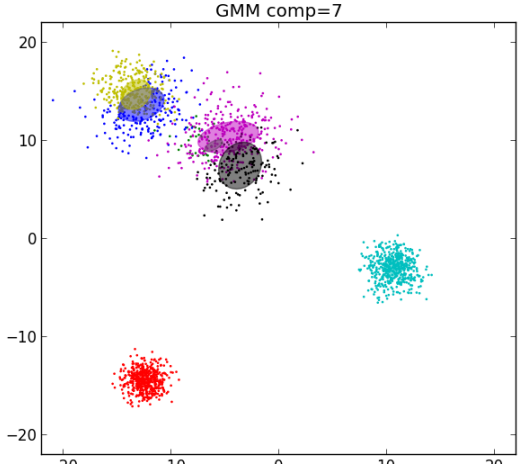


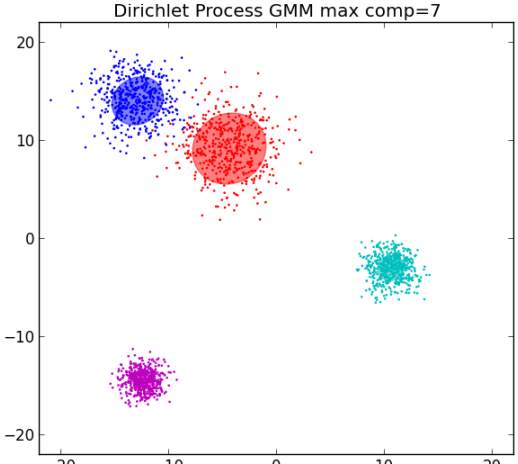
Expectation + Maximization = new parameters



- K-means is a particular case of this algorithm (hard partition)
- The main advantage is that we obtain a membership as a probability (soft assignments)
- Using different probability distributions we can find different kinds of structures in the data
- For each probability model we use we need to derive the calculations for the iterative updating of their parameters
- This is a generative model, we can obtain new data sampling from the estimated distribution

- One of the problems of GMM is to decide a priori the number of components
- This can be included in the model following a bayesian approach using a mixture model that includes as priors for the weights of the components a Dirichlet Process distribution (represents the distribution of the number of mixtures and their weights)
- Dirichlet Process distribution assumes an unbound number of components
- A finite weight is distributed among all the components
- The fitting of the model will optimize the weights of the components, and some can be reduced to 0, eliminating some of them


GMM comp=7
20
10
0
−10
−20
−20
−10
0
10
20


Dirichlet Process GMM max comp=7
20
10
0
−10
−20
−20
−10
0
10
20

- Fuzzy clustering relax the hard partition constraint of K-means
- Each example has a continuous membership to each partition
- A new optimization function is introduced:

$$L = \sum_{i=1}^{N} \sum_{k=1}^{K} \delta(C_k, x_i)^b \|x_i - \mu_k\|^2$$

with $\sum_{k=1}^{K} \delta(C_k, x_i) = 1$, and b is a blending factor

- When the clusters are overlapped this is an advantage over hard partition algorithms

- C-means is the most known fuzzy clustering algorithm, it is the fuzzy version of K-means
- Membership is computed as the normalized inverse distance to all the clusters
- The updating of the cluster centers is computed as:

$$\mu_j = \frac{\sum_{i=1}^{N} \delta(C_j, x_i)^b x_i}{\sum_{i=1}^{N} \delta(C_j, x_i)^b}$$

- And the updating of the memberships:

$$\delta(C_j, x_i) = \frac{(1/d_{ij})^{1/(1-b)}}{\sum_{k=1}^{K} (1/d_{ik})^{1/(1-b)}}, d_{ij} = \|x_i - \mu_j\|^2$$

Notice that this is similar to a **softmax** of the distances to the centroids

- The C-means algorithm looks for spherical clusters, other alternatives:
  - Gustafson-Kessel algorithm: A covariance matrix is introduced for each cluster in the objective function that allows elipsoid shapes and different cluster sizes
  - Gath-Geva algorithm: Adds to the objective function the size, and an estimation of the density of the cluster
- Different objective functions can be used to detect specific shapes in the data (lines, rectangles...)

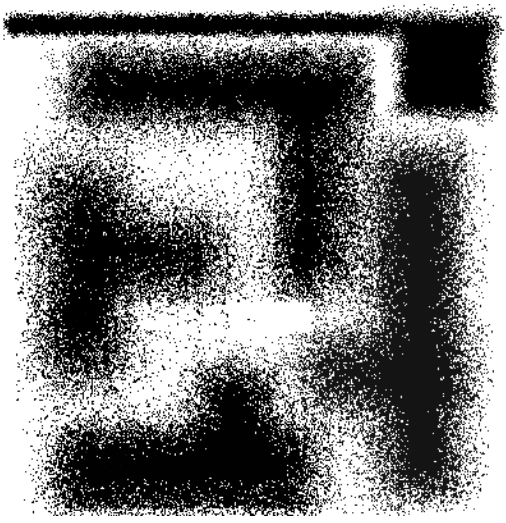
This Python Notebook shows examples of using different the K-means and GMM and their problems

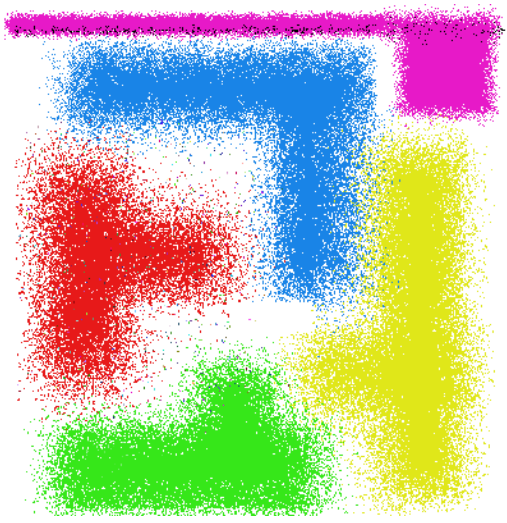
- Prototype Based Clustering Algorithms Notebook (click here to open the notebook in colab)

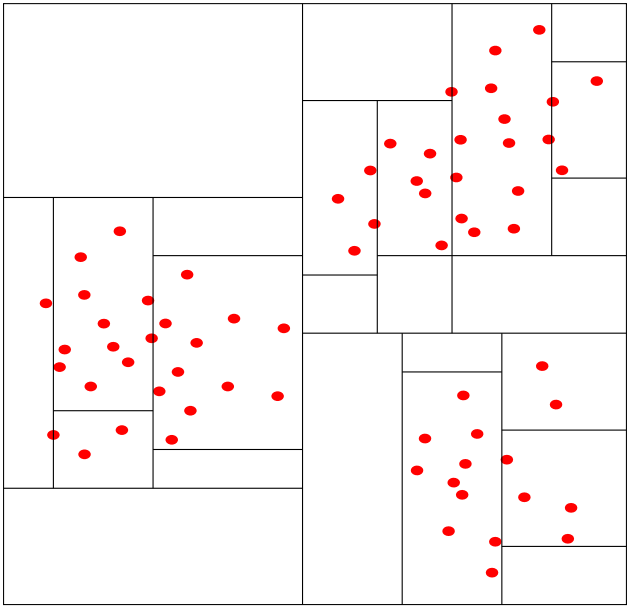
If you download the notebook you will be able to use it locally (run jupyter notebook to open the notebooks)

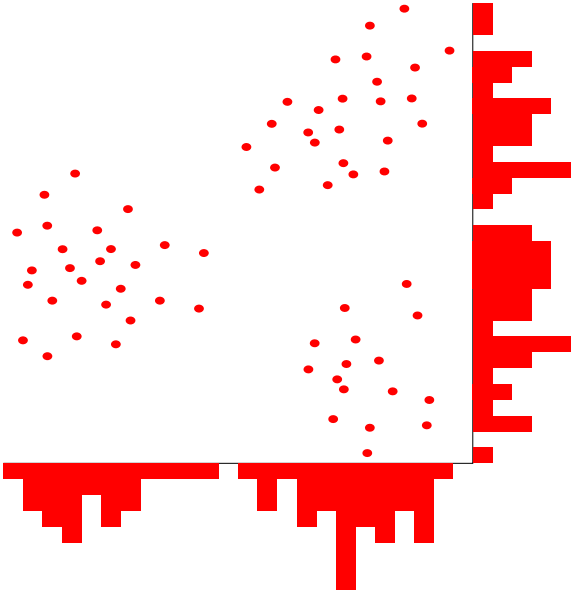
Density/Grid Clustering

- The number of clusters is not decided beforehand
- We are looking for regions with high density of examples
- We are no limited to predefined shapes (there is no model)
- Different approaches:
  - Density estimation
  - Grid partitioning
  - Multidimensional histograms
- Usually applied to datasets with low dimensionality









Ester, Kriegel, Sander, Xu **A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise (DBSCAN)** (1996)

Ankerst, Breunig, Kriegel, Sander **OPTICS: Ordering Points To Identify the Clustering Structure** (2000)

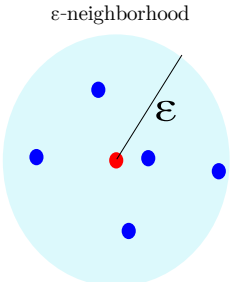
- Used in spatial databases, but can be applied to data with more dimensionality
- Based on finding areas of high density, it finds arbitrary shapes

- We define ε-**neighbourhood**, as the examples that are at a distance less than ε to a given instance

$$N_\varepsilon(x) = \{y \in X | d(x, y) \leq \varepsilon\}$$

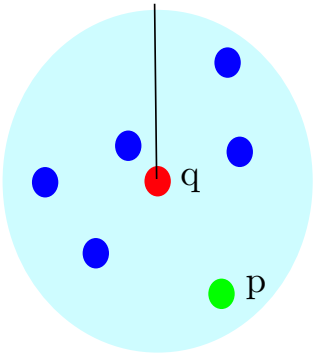
- We define **core point** as the examples that have a certain number of elements in $N_\varepsilon(x)$

$$Core\_point \equiv |N_\varepsilon(x)| \geq MinPts$$

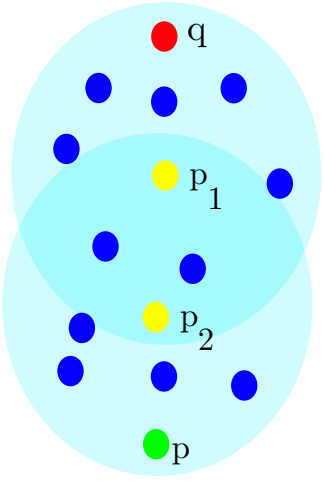


- Two examples p and q are **Direct Density Reachable** with respect to ε, and $MinPts$ if:
  1. $p \in N_\varepsilon(q)$
  2. $|N_\varepsilon(q)| \geq MinPts$
- Two examples p and q are **Density Reachable** if there is a sequence of examples $p = p_1, p_2, \ldots, p_n = q$ where for all p_i, p_{i+1} is **Direct Density Reachable** from p_i
- Two examples p and q are **Density connected** if there is an example o such that both p, and q are **Density Reachable** from o

p DDR q



p DR q



Cluster

Given a dataset D, a cluster C with respect ε and $MinPts$ is any subset of D that:

1. $\forall p, q \; p \in C \wedge density\_reachable(q, p) \longrightarrow q \in C$
2. $\forall p, q \in C \; density\_connected(p, q)$

Any example that can not be connected using these relationships is treated as noise

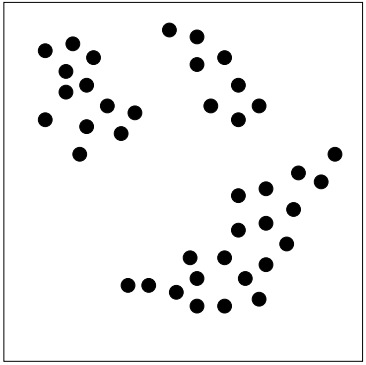
Algorithm

1. We start with an arbitrary example and compute all density reachable examples with respect ε and $MinPts$.
2. If it is a core point we will obtain a group, otherwise, it is a border point and we will start from other unclassified instance

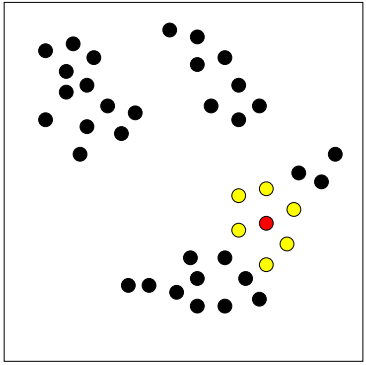
To decrease the computational cost R^* trees are used to store and compute the neighborhood of instances

ε and $MinPts$ are set from the thinnest cluster

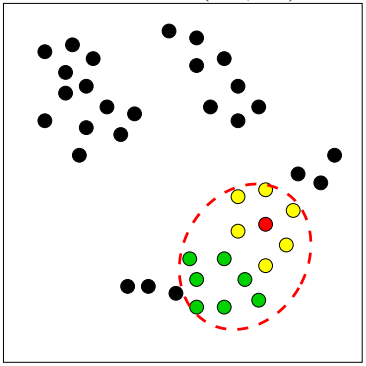
Datos Epsilon, MinPts=5



First Iteration (DDR)



First Iteration (DR,DC)



- The OPTICS algorithm is an improvement over DBSCAN to detect automatically the clusters in the data
- Two distances are defined used to compute the clusters that can be obtained for any value less than a given ε.
  - The core distance of an example, that is the smallest distance of a core point to the closest example in its ε neighborhood
  - The reachability distance between two examples p and o (core point), the smallest distance such p is directly density reachable from o
- This information can also be used to order the examples by cluster and reachability distance obtaining a reachability plot
- The assignments can be decided using a parameter ξ that defines how much must change the reachability distance between two consecutive examples to consider that we have reached the border of the cluster

This Python Notebook compares Prototype Based and Density Based Clustering algorithm

- Density Based Clustering Notebook (click here to go to the url)

If you download the notebook you will be able to use it locally (run jupyter notebook to open the notebooks)

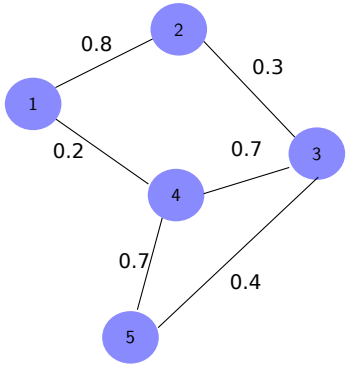
Other Approaches

- Spectral Graph Theory defines properties that hold the eigenvalues and eigenvectors of the adjacency matrix or Laplacian matrix of a graph
- Spectral clustering uses the spectral properties of the similarity matrix
- The distance matrix represents the graph that connects the examples
  - Complete graph
  - Neighborhood graph (different definitions)
- Different clustering algorithms can be defined from the diagonalization of this matrix

- We start with the similarity matrix (W) of the data
- The degree of a vertex is defined as:

$$d_i = \sum_{j=1}^{n} w_{ij}$$

- We define the degree matrix D as the diagonal matrix with values $d_1, d_2, \ldots, d_n$
- We can define different Laplace matrices:
  - Unnormalized: $L = D - W$
  - Normalized: $L_{sym} = D^{-1/2} L D^{-1/2}$ or also $L_{rw} = D^{-1} L$

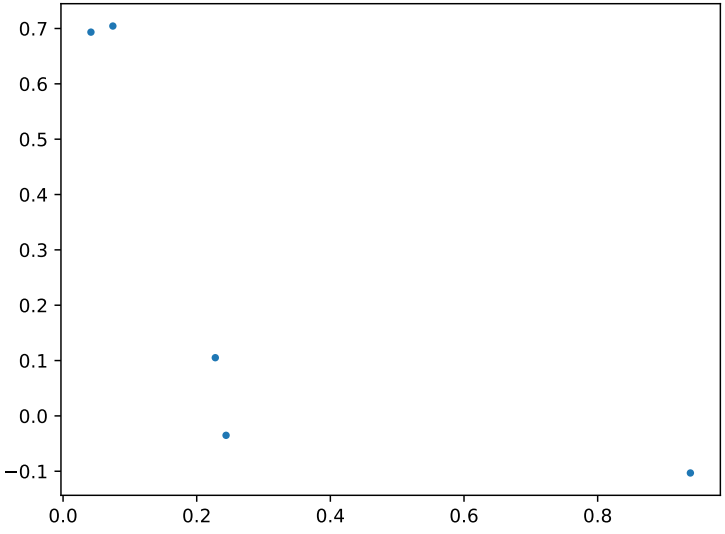


$$W = \begin{pmatrix} 0 & 0.8 & 0 & 0.2 & 0 \\ 0.8 & 0 & 0.3 & 0 & 0 \\ 0 & 0.3 & 0 & 0.7 & 0.4 \\ 0.2 & 0 & 0.7 & 0 & 0.7 \\ 0 & 0 & 0.4 & 0.7 & 0 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.1 & 0 & 0 & 0 \\ 0 & 0 & 1.4 & 0 & 0 \\ 0 & 0 & 0 & 1.6 & 0 \\ 0 & 0 & 0 & 0 & 1.1 \end{pmatrix}$$

- **Algorithm:**
  1. Compute the Laplace matrix from the similarity matrix
  2. Compute the first K eigenvalues of the Laplace matrix
  3. Use the eigenvectors as new datapoints
  4. Apply K-means as clustering algorithm
- We are actually embedding the dataset in a space of lower dimensionality

$$L = \begin{pmatrix} 1 & -0.8 & 0 & -0.2 & 0 \\ -0.8 & 1.1 & -0.3 & 0 & 0 \\ 0 & -0.3 & 1.4 & -0.7 & -0.4 \\ -0.2 & 0 & -0.7 & 1.6 & -0.7 \\ 0 & 0 & -0.4 & -0.7 & 1.1 \end{pmatrix}$$

$$Eigvec(1,2) = \begin{pmatrix} 0.07 & 0.70 \\ 0.04 & 0.69 \\ 0.22 & 0.10 \\ 0.93 & -0.10 \\ 0.24 & 0.03 \end{pmatrix}$$


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0.0
0.2
0.4
0.6
0.8

- From the similarity matrix and its Laplacian it is possible to formulate it as a graph partitioning problem
- Given two disjoint sets of vertex A and B, we define:

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

- We can partition the graph solving the mincut problem choosing a partition that minimizes :

$$cut(A_1, \ldots, A_k) = \sum_{i=1}^{k} cut(A_i, \overline{A_i})$$

- Using directly the weights of the Laplacian not always gives good results, alternative objective functions are:

$$RatioCut(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \overline{A_i})}{|A_i|} \quad (1)$$

$$Ncut(A_1, \ldots, A_k) = \sum_{i=1}^{k} \frac{cut(A_i, \overline{A_i})}{vol(A_i)} \quad (2)$$

- Where $|A_i|$ is the size of the partition and $vol(A_i)$ is the sum of the degrees of the vertex in A_i

- Affinity clustering is a message passing algorithm related to graph partitioning and belief propagation in probabilistic graphical models
- It chooses a set of examples as the cluster prototypes and computes how the rest of examples are attached to them
- Each pair of examples have a similarity defined $s(i, k)$
- Each example has a value $r(k, k)$ that represents the preference for each point to be an exemplar
- The algorithm does not set apriori the number of clusters,

- The examples pass two kind of messages
  - Responsibility $r(i,k)$, this is a message that an example i passes to the candidate to exemplars k of the point. This represents the evidence of how good is k for being the exemplar of i
  - Availability $a(i,k)$, sent from candidate to exemplar k to point i. It represents the accumulated evidence of how appropriate would be for point i to choose point k as its exemplar

- All availabilities are initialized to 0
- The responsibilities are updated as:

$$r(i,k) = s(i,k) - \max_{k' \neq k}\{a(i,k') + s(i,k')\}$$

- The availabilities are updated as:

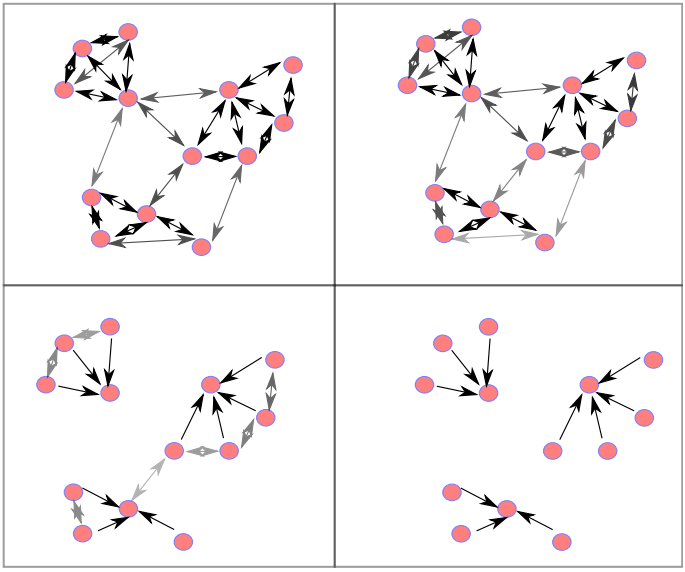
$$a(i,k) = \min\{0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k))\}$$

- The self availability $a(k,k)$ is updated as:

$$a(k,k) = \sum_{i' \neq k} \max(0, r(i',k))\}$$

- The exemplar for a point is identified by the point that maximizes $a(i,k) + r(i,k)$, if this point is the same point, then it is an exemplar.

1. Update the responsibilities given the availabilities
2. Update the availabilities given the responsibilities
3. Compute the exemplars
4. Terminate if the exemplars do not change in a number of iterations



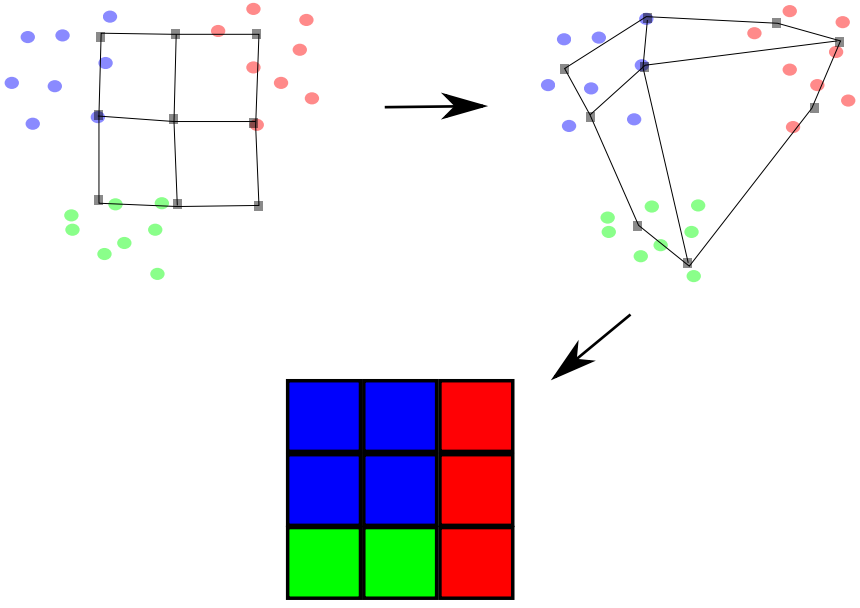
- Self-organizing maps are unsupervised neural networks
- Can be seen as an on-line constrained version of K-means
- The data is transformed to fit in a 1-d or 2-d regular mesh
- The nodes of this mesh are the prototypes
- This algorithm can be used as a dimensionality reduction method (from N to 2 dimensions)

- To build the map we have to decide the size and shape of the mesh (rectangular/hexagonal)
- Each node a multidimensional prototype of p features

Algorithm: Self-Organizing Map algorithm

Initial prototypes are distributed regularly on the mesh
for *Predefined number of iterations* **do**
  foreach *Example* x_i **do**
    Find the nearest prototype (m_j)
    Determine the neighborhood of m_j ($\mathcal{M}$)
    foreach *Prototype* $m_k \in \mathcal{M}$ **do**
      $m_k = m_k + \alpha(x_i - m_k)$

- Each iteration transforms the mesh closer to the data, maintaining the 2D relationship between prototypes
- The neighborhood of a prototype is defined by the adjacency of the cells and the distance of the prototypes
- Performance depends on the learning rate α, that is usually decreased from 1 to 0 during the iterations
- The number of neighbors used in the update is decreased during the iterations from a predefined number to 1
- Some variations of the algorithm use the distance of the prototypes as weights for the update

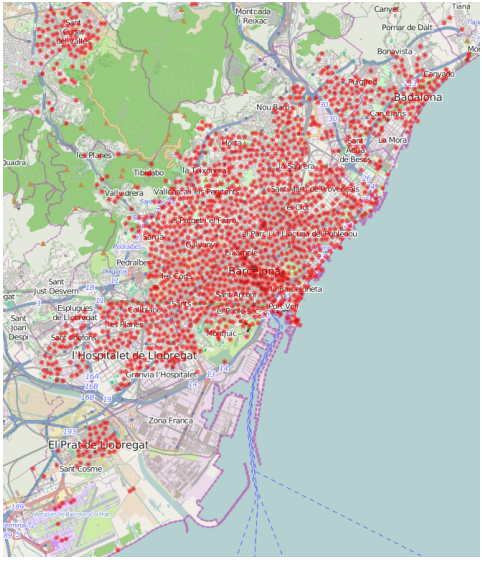


Applications

- Goal: To analyze the geographical behavior of people living/visiting a city
- Dataset: Tweets/posts inside a geographical area
- Attributes: geographical information / time stamp of the post
- Processes:
  - Geographical discretization for user representation
  - Discovery of behavior profiles

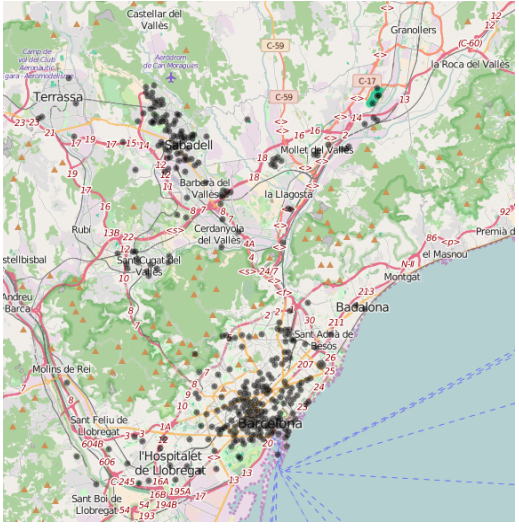
- Focus on the discovery of different groups of people
- The hypothesis is that users can be segmented according to where they are at different times of the day
- We could answer questions like:
  - people from one part of the city usually stay in that part?
  - people from outside the city go to the same places?
  - public transportation is preferred by different profiles? ...

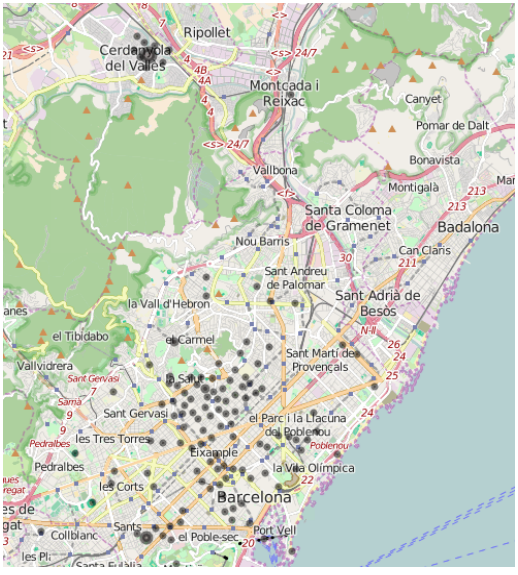
- Raw geolocalization and timestamp are too fine grained
- Even when we have millions of events the probability of having two events at the same place and at the same time is low
- Discretizing localization and time will increase the probability of finding patterns
- How do we discretize space and time? Clustering

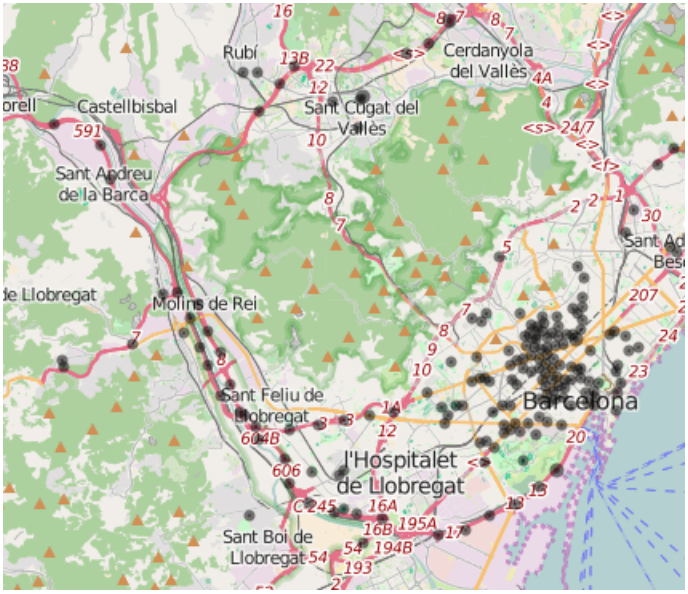

Badalona
Barcelona
l'Hospitalet de Llobregat
El Prat de Llobregat
Zona Franca
Sant Cosme
Montcada i Reixac
Canyet
Tiana
Pomar de Dalt
Bonavista
Carmelo
Nou Barris
Can Peixau
La Mora
Les Planes
Tibidabo
la Teixonera
Valldrebra
Vallvidrera, el Tibidabo i les Planes
Sarrià
Pedralbes
les Corts
Sant Andreu
Sant Martí
Poble Nou
Sant Just Desvern
Esplugues de Llobregat
Sant Joan Despí
Sant Feliu de Llobregat
Montjuïc
Granvia l'Hospitalet
Cerdanyola del Vallès
Sant Cugat del Vallès
Portal
la Sagrera
Sant Martí de Provencals
el Clot
el Parc i la Llacuna del Poblenou
Gràcia
Eixample
Sants
el Poble Sec
Port Vell
la Barceloneta
Sant Antoni
Collblanc
les Planes
Sant Ildefons

- We could explore the aggregated behavior of a user for a long period (month, year)
- Each example represents the places/times of the events of a user for the period
- We obtain a representation similar to the *Bag of Words* used in text mining
  - User $\Rightarrow$ Document
  - Time $\times$ Place $\Rightarrow$ Word

- Difficult to choose the adequate clustering algorithm
- Some choices will depend on:
  - Types of attributes: continuous or discrete values, sparsity of the data
  - Size of the dataset: Aggregating the events reduces largely the size of the data (no scalability issues)
  - Assumptions about the model that represents our goals: Shape of the clusters, Separability of the clusters/Distribution of examples
  - Interpretability/Representability of the clusters


Castellar del Vallés
Granollers
la Roca del Vallés
Terrassa
Sabadell
Mollet del Vallés
Barberà del Vallés
la Llagosta
Rubí
Cerdanyola del Vallés
Premià d
el Masnou
Montgat
Badalona
Sant Adrià de Besós
Molins de Rei
Barcelona
Sant Feliu de Llobregat
l'Hospitalet de Llobregat
Sant Boi de Llobregat


Ripollet
Cerdanyola del Vallés
Montcada i Reixac
Canyet
Pomar de Dalt
Bonavista
Vallbona
Montigalà
Santa Coloma de Gramenet
Badalona
Nou Barris
Can Clanis
Sant Andreu de Palomar
Sant Adrià de Besós
la Vall d'Hebron
el Tibidabo
el Carmel
Sant Martí de Provençals
Vallvidrera
Sant Gervasi
el Parc i la Llacuna del Poblenou
les Tres Torres
Eixample
Pedralbes
la Vila Olímpica
les Corts
Barcelona
Sants
Collblanc
el Poble-sec
Port Vell


Rubí
Cerdanyola del Vallès
Castellbisbal
Sant Cugat del Vallès
Sant Andreu de la Barca
Molins de Rei
Sant Feliu de Llobregat
Barcelona
l'Hospitalet de Llobregat
Sant Boi de Llobregat