Unsupervised Machine Learning and Data Mining

(Lecture Notes)

Javier Béjar

AMLT

Master in Artificial Intelligence

Contents

Introduction

These are the lecture notes for the first part of the course Advanced Machine Learning Techniques (AMLT). The idea of these notes is to facilitate to follow the course and to allow a more dynamic work in class. It is necessary that you read the material corresponding to each scheduled class beforehand, so you can use and practice the different techniques programmed for the class.

The different chapter have examples using several machine learning libraries mainly developed in python. It is important that you have installed these libraries in your computer so you can run the different examples and make your own experiments.

Most of the examples use the `scikit-learn` library, that is available for any python distribution using `pip` or `easy_install`. Some methods are implemented in the library `kemlg-learn` that you can find in

1 Data Preprocessing

1.1 Datasets

Each domain has its own representation according to its characteristics. This representation will result on a set of attributes that describe what interest us from the data and what we want to use for the discovery process. According to this representation we can categorize the different kinds of datasets in two groups.

The first kind is those that can be represented by a flat structure, we will call them *unstructured* datasets. These datasets can be described using a list of attributes and an attribute-value matrix. We assume that there is no structural/semantic relation among the different attributes that describe the data (or we are not interested on it). There will be only one database table/file of observations, where each example is an instance of the problem. The examples will be represented by a set of attributes (discrete, continuous, . . .), where each row corresponds to a realization of the attributes. For example, the table 1.1 shows a dataset described by three attributes: A (binary), B (real) and C (discrete).

In the second kind, the *structured* datasets, the attributes of the examples or the examples itself are related. In this case we are more interested in the patterns that the relationships contain. These datasets correspond to sequences (time related or with a domain specific order relationship) and tree/graph structures. In a sequence we have an order (total or partial) relationship among all the examples of the dataset. This order usually corresponds with a time relationship (time series) or a domain defined precedence like nucleotids order in DNA. This sequence can contain itself structured examples, being a sequence of unstructured data, for instance, text obtained from a news feed or purchases from an online store. It also can be a sequence of complex data, like small sequences interleaved in time, such as click behavior of users in a website, where each user generates a sequence of events, with all that sequences

A	B	C	⋯
1	3.1	green	⋯
1	5.7	blue	⋯
0	-2.2	blue	⋯
1	-9.0	red	⋯
0	0.3	yellow	⋯
1	2.1	green	⋯
⋮	⋮	⋮	⋱

Figure 1.1: Example of the representation of an unstructured dataset

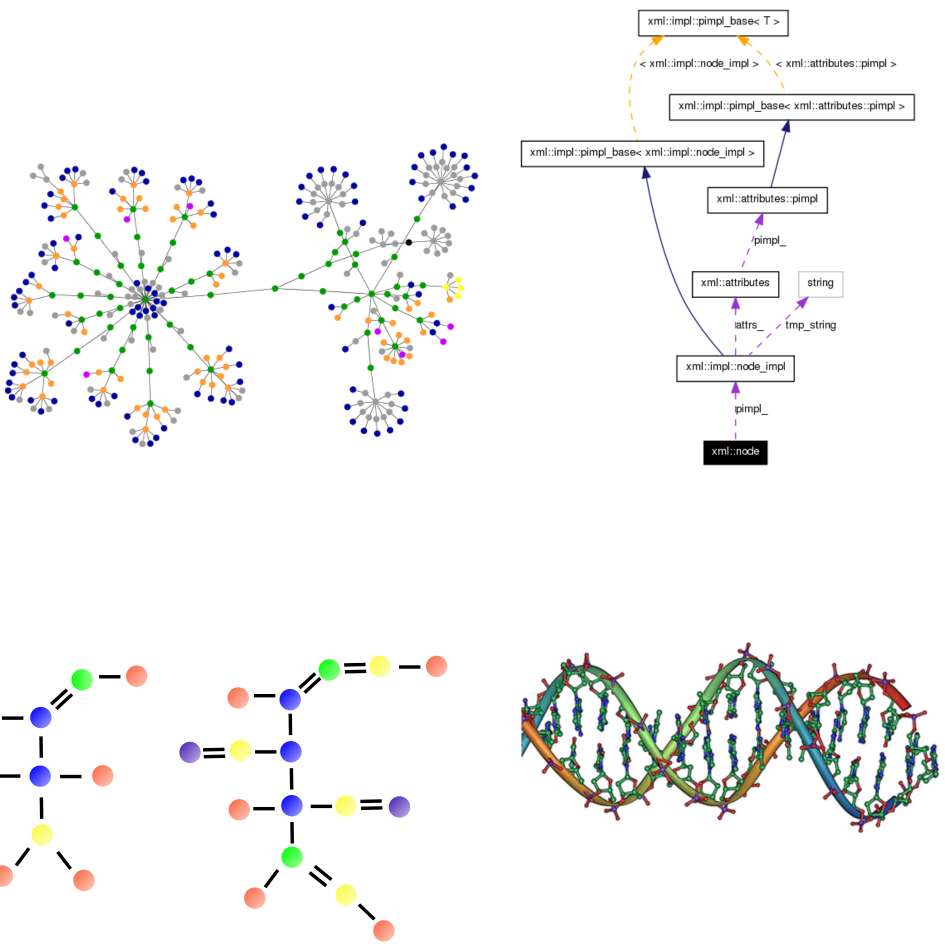

xml::impl::pimpl_base< T >
< xml::impl::node_impl >
< xml::attributes::pimpl >
xml::impl::pimpl_base< xml::attributes::pimpl >
xml::impl::pimpl_base< xml::impl::node_impl >
xml::attributes::pimpl
pimpl_
xml::attributes
string
attrs_
tmp_string
xml::impl::node_impl
pimpl_
xml::node

Figure 1.2: Examples of structured datasets (Social Network, XML graph, Molecules database, DNA sequence)

conforming the sequential dataset.

With tree and graph structures we have more complex relationships among the attributes. These kinds of data can also be just and instance, (e.g. a social network) or be a set of structured examples (eg. XML documents, graph representation of molecules, relational database). Figure 1.2 shows some examples of these kinds of datasets.

1.2 Data Preprocessing

Usually raw data is not directly adequate for analysis for different reasons. For instance, the quality of the data could not be adequate because of noise in their attribute values, the existence of examples with missing values, or extreme values on their attributes (outliers). Also, sometimes the dimensionality of the attributes is too high to be efficiently processed because there are too many attributes or too many examples.

The first step before performing any data analysis task is to assess the quality of the data checking for any of the mentioned problems. Each one of them can be addressed using different techniques and algorithms.

1.2.1 Outliers

Data outliers are defined as examples that have values in some or all of their attributes that deviate significantly from the rest of the values of the dataset. A simple way of deal with these

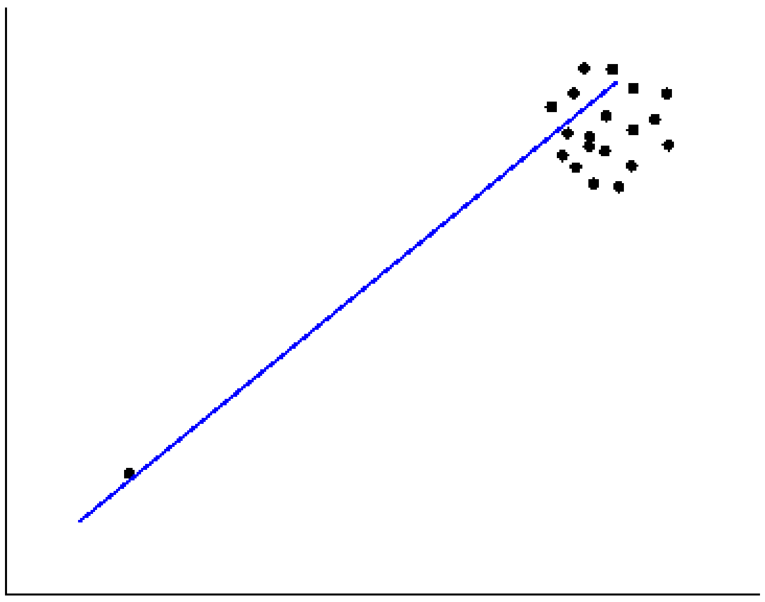


Figure 1.3: Effect of outliers

data is to delete the examples. If the exceptional values appear only in a few of the attributes the actual values could be ignored and treated as missing values.

There are different algorithms for detecting outliers in data that can be divided into parametric and non-parametric methods.

Parametric methods assume a specific probability distribution for the attributes, uses the data to estimate its parameters, and then compute the probability of the values. Those that have a probability lower than a threshold are marked as outliers. Other possibility is to use the upper and lower quartiles of the distribution and mark as outliers all the values that are at a distance larger than a threshold from the quartiles.

Sometimes it is not possible to assume a model for the distribution of the values, so a non parametric method have to be used. There are different possibilities, for example, to compute a histogram for the attributes (deciding an adequate binning) and discarding the examples that are from bins that have a low number of examples. Other possible solution is to apply kernel density estimation (Parzen Window) and labeling as outliers the examples that belong to low density areas.

All these methods detect attributes that have outlier values. An example with only a few outliers has still valuable information, so a threshold has to be determined to decide when the example is different enough to be considered an outlier respect to the rest of the data.

Another non parametric methods use a proximity criteria to determine if the example itself is an outlier by using the distribution of the distances to the k-nearest neighbors of each example. If this distribution is different enough from the mean behavior, then the example can be discarded.

An excellent review on this topic can be found on [30].

1.2.2 Missing Values

Missing values usually appear because of errors or omissions during the data collection. This circumstance has an impact on the quality of the data. The process of substituting these data is called missing values *imputation*.

There are several methods for computing the value to be used to substitute the missing data. These are some of them:

- To use a global constant for all the values (domain/variable dependent)

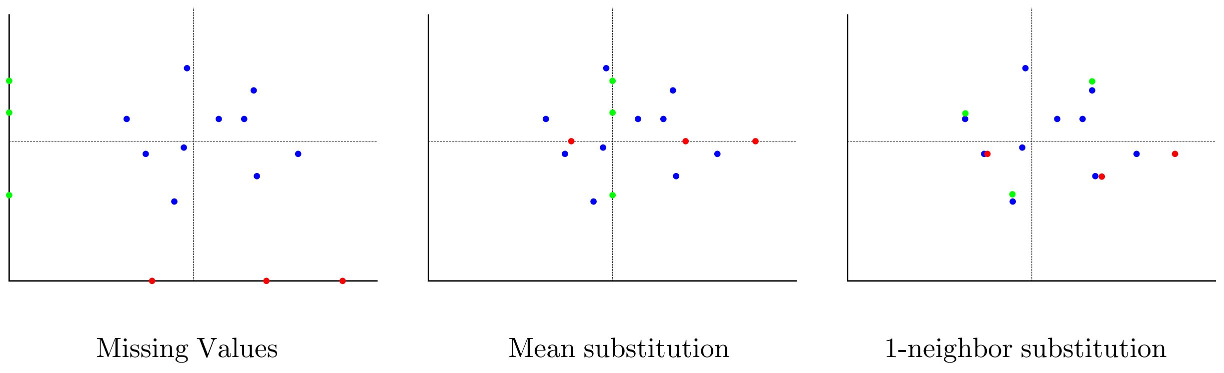


Figure 1.4: Effect of different strategies of missing values imputation

- To use the mean or mode of the attribute (global central tendency)
- To use the mean or mode of the attribute but only of the k nearest examples (local central tendency)
- To learn a model for the data (regression, bayesian) and use it to predict the missing values

Each one of them has different computational cost, the first one is $O(1)$ for each missing value. The second one requires first computing the statistical moments of the variable $O(|examples|)$ and then the cost is constant for each missing value. The third one needs to compute for each example with missing values the k nearest examples, this involves $O(|examples| \times \log(k))$, and then computing the moments of the k examples. Finally, the last one involves to learn a model for each variable, the cost will depend on the cost of building the model.

Something that we have to be aware of is that the statistical distribution of the data is changed by the imputation of missing values. Using the same value for all the imputations will reduce the variance of the variable, this is what the first two methods do. The rest will have less impact in the variance. Figure 1.4 represents the effect of missing values imputation using the mean and the 1-nn.

1.2.3 Data normalization

Data normalization is applied to quantitative attributes in order to eliminate the effects of having different scale measures. To have disparate scales in the attributes makes difficult to compare the data and affect the measure of similarity or distance among examples. This normalization can be explicit, a new dataset is generated, or implicit, the data is normalized when it is processed by the algorithms that are used.

Range normalization

It transforms all the values of the attributes to a predetermined scale (for instance $[0, 1]$ or $[-1, 1]$). This can be computed from the range of values of an attribute, the following transformation changes the values of a variable to the range $[0, 1]$

$$range\_norm(x) = \frac{x - x_{min}}{x_{max} - x_{min}}$$

From this normalization other ranges can be obtained simply by translation and scaling.

Standard score normalization or Z-normalization:

It transforms the data assuming gaussian distribution, using the attributes empirical moments (μ_x, σ_x) so all the attributes have $\mu = 0$ and $\sigma = 1$. The following transformation performs this change:

$$Z\_norm(x) = \frac{x - \mu_x}{\sigma_x}$$

1.2.4 Data discretization

Sometimes the process that we are going to apply to the data does not admit continuous values, or it is more simple to understand the results if the values are discrete. Data discretization allows transforming attribute values from quantitative to qualitative. There are different ways of performing this task, we are only going to describe the unsupervised ones.

The task is to decide how to partition the set of continuous values in a set of bins computing the limits between consecutive bins. The number of bins is usually fixed beforehand using domain information, or can be adjusted using some quality measure.

- **Equal length bins:** The range between the maximum and the minimum value is divided in as many intervals of equal length as the desired bins. Values are discretized accordingly to the interval they belong to.

- **Equal frequency bins:** The intervals are computed so each bin has the same number of examples, the length of the intervals will be consequently different. The cutting points are usually computed as the mean of the examples that are in the extremes of consecutive bins.

- **Distribution approximation:** A histogram of the data can be computed (using a specific number of bins) and a parameterized function is fitted to approximate the frequencies (e.g. a polynomial). The intervals are selected computing the different minimums of the function. Alternatively, Kernel Density Estimation (non parametric) can be applied to approximate the different densities of the data, and determining the intervals by finding the areas with lower density.

These three methods are represented in figure 1.5. Several other techniques can be applied to unsupervisedly detect the different modalities of the data, like for example entropy based measures, Minimum Description Length or even clustering.

As an example, figure 1.6 shows the effect of equal width and equal frequency discretization using five bins for the well known Iris dataset. It can be observed the different distribution of the examples on the bins for each one of the discretizations, so jittering has been used so the amount of examples in each combination can be appreciated.

Supervised discretization methods use class information to decide the best way to slit the data and the optimal number of bins. Unfortunately, this information is not always available.

1.3 Dimensionality reduction

Before applying a specific mining task on a dataset, several preprocessing steps can be done. The first goal of the preprocessing step is to assure the quality of the data by reducing the noisy and irrelevant information that it could contain, that be obtained with the methods described on the previous sections. The second goal is to reduce the size of the dataset, so the

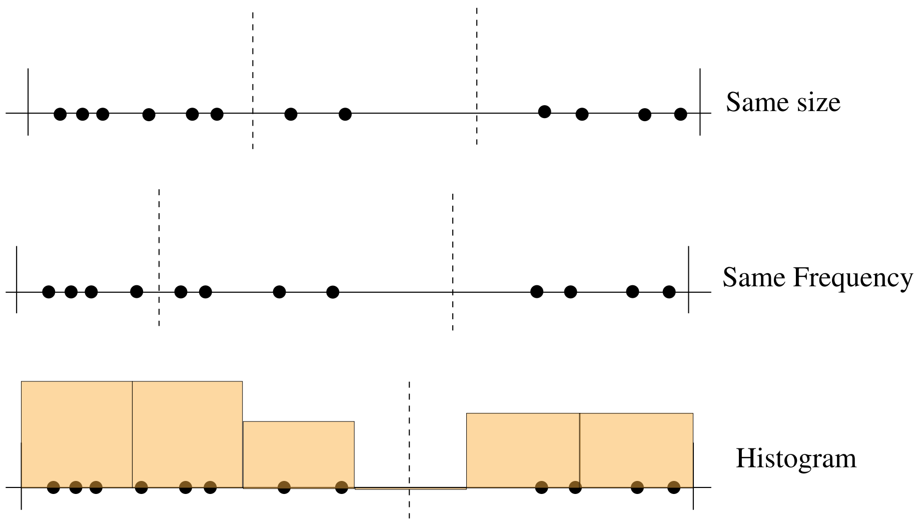


Figure 1.5: Attribute discretization methods

computational cost of the discovery task is also reduced. This reduction process can also have the purpose of visualizing the data to help the discovery process.

There are two dimensions that have to be taken into account when reducing the size of a dataset. The first one is the number of instances. This problem can be addressed by sampling techniques when it is clear that a smaller subset of the data holds the same information that the whole dataset. Not for all applications it is the case, and sometimes the specific goal of the mining process is to find particular groups of instances with low frequency, but of high value. These data could be discarded by the sampling process, making unfruitful the process. In other applications, the data is a stream, this circumstance makes more difficult the sampling process or carries the risk of losing important information from the data if its distribution changes over time.

Dimensionality reduction addresses the number of attributes of the dataset by transforming it from its original representation to one with a reduced set of features. These techniques come from very different areas, but the common goal is to obtain a new dataset that preserves, up to a level, the original structure of the data, so its analysis will result in the same or equivalent patterns present in the original data.

Broadly, there are two kinds of methods for reducing the attributes in a dataset, feature selection and feature extraction.

1.3.1 Feature selection

Most of the research on feature selection is related to supervised learning [33]. More recently, methods for unsupervised learning have appeared in the literature [13], [27], [46], [48]. These methods can be divided on *filters*, that use characteristics of the features to determine their salience so the more relevant ones can be kept, and *wrappers*, that involve the exploration of the space of subsets of features and the use of a clustering algorithm to evaluate the quality of the partitions generated with each subset, according to an internal or external quality criteria. The main advantage of all these methods is that they preserve the original attributes, so the resulting patterns can be interpreted more easily.

Wrapper methods need a model to evaluate the relevance of subsets of attributes. This is more easy for supervised learning because the labels for the examples are available. The success of the methods depend on the chosen model and how well it captures the inherent structure of the data. Typical unsupervised wrapper methods involve the use of a clustering algorithm that

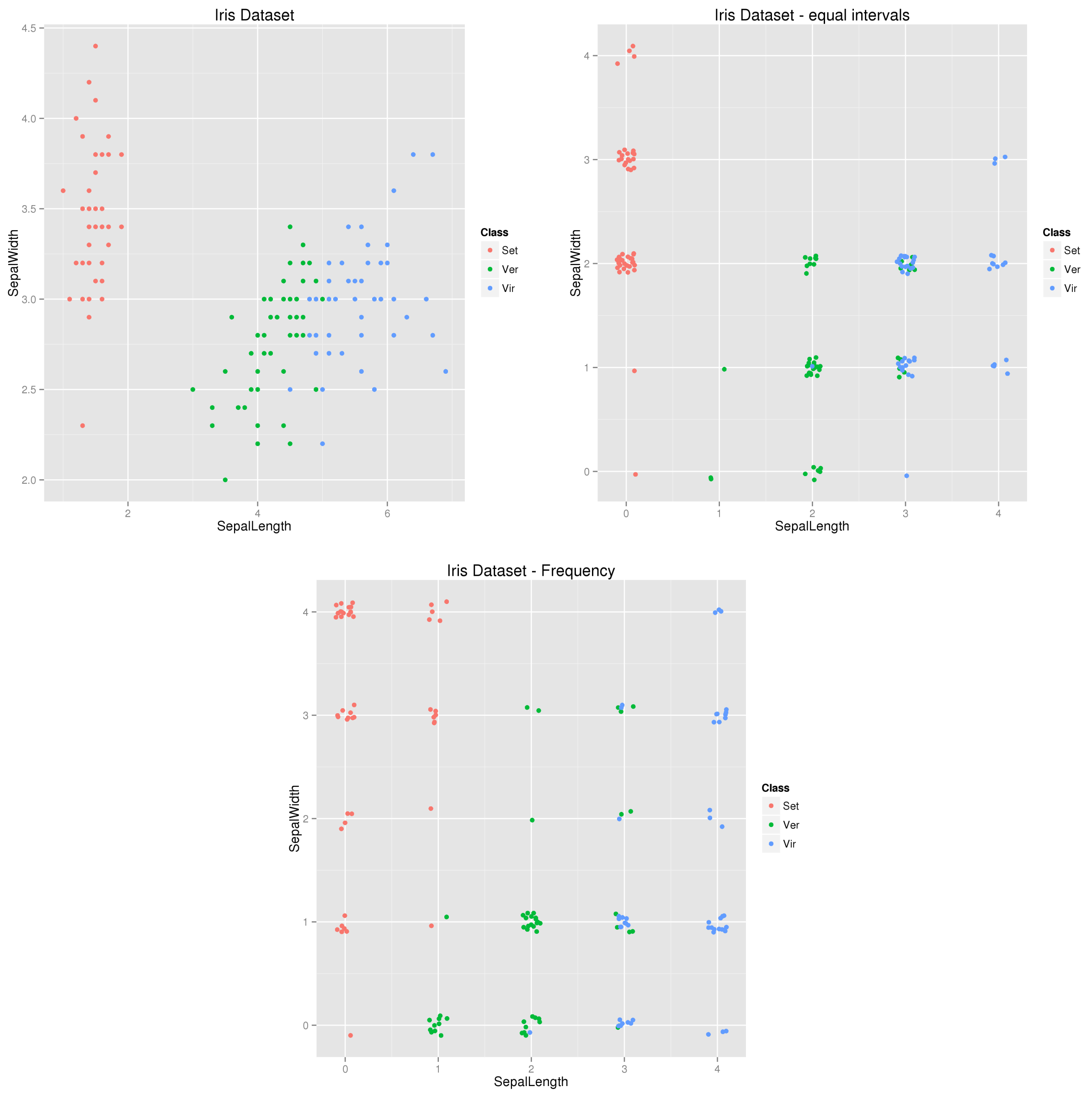


Figure 1.6: Equal sized and frequency discretization applied to the iris dataset

computes, as part of the fitting of the model, a set of weights for each attribute or the use of an objective function that penalizes the size of the model.

Filter methods use measures that assess the relevance of each attribute. This is also easier for supervised methods, in the unsupervised case the idea is to obtain a measure that evaluates the capacity of each attribute to reveal the structure of the data (class separability, similarity of instances in the same class). It is typical to use measures for properties of the spatial structure of the data (Entropy, PCA based scores, laplacian matrix based scores) or measures of feature correlation/dependence.

Laplacian score

The Laplacian Score ([27]) is a filter method that ranks the features respect to their ability of preserving the natural structure of the data. It is assumed that there are compact clusters in the data and this will reflect on the local relations among the examples.

In order to measure these local relations, this method uses the spectral matrix of the graph computed from the near neighbors of the examples. The edges of the graph use as weights the similarity among the examples. This means that two parameters have to be decided. The first one is the number of neighbors used to build the graph and, the second one, the similarity measure to be used to compute the weights.

This similarity is usually computed using a gaussian kernel with a specific bandwidth (σ), so the weights of the edges are:

$$S_{ij} = e^{\frac{||x_i - x_j||^2}{\sigma}}$$

And all edges not present have a value of 0.

Being S the similarity matrix and D a diagonal matrix where the elements of the diagonal are the sum of the weights of the row, the Laplacian matrix is computed as $L = S - D$.

Using the values of an attribute of the dataset f_r and being 1 a vector of ones, the score first computes $\tilde{f}_r$ as:

$$\tilde{f}_r = f_r - \frac{f_r^T D1}{1^T D1} 1$$

The score L_r is then computed as:

$$L_r = \frac{\tilde{f}_r^T L \tilde{f}_r}{\tilde{f}_r^T D \tilde{f}_r}$$

This gives a ranking for the relevance of the attributes, only the k attributes with the higher score are kept.

1.3.2 Feature extraction

Feature extraction is an area with many methods. The goal is to build a new set of attributes that maintains some characteristics of the data that also preserve the patterns. New attributes are computed as combinations (linear or not) of the original attributes. The following sections describe the principal methods used in this area, a more extensive tutorial can be found in [9].

Principal Component Analysis (PCA)

The most popular method for feature extraction is Principal Component Analysis [26]. This method computes a linear transformation of the data that results in a set of orthogonal dimensions that account for all the variance of the dataset. It is usual for only a few of these dimensions to hold most of the variance, meaning that with only this subset should be enough to discover the patterns in the data. This also allows visualizing the data, if two or three dimensions are enough.

PCA is best suited when the attributes follow a gaussian distribution. The new set of features, called principal components, are uncorrelated so each one explains a part of the total variance.

The method can be described geometrically as a rotation/scaling of the axes looking for the projections where the variance of the data can be explained by a few variables (see figure).

The computation of the principal components obtains the best linear approximation of the data:

$$f(\lambda) = \mu + V_q \lambda$$

μ is a location vector in $\mathbb{R}^p$, V_q is a $p \times q$ matrix of q orthogonal unit vectors and λ is a q vector of parameters We want to minimize the reconstruction error for the data:

$$\min_{\mu,\{\lambda_i\},V_q} \sum_{i=1}^{N} ||x_i - \mu - V_q\lambda_i||^2$$

Optimizing partially for μ and λ_i we have:

$$\begin{aligned} \mu &= \bar{x} \\ \lambda_i &= V_{iq}^T(x_i - \bar{x}) \end{aligned}$$

We can obtain the matrix V_q by minimizing:

$$\min_{V_q} \sum_{i=0}^{N} \|(x_i - \bar{x}) - V_q V_q^T (x_i - \bar{x})\|_2^2$$

Assuming $\bar{x} = 0$ we can obtain the projection matrix $H_q = V_q V_q^T$ by Singular Value Decomposition of the data matrix X

$$X = UDV^T$$

U is a $N \times p$ orthogonal matrix, its columns are the *left singular vectors* V is a $p \times p$ diagonal matrix with ordered diagonal values called the *singular values* The columns of UD are the *principal components* The solution to the minimization problem are the first q principal components The singular values are proportional to the reconstruction error

Independent Component Analysis [26] also transforms the dataset projecting the data to a new set of variables, but in this case these are statistically independent (all statistical moments are independent) and usually it is assumed that the attributes do not follow a gaussian distribution.

PCA variations

PCA is a linear transformation, this means that if data is linearly separable, the reduced dataset will be also linearly separable (given enough components)

We can use the *kernel trick* to map the original attribute to a space where non linearly separable data is linearly separable

Distances among examples are defined as a dot product that can be obtained using a kernel:

$$d(x_i, x_j) = \Phi(x_i)^T \Phi(x_j) = K(x_i, x_j)$$

Different kernels can be used to perform the transformation to the feature space (polynomial, gaussian, ...)

The computation of the components is equivalent to PCA but performing the eigen decomposition of the covariance matrix computed for the transformed examples

$$C = \frac{1}{M} \sum_{j=1}^{M} \Phi(x_j)\Phi(x_j)^T$$

The main advantage of this method is that allows discovering patterns that are non linearly separable in the original space. Also, this method is that can be applied using any kind of kernel, this means that the features do not have to be continuous or even gaussian. This extends the method to structured data like for example graphs or strings.

PCA transforms data to a space of the same dimensionality (all eigenvalues are non zero) Another variation of PCA is to solve the minimization problem posed by the reconstruction

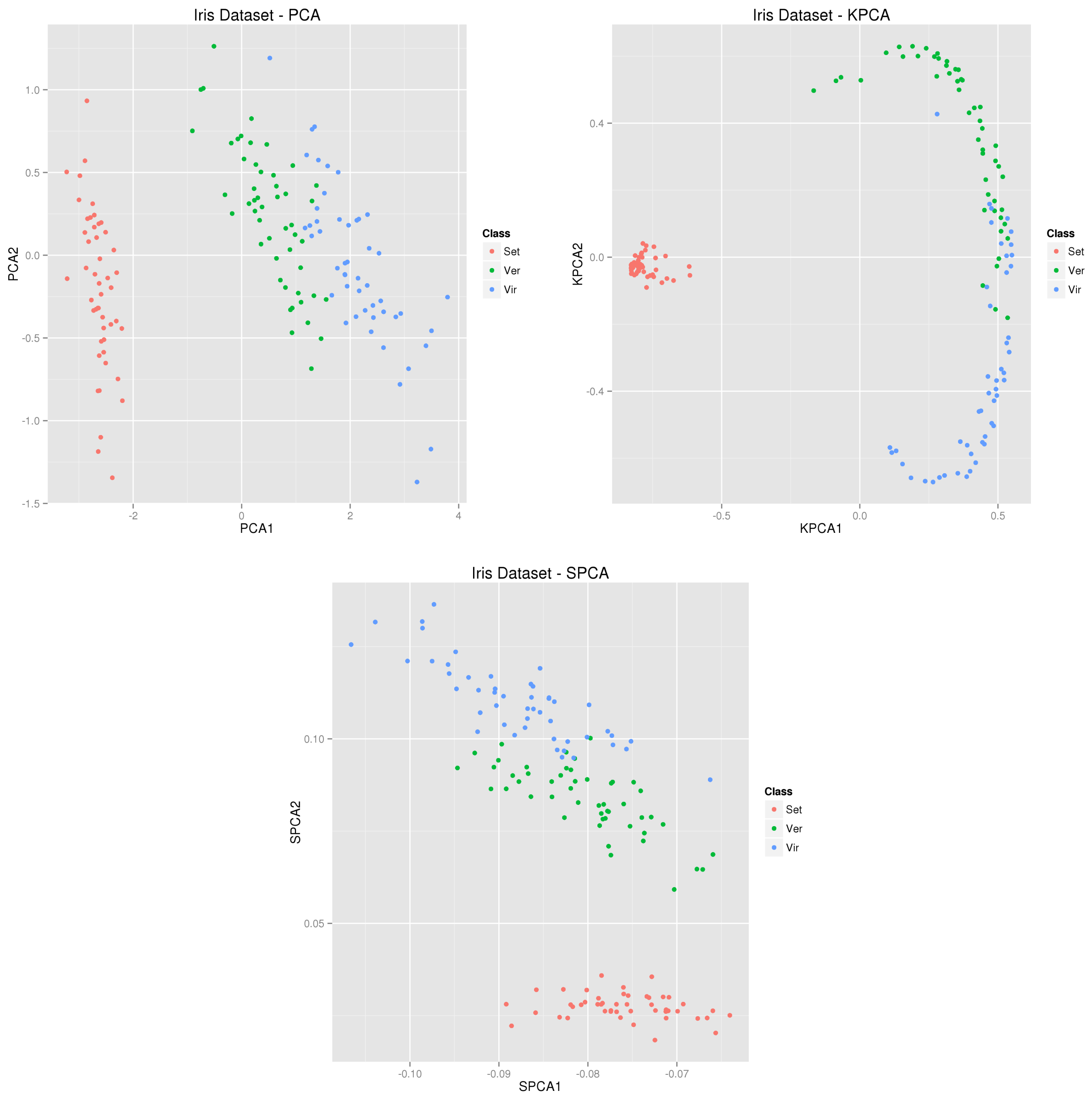


Figure 1.7: PCA, Kernel PCA (RBF kernel) and Sparse PCA transformation of the iris dataset

error using ℓ-1 norm regularization, this in known as *sparse PCA*. A penalization term is added to the objective function proportional to the norm of the matrix of eigenvalues

$$\min_{U,V} \|X - UV\|_2^2 + \alpha \|V\|_1$$

The ℓ-1 norm regularization will encourage sparse solutions (zero eigenvalues).

Figure 1.7 shows PCA and the two described PCA variations applied to the iris dataset. It is interesting to notice in this example, that the representation with the principal components does not differ much from the representation with only two features (Sepal Length, Sepal Width) shown in figure 1.6. In this case just two dimensions are enough for discriminating the groups in the data.

Multidimensional Scaling

The methods in this class of transformations obtain a matrix that projects the dataset directly from M to N dimensions, preserving pairwise data distances. This results in a new dataset that maintains only the distance relations between pairs examples, but usually this is enough to uncover the structure in the data.

The method for computing the projection matrix is based on the optimization of a function of the pairwise distances (called stress function). This means that the actual attributes are not used in the transformation, only the distances, this makes the method suitable for any data where a distance function can be defined, independently of the representation of the attributes (they do not have to be continuous features).

This method allows different objective functions for optimization that lead to several variants, like *least squares MDS*, *Sammong mapping* or *classical scaling*. The optimization problem is usually solved by gradient descent.

For example, for Least Squares Multidimensional Scaling (MDS) the distortion is defined as the sum of the square distance between the original distance matrix and the distance matrix of the new data:

$$S_D(z_1, z_2, ..., z_n) = \left[\sum_{i \neq i'} (d_{ii'} - \|z_i - z_i'\|_2)^2\right]$$

The optimization problem is defined as:

$$\underset{z_1, z_2, ..., z_n}{\arg\min} S_D(z_1, z_2, ..., z_n)$$

Several strategies can be used to solve this problem. If the distance matrix is euclidean, it can be solved applying eigen decomposition just like PCA. In other cases gradient descent can be used using the derivative of $S_D(z_1, z_2, ..., z_n)$ and a step α in the following fashion:

1. Begin with a guess for Z

2. Repeat until convergence:
$$Z^{(k+1)} = Z^{(k)} - \alpha \nabla S_D(Z)$$

Sammong Mapping puts emphasis on smaller distances, using as objective function:

$$S_D(z_1, z_2, ..., z_n) = \left[\sum_{i \neq i'} \frac{(d_{ii'} - \|z_i - z_i'\|)^2}{d_{ii'}}\right]$$

Classical Scaling uses similarity instead of distance, with the following objective function:

$$S_D(z_1, z_2, ..., z_n) = \left[\sum_{i \neq i'} (s_{ii'} - \langle z_i - \bar{z}, z_i' - \bar{z} \rangle)^2\right]$$

Non-negative Matrix Factorization

Non-negative Matrix Factorization uses a similar idea than PCA, an eigen decomposition of a matrix that transforms the original data in the product of two matrices. The main difference is that the values of the matrices are constrained to be positive. The formulation assumes that the data is the sum of unknown positive latent variables and the positiveness assumption helps to interpret the result. This is interesting in domains like for instance text mining, where it can be assumed that documents can be described as the composition of topics, image recognition,

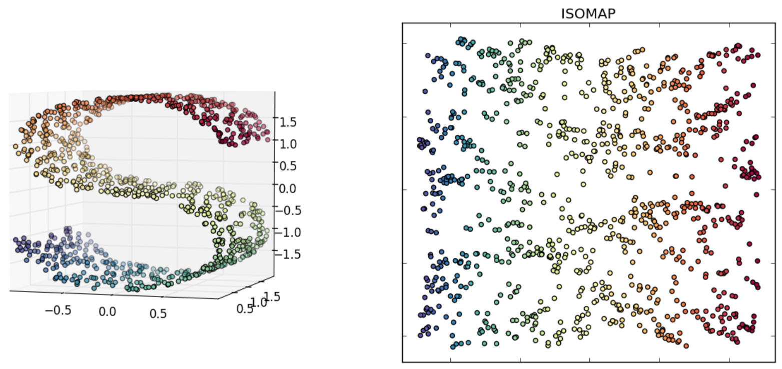


Figure 1.8: ISOMAP

where an image can be obtained by joining object features, or recommender systems, where a recommendation depends on topics of interest of the users.

This area is a hot topic because of its practical applications and many methods exists, a recent review can be found in [45].

ISOMAP

This method computes a non-linear transformation between the original space and the reduced new space. It is assumed that there is a low dimensional embedding of the data able to represent the relations among the patterns in the data. The geodesic distance among the examples is used for this transformation instead of the usual euclidean distance. This distance is computed as the length of the shortest path that connects two nodes in the neighborhood graph of the data.

The choice of this measure is based on the assumption that the relation of an instance with its immediate neighbors holds the most representative information about the structure of the data. Obtaining a new space that preserves this distance would preserve these neighborhood relations. In fact, we are obtaining a new space where the locality of the data behavior is preserved.

The algorithm for computing this transformation is as follows:

1. For each data point find its k-closest neighbors (points at minimal euclidean distance)
2. Build a graph where each point has an edge to its closest neighbors
3. Approximate the geodesic distance for each pair of points by the shortest path in the graph (using Kruskal algorithm for example)
4. Apply a MDS algorithm to the distance matrix of the graph to obtain the new set of coordinates

Local Linear Embedding

Local Linear Embedding ([39, 16, 51]) performs a transformation that preserves the local structure of the data. It assumes that each instance can be reconstructed by a linear combination of its neighbors. The method computes for each example a set reconstruction weights from its k-nearest neighbors. From these weights a new set of coordinates in a lower dimensional space is computed for each data point. Different variants of the algorithm have been defined using different properties.

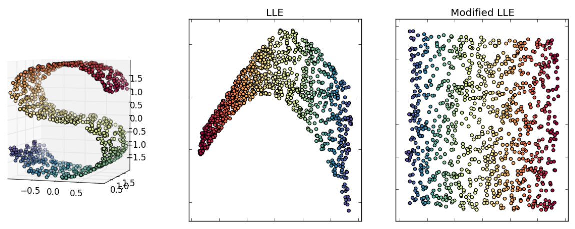


Figure 1.9: Variants of LLE

The general algorithm for this method is as follows:

1. For each data point find the K-nearest neighbors in the original p-dimensional space ($\mathcal{N}(i)$)
2. Approximate each point by an mixture of the neighbors solving the optimization problem:

$$\min_{W_{ik}} \|x_i - \sum_{k\in\mathcal{N}(i)} w_{ik}x_k\|^2$$

   where $w_{ik} = 0$ if $k \notin \mathcal{N}(i)$, $\sum_{i=0}^{N} w_{ik} = 1$ and $K < p$

3. Find points y_i in a space of dimension $d < p$ that minimize:

$$\sum_{i=0}^{N} \|y_i - \sum_{k=0}^{N} w_{ik}y_k\|^2$$

Local MDS

This is a non-linear version of MDS that uses an objective function that gives more emphasis to local distances. This locality is also derived from the neighborhood graph that is used in the computation of the objective function. The idea is to obtain a new space where neighbor examples are closer and the distances to non neighbors are magnified, improving separability.

Given a set of pairs of points $\mathcal{N}$ where a pair (i, i') belongs to this set if i is among the k neighbors of i' or viceversa, the objective function that is minimized is:

$$S_L(z_1, z_2, \ldots, z_N) = \sum_{(i,i')\in\mathcal{N}} (d_{ii'} - \|z_i - z_{i'}\|)^2 - \tau \sum_{(i,i')\notin\mathcal{N}} (\|z_i - z_{i'}\|)$$

The parameter τ controls how much the non neighbors are scattered.

Random Projections

All the previous methods are computationally expensive because a projection/transformation matrix has to be obtained to perform the dimensionality reduction with the corresponding computational cost. Random projections ([7], [34]) takes advantage of the properties of certain transformation matrices to obtain a reasonable approximation with a reduced computational cost.

The idea is that if we generate randomly a transformation matrix with certain conditions, it is obtained a projection to a new space of reduced dimensionality that maintains the distances among the examples in the original space.

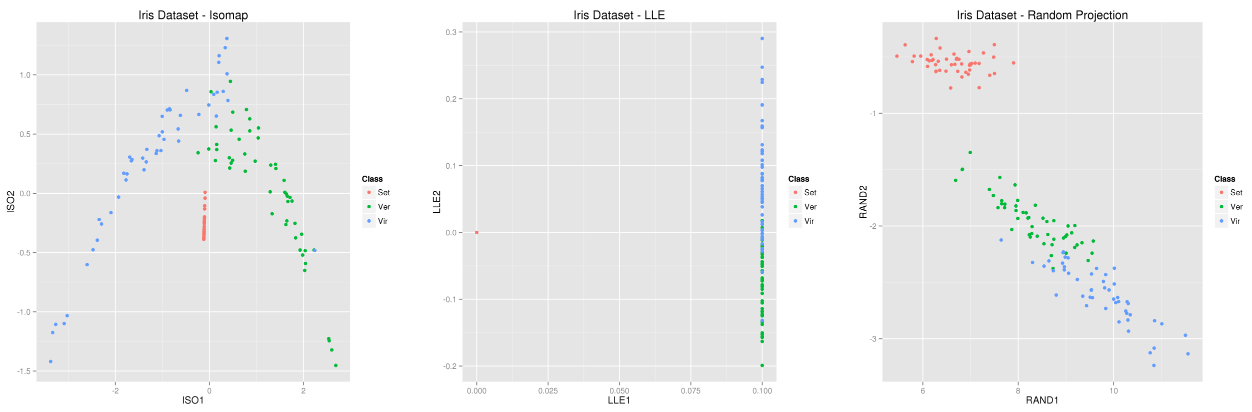


Figure 1.10: ISOMAP, LLE and Random Projection transformation of the iris dataset

The matrix generated has as dimensions $n\_features \times n\_components$, being $n\_components$ the number of dimensions of the new space

There are two main strategies for generating the random matrix:

- To generate the matrix using a gaussian distribution $N(0, \frac{1}{n\_components})$, this matrix is dense.
- To generate a sparse matrix of certain density (s) with values:

$$\begin{cases} \sqrt{\frac{s}{n\_components}} & & \frac{1}{2s} \\ 0 & with\ \ probability & 1 - \frac{1}{s} \\ -\sqrt{\frac{s}{n\_components}} & & \frac{1}{2s} \end{cases}$$

These matrices are almost orthogonal (close to matrix obtained by PCA). The effectiveness of the transformation usually depends on the number of dimensions, the Johnson-Lindenstrauss lemma ([12]) can be used to obtain an estimate of the number of dimensions from the number of examples in the dataset and the expected distorsion.

1.4 Similarity/distance functions

Many unsupervised algorithms (like clustering algorithms) are based on the measure of the similarity/distance among examples. The values for this comparison are obtained applying these functions to the attribute representation of the examples. This means that we assume that there is a N-dimensional space where the examples are embedded and where such functions represent the relationships of the patterns in the data. There are domains where this assumption is not true so some other kind of functions will be needed to represent instances relationships.

Mathematically speaking, a function is a *similarity* if it holds the following properties:

1. $s(p, q) = 1 \iff p = q$
2. $\forall p, q\ s(p, q) = s(q, p)$ (symmetry)

Some examples of similarity function are:

- **Cosine similarity**

  $$d(i,k) = \frac{x_i^T \cdot x_j}{\parallel x_i \parallel \cdot \parallel x_j \parallel}$$

  The similarity is computed as the cosine of the angle between two vectors, so there is a geometrical interpretation of the similarity.

- **Pearson correlation measure**

  $$d(i,k) = \frac{(x_i - \overline{x}_i)^T \cdot (x_j - \overline{x}_j)}{\parallel x_i - \overline{x}_i \parallel \cdot \parallel x_j - \overline{x}_j \parallel}$$

This distance corresponds to the cross correlation between two variables. The only difference with the cosine distance is that data is centered before computing the distance, so the result is the same if data is already centered.

On the other hand, a function is a *distance* if it holds the following properties:

1. $\forall p,q \quad d(p,q) \geq 0 \;\; and \;\; \forall p,q \quad d(p,q) = 0 \Longleftrightarrow p = q$
2. $\forall p,q \quad d(p,q) = d(q,p)$ (symmetry)
3. $\forall p,q,r \quad d(p,r) \leq d(q,p) + d(p,r)$ (triangular inequality)

Distance functions can be unbounded, but in practice there is a maximum value that can be obtained defined by the ranges of the attributes that are used to describe the data. This can be used to normalize the values of the distances for example to a $[0,1]$ range. Some examples of distance functions are:

- **Minkowski metrics**

  $$d(i,k) = \left(\sum_{j=1}^{d} |x_{ij} - x_{kj}|^r \right)^{\frac{1}{r}}$$

  The special cases of d=1 and d=2 are respectively the Manhattan (L_1) and Euclidean (L_2) distance functions that are the most common choices in applications.

- **Mahalanobis distance**

  $$d(i,k) = (x_i - x_k)^T \cdot \varphi^{-1} \cdot (x_i - x_k)$$

  where φ is the covariance matrix of the attributes. This distance makes the assumption that attributes are not independent represented by the covariance matrix. In the case of truly independent data this distance is the euclidean distance.

- **Chebyshev Distance**

  $$d(i,k) = \max_j |x_{ij} - x_{kj}|$$

  It is also known as the maximum distance and it is also a Minkowski metric where $r = \infty$ (L_∞ metric).

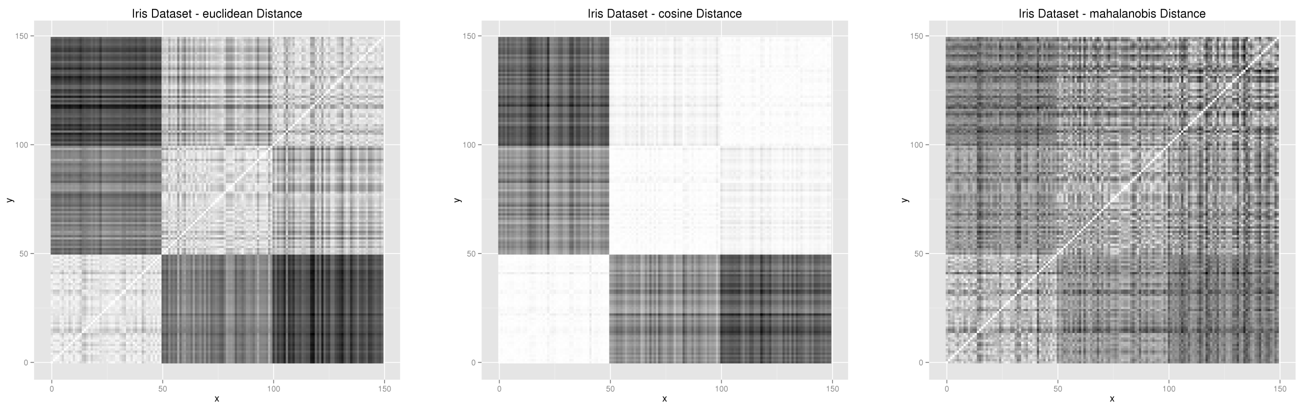


Figure 1.11: Distance/similarity matrices for the Iris dataset using Euclidean, Cosine and Mahalanobis functions

- **Canberra distance**

$$d(i,k) = \sum_{j=1}^{d} \frac{|x_{ij} - x_{kj}|}{|x_{ij}| + |x_{kj}|}$$

  This is a weighted variation of the Manhattan distance.

Figure 1.11 shows the distance/similarity matrix for the Iris dataset (among all the examples) computed for three functions (euclidean, cosine and mahalanobis). The chosen function changes the relations among examples, resulting in a different space embedding, so they have to be chosen carefully so the patterns in the data are correctly identified.

Apart from the examples of similarity/distance functions presented, there are many more that are specific to different types of data or that are used in specific domains. For example in the case of *binary data* the Coincidence coefficient or the Jaccard coefficient are used among others, for *strings* a popular choice is the edit distance (also known as Levenstein distance) and its variations, if data corresponds to *temporal data* other distances that adapt to different time scales/speed like dynamic time warping (DTW) are more appropriate than for example the euclidean distance.

A very good reference on this topic is [14].

1.5 Examples

This section presents different examples of data preprocessing methods applied to real data extracted from the UCI ML repository and other practical applications.

1.5.1 Dimensionality reduction and discrete data

This example uses the mushroom dataset from the UCI ML repository. The data consist on 18 attributes describing qualitative characteristics of over 8000 mushrooms. This is a supervised dataset where the label classifies each mushroom as edible or poisonous.

The goal of this example is to visualize the data to obtain some insight about its structure. One problem with this dataset is that all the attributes are discrete, making it difficult to visualize the data. Figure 1.12 represents two of the dimensions for a subsample of the data,

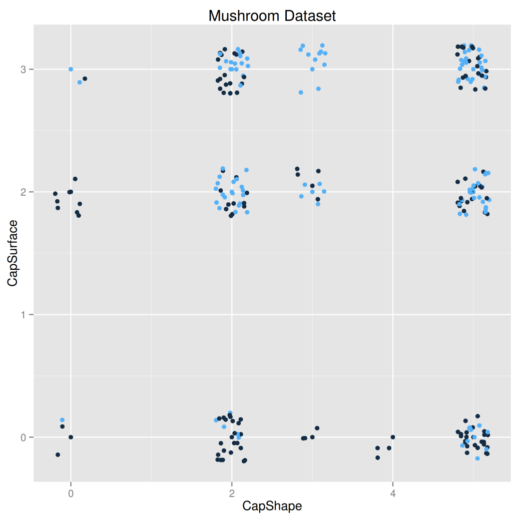


Figure 1.12: Mushroom data visualization for two attributes

some jitter has been added to the graphic so overlapping points can be seen, otherwise only the $n \times m$ crossings of the values of the attributes would be visible.

The values of the attributes correspond to categories, so there is not an order that can be used to map the values to a continuous range. A possible transformation is to obtain binary attributes by *one-hot encoding*. This transformation generates as many binary attributes as values has the original attribute and each example is coded assigning a value 1 to the variable corresponding to the actual value and a 0 for the rest of attributes.

For example, one of the attributes of the dataset is the *gill-spacing* with values {`close`, `crowded`, `distant`}. The encoding will transform this variable into three variables (gill-spacing-close, gill-spacing-crowded, gill-spacing-distant) and for example the value `crowded` would be encoded as (gill-spacing-close=0, gill-spacing-crowded=1, gill-spacing-distant=0).

One has to be careful interpreting the results using this kind of transformations because the attribute generated are not independent, so some assumptions of dimensionality reduction methods would not hold.

Figure 1.13 presents the transformation of the one-hot encoding of the dataset using random projection, PCA and ISOMAP. As can be seen in the graphic, it seems to be more structure in the data than the one coded in the examples labels. The random projection transformation looks less evident, but despite that the projection only has two dimensions, each class seems to have some internal structure. The conditions for applying PCA to the data do not hold because attributes are not gaussian distributed, so the representation has to be cautiously interpreted. Regardless, it is obvious that each class appears to be composed by several clusters indicating a more complex relationships in the data.

The ISOMAP transformation uses the euclidean distances among the examples, given that we are using one-hot encoding actually we are computing the hamming distance for the original nominal attributes. The number of neighbors used in the computations of ISOMAP has an influence on the results, so two values are presented. In the case of 100 neighbors the structure that shows the data is similar to the one obtained by PCA, using only 5 neighbors structure seems simpler, but there is also evidence of more information than the two labels used in the dataset.

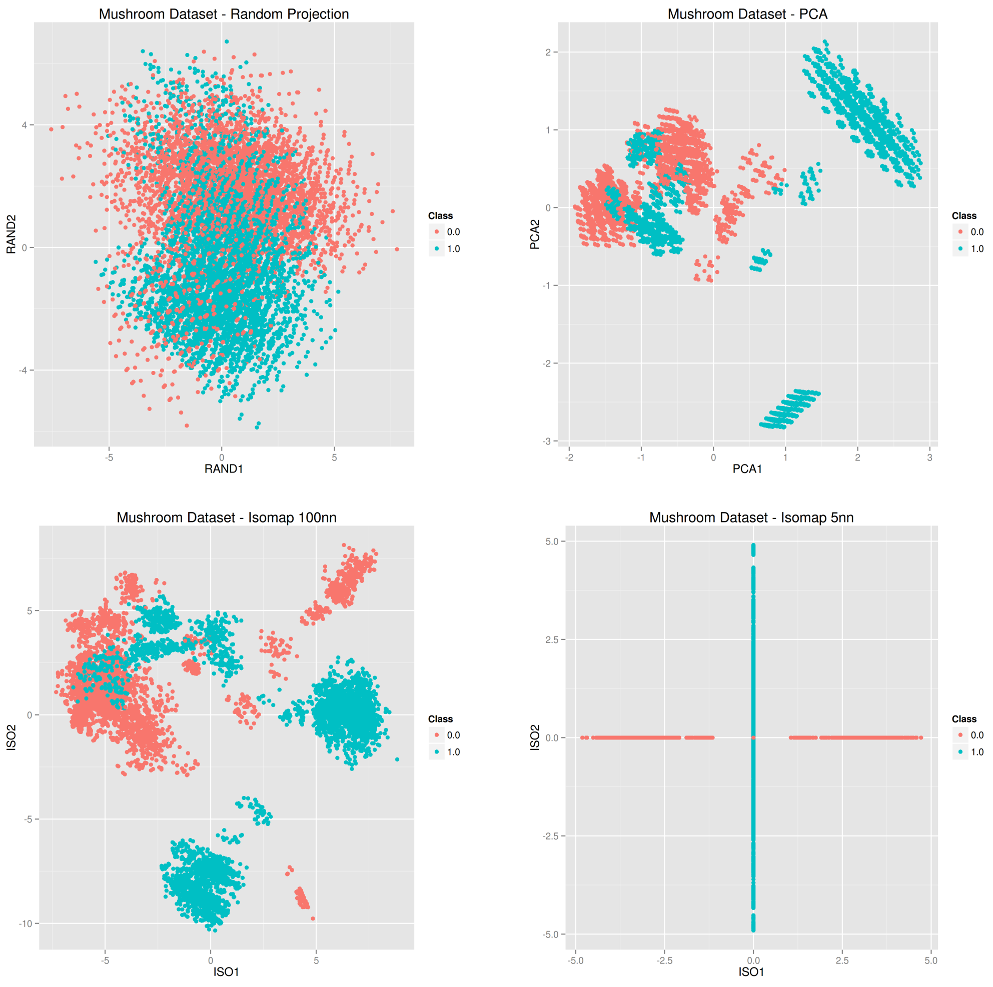


Figure 1.13: Mushroom data visualization after Random Projection, PCA and ISOMAP (100 and 5 neighbors) transformation

1.5.2 Application: Wheelchair control

This example uses data collected from a research project about elderly patients with mobility problems. The goal is to study the patterns generated by the users of a wheelchair that has implemented a shared control mechanism. The idea is that the patient receives help from the chair in the case it detects that the movement is outside a set of quality parameters, mainly related to the avoidance of obstacles (walls, doors).

The wheelchair (see figure 1.14) is controlled by the patient using a joystick. The shared control mechanism is able to correct the input received in case it is needed. There is an array of laser sensors able to measure the distance from the front of the chair (210 degrees) to the nearest obstacle and an odometer for computing the position of the chair.

The dataset consists of collected trajectories of several patients in different predefined situations, like entering in a room from a corridor through a door and then turning left to arrive to the goal point (see figure 1.15). There is a total of 88 attributes (see figure 1.16) consisting in the angle/distance to the goal, and the angle/distance measured by the sensors to the nearest obstacle around the chair.

The analysis goal is to characterize how the computer helps the patients with different handicaps and discover what patterns/structure appear in the trajectory data.

As in the previous example, we will test different dimensionality reduction methods to

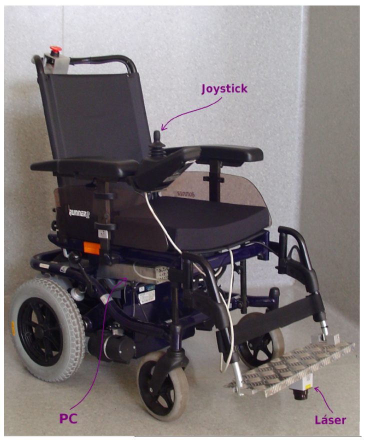


Figure 1.14: Wheel Chair with shared control

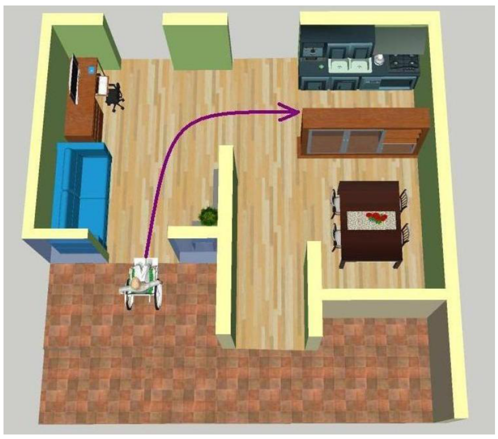
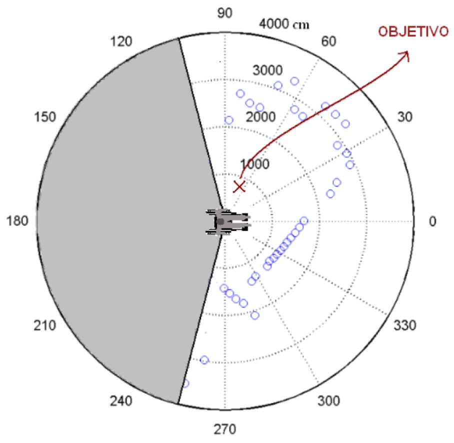


Figure 1.15: Crossing a door and left turn situation



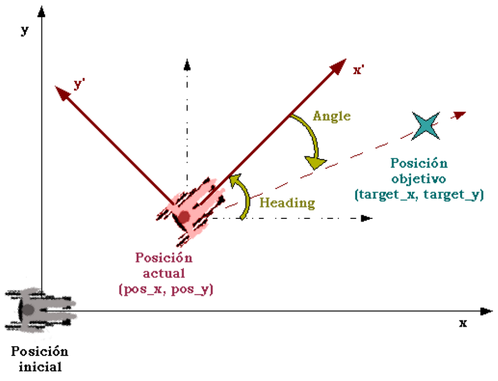

y
y'
x'
Angle
Posición objetivo (target_x, target_y)
Heading
Posición actual (pos_x, pos_y)
x
Posición inicial

Figure 1.16: Attributes of the dataset

visualize the data and check for indications of structure. The first transformation using PCA reveals several clusters in the data with elongated shapes, there is one cluster well separated and a density area that is difficult to appreciate, and that seems to have different densities.

This result gives the idea that probably a nonlinear transformation of the data will help to highlight the different density areas. A transformation using Kernel PCA with a quadratic kernel still maintains the separated cluster and gives a more spherical shape

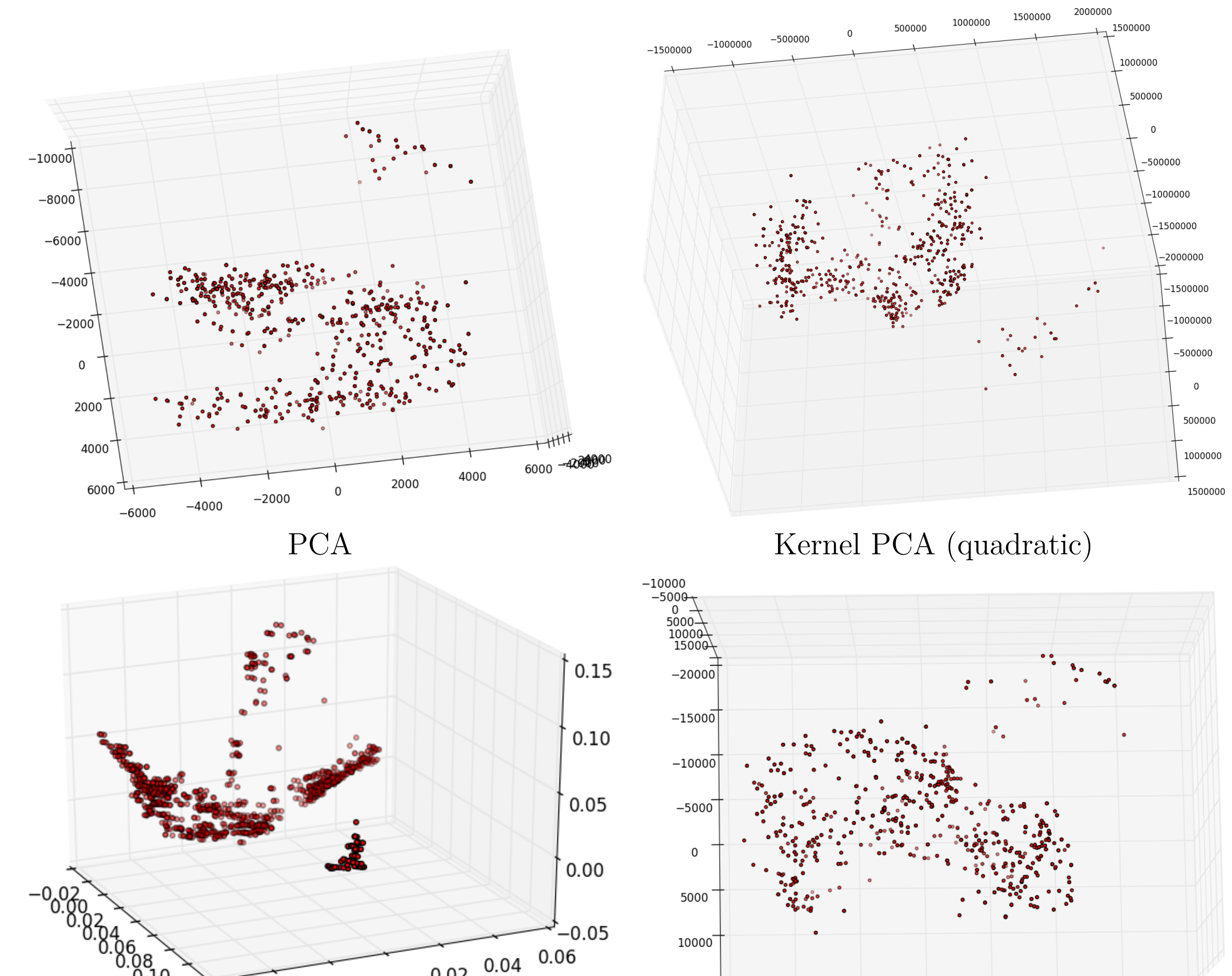


Figure 1.17: Wheelchair dataset visualization using PCA, Kernel PCA, Locally Linear Embedding (50n) and ISOMAP (50n)

2 Clustering

2.1 Clustering algorithms

To learn/discover patterns from data can be done in a supervised or unsupervised way. The machine learning community has a strong bias towards supervised learning methods because the goal is well defined and the performance of a method given a dataset can be measured objectively. Clustering can be qualified as an ill-posed problem in the sense that there is not a unique solution, because different clusterings of a dataset can be correct depending on the goals or perspective of the task at hand. It is obvious that we learn a lot of concepts unsupervisedly and that the discovery of new ideas is always unsupervised, so these methods are important from a knowledge discovery perspective.

The clustering task can be defined as a process that, using the intrinsic properties of a dataset $\mathcal{X}$, uncovers a set of partitions that represents its inherent structure. It is, thus, a task that relies in the patterns that show the values of the attributes that describe the dataset. The goal of the task will be to either obtain a representation that describes an unlabeled dataset (summarization) or to discover concepts inside the data (understanding).

Partitions can be either nested, so a hierarchical structure is extracted, or flat, no relation among the groups. The groups can also be hard partitions, so each example only belongs to one of the clusters, or soft partitions, so there is overlapping among them, that can be represented by a function measuring the membership of the examples to the clusters.

There are several approaches to obtain a partition from a dataset, depending on the characteristics of the data or the kind of the desired partition. It is usual to assume that the data is embedded in an N-dimensional space that has a similarity/dissimilarity function defined. The main bias that clustering strategies apply is to assume that examples are more related to the nearest examples than to the farthest ones, and that clusters have to be compact (dense) groups that are maximally separated from each other. Broadly approaches that follow that bias can be divided in:

- *Hierarchical algorithms*, that result in a nested set of partitions, representing the hierarchical structure of the data, being commonly hard partitions. These methods are usually based on a matrix of examples distances/similarities and a recursive divisive or agglomerative strategy.

- *Partitional algorithms*, that result in a set of disjoint (hard) or overlapped (soft) partitions. They are based on the optimization of a criterion (assumptions about the characteristics of the cluster model) There is a more wide variety of methods of this kind, depending on the model assumed for the partition or the discovery strategy used. The more representative ones include algorithms based on prototypes or probabilistical models, based on the discovery of dense regions, based on the subdivision of the space of features into a multidimensional grid and based on defining a graph structure among the examples.

In the following sections the main characteristics of these methods will be described with an outline of the main representative algorithms. As we will see, the methods for clustering data come from different areas like statistics, machine learning, graph theory, fuzzy sets theory or physics.

2.2 Statistical Hierarchical clustering

Hierarchical methods [17] use two strategies for building a tree of nested clusters that partitions a dataset, divisive and agglomerative. Divisive strategies begin with the entire dataset, and each iteration it is determined a way to divide the data into two partitions. This process is repeated recursively until individual examples are reached. Agglomerative strategies iteratively merge the most related pair of partitions according to a similarity/distance measure until there is only one partition. Usually agglomerative strategies are computationally more efficient.

These methods are based on a distance/similarity function that compares partitions and examples. Initially, these values for each pair of examples are stored in a matrix that the algorithm updates during the clustering process.

The strategy used by some algorithms considers this matrix as a graph that is created by adding each iteration new edges in an ascending/descending order of length. In this case, a criterion determines when the addition of a new edge results in a new cluster. These are *graph based* algorithms. Other algorithms reduce the distance/similarity matrix each iteration by merging two groups, deleting these groups from the matrix and then adding the new merged group. These are *matrix algebra based* algorithms.

2.2.1 Graph based algorithms

These algorithms assume that the examples form a fully connected graph, the length of the edges is defined by the similarity/distance function and there is a connectivity criteria used to define when a new cluster is created. The most common ones are, the *single linkage* criteria, that defines a new cluster each time a new edge is added if it connects two disjoint groups, and the *complete linkage* criteria that considers that a new cluster appears only when the union of two disjoint groups form a clique in the graph. Both criteria have different bias, single linkage allows discovering elongated clusters and complete link has preference for globular clusters.

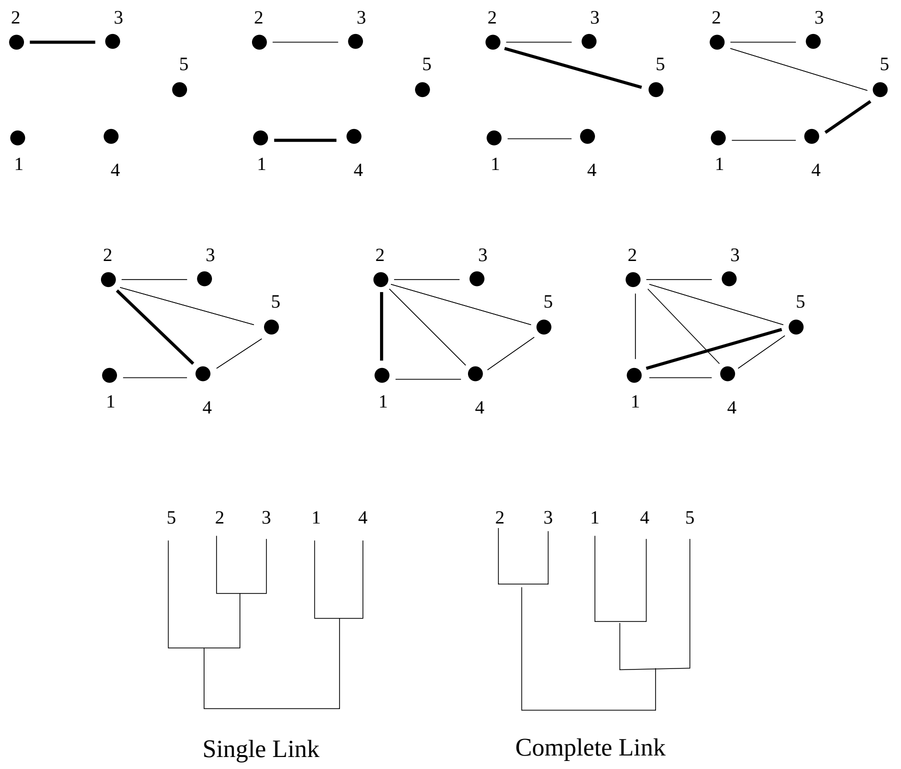
Algorithm 2.1 shows the agglomerative version of this clustering, the criteria used to determine the creation of new clusters can be any of the mentioned ones. The cost of this algorithm is $O(n^3)$ being n the number of examples, it can be improved using a fast data structure to find the closest examples too $O(n^2 \log(n))$. It is also possible to assume that the graph is not fully connected to improve the computational cost for example using a minimum spanning tree of a graph with the k-nearest neighbors but sacrificing the quality of the clustering.

As an example, lets consider the following matrix as the distances among a set of five examples:

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

Algorithm 2.1 Algorithm for graph agglomerative hierarchical clustering

```
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
    end
until Only one cluster exists
```



2.2.2 Matrix algebra based algorithms

Matrix algebra based algorithms work similar. The process initiates with the full distance matrix computed from the examples given a distance/similarity function. Each step of the process first finds the two most similar examples/clusters to merge in a new group, then a new distance is computed from the merged group to the remaining group. Different updating criteria can be chosen.

The distances of this new group to the other groups is a combination of the distances to the two merged groups. Popular choices for this combination are the maximum, minimum and mean of these distances. The size of the group or its variance can be also used to weight the combination. Some of these choices replicate the criteria used by graph based algorithms, for example single linkage is equivalent to substitute the distances by the minimum distance among the examples from the new group to the examples to the other groups, choosing the largest distance is equivalent to complete linkage.

This algorithm allows for a continuous variety of choices in the criteria for merging the partitions and updating the distances, creates a continuous of algorithms that allows obtaining very different partitions from the same data. Most of the usual criteria can be described using

Algorithm 2.2 Algorithm for matrix algebra based agglomerative hierarchical clustering

```
Algorithm: Agglomerative Clustering
Compute Distance/similarity matrix
repeat
    Find the pair of groups/examples with the smallest similarity
    Merge the pair of groups/examples
    Delete the rows and columns corresponding to the pair of groups/examples
    Add a new row and column with the new distances to the new group
until Matrix has one element
```

the following formula:

$$d((i,j),k) = \alpha_i d(i,k) + \alpha_j d(j,k) + \beta d(i,j) + \gamma[d(i,k) - d(j,k)]$$

For example, using $\alpha_i = \frac{1}{2}$, $\alpha_j = \frac{1}{2}$, $\beta = 0$ and $\gamma = -\frac{1}{2}$ we have the single linkage criteria. If we introduce the sizes of the groups as weighting factors, $\alpha_i = \frac{n_i}{n_i+n_j}$, $\alpha_j = \frac{n_j}{n_i+n_j}$ and $\beta = \gamma = 0$ computes the group average linkage. The Ward method that minimizes the variance of the groups corresponds to the weights $\alpha_i = \frac{n_i+n_k}{n_i+n_j+n_k}$, $\alpha_j = \frac{n_j+n_k}{n_i+n_j+n_k}$, $\beta = \frac{-n_k}{n_i+n_j+n_k}$ and $\gamma = 0$.

As an example, lets consider the same distance matrix as in the previous example:

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

After merging the examples 2 and 3 and using the mean distance as criteria for computing the new distances we have:

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

Now the closest groups are the examples 1 and 4, and after the distance recomputation:

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

And finally the example 5 is the closest to the group (1,4).

	1,4,5
2,3	6.725

This finishes the clustering, the different mergings and their distances allows computing the dendrogram.

2.2.3 Drawbacks

The main drawback of these algorithms is their computational cost. The distance matrix has to be stored and this scales quadratically with the number of examples. Also, the computational cost of these algorithms is cubic with the number of examples in the general case and it can be reduced in some particular cases to $O(n^2 \log(n))$ or even $O(n^2)$.

Also, the dendrogram as a representation of the relationship of the examples and clusters is only practical as representation when the size of the dataset is small. This implies that the result of the algorithm has to be reduced to a partition instead of using the hierarchy. There

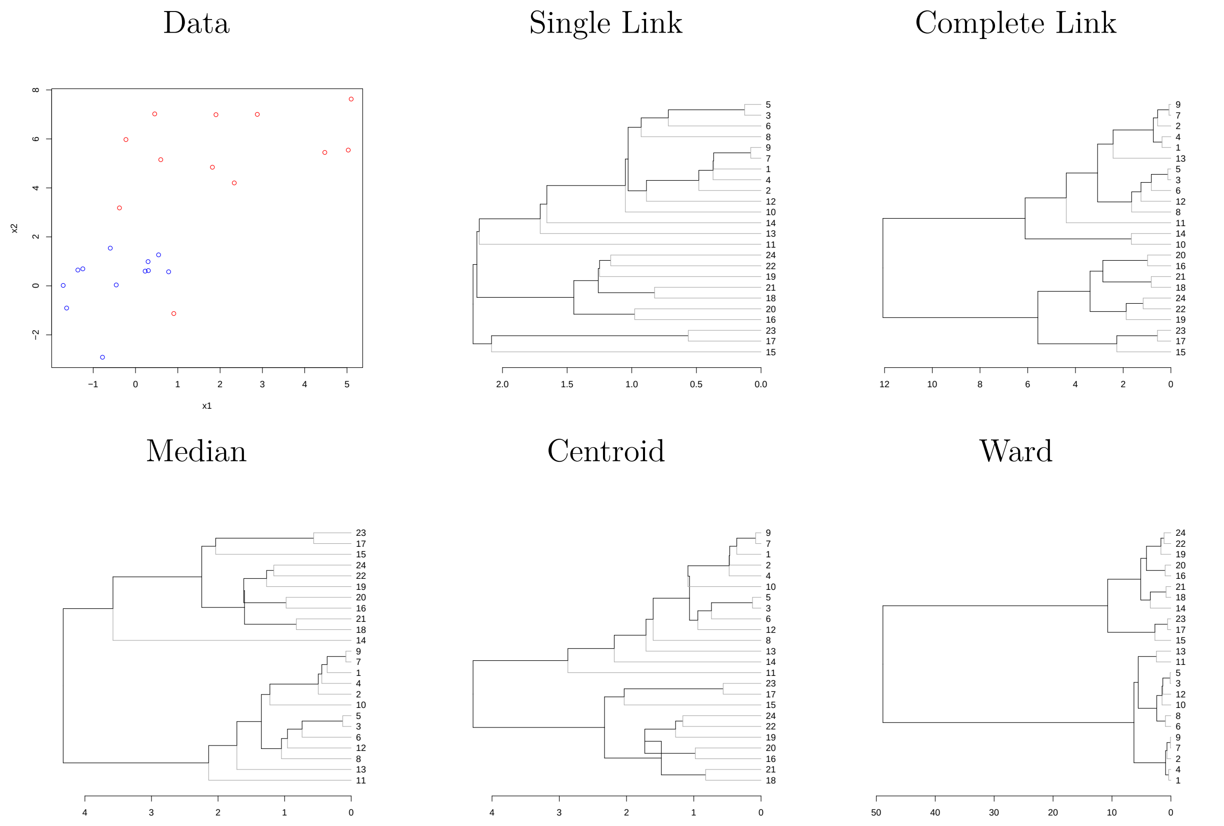


Figure 2.1: Dendrograms obtained applying different linkage criteria to the same dataset

are different criteria for obtaining a partition from the hierarchy that involves testing the gap in the distances among the cluster merges in the dendrogram or applying one of the several criteria for assessing cluster validity that will be described on the next chapter.

It is also a problem to decide the best linkage criterion for the data. Each one assumes that different kinds of patterns exist in the data. For example, single linkage is able to discover elongated and non convex shaped clusters, but complete linkage or average linkage prefers globular clusters. Some criteria can also have problems when clusters of different sizes and densities appear in the data. This makes that the results using one or other criterion could be very different for the same dataset as can be seen in figure 2.1.

Some criteria present also some undesirable behaviors related to the behavior of the distance/similarity measure used to compare the examples. For example, *chaining effects* can usually appear when single linkage criterion is applied, this means that most or all of the merges consist in one individual example and an existing cluster, so cutting the dendrogram yields a large cluster and several individual examples. Also, the phenomenon known as *inversions* appear when the criteria considers the centroids of the clusters, this means that the distances in the dendrogram are not monotonically increasing with the height of the dendrogram, and two clusters merge at a distance that is lower than the previous merge. This can be observed clearly in figure 2.1 for the centroid linkage, but also appear in other dendrograms of the example.

2.3 Conceptual Clustering

Related to hierarchical clustering but from the perspective of unsupervised machine learning, conceptual clustering ([23]) includes algorithms that build a hierarchy of concepts aimed to describe the groups of the data at different levels of abstraction.

These algorithms use as bias the ideas from cognitive science about how people build their concept structures from their observation of the environment. There are different theories that try to explain how human categories form and are organized in the mind. One of these theories describes categorization as the construction of a hierarchy of concepts based on generality/specialization and described by probabilistic prototypes.

This theory explains some phenomenon observed in cognitive psychology experiments about human conceptualization. Mainly that we use a hierarchy to organize concepts and we have a preferred level that we apply to categorize (basic level) and identify new examples and that concepts are not defined by necessary and sufficient conditions, but that we have prototypes that are use for this categorization. The decision about if an example belongs or not to a category is explained by the similarity to the prototype. This explains that there are examples that we consider more prototypical of a concept or that there are examples that are difficult to categorize because they share similarity to different prototypes.

The algorithms included in this area have some specific assumptions about how the learning of new concepts has to be. It has to be incremental in nature, not at once, because experience is acquired from continuous observation. Concepts are learned with their relationships and their relationship is complex, so a hierarchy of concepts has to be polythetic (not binary).

The learning algorithm has to search in the space of hierarchies, so it begins with an empty hierarchy and at each step of the process a full hierarchy always exists. There is an objective function that measures the utility of the learned structure and that is used to decide how the hierarchy is modified when new examples appear. And finally, the updating of the structure is performed by a set of conceptual operators that decide what possible changes can be applied to the hierarchy. Being the learning incremental this means that the final hierarchy depends on the order the examples are presented to the algorithm, this is plausible for modeling human learning but it is not always practical from a data mining perspective.

The COBWEB algorithm ([21]) is an example of conceptual clustering algorithm. It is an incremental algorithm that builds a hierarchy of probabilistic concepts that act as prototypes (see figure 2.2). To build the hierarchy a heuristic measure called *category utility* (CU) is used. This is related to the *basic level* phenomenon described by cognitive psychology. This measure allows COBWEB to have as a first level the concepts that better categorize a set of examples, the subsequent levels specialize this first level.

Defined for categorical attributes, category utility balances intra class similarity ($P(A_i = V_{ij}|C_k)$) and inter class similarity ($P(C_k|A_i = V_{ij})$) combining both measures as trade off between the two measures for a given set of categories:

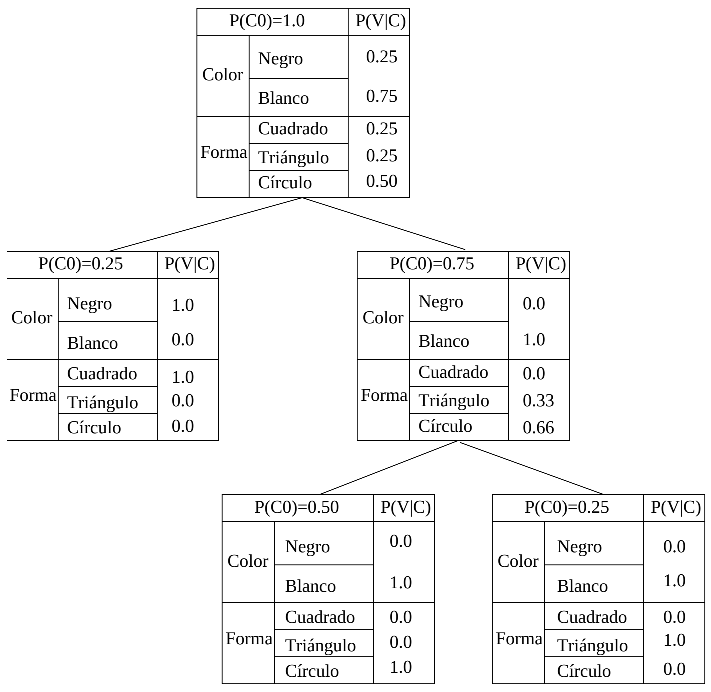
$$\sum_{k=1}^{K} P(A_i = V_{ij}) \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij}|C_k) P(C_k|A_i = V_{ij})$$

Using Bayes theorem this measure can be transformed using only the information that can be estimated from the examples:

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

The term $\sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij}|C_k)^2$ represents the number of attributes that can be correctly predicted for a class. We look for a partition that increases this number of attributes compared to a baseline (no partition):

$$\sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij})^2$$



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

Figure 2.2: Example of COBWEB hierarchy of probabilistic concepts

The category utility measure compares the difference between having a specific partition of the data and no having any partition at all in the following way:

$$CU(\{C_1, \ldots C_K\}) = \frac{\sum_{k=1}^{K} P(C_k) \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij}|C_k)^2 - \sum_{i=1}^{I} \sum_{j=1}^{J} P(A_i = V_{ij})^2}{K}$$

The measure scales by the number of partitions to be able to compare categorization of different sizes. This formula can be extended to continuous attributes assuming they have a specific distribution, for example for gaussian attributes:

$$CU(\{C_1, \ldots C_K\}) = \frac{\sum_{k=1}^{K} P(C_k) \sum_{i=1}^{I} \frac{1}{\sigma_i k} - \sum_{i=1}^{I} \frac{1}{\sigma_i p}}{K}$$

The algorithm (see algorithm 2.3) begins with an empty hierarchy and incrementally incorporates new examples traversing hierarchy and modifying the concepts at each level using four conceptual operators.

- **Incorporate:** Put the example inside an existing class
- **New class:** Create a new class at this level
- **Merge:** Two concepts are merge and the example is incorporated inside the new class (see figure 2.3)
- **Split:** A concept is substituted by its children (see figure 2.3)

The category utility is used to decided what conceptual operator to apply observing what modification of the concepts in a level has the higher value. In order to reduce the computational cost, not all the splits and merge are considered, only the two concepts with higher CU when include the new example can be merged and only the concepts with higher CU can be split.

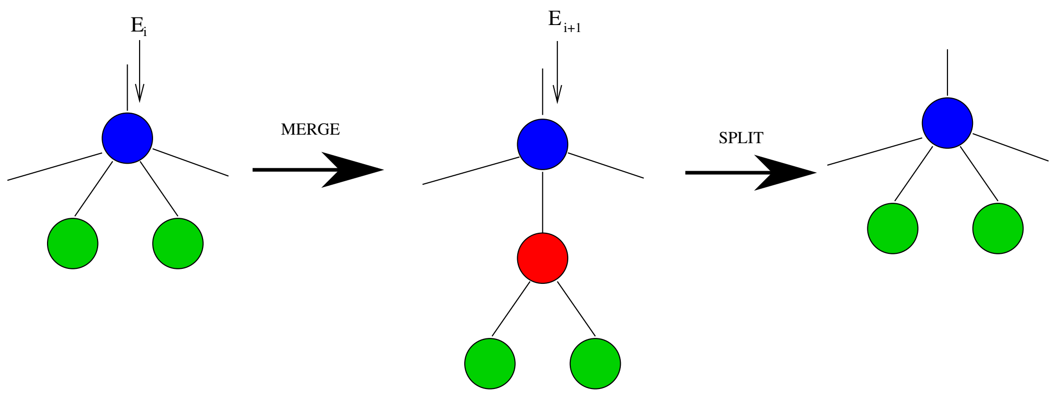


Figure 2.3: Merge and split operators of COBWEB

Algorithm 2.3 COBWEB algorithm

```
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
end
```

2.4 Partitional clustering Algorithms

Partitional clustering approximates the solution to finding the optimal partition of N objects in K groups. This problem is NP-hard requiring the enumeration of all possible partitions of the objects. There is a large variety of algorithms that perform this task using different criteria and information. The next subsections will review some of them including:

- 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

Algorithm 2.4 K-means algorithm

```
Algorithm: K-means (X: Examples, k: integer)
Generate k initial prototypes (e.g. the first k examples)
Assign the k examples to its nearest prototype
SumD = Sum of square distances examples-prototypes
repeat
    Recalculate prototypes
    Reassign examples to its nearest prototype
    SumI = SumD
    SumD = Sum of square distances examples-prototypes
until SumI - SumD < $\epsilon$
```

2.4.1 Prototype/model based clustering

K-means and variants

Prototype and model based clustering assume that clusters fit to a specific shape, so the goal is to discover how different numbers of these shapes can explain the spatial distribution of the data.

The most used prototype based clustering algorithm is K-Means [17]. This algorithm assumes that clusters are defined by their center (the prototype) and they have spherical shapes. The fitting of this spheres is done by minimizing the distances from the examples to these centers. Examples are only assigned to one cluster, this is known as hard clustering or hard partitioning..

Different optimization criteria can be used to obtain the partition, but the most common one is the minimization of the sum of the square euclidean distance of the examples assigned to a cluster and the centroid of the cluster. The problem can be formalized as:

$$\min_{\mathcal{C}} \sum_{\mathcal{C}_i} \sum_{x_j \in \mathcal{C}_i} \parallel x_j - \mu_i \parallel_2$$

This minimization problem is solved iteratively using a gradient descent algorithm, being k a parameter. Algorithm 2.4 outlines the canonical K-means algorithm. Its computational complexity depends on the number of examples (n), the number of features (f), the number of clusters (k) and the number of iterations that the algorithm performs until convergence (I), being linear on all these factors in time with a complexity $O(nfkI)$ The number of iterations can not be predicted apriori and depends on how cluster-like is the data. If the clusters are well separated and there are actually k clusters, the convergence is really fast. This algorithm needs all the data in memory, and the space needed for storing the clusters is negligible compared with the size of the dataset, so the space complexity is $O(nf)$. This spatial complexity makes this algorithm non suitable for large datasets. Using secondary memory to store the data while executing the algorithm is not a good idea, because all the examples have to be check eat iteration to see if their assignment to a cluster has changed.

Figure 2.6 presents a simple example of how the algorithm works. As can be seen, the first iteration assigns the examples to two randomly chosen centers. As the clusters are well separated each iteration the new centers are closer to the centroids, converging in just tree steps of the algorithm in this case to the true clusters.

This algorithm has some drawbacks, being an approximate algorithm, there is no guarantee about the quality of the solution. It converges to a local minima that depends on the initial solution. Because of this, a common strategy to improve the quality of the solution is to run the

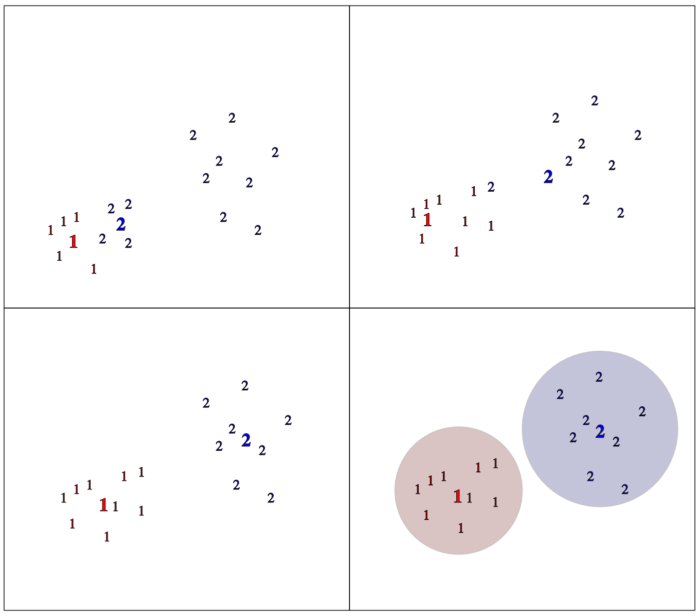


Figure 2.4: K-means algorithm example

algorithm from several random initializations keeping the best one. This increases the actual computational cost of the algorithm.

There are different studies to deal with the sensitivity to initialization. The original algorithm uses random chosen examples as initial centroids. The algorithm *K-means++* ([4]) proposes a seeding technique based on randomization that has proven to yield better results. The idea is to try to maximize distance among initial centers so they cover most of the space of examples. the algorithm is as follows:

1. Choose one center uniformly at random from among the data points
2. For each data point x, compute $d(x, c)$, the distance between x and the nearest center that has already been 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

K-means is also sensitive to clusters with different sizes and densities. Figure 2.5 shows an example of this problem. Both clusters are well separated, but being one small and compact and the other larger and spread makes that the best solution for K-means is to integrate part of the larger cluster in the smallest one.

Outliers it is also an issue for this algorithm, because it is possible that isolated examples would appear as one example clusters. This can be used as a feature of the algorithm as alternative method for outlier detection, but having also in mind the problem of K-means with small clusters. If outliers are not really far from other densities, they will be integrated to other clusters, deviating the centroids from its true position.

Other practical problems of K-means, is how to determine k, the number of clusters. It is usually solved by exploring the range of different possibilities, but information from the domain can help to narrow it down. In the next chapter we will treat the problem of cluster validation

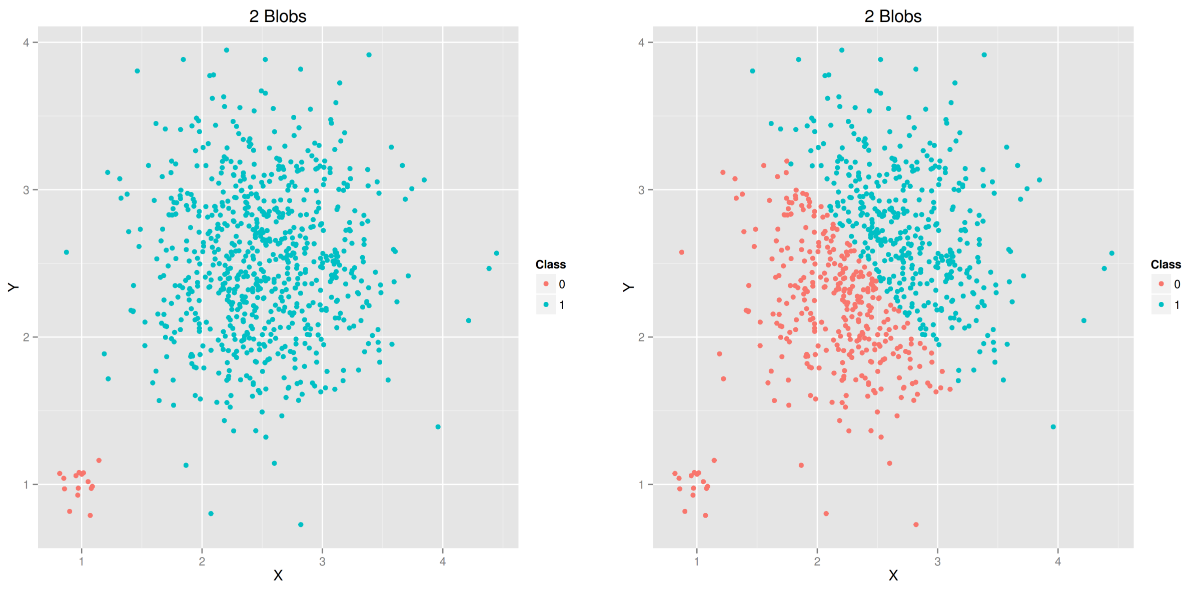


Figure 2.5: Clusters with different sizes and densities are problematic for K-means, left actual data labels, right partition obtained by k-means

that it is closely related to this problem and some indices that help to determine how well a number of clusters fit the data will be discussed.

There are different variants of K-means that try to tackle some of the problems the original algorithm has or that include other methodologies that allow to discover other than spherical clusters. The first one that is worth mentioning is *Bisecting K-means* ([**?**]). This algorithm begins with two clusters and iteratively decides what of the current clusters to split, also in in two, until the desired number of clusters is obtained. This makes that the clusters obtained are hierarchically nested making it similar to the hierarchical method, but in this case there is not a complete hierarchy, only k splits are obtained. There are different criteria that can be used to decide what cluster is split in two. For instance, the size of the clusters can be used, so the largest cluster is the chosen one, or the variance of the distances of the examples to the centroids, so the cluster with larger spread is chosen. Other criteria can be used, but the idea is to use a criteria that selects the less adequate cluster from the current set, so the split results in better ones. This criteria could be used also to decide the parameter k, so the splitting can be performed until all the clusters have a similar quality or the criteria has a value over a specific threshold. This is an outline of the 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 split using a quality criteria
5. Apply K-means to the cluster with k=2
6. If the number of clusters is less than desired repeat from step 3

Other interesting variant is *Global K-means* ([35] that tries to minimize the clustering initialization dependence by exploring the clusterings that can be generated using all the examples as initialization points. For generating a partition with K clusters explores all the alternative partitions from 1 to K clusters, so it includes by construction the exploration of the range of possible number of clusters. It has been show that this algorithm can generate better results than the original K-means but with a larger computational cost because it has to run $K \times N$ the K-means algorithm, so it is prohibitive for large datasets. There some developments alleviating this temporal cost like [5] but with the cost of reducing the quality of the clusters. The following is an outline of the algorithm:

- Compute the centroid of the partition with 1 cluster
- For C from 2 to k:
  - for each example e, compute K-means initialized with the $C-1$ centroids from the previous iteration and an additional with e as the C-th centroid
  - Keep the clustering with the best objective function as the C-clusters solution

Kernel K-means ([15]) is kernelized version of the original algorithm where distances are computed using a kernel in feature space. This helps to discover non spherical clusters/non convex, transforming the original space to one where the cluster shapes are more separable. The first problem is to determine what kernel is suitable for discovering the cluster, what lead to have to explore different possibilities. Also the computation of the kernel increases the computational cost and the centroids that are used to cluster the data are in the feature space, so there is not an image in the original space that can be used to describe the cluster prototypes

K-means assumes that the data is represented using continuous attributes, this means that a centroid can be computed and correspond to a possible example. This is not always the case, sometimes data has nominal attributes or is structured data, so a prototype computed as the mean of the examples of a cluster make no sense. In this case, the approach is to use one or more examples for each cluster as their representative, this is called the medoid. Now the optimization criteria has to change so the sum of distances from each examples to the medoid of their cluster is optimized. This increases also the computational cost of the algorithm. In the case of only using one example as medoid, the cost per iteration for recomputing the is $O(n^2)$, using more examples makes the algorithm NP-hard. This algorithm has the advantage of not being sensitive to outliers. The representative algorithm of this kind is itPartitioning Around Medoids (PAM) ([**?**]), that is as follows:

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

Leader algorithm

The *leader algorithm* is a simple clustering algorithm that has the advantage of being incremental and with a linear time complexity. This makes it suitable for processing streams of data and using different updating strategies to adapt to the changes in the model of the data known as *concept drift*.

Algorithm 2.5 Leader algorithm

```
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
        Introduce the example in the class
        Recompute the prototype
    else
        Create a new prototype with this example
    end
end
```

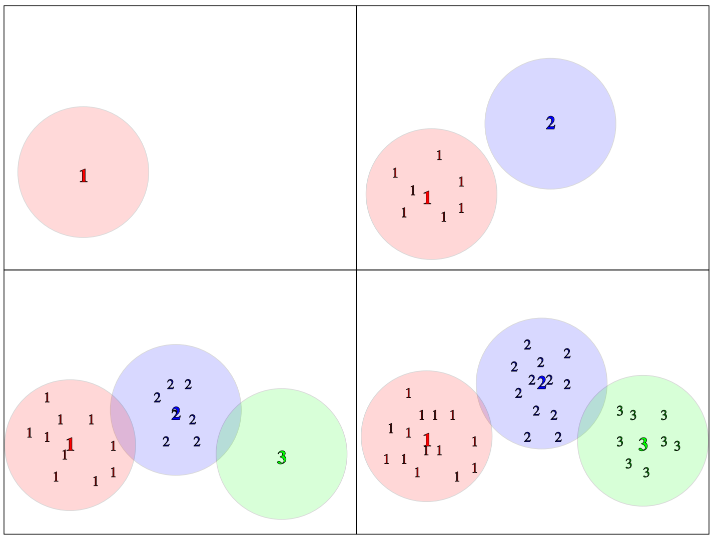


Figure 2.6: Leader algorithm example

The parameter that controls this algorithm represents the radius of the sphere that encloses a cluster. This has an effect similar to the number of clusters in K-means, but in this case all the clusters are assumed to fit inside the radius, meanwhile K-means does not constraint the size of the clusters.

Data is processed incrementally and clusters appear as needed, so the actual number of clusters emerges from the data. A new cluster is generated if there is no current cluster that includes the new example inside the radius, the new cluster will have as center the example. Usually the clusters are represented by the centroids and there is no need of keeping all the seen examples if sufficient statistics are stored. Only storing the number of examples in the clusters, the sum of the examples an the squared sum of examples (that can be computed incrementally) allow to recompute the new center. This makes this algorithm also space efficient. An outline of this algorithm appears in algorithm 2.5.

The quality of the clusters obtained by this algorithm is not usually as good as the one obtained by others, but it can be used as a preprocessing step for scalability, reducing the granularity of the data so it can fit in memory and then apply a better algorithm to the reduced data.

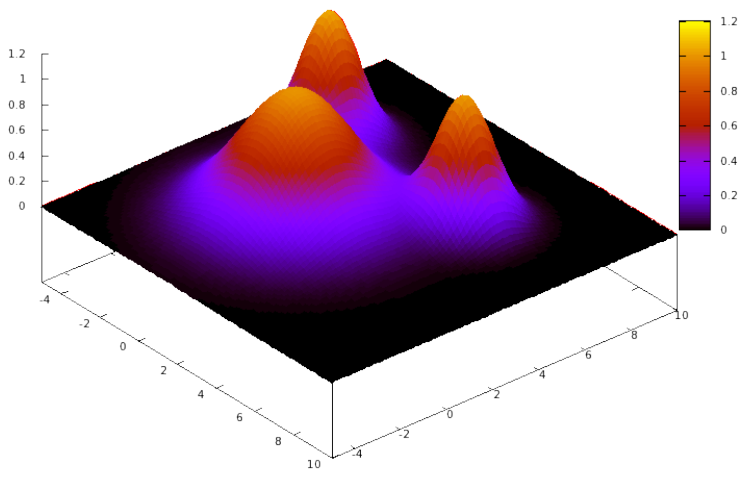


Figure 2.7: Clusters defined by bivariate gaussian distributions

Gaussian Mixtures

Model based clustering assumes that the dataset can be fit to a mixture of probability distributions. The shape of the clusters depends on the specific probability distribution used. A common choice is the gaussian distribution. In this case, depending on the choice about if covariances among attributes are modeled or not, the clusters correspond to arbitrarily oriented ellipsoids or spherical shapes (see figure 2.7). The model fit to the data can be expressed in general as:

$$P(x|\theta) = \sum_{i=1}^{K} P(w_i)P(x|\theta_i, w_i)$$

Being K the number of clusters, with $\sum_{i=1}^{K} P(w_i) = 1$. This model is usually fit using the Expectation-Maximization algorithm (EM), assigning to each example a probability to each cluster, that means that we are obtaining a soft clustering of the data.

The main limitation of all these methods is to assume that the number of partition is known. Both types of algorithms need to have all the data in memory for performing the computations, so their spatial needs scale linearly with the size of the dataset. The storage of the model is just a fraction of the size of the dataset for prototype based algorithms, but for model base algorithms depends on the number of parameters needed to estimate the probability distributions, this number can grow quadratically with the number of attributes if, for example, gaussian distributions with full covariance matrices are used.

As we discussed in the section about k-means, the computational time cost for the prototype based algorithms each iteration is $O(nfk)$. For the model based algorithms, each iteration has to estimate all the parameters (p) of all the distributions of the model (k) for all instances (n), the number of parameters is proportional to the number of features and also depends on what assumptions make about the independence of the attributes. In the case of full covariance estimation, each iteration results in a total time complexity of $O(nf^2k)$, if we assume attribute independence time complexity is $O(nf^2k)$.

For learning this model the goal is to estimate the parameters of the distribution that describes each class (e.g.: means and standard deviations). The Expectation Maximization

(EM) algorithm is the usual choice. It maximizes the likelihood of the distribution respect the dataset iteratively beginning with an initial estimate of the parameters. It performs to steps until convergence (no further improvements of the likelihood):

- **Expectation:** a function that assigns a degree of membership to all the instances to any of the K probability distributions is computed using the current estimation of the parameters of the model.
- **Maximization:** the parameters of the distributions are reestimated using as weights the memberships obtained in the previous step to maximize the memberships of the examples to the new model.

Each specific model has a set of parameters that have to be estimated using the EM algorithm. In the case of choosing the Gaussian distribution parameters are the mean and covariance matrix, so the model is:

$$P(x|\overrightarrow{\mu},\Sigma) = \sum_{i=1}^{K} P(w_i)P(x|\overrightarrow{\mu_i},\Sigma_i,w_i)$$

The actual computations depend on the assumptions that we make about the attributes (independent or not, same σ, ...). If the attributes are assumed independent μ_i and σ_i have to be computed for each class ($O(k)$ parameters) (the model is described by hyper spheres or ellipsoids parallel to coordinate axis. If the attributes are modeled as not independent μ_i, σ_i and σ_{ij} have to be computed for each class ($O(k^2)$ parameters) (the model is described by hyper ellipsoids non parallel to coordinate axis)

For the case of A independent attributes, each cluster has the model:

$$P(x|\overrightarrow{\mu_i},\Sigma_i,w_i) = \prod_{j=1}^{A} P(x|\mu_{ij},\sigma_{ij},w_i)$$

Being the model to fit:

$$P(x|\overrightarrow{\mu},\overrightarrow{\sigma}) = \sum_{i=1}^{K} P(w_i) \prod_{j=1}^{A} p(x|\mu_{ij},\sigma_{ij},w_i)$$

The update rules used by the EM algorithm for any model are obtained from the log likelihood of the model, differentiating for each parameter and finding where the gradient is zero. For the case of gaussian distributions the update of the parameters (we omit the derivations) in the maximization step is:

$$\begin{aligned}
\hat{\mu}_i &= \frac{\sum_{k=1}^{N} P(w_i|x_k,\overrightarrow{\mu},\overrightarrow{\sigma})x_k}{\sum_{k=1}^{N} P(w_i|x_k,\overrightarrow{\mu},\overrightarrow{\sigma})} \\
\hat{\sigma}_i &= \frac{\sum_{k=1}^{N} P(w_i|x_k,\overrightarrow{\mu},\overrightarrow{\sigma})(x_k-\hat{\mu}_i)^2}{\sum_{k=1}^{N} P(w_i|x_k,\overrightarrow{\mu},\overrightarrow{\sigma})} \\
\hat{P}(w_i) &= \frac{1}{N}\sum_{k=1}^{N} P(w_i|x_k,\overrightarrow{\mu},\overrightarrow{\sigma})
\end{aligned}$$

The EM algorithm is initialized with a set of K initial distributions usually obtained with K-means. The μ_i and $\sigma_{i,j}$ for each cluster and attribute are estimated from the hard partition. The algorithm iterates until there is not improvement in the log likelihood of the model given the data.

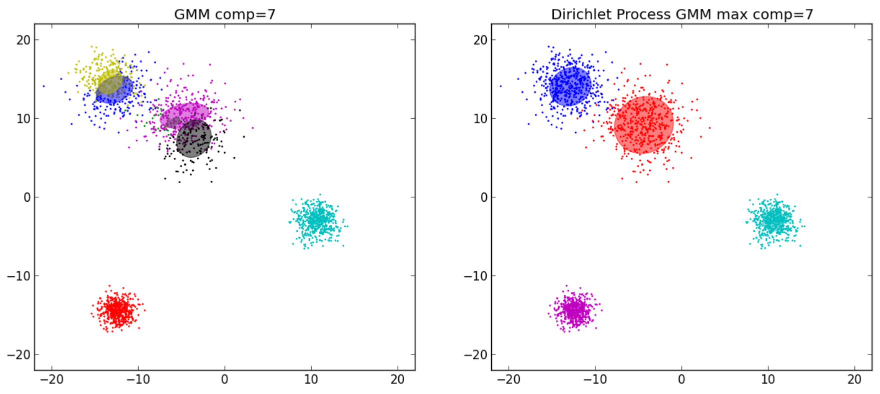


Figure 2.8: plain GMM versus Dirichelet process

It can be seen that K-means is a particular case of this algorithm where only the mean of the distributions is estimated and each example has only weights 1/0 determined by the closest centroid. The main advantage of GMM is that we obtain a membership as a probability (soft assignments) that can be useful for some tasks. Also, using different probability distribution we can find different kinds of structures in the data, not only spherical ones. The main drawback is the derivation of the update rules for the EM algorithm. In the case of gaussian distributions there is a closed form for the updates, but other models has to resort to other methods for the estimation like MonteCarlo Markof Chains (MCMC) or variational inference, increasing the computational cost of the maximization step.

Just as in K-means, the main issue of GMM is to decide a priori the number of components. The mixture model paradigm gives more flexibility and allows introducing the number of clusters as a part of the estimation parameters. This is introduced by adding to the model a prior described as a Dirichlet distribution. The Dirichlet distribution makes the assumption of an unbound number of components, introducing a finite weight that is distributed among all the components. The estimation process allocates the examples and distributes the weights to the components deciding what number of components better suits the data. This method is known as the Dirichlet process Gaussian Mixture Model. Figure 2.8 shows a GMM and a Dirichlet Process GMM fitted to a dataset with only 4 clusters but using 7 as the guess of the number of clusters. As can be seen the Dirichlet process GMM nullifies the weights of the extra components while the GMM divides some cluster to obtain the requested number of partitions.

Fuzzy C-means

Fuzzy clustering relax the hard partition constraint of K-means like gaussian mixture models, but in this case fuzzy membership functions are used for distributing the assignment of examples to clusters, so we have soft clusters. This is an advantage from other algorithms when the groups are overlapped.

The Fuzzy clustering algorithms optimize the following function:

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

where $\sum_{k=1}^{K} \delta(C_k, x_i) = 1$ and b is a blending factor that allows a continuum between hard and soft partitions.

Fuzzy C-means is the most known fuzzy clustering algorithm, and corresponds to a fuzzy version of K-means. The algorithm performs an iterative optimization of the above objective function, updating of the cluster centers using the following equation:

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

and the updating of the memberships of the examples to the clusters as:

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

This behavior can be assimilated to the EM algorithm used by GMM.

The bias of the C-means algorithm is to look for spherical clusters like the previous algorithms, but there are other alternative optimization functions that allow for more flexible forms but increasing the computational cost, like for instance:

- 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

This method allows also for using other objective functions that look for specific geometrical shapes in the data (lines, rectangles, ...) making this characteristic specially useful in image segmentation and recognition.

2.4.2 Density based clustering

Density based clustering does not assume a specific shape for the clusters or that the number of clusters is known. The goal is to uncover areas of high density in the space of examples. There are different strategies to find the dense areas of a dataset, but the usual methods are derived from the works of the algorithm DBSCAN [20]. Initially was defined for spatial databases, but can be applied to data with more dimensionality.

This algorithm is based on the idea of *core points*, that constitute the examples that belong to the interior of the clusters, and the neighborhood relations of these points with the rest of the examples.

We define the set *ε-neighborhood*, as the instances 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 example that have a certain number of elements in $N_\varepsilon(x)$

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

From this neighborhood sets, different reachability relations are defined allowing to connect density areas defined by these core points. We say that two instances p and q are *Direct Density Reachable* with respect to ε and $MinPts$ if:

1. $p \in N_\varepsilon(q)$
2. $|N_\varepsilon(q)| \geq MinPts$

We say that two instances p and q are *Density Reachable* if there are a sequence of instances $p = p_1, p_2, \ldots, p_n = q$ where p_{i+1} is direct density reachable from p_i. And, finally, p and q are **Density connected** if there is an instance o such that both p and q are *Density Reachable* from o.

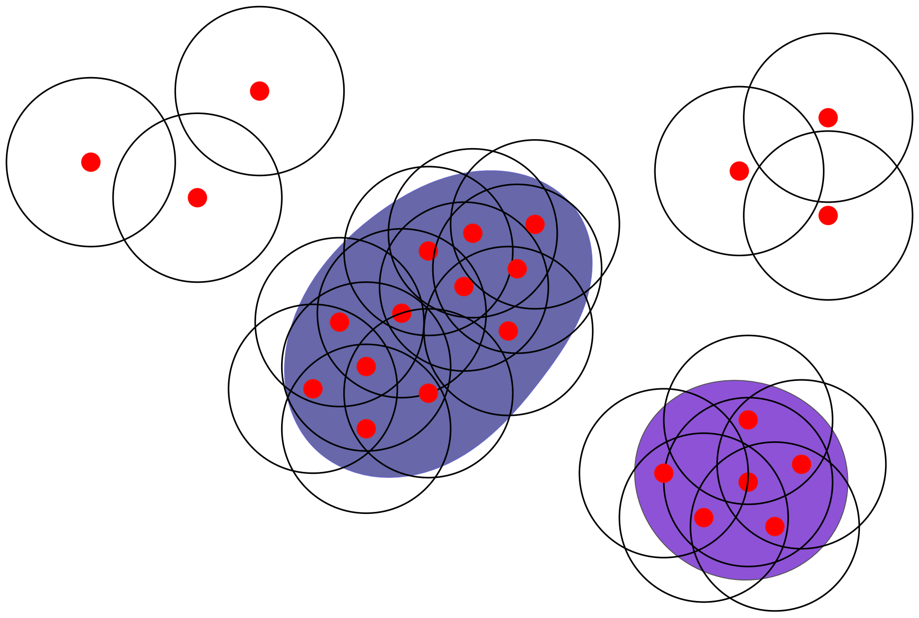


Figure 2.9: DBSCAN corepoints and ε-neighborhood

A cluster is defined as all the core points that are connected by this reachability relations and the points that belong to their neighborhood. Formally, 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 point that can not be connected using these relationships is treated as noise.
This is an outline of the DBSCAN algorithm:

1. Start with an arbitrary instance and compute all density reachable instances with respect ε and $MinPts$.
2. If it is a core point we will join all examples related to this example and obtain a group (all these examples are labeled and discarded from further consideration), otherwise, the instance belongs to the border of a cluster or is an outlier, so it is not considered as starting point.
3. Repeat the first step until no more core points are found.
4. Label the non clustered examples as noise.

See figure 2.10 for an example.
The key point of this algorithm is the choice of the ε and $MinPts$ parameters. A possibility is to set heuristically their values using the thinnest cluster from the dataset. As an extension of DBSCAN, the algorithm OPTICS [2] uses heuristics based on the histogram of the distances among the examples to find good values for these parameters.

The main drawback of these methods come from the cost of finding the nearest neighbors for an example. To decrease the computational cost an R* tree or a ball tree can be used to index all examples, but these structures degrade with the number of dimensions to a linear search. This makes the computational time of these algorithms proportional to the square of the number of examples for datasets with many dimensions.

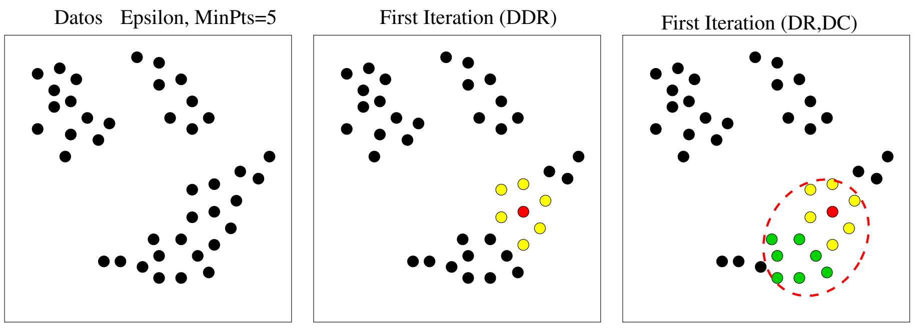


Figure 2.10: Example of the DBSCAN algorithm

Other example of density based clustering algorithm is DENCLUE ([28]). It is based on kernel density estimation and defines the influence of an example in a dataset as the sum of the densities based on the examples computed using a kernel for all the dataset (for instance a gaussian kernel)

$$f_B^y(x) = f_B(x, y) = \sum_{i=1}^{N} e^{-\frac{1}{2}\frac{|x-y_i|^2}{\sigma^2}}$$

It defines the density function of a dataset as the sum of the influences of all examples of the dataset

$$f_B^D(x) = \sum_{i=1}^{N} f_B^{x_i}(x_x i)$$

This function takes the role of the ε-neighborhood from DBSCAN, but in this case as a continuous function. Instead of using density relations among points, it defines the gradient of $f_B^D(x)$ as:

$$\bigtriangledown f_B^D(x) = \sum_{i=1}^{N}(x_i - x) f_B^{x_i}(x)$$

so the direction of the gradient points to the more dense parts of the datasets, assimilate to the core points in DBSCAN.

The algorithm defines that a point x^* is a *density-attractor* iff is a local maximum of the density function f_B^D. A point x is *density-attracted* to a density-attractor iff $\exists k \in N; d(x^k, x^*) \leq \epsilon$

$$x^0 = x, x^i = x^{i-1} + \delta \cdot \frac{\bigtriangledown f_B^D(x^{i-1})}{\| \bigtriangledown f_B^D(x^{i-1})\|}$$

Two kinds of clusters can be defined, *Center-defined Cluster* (wrt to σ,ξ) for a density-attractor x^* as the subset of examples being density-attracted by x^* and with $f_B^D(x^*) > \xi$. This bias the algorithm towards finding spherical clusters centered on the more dense areas of a dataset defined by a central point. We have also an *Arbitrary-shape cluster* (wrt to σ,ξ) for a set of density-attractors X as the subset of examples being density-attracted to any $x^* \in X$ with $f_B^D(x^*) > \xi$ and with any density-attractor from X connected by a path P with $\forall p \in P : f_B^D(p) > \xi$. This allows obtaining a cluster as the sum of different close dense areas. Figure 2.11 show an example of the gradients of the density function and the central points of the densities as density attractors.

This algorithm is more flexible than DBSCAN because of the use of the approximation of the densities using kernel density estimation. This makes that different algorithms can be

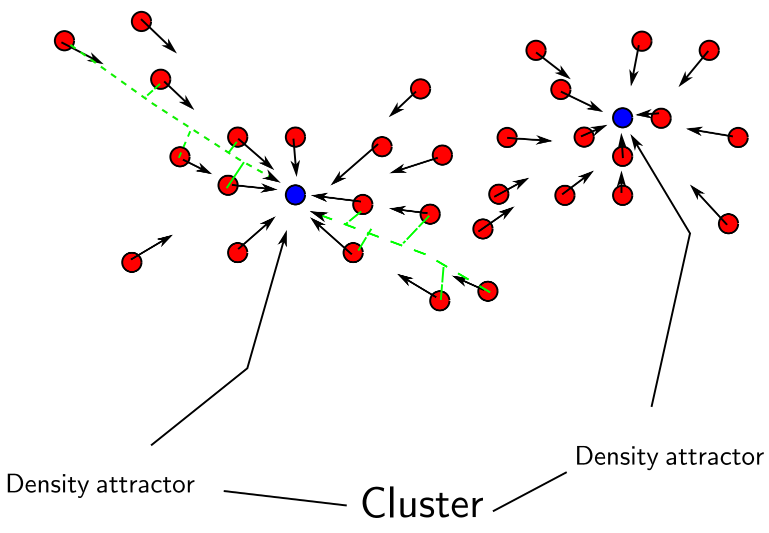


Figure 2.11: DENCLUE algorithm

reproduced with the right choice of σ and ξ parameters, for example the own DBSCAN can be obtained using arbitrary shape clusters, K-means using center defined clusters and hierarchical clustering merging different densities hierarchically.

The algorithm is divided in two phases:

1. *Preclustering:* The dataspace is divided in d-dimensional hypercubes, only using the cubes with datapoints. These hypercubes are mapped to a tree structure for efficient search. From these hypercubes, given a threshold, only the highly populated ones and their neighbors are considered.

2. *Clustering:* For each datapoint in the hypercubes a local density function and a local gradient are computed. Using a Hill-Climbing algorithm the density-attractor for each point is computed. For efficiency reasons each point near the path computed during the search of a density-attractor is assigned to that density-attractor.

Due to efficiency reasons, the density functions are only computed for the neighbors of each point.

2.4.3 Grid based clustering

Grid based clustering is another approach to finding dense areas of examples. The basic idea is to divide the space of instances in hyperrectangular cells by discretizing the attributes of the dataset. In order to avoid generating a combinatorial number of cells, different strategies are used. It has to be noticed the fact that, the maximum number of cells that contain any example is bounded by the size of the dataset.

These methods have the advantage of being able to discover clusters of arbitrary shapes and also the number of cluster has not to be decided beforehand. The different algorithms usually rely on some hierarchical strategy to build the grid top down or bottom up.

For example, the algorithm STING [44] assumes that the data has spatial relation (usually two dimensional) and, beginning with one cell, recursively partitions the current level into four cells obtaining a hierarchical grid structure determined by the density of the examples (see figure 2.12). Each level divides the cells from the previous level in rectangular areas and the size of the cells depends on the density of the examples. Each cell is summarized by the

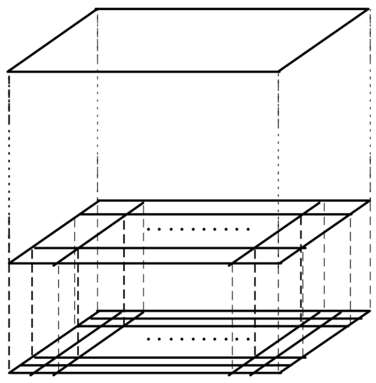


Figure 2.12: Hierarchical grid stucture built by STING

sufficient statistics of the examples it contains. (number of instances, mean, deviation, min value, max value, type of distribution)

This structure allows querying for regions that hold certain conditions. The structure is examined and the cells containing data relevant to the query are returned. The algorithm is as follows:

1. For each cell of a layer determine if it is relevant to the query
2. If it is not the bottom layer repeat the process with the cells marked relevant
3. If it is the bottom layer, find all the relevant cells that are connected and return the region that form

The results of this algorithm approximate those returned by DBSCAN as the granularity approaches zero.

The algorithm of CLIQUE [1] uses a more general approach. It assumes that the attributes of the dataset have been discretized and the one dimensional dense cells for each attribute can be identified. These cells are merged attribute by attribute in a bottom up fashion, considering that a merging only can be dense if the cells of the attributes that compose the merger are dense. This antimonotonic property allows to prune the space of possible cells. Once the cells are identified, the clusters are formed by finding the connected components in the graph defined by the adjacency relations of the cells.

The algorithm can generate CNF descriptions from the groups that discovers and its goal is to find a space with fewer dimensions where the groups are easier to identify. The process is divided in three steps:

1. Identify the subspaces with clusters

   A bottom up strategy is applied identifying first the bins of higher density in one dimension and combining them (see figure 2.13). A combination of $k + 1$ dimensions can only have high density bins if there are high density bins in k dimensions, this rule gives the set of candidate to high density bins when we increase the dimensionality by one. Some other heuristics are used to reduce the computational cost of the search

2. Identify the clusters

   A set of dense bins are received from the previous step, the contiguous bins are considered as forming part of the same cluster (see figure 2.14). Two bins are connected if they share a side in one dimension or there is an intermediate bin that connect them. This problem is equivalent to find the connected components of a graph.

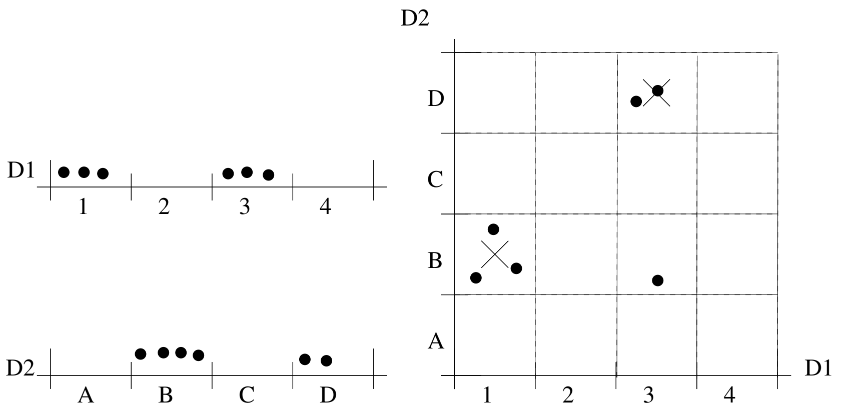


Figure 2.13: CLIQUE algorithm step 1

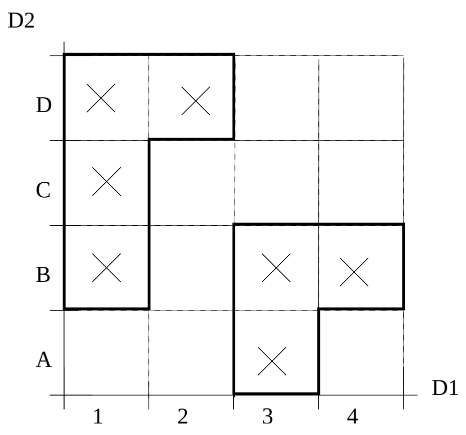


Figure 2.14: CLIQUE algorithm step 2

3. Generate the minimal description of the clusters

   Now a set of connected components in k dimensions are received and the problem solved is to look for the minimal cover for each connected component (see figure 2.15). Being this problem NP-hard, an approximation is computed using rectangles that cover part of the group maximally and after this the redundancies are reduced, obtaining a computational affordable approximation to the problem.

All these methods can usually scale well, but it depends on the granularity of the discretization of the space of examples. The strategies used to prune the search space allow to largely reduce the computational cost, that scales on the number of examples and a quadratic factor in the number of attributes.

2.4.4 Other approaches

There are several other approaches that use other methodologies for obtaining a set of clusters from a dataset. We are going to outline three different approaches: methods based on graph theory and properties of neighborhood graphs, methods based on unsupervised neural networks and methods based on support vector machines.

Graph based

There are some classical approaches to graphs clustering that use basic principles as criteria for partitioning the data and are the base for more complex algorithm. The idea is that different

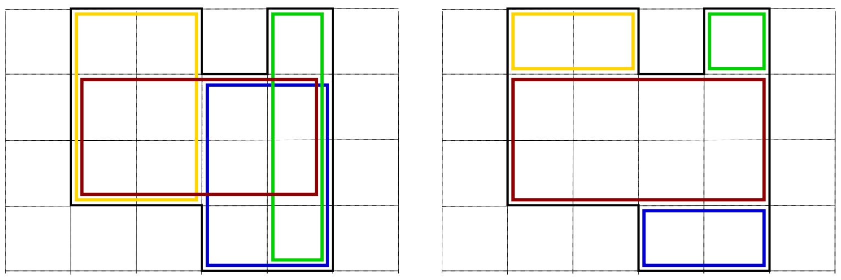


Figure 2.15: CLIQUE algorithm step 3

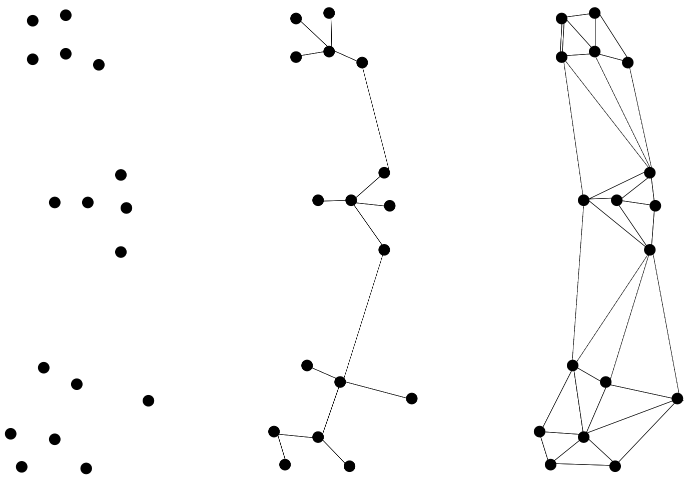


Figure 2.16: Different graphs (MST/Delanau) computed from a set of examples

kinds of graphs can be computed using the similarity matrix defined by the examples like the Minimum Spanning Tree (MST), Voronoi tesselation or Delanau triangularization (see figure 2.16) that capture at different levels the local structure of the data. Using these graphs, a consistency criteria for the edges of the graph is defined.

This criterion can be used agglomeratively or divisively, merging examples/groups that have edges below a threshold or deleting inconsistent edges above a threshold until some unconnected component are obtained from the initial graph (see figure 2.17). These components define the clusters of the data. These algorithms have two advantages, first we do not need to define the number of classes beforehand, this number is controlled by the consistency criteria and the densities in the data. Second, we do not impose a specific model for the clusters, so the can have any shape, being more adaptive to the actual densities.

An example of this methodology is CHAMELEON ([32]). This is an agglomerative algorithm that is based on the graph obtained from the matrix distance using only the k nearest neighbors of each example. It defines two measures for the decision about the consistency of the edges of the graph, the relative interconnectivity $RI(C_i, C_j)$ and relative closeness $RC(C_i, C_j)$ between subgraphs. These measures are defined from the cost of the edges that divide two groups. Two groups could be merged if the interconnectivity between their graphs and the closeness among instances is high.

$RI(C_i, C_j)$= (sum of the edges that connect the two groups)/ (sum of the edges that partitions each group in two equal sized groups)

$RC(C_i, C_j)$= (mean of the edges that connect the two groups)/ (mean of the edges that partitions each group in two equal sized groups)

Initially the algorithm divides the generated graph deleting edges from the distance matrix until many small groups are obtained (minimizing the cost of the edges inside the group).

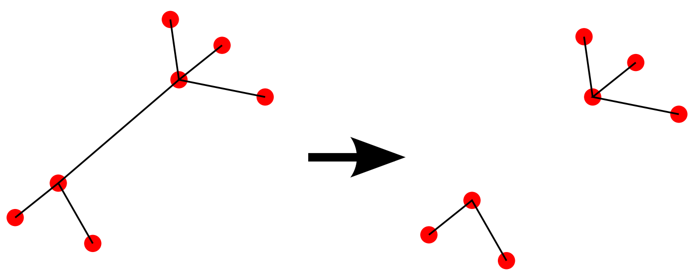


Figure 2.17: Deleting inconsistent edges from an examples graph generates clusters

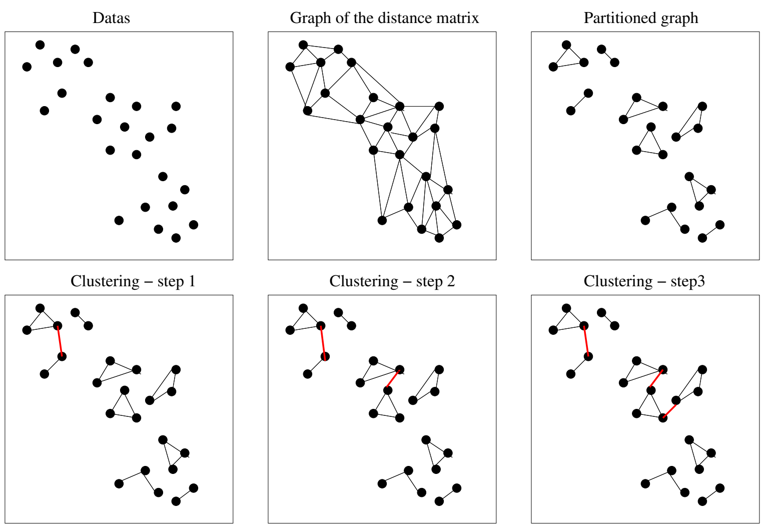


Figure 2.18: Example of the CHAMALEON algorithm in action. First the k-neigbor graph is defined and the graph is partitioned so small components are obtained, then new edges are added in order optimizing the RI/RC combination measure.

Then it uses an agglomerative process to merge the groups using the measures of relative interconnectivity and closeness ($RI(C_i, C_j) \cdot RC(C_i, C_j)^\alpha$ is used as goal function). Figure 2.18 presents an example of how this algorithm works.

In the latest years other methods also related to graphs but using different perspectives have gained popularity spectral clustering and affinity clustering are two examples.

Spectral graph theory defines properties that hold the eigenvalues and eigenvectors of the adjacency matrix or Laplacian matrix of a graph. Spectral clustering [36] uses the Laplacian matrix defined from the similarity matrix of the dataset. Different clustering algorithms can be defined depending on how it is computed. For example, the whole matrix can be used or only the neighborhood graph. Also, different normalizations can be applied to the Laplacian matrix.

If we start with the similarity matrix (W) of a dataset (complete or only k-neighbors). The degree of an edge is defined as:

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

The degree matrix D is defined as the matrix with values $d_1, d_2, \ldots, d_n$ as diagonal. From this matrix different Laplace matrices can be computed:

- Unnormalized: $L = D - W$
- Normalized: $L_{sym} = D^{-1/2}LD^{-1/2}$ or also $L_{rw} = D^{-1}L$

Assuming that the Laplacian matrix represents the neighborhood relationships among examples, applying an eigen decomposition of this matrix, the eigenvectors of this matrix define a new set of coordinates that can be interpreted as a dimensionality reduction method if only the k components corresponding to the k highest values are used.

It is known that if there are well defined clusters in the data, the decomposition of the Laplacian matrix will have only an specific number of eigenvalues larger than zero, corresponding to the number of clusters. Also, the eigenvectors are in a space where the clusters are more easy to identify. This means that traditional clustering algorithms can be applied to discover these clusters, for instance, K-means.

Also, graph partitioning algorithms can be applied to the matrix for obtaining a set of unconnected components. For instance, the min-cut problem can be applied to the defined graph. Several objective functions have been defined for this purpose. 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 :

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

Sometimes, using directly the weights of the Laplacian does not always obtain a good result. There are alternative objective functions that are also used like:

$$\begin{aligned} RatioCut(A_1,\ldots,A_k) &= \sum_{i=1}^{k} \frac{cut(A_i,\overline{A_i})}{|A_i|} \\ Ncut(A_1,\ldots,A_k) &= \sum_{i=1}^{k} \frac{cut(A_i,\overline{A_i})}{vol(A_i)} \end{aligned}$$

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

The computational complexity of this family of methods is usually high because the computation of the eigenvectors of the Laplacian matrix is needed. This cost can be reduced by using approximate methods for estimating the first k eigenvalues of the matrix.

Affinity clustering [22] is an algorithm based on message passing but related to graph partitioning in the graph obtained from the data similarity matrix. The algorithm establishes the set of examples that have to be the cluster prototypes and how the examples are attached to them. Each pair of objects have a distance defined $s(i,k)$ (e.g.: euclidean distance). The number of clusters is not fix apriori, but depends of a parameter called *damping factor* that controls how the information about the assignment of examples to clusters is computed.

The method define two measures, *responsibility*, that accounts for the suitability of an exemplar for being a representative of a cluster and *availability*, that accounts for the evidence that certain point is the representative of other example.

- *Responsibility* $r(i,k)$, that 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. This represents the accumulated evidence of how appropriate would be for point i to choose point k as its exemplar

These measures are initialized using the similarity among the examples, and also, each example begins with a value for $r(k,k)$ that represents the preference for each point to be an exemplar. Iteratively, the number of clusters and their representatives are determined by refining these measures, that are linked by the following set of equations:

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

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

- 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.

The algorithm proceeds iterativelly as follows:

1. Updates 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

See figure 2.19 for an example of this algorithm in action. The temporal computational complexity of this method is quadratic on the number of examples and the space complexity is also quadratic, making it not suitable for large datasets. This algorithm is closely related to other message passing algorithms used for belief propagation in probabilistic graphical models

2.4.5 Unsupervised neural networks

There are some unsupervised neural networks methods, among them *self-organizing maps* are widely used for visualization tasks. This method can be interpreted as an on-line constrained version of K-means. It transforms the data to fit in a 1-d or 2-d regular mesh (rectangular or hexagonal) that represents the clusters in the data. The nodes of this mesh correspond to the prototypes of the clusters. This algorithm can also be used as a dimensionality reduction method (from N to 2 dimensions).

To build the map we have to fix the size and shape of the mesh (rectangular/hexagonal). This constrains the number of clusters that can be obtained and how are they related. Each node of the mesh is a multidimensional prototype of p features.

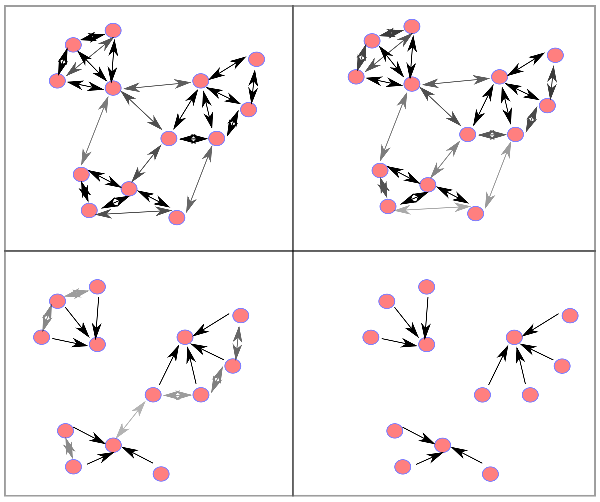


Figure 2.19: Affinity propagation in action. Each iteration the resposibility increases for some examples (cluster prototypes) and decreases for the rest.

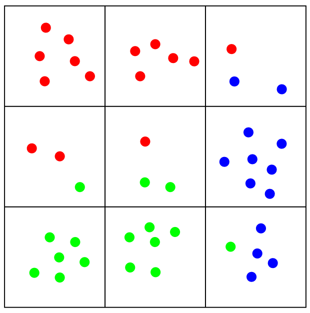


Figure 2.20: Example of self-organizing map using a 3×3 grid

The canonical SOM algorithm is outlined in 2.6. There can be seen some differences to the standard batch K-means algorithm. Each iteration each example is assigned sequentially to the closer prototype and it is moved a fraction in the direction of the example (instead of computing the centroid). Also, the neighbor prototypes are also affected. The effect of this procedures is to transform the low dimensional mesh to be closer to the data, but maintaining the fixed neighborhood relationships among the prototypes.

This algorithm has different variations that affect to the update strategy and the behavior of the parameters. The performance of the algorithm can be controlled by the learning rate α, that represents how much the prototype is moved towards the data each iteration. It is usually is decreased from 1 to 0 during the iterations to accelerate convergence. The neighborhood of a prototype is defined by the adjacency of the cells and the distance among the prototypes. The number of neighbors used in the update can also be decreased during the iterations from a predefined number to 1 (only the prototype nearest to the observation). Also, the algorithm could give different weights to the effect on the neighbors depending on the distance among the prototypes, so if a prototype gets apart from the rest it will have less or no effect on them.

2.4.6 SVM Clustering

Support Vector Machines are a supervised learning method that uses kernel transformation of the feature of the data for obtaining a separating hyperplane. This procedure can be extended in different ways to the unsupervised counterpart.

The simplest formulation is the approach using the one class-SVMs. This machines are able to work with only positive examples and can be used to estimate their probability density. They

Algorithm 2.6 SOM algorithm

```
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
            $m_k = m_k + \alpha(x_i - m_k)$
        end
    end
end
```

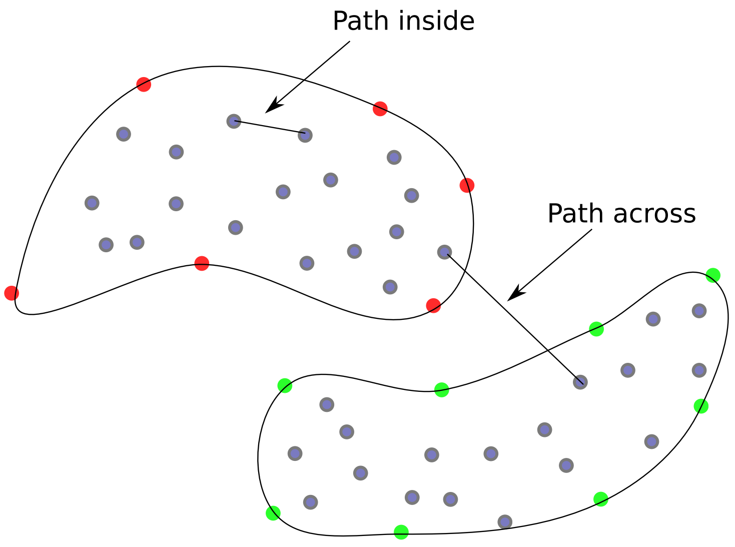


Figure 2.21: Examples of SVM clustering. The labeling of the examples is determined by determining the sphere that contains each example

are usually applied to outlier detection, so examples that are outside the separating hyperplane are considered negative examples.

This allows defining different algorithms, the original one was defined in [6], that finds the smallest enclosing sphere in feature space for the data using a gaussian kernel:

$$||\Phi(x_j) - a||^2 \leq R^2 \ \ \forall j$$

with a the center of the sphere and R the minimum radius. The data is then mapped back to the original space and the support vectors (examples that lie in the border of the sphere) are used to partition the dataset. The labeling of the examples is determined assuming that the sphere is divided in different spheres in the original space and the path that connects examples from different clusters has to cross outside the sphere borders (see figure 2.21).

Other approaches use an algorithm similar K-means like [10] or define a more general optimization problem that looks for a set of dividing planes for the examples like Maximum Margin Clustering ([49]).

2.5 Examples

2.5.1 Twitter Geoprofiling

This is an application to the study of the behavior of people living and visiting a large city. The goal is to analyze if there are patterns on their geographical behavior. The dataset used consist on the geolocalization of Tweets and Instagram posts inside a constrained geographical area. The data used is restricted to geographical information (latitude/longitude) and the time stamp of the post.

The process of analysis is divided in two steps, first the Geographical discretization of the data for obtaining a representation of the users characteristics, and second, the discovery of different behavior profiles in the data.

The dataset is composed of a few millions of events during several months (represents less than 5% of the actual events)inside an area of 30×30 Km^2 of the city of Barcelona. Each event contains a geographical position and a time stamp. For the purpose of the analysis (geoprofiles) the actual data is difficult to analyze because the resolution of the coordinates is too fine. There is a low probability of having two events in the exact same place. Clustering geographically the events can help to make more sense of the data.

Being the data geographically distributed there are few alternatives (clusters of arbitrary shapes), the more adequate methods seem Density based clustering or Grid based clustering. The main problem is that the size of the dataset ($\sim$ 2.5 million events) could arise scalability issues, so a preliminary coarse grained clustering could be helpful (for example using k-means, or the leader algorithm).

The *epsilon* parameter of DBSCAN in this case has an intuitive meaning, how close geographically the points have to be so they are considered in a cluster. The *minpoints* is more difficult, for this dataset, some areas have a lot of events and others have a few of them. Another curious thing about this dataset is that people generate events from almost everywhere, so we can end having just one cluster that connects everything.

Unfortunately the results using DBSCAN are not so good, apart from taking a long time to obtain the results ($\sim$ 4 hours), only a few clusters are found and many of examples is discarded as noise. It was decided that was more informative to have a coarse discretization based on the leader algorithm. The granularity is defined by the radius parameter of the algorithm. The linear computational cost made possible to experiment with this parameter to find an adequate value. Figure 2.22 shows two examples of the resulting clusters.

We want to find groups of users that have similar behavior (in geographical position and time) along the period of data collection. The clusters obtained from the discretization can be the basis for the geographical profiles. Additionally, a discretization of time can provide a finer grained representation. Different representation can be used to generate the dataset:

- Presence/absence of a user in a place at a specific time interval
- Absolute number of visits of a user to a place in a time interval
- Normalized frequency of the visits of a user to a place in a time interval
- ...

Different choices of representation and discretization allow for different analysis.

For obtaining the profiles, it is more difficult to choose what clustering algorithm is adequate because there is not an intuition about how are distributed. We can explore different alternatives and analyze the results. Some choices will depend on:

- If the dataset generated is continuous or discrete

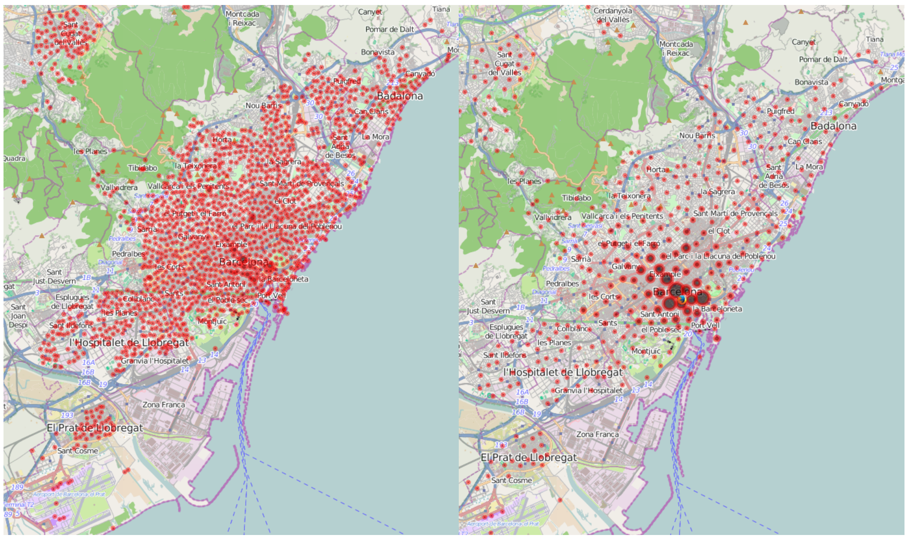


Figure 2.22: Clustering of events using the leader algorithm with different radius

- The size of the dataset
- Our assumptions about the model that represents our goals (Shape of the clusters, Separability of the clusters/Distribution of examples)
- Interpretability/Representability of the clusters

Experts on the domain have to validate the results, but some general assumptions can be used to evaluate the alternatives. We will explore two alternatives:

- K-means algorithm (simple and efficient, spherical clusters)
- Affinity propagation clustering (adaptive and non predefined shape)

Clustering results depend on the chosen representation, our assumption is that several profiles should arise from the data and the number of people for each profile should be large.

For K-means, if a large value of k is used, very small clusters are bound to appear. We can experiment with a reasonable range of values. From the results, most of clusterings result in a very large cluster (most of the data) and several very small ones. The clustering that is closer to our assumptions is the one that uses binary representation normalized using TF-IDF. This obtains a relatively large number of clusters (around 25) with a large cluster, some medium sized clusters and a few small ones. Profiles seem more biased to define people that moves around large regions and the small clusters are not supported by many examples. Figure 2.23 shows some examples of the profiles obtained using this algorithm with localized behavior at different places of the city.

Using affinity propagation the only parameter of the algorithm to tune is the damping factor. We explored the range of possible values (from 0.5 to 1) to see how many clusters appear natural to this algorithm. Like for K-means the results vary depending on the representation used for the examples. There are a slightly larger number of clusters and the examples are more evenly distributed among them. As for the previous algorithm, the clustering closer to our assumptions are the ones obtained using the binary representation for the attributes. A larger number of clusters accounts for general and also more specific behaviors. They have a more evenly distributed of sizes and the different profiles are supported by a number of examples enough to be significant. Figure 2.24 shows some examples of the profiles obtained using this algorithm.

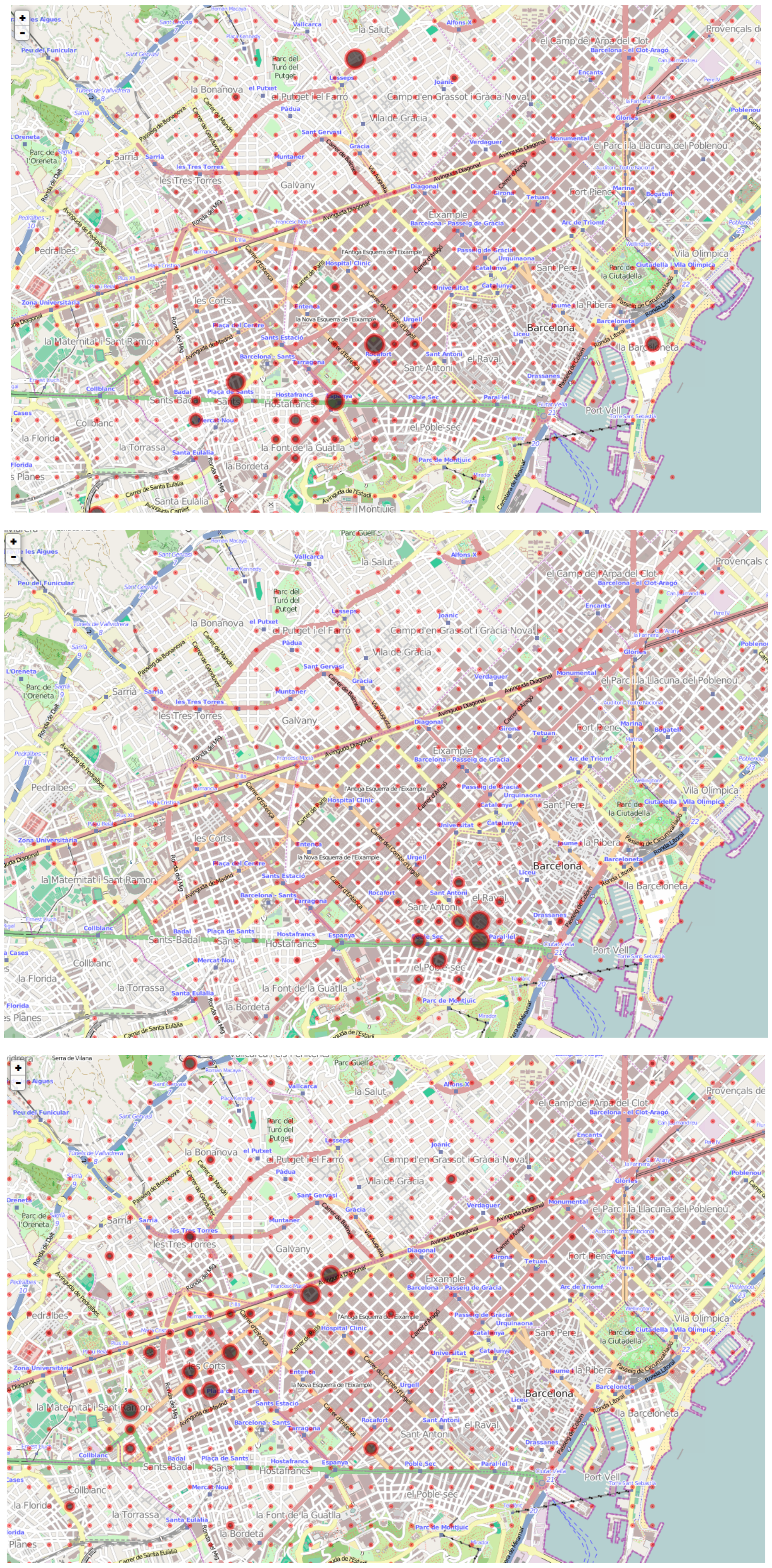


Figure 2.23: Examples of the geographical cluster obtained using K-means, they show different localized activity profiles.

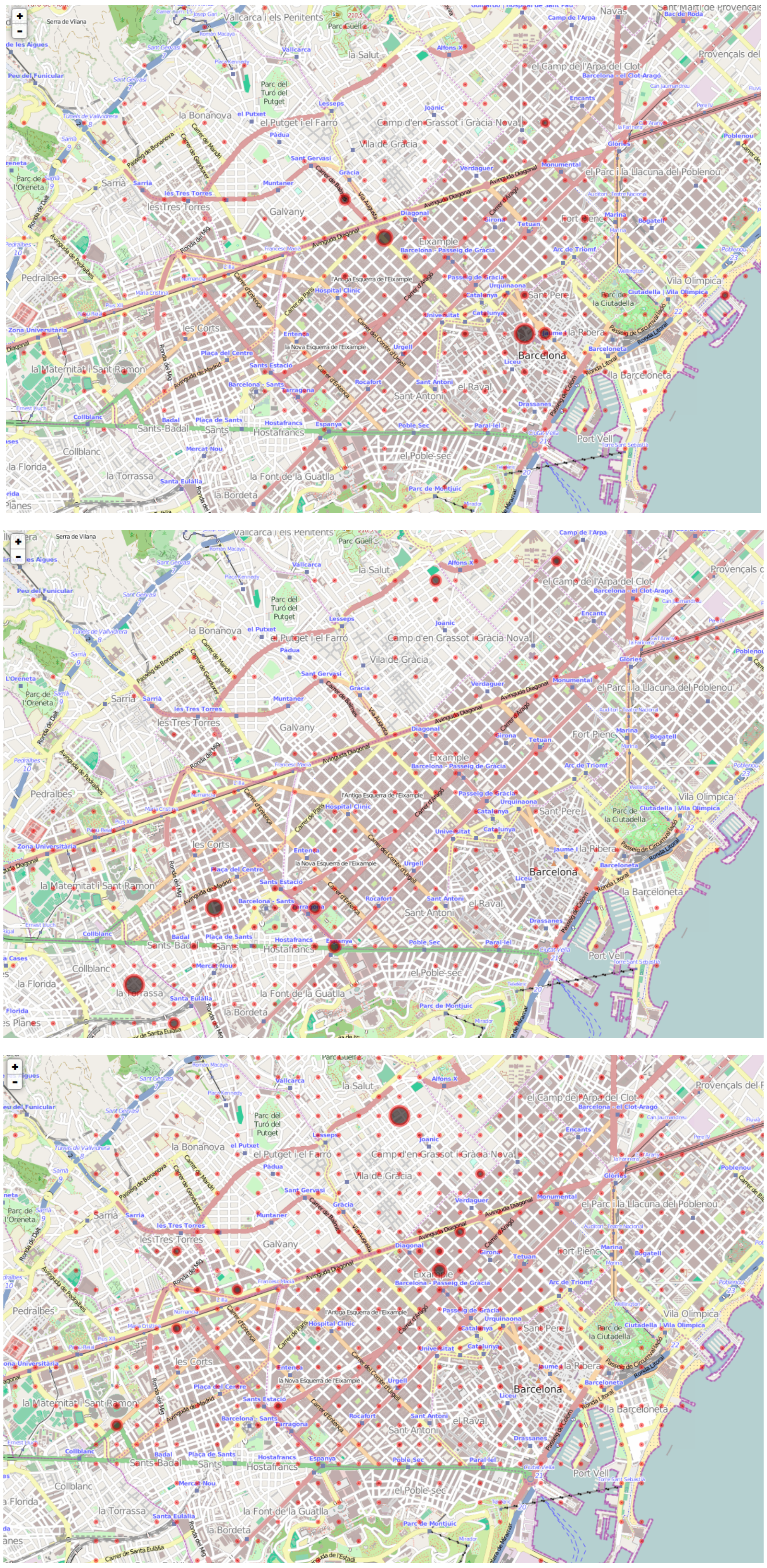


Figure 2.24: Examples of the geographical cluster obtained using affinity propagation, they show other more specific behavior profiles.

3 Clustering Validation

After processing a dataset using a unsupervised algorithm, the first question that arises is whether an actual structure has been found on the data or only a product of the bias of the algorithm. A disadvantage of the clustering task is that its evaluation is difficult. Supervised algorithms have available the actual labels so they have an answer they can compare their predictions. This is not the case for unsupervised algorithms, there is no predefined model to compare with, the true result is unknown and moreover, it may depend on the context or the actual the task to perform with the discovered model.

Obviously, assessing the quality of the patterns discovered is important, so we need measures able to quantify cluster quality. There will be not only one measure available for this task because of the lack of definition of the actual goal of cluster validation. We can measure how cluster-like are the patterns discovered, considering that the goal is to obtain well separated and compact clusters, but there is not a unique way to do this.

We can enumerate different applications for these measures. First of all, to avoid finding patterns in noise. This is something that will not be detected by most of the algorithms that we described in the previous chapter. For example, if we apply K-means to a dataset we will always find a partition. We can suspect that something is wrong if we run the algorithm several times and we obtain very different partitions, but the way the algorithm works, it will always result in a partition, even for random data. The same happens for hierarchical clustering, we can also suspect if cutting the dendrogram results in very strange partitions, but a dendrogram can always be computed.

The second reason for clustering validation measures is to compare different clustering algorithms or the results obtained by the tuning of the parameters of a single algorithm. Each algorithm has its own bias and their parameters can have a huge impact on the discovered patterns, so it is interesting to know if is there is a difference among the structure of the data that different choices are to capture. These measures can also useful if the labels for the data are available, so we can measure how well the known structure of the data is discovered by the chosen algorithm, but the measures used for supervised learning can be used in this case instead.

These measures can also be used for model selection (for example the number of clusters). All these measures capture heuristically some characteristic of the patterns that can be optimized. We can use this optimization as a means of finding the best parameters considering that the measure really captures the kind of patterns we are interested in.

3.1 Measures for assessing clusterness

Before clustering a dataset we can test if there are actually clusters to be found. We have to test the hypothesis of the existence of patterns in the data versus a dataset that is uniformly distributed (homogeneous distribution). This can be measured by the *Hopkins Statistic*. The procedure to compute this measures is as follows:

1. Sample n points (p_i) from the dataset (D) uniformly and compute the distance to their nearest neighbor ($d(p_i)$)

2. Generate n points (q_i) uniformly distributed in the space of the dataset and compute their distance to the nearest neighbors in D ($d(q_i)$)

3. Compute the quotient:

$$H = \frac{\sum_{i=1}^{n} d(p_i)}{\sum_{i=1}^{n} d(p_i) + \sum_{i=1}^{n} d(q_i)}$$

If the data is uniformly distributed, the value of H will be around 0.5.

3.2 Measures for clustering validation

We can divide the different criterion to evaluate the quality of a clustering in three groups:

- Methods that compare with a model partition/labeled data (External criteria)
- Quality measures based on the characteristics of the examples (Internal criteria)
- Measures to compare clusterings with each other(Relative criteria)

3.2.1 Internal validity criteria

These measures use the properties that are expected in a good clustering: compact and well separated groups. This is the bias that one way or another all clustering algorithm use. The indices compute a heuristic function that is optimized by a good clustering.

The computation of the criteria is based on the characteristics of the model of the groups and the statistical properties of the attributes of the data. These measures assume that we have continuous attributes and a distance defined for the data, so the statistical distribution of the values of the attributes and the distribution of the examples/clusters distances can be obtained.

Some of these indices correspond directly to the objective function optimized, for example:

- Quadratic error/Distorsion (k-means)

$$SSE = \sum_{k=1}^{k} \sum_{\forall x_i \in C_k} \parallel x_i - \mu_k \parallel^2$$

- Log likelihood (Mixture of gaussians/EM)

For prototype based algorithms several measures can be obtained using the statistical distribution of the values of the cluster prototypes. They are based on three quantities defined by the scatter matrices of the clusters: interclass distance (within distances), intraclass distance (between distances) and cross-intercluster distance.

$$\begin{aligned}
S_{W_k} &= \sum_{\forall x_i \in C_k} (x_i - \mu_k)(x_i - \mu_k)^T \\
S_{B_k} &= |C_k|(\mu_k - \mu)(\mu_k - \mu)^T \\
S_{M_{k,l}} &= \sum_{\forall i \in C_k} \sum_{\forall j \in C_l} (x_i - x_j)(x_i - x_j)^T
\end{aligned}$$

The simplest criteria use the trace of the interclass distance (S_W) and the intraclass distance (S_B) (trace criteria):

$$Tr(S_W) = \frac{1}{K} \sum_{i=1}^{K} S_{W_k}$$

$$Tr(S_B) = \frac{1}{K} \sum_{i=1}^{K} S_{B_k}$$

A cluster is better if it has a lower overall intracluster distance and a higher overall intercluster distance. This means compact and separated clusters.

From the cross-intercluster distance it is defined the determinant criteria

$$Det(S_M) = \frac{1}{K^2} \sum_{i=1}^{K} \sum_{j=1}^{K} S_{M_{i,j}}$$

A higher value means that the examples from each cluster are separated from the examples of the other clusters.

These criteria have their limitations, basically they fail when natural clusters are not well separated or have different densities. Also, it is not possible to compare clusters that have a different number of partitions, for example intraclass distance decreases monotonically with the number of partitions, and is zero (the lower the better) when there are as many partitions as examples.

More complex criteria that try to avoid these problems have been defined, these are the more frequently used:

- Kalinski Harabasz index (interclass/intraclass distance ratio)

$$CH = \frac{\sum_{i=0}^{K} |C_i| \times \|\mu_i - \mu\|^2 / (K-1)}{\sum_{k=1}^{K} \sum_{i=0}^{|C_i|} \|x_i - \mu_i\|^2 / (N-K)}$$

  This index computes a ratio between the interclass and intraclass distances and takes in account the number of clusters in the partition.

- Davies-Bouldin criteria (maximum interclass/cross-intraclass distance ratio)

$$\bar{R} = \frac{1}{K} \sum_{i=1}^{K} R_i$$

where

$$\begin{aligned} R_{ij} &= \frac{S_{W_i} + S_{W_j}}{S_{M_{ij}}} \\ R_i &= \max_{j:j \neq i} R_{ij} \end{aligned}$$

This index takes into account only the maximum ration of the separation among each pair of the clusters respect to their compactness.

- Silhouette index (maximum class spread/variance)

$$S = \frac{1}{N} \sum_{i=0}^{N} \frac{b_i - a_i}{max(a_i, b_i)}$$

Where

$$a_i = \frac{1}{|C_j| - 1} \sum_{y \in C_j, y \neq x_i} \|y - x_i\|$$

$$b_i = \min_{l \in H, l \neq j} \frac{1}{|C_l|} \sum_{y \in C_l} \|y - x_i\|$$

with $x_i \in C_j, H = \{h : 1 \leq h \leq K\}$

This index considers how the examples are scattered in the clusters, so the more compact the better.

In the literature it can be found more than 30 different indices and there is a consistent effort on studying their performance. A recent study ([3]) has exhaustively studied many of these indices, finding that some have a performance significantly better that others and some show a similar performance (not statistically different). The study concludes that Silhouette, Davies-Bouldin and Kalinski-Harabasz perform well in a wide range of situations.

Figure 3.1 shows the plot of the trace of the Within Scatter (SW) matrix, the Calinski-Harabasz (CH) and Davis-Bouldin (DB) indices for a dataset with 5 well separated clusters. Usually the criteria for deciding what clusters are better is to choose the optimal value (minimum or maximum depending on the direction of the criteria) or detecting a jump in the plot. Figure 3.2 shows the same indices for non well separated clusters, for the SW criteria now the change in the tendency of the values is not so clear as before, the CH criteria now has a jump in the correct number of clusters instead of a minimum but the DB criteria shows correctly the number of clusters.

3.2.2 External criteria

They measure if a clustering is similar to a model partition P. This mean that either we have the labels corresponding to the examples or we want to assess the difference among different ways of obtaining a partition of the data.

This second possibility is useful for model assessment, comparing the results of using different parameters of the same algorithm or completely different algorithms. For instance, it can be used to assess the sensitivity to initialization. These indices are independent of the attributes representing the data and how the cluster model is described, this means that it can be used to compare any clustering algorithm.

All the indices are based on the coincidence of each pair of examples in the groups of two clusterings. The computations are based on we have four values:

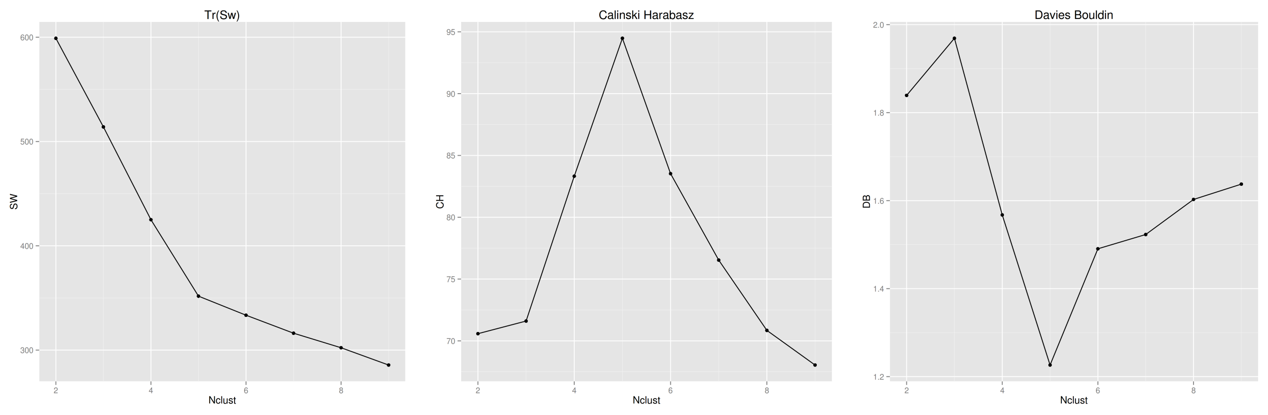


Figure 3.1: Trace of the Within Scatter matrix, Calinski-Harabasz and Davis-Bouldin indices for a dataset with 5 well separated clusters

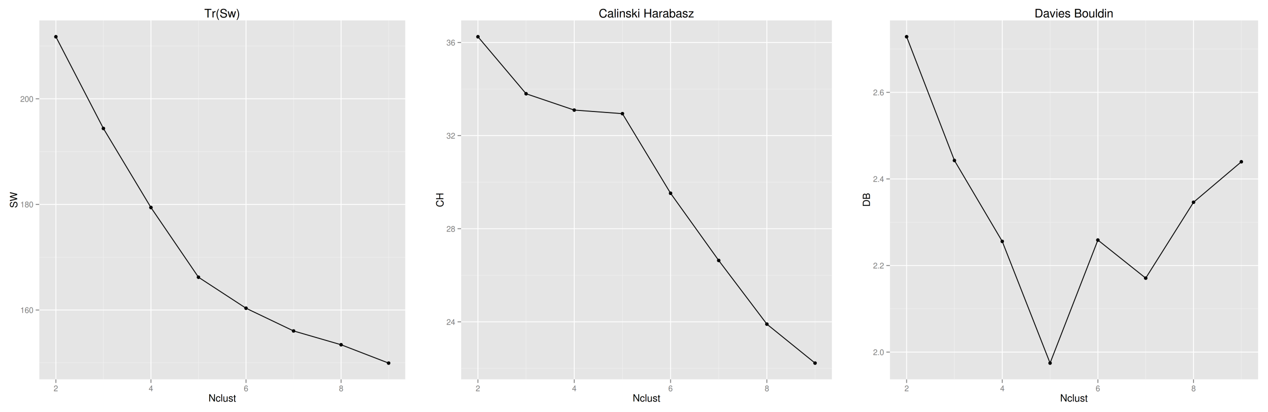


Figure 3.2: Trace of the Within Scatter matrix, Calinski-Harabasz and Davis-Bouldin indices for a dataset with 5 noisy non well separated clusters

- The two examples belong to the same class in both partitions (a)
- The two examples belong to the same class in C, but not in P (b)
- The two examples belong to the same class in P, but not in $C(c)$
- The two examples belong to different classes in both partitions (d)

From these quantities different measures can be defined, these are the most used:

- Rand/Adjusted Rand statistic:

$$R = \frac{(a+d)}{(a+b+c+d)}; \quad ARand = \frac{a - \frac{(a+c)(a+b)}{a+b+c+d}}{\frac{(a+c)+(a+b)}{2} - \frac{(a+b)(a+c)}{a+b+c+d}}$$

  The adjusted version takes in account that the agreement of two random partitions is not zero and scales the range of values of the index correspondingly.

- Jaccard Coefficient:

$$J = \frac{a}{(a+b+c)}$$

This index assumes that only are important the events when two examples agree on both partitions or completely disagree. Usually the events when two examples belong to different clusters in both partitions are larger than the rest overshadowing the contribution of the other possibilities.

- Folkes and Mallow index:

$$FM = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}$$

This index uses the geometric mean instead of the arithmetic.

An alternative way to measure the agreement of partitions is to use theoretical information measures. Assuming that the partition correspond to discrete statistical distributions, their similarity from this perspective can be assimilated to agreement. We can define the Mutual Information between two partitions as:

$$MI(Y_i, Y_k) = \sum_{X_c^i \in Y_i} \sum_{X_{c'}^k \in Y_k} \frac{|X_c^i \cap X_{c'}^k|}{N} \log_2 \left(\frac{N|X_c^i \cap X_{c'}^k|}{|X_c^i||X_{c'}^k|}\right)$$

and the Entropy of a partition as:

$$H(Y_i) = -\sum_{X_c^i \in Y_i} \frac{|X_c^i|}{N} \log_2 \left(\frac{|X_c^i|}{N}\right)$$

where $X_c^i \cap X_{c'}^k$ is the number of objects that are in the intersection of the two groups. From these quantities different measures can be defined as:

- Normalized Mutual Information:

$$NMI(Y_i, Y_k) = \frac{MI(Y_i, Y_k)}{\sqrt{H(Y_i)H(Y_k)}}$$

- Variation of Information:

$$VI(C, C') = H(C) + H(C') - 2I(C, C')$$

- Adjusted Mutual Information:

$$AMI(U, V) = \frac{MI(U, V) - E(MI(U, V))}{\max(H(U), H(V)) - E(MI(U, V))}$$

These measures allow comparing partitions without using the actual description of the partitions. An alternative when we have continuous attributes and a centroid based description of partitions is to use the distances among the centroids of different partitions and their variance as a measure of agreement. In this case this can be interpreted as measure of the correlation among the partitions. The following measures can be used for this purpose:

- Modified Hubert Γ statistic

$$\Gamma = \frac{2}{N \cdot N - 1} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} P(i, j) \cdot Q(i, j)$$

Where $P(i, j)$ is the distance matrix of the dataset and $Q(i, j)$ is a $N \times N$ matrix with the distance between the centroids the examples x_i and x_j belong to.

- Normalized Hubert Γ statistic

$$\Gamma = \frac{\frac{2}{N \cdot N-1}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(P(i,j)-\mu_P)\cdot(Q(i,j)-\mu_Q)}{\sigma_P\sigma_Q}$$

  A significant increase of this value indicates the number of clusters

- *SD* index

$$SD(n_c) = Dis(n_{max})\cdot Scat(n_c) + Dis(n_c)$$

  Where n_{max} is the maximum number of clusters, $\sigma(v_i)$ is the variance of a cluster, $\sigma(X)$ is the variance of the data set, D_{max} and D_{min} are the maximum and minimum cluster distances and

$$Scat(n_c) = \frac{\frac{1}{n_c}\sum_{i=1}^{n_c}\|\sigma(v_i)\|}{\|\sigma(X)\|}$$

$$Dis(n_c) = \frac{D_{max}}{D_{min}}\sum_{i=1}^{n_c}\left(\sum_{j=1}^{n_c}\|v_i - v_j\|\right)^{-1}$$

3.2.3 Assessing the number of clusters

A topic related to cluster validation is to decide if the number of clusters obtained is the correct one. This is part of model assessment/validation and is important specially for the algorithms that need this value as a parameter. The usual procedure is to compare the characteristics of clusterings of different number of clusters. Internal criteria indices are preferred in this comparison. A plot of the variation of these indices with the number of partitions can reveal how the quality of the clustering changes. The optimal value in the plot according to a specific criterion can reveal what number of clusters is more probable for the dataset.

Several of the internal validity indices explained in section 3.2.1 can be used for this purpose as shown in figures 3.1 and 3.2, for instance the Calinsky Harabasz index or the Silhouette index. The usual way to apply these indices is to look for a jump in the value of the criteria (commonly known as a *knee* or *elbow*) or to detect the number of clusters that yields the optimal value (minimum or maximum).

There are specialized criteria for this purpose that use the within class scatter matrix (S_W) such as:

- Hartigan index:

$$H(k) = \left[\frac{S_W(k)}{S_W(k+1)} - 1\right](n-k-1)$$

  This criterion is frequently used in hierarchical clustering to decide the level where to cut the dendrogram to obtain a flat partition.

- Krzanowski Lai index:

$$KL(k) = \left|\frac{DIFF(k)}{DIFF(k+1)}\right|$$

  with $DIFF(k) = (k-1)^{2/p}S_W(k-1) - k^{2/p}S_W(k)$

These are relative indices that compare the change in quality when the number of partitions is increased in one unit. The optimal number of clusters is decided also looking for a significant jump in the plot of its values. The effectiveness of these indices depends on how separable are the clusters and sometimes the plot presents different optimal number of clusters.

An alternative to assess the number of clusters in a dataset is to define it as a hypothesis test. One way is to compare the clustering with the expected distribution of data given the null hypothesis (no clusters). The procedure needs the generation of different clusterings of the data increasing the number of clusters and to compare each one to clusterings of datasets (B) generated with a uniform distribution.

A very successful test that uses this method is the Gap statistic, defined as

$$Gap(k) = (1/B)\sum_b log(S_W(k)_b) - log(S_W(k))$$

From the standard deviation (sd_k) of $\sum_b log(S_W(k)_b)$ is defined s_k as:

$$s_k = sd_k\sqrt{1+1/B}$$

The probable number of clusters is the smallest number that holds:

$$Gap(k) \geq Gap(k+1) - s_{k+1}$$

A third method is to use the concept of *cluster stability*. The idea behind this method is that if the model chosen for clustering a dataset is correct, it should be stable for different samplings of the data. The procedure is to obtain different subsamples of the data and cluster them and test how stable they are.

There are two methods:

- Using disjoint samples:

  The dataset is divided in two disjoint samples that are clustered separately. Several indices can be defined to assess stability, for instance, to the distribution of the number of neighbors that belong to the complementary sample. An adequate number of clusters should generate similar partitions so there should not be large deviation in the number of neighbors that belong to one or the other sample.

- Using non disjoint samples:

  The dataset is divided in three disjoint samples (S_1,S_2, S_3). Two clusterings are obtained from $S_1 \cup S_3$, $S_2 \cup S_3$. Different indices can be defined about the coincidence of the common examples in both partitions. The adequate number of clusters should have many common examples among the different samples.

3.2.4 Other methods

There are less systematic methods for assessing cluster validity or clusterness, but that can be used as a preliminary way of studying the dataset. The simplest way to gain some intuition about the characteristics of our data is to try to visualize it and see if there are some natural clusters.

Usually the dimensionality of the data will not allow directly observing the patterns in the data, but the different dimensionality reduction techniques explained in 1.3.1 can be used to project the dataset to 2 or 3 dimensions. Depending on our knowledge about the domain we can decide the method more adequate or we can begin with the simpler method (e.g.: PCA) and decide how to continue from there depending on the results. The hope is that the clusters that

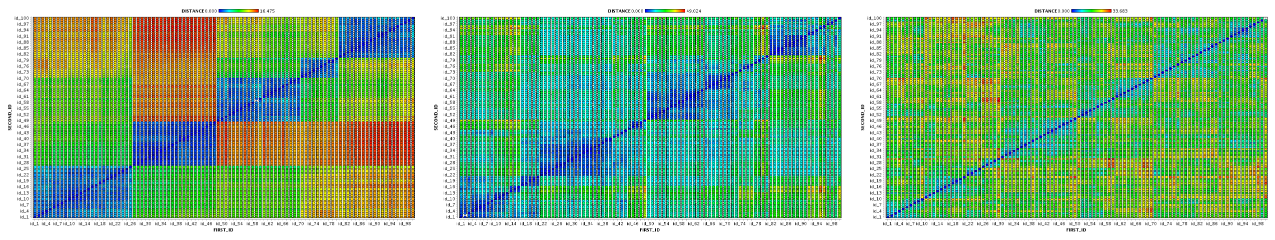


Figure 3.3: Distance matrix visualization. The first plot correspond to a dataset with 5 well separated clusters, the second one to 5 overlapping clusters and the third one to random generated data

can be observed in the new space could represent clusters in the original space. The confidence on this being the case depends on the reconstruction error of the transformed data and that the transformation maintains the relations in the original space.

Other approach is to visualize the distance matrix of the data. This matrix represents the examples relationships, using different distance functions will generate different relationsiphs so they have to be chosen carefully. The values of the matrix have to be rearranged so the closer examples appear in adjacent columns. The apparition of patterns in the rearranged matrix can be a sign of cluster tendency.

There are several methods for rearranging the distance matrix. The simplest one uses a hierarchical clustering algorithm and rearranges the matrix using an inorder traversal of the tree. Results will depend on the algorithm and the distance/similarity function used. The main advantage of this method is that it can be applied to quantitative and qualitative data, because only the distance are used and not the values of the attributes.

The main problem of these methods is that they are computationally expensive (at least quadratic on time and space). Also, to see patterns in the distance matrix is not always a guarantee of having actual clusters in the data. As an example, figure 3.3 shows the distance matrix of three datasets. The first one corresponds to five well separated clusters, it can be seen a clear structure in the distance matrix as five squares in the diagonal. The second one is for five overlapped clusters, the overlapping results is fuzzy defined square patterns in the diagonal that makes difficult to appreciate the actual number of clusters. The third one corresponds to a uniform distributed dataset, where no pattern appears in the rearranged distance matrix.

Figure 3.4 shows the distance matrix of three concentric rings dataset using the euclidean and the cosine distances. Being the data non linearly separable it is difficult to appreciate the actual patterns in the data just by matrix visualization and it can be usually misleading. The distance matrix using the euclidean distance shows a square pattern corresponding to the inner ring, but the other rings appear like random noise. Using the cosine distance shows a five cluster pattern in the data matrix. This means that one has to be careful when interpreting the results of these methods and has to try different alternatives before making a decision about how to proceed when processing a dataset.

3.3 Examples

3.3.1 Application: Wheelchair control

This example continues the application described in section 1.5.2. As a remainder, the domain is about the trajectories recorded from a wheelchair with shared control (patient/computer) for several patients solving different situations. The examples are described by the values obtained from sensors measuring angle/distance to the goal, angle/distance to the nearest obstacle from around the chair (210 degrees).

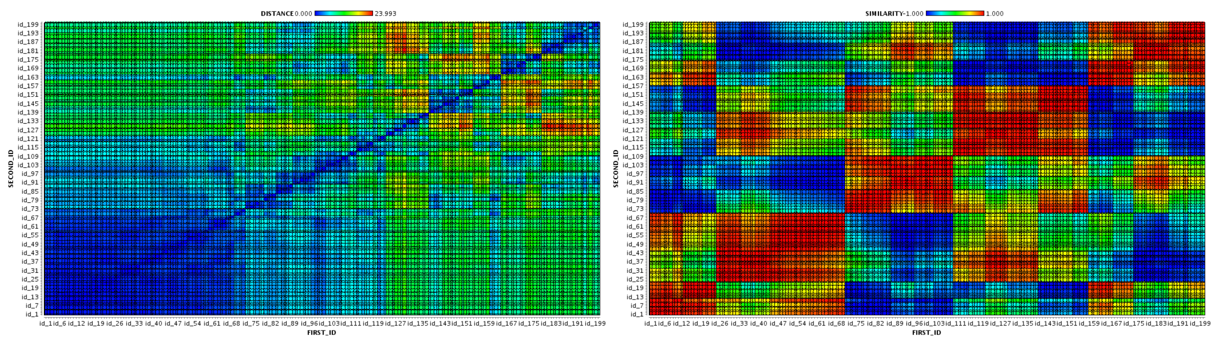


Figure 3.4: Distance matrix visualization for a three concentric rings dataset

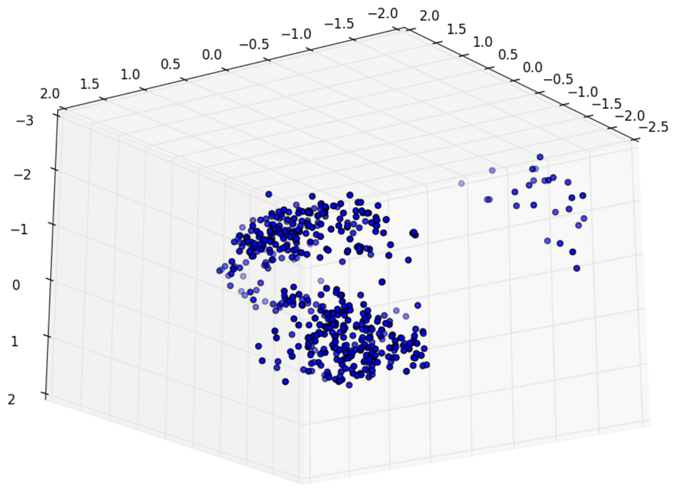


Figure 3.5: Visualization of the three principal components obtained from PCA

Now we are asking, given that the visualization of the data (see figure 3.5) has revealed some structure, how many clusters characterize the trajectories? The visualization reveals three clear densities using the tree first components of the PCA, but as commented in the previous section, visualization specially when reducing largely the number of dimensions can be misleading.

To confirm or refine the number of clusters, two different approaches are applied measuring the quality of clusters obtained using the K-means algorithm for different number of clusters. First using the silhouette index as internal validation criteria. Figure 3.6 (first plot) shows the variation of this criterion with the number of clusters. It presents a clear peak at tree clusters, confirming the intuition obtained from the visualization. The second approach used cluster stability as validation method. In this case several clusterings for each number of clusters have been generated and the adjusted rand and NMI indices have been used to compare the clusters. The idea is that if a number of clusters is adequate for the data, the partitions generated will be more similar among them. The second and third plot from figure 3.6 show the evolution of these indices. It can be seen that 2, 3 and 5 have the highest values for both indices. From the visualization of the data it is clear why two clusters has the higher value, the small cluster is very well separated from the rest of the data. Three clusters was expected also from the visualization. Five clusters is also a stability point that is worth exploring using other methods or that can be explained because the larger clusters are evidently not spherical and are consistently divided in half by K-means.

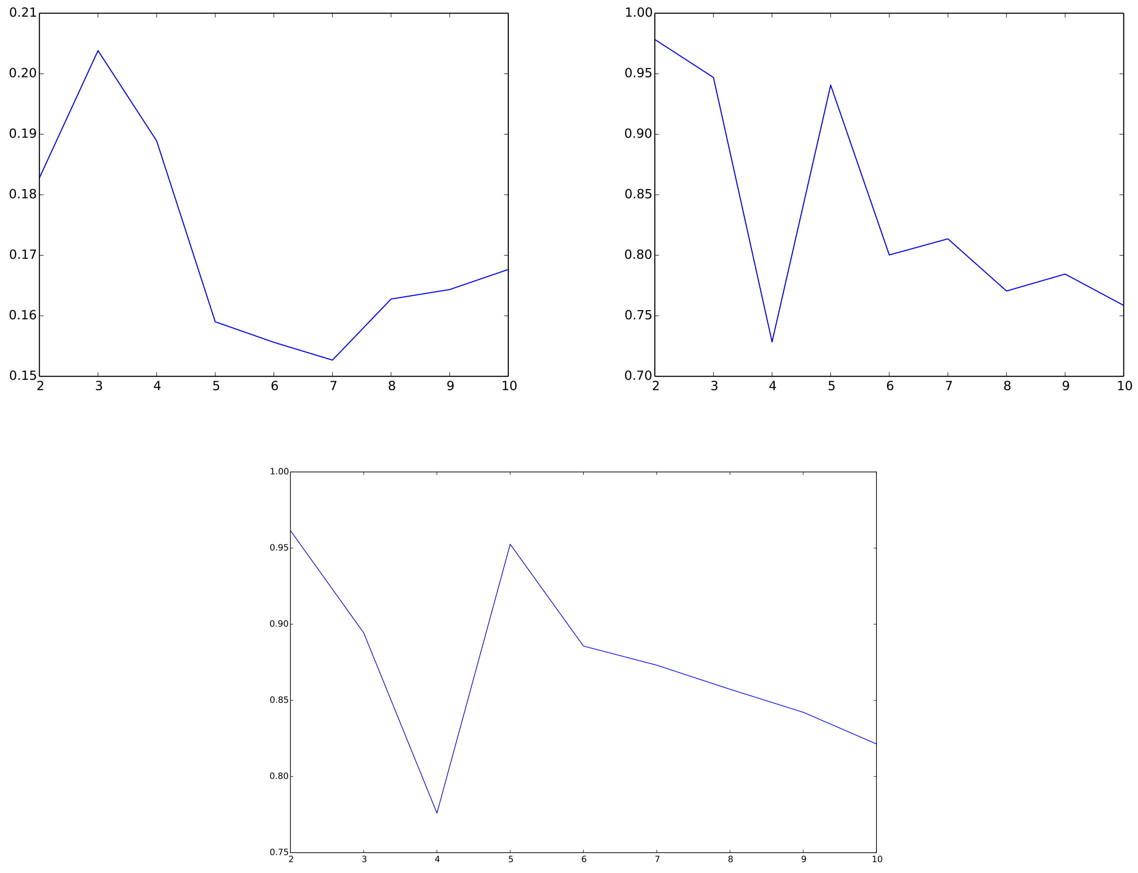


Figure 3.6: Plot of the silhouette, adjusted rand and normalized mutual information indices for different number of clusters for the wheelchair dataset

4 Clustering Large Datasets

According to a recent poll about the most frequent tasks and methods employed in data mining projects (KDNuggets, 2011), clustering was the third most frequent task. It is usual that these projects involve areas like astronomy, bioinformatics or finance, that generate large quantities of data. Also according to a recurrent poll of KDNuggets the most frequent size of the datasets being processed has shifted from tens of gigabytes in 2011 to terabytes in 2013. It is also common also that, in some of these domains, data is a continuous stream representing and boundless dataset, that is collected and processed in batches to incrementally update or refine a previously built model.

The classical methods for clustering (e.g.: K-means, hierarchical clustering) are not able to cope with this increasing amount of data. The reason is mainly because either the constraint of maintaining all the data in main memory or the temporal complexity of the algorithms. This makes them impractical for the purpose of processing these increasingly larger datasets. This means that the need of scalable clustering methods is a real problem and in consequence some new approaches are being developed.

There are several methodologies that have been used to scale clustering algorithms, some inspired in methodologies successfully used for supervised machine learning, other specific for this unsupervised task. For instance, some of these techniques use different kinds of sampling strategies, in order to store in memory only a subset of the data. Others are based on the partition of the whole dataset in several independent batches for separate processing and the merging of the result in a consensuated model. Some methodologies assume that the data is a continuous stream and has to be processed on-line or in successive batches. Also, these techniques are integrated in different ways depending on the model that is used for the clustering process (prototype based, density based, ...). This large variety of approaches makes necessary to define their characteristics and to organize them in a coherent way.

4.1 Scalability strategies

The strategies used to scale clustering algorithms range from general strategies that can be adapted to any algorithm to specific strategies that exploit the characteristics of the algorithm in order to reduce its computational cost.

Some of the strategies are also dependent on the type of data that is used. For instance, only clustering algorithms that incrementally build the partition can be used for data streams. For this kind of datasets it means that the scaling strategy has to assume that the data will be processed continuously and only one pass through the data will be allowed. For applications where the whole dataset can be stored in secondary memory, other possibilities are also available.

The different strategies applied for scalability are not disjoint and several strategies can be used in combination. These strategies can be classified in:

One-pass strategies: The constraint assumed is that the data only can be processed once and in a sequential fashion. A new example is integrated in the model each iteration. Depending on the type of the algorithm a data structure can be used to efficiently determine how to perform this update. This strategy does not only apply to data streams and can be actually used for any dataset.

Summarization strategies: It is assumed that all the examples in the dataset are not needed for obtaining the clustering, so an initial preprocess of the data can be used to reduce its size by combining examples. The preprocess results in a set of representatives of groups of examples that fits in memory. The representatives are then processed by the clustering algorithm.

Sampling/batch strategies: It is assumed that processing samples of the dataset that fit in memory allows to obtain an approximation of the partition of the whole dataset. The clustering algorithm generates different partitions that are combined iterativelly to obtain the final partition.

Approximation strategies: It is assumed that certain computations of the clustering algorithm can be approximated or reduced. These computations are mainly related with the distances among examples or among the examples and the cluster prototypes.

Divide and conquer strategies: It is assumed that the whole dataset can be partitioned in roughly independent datasets and that the combination/union of the results for each dataset approximates the true partition.

4.1.1 One-pass strategies

The idea of this strategy is to reduce the number of scans of the data to only one. This constraint may be usually forced by the circumstance that the dataset can not fit in memory and it has to be obtained from disk. Also the constraint could be imposed by a continuous process that does not allow to store all the data before processing it.

Sometimes this strategy is used to perform a preprocess of the dataset. This results in two stages algorithms, a first one that applies the one-pass strategy and a second one that process in memory a summary of the data obtained by the first stage.

The assumption of the first stage is that a simple algorithm can be used to obtain a coarse representation of the clusters in the data and that these information will be enough to partition the whole dataset.

Commonly this strategy is implemented using the leader algorithm. This algorithm does not provide very good clusters, but can be used to estimate densities or approximate prototypes, reducing the computational cost of the second stage.

4.1.2 Summarization Strategies

The purpose of this strategy is to obtain a coarse approximation of the data without losing the information that represent the different densities of examples. This summarization strategy assumes that there is a set of sufficient quantities that can be computed from the data, capable of representing their characteristics. For instance, by using sufficient statistics like mean and variance.

The summarization can be performed single level, as a preprocess that is feed to a cluster algorithm able to process summaries instead of raw examples, or also can be performed in a hierarchical fashion. This hierarchical scheme can reduce the computational complexity by using a multi level clustering algorithm or can be used as an element of a fast indexing structure that reduces the cost of obtaining the first level summarization.

4.1.3 Sampling/batch strategies

The purpose of sampling and batch strategies is to allow to perform the processing in main memory for a part of the dataset.

Sampling assumes that only a random subset or subsets of the data are necessary to obtain the model for the data and that the complete dataset is available from the beginning. The random subsets can be or not disjoint. If more than one sample of the data is processed, the successive samples are integrated with the current model. This is usually done using an algorithm able to process raw data and cluster summaries. The algorithms that use this strategy do not process all the data, so they scale on the size of the sampling and not on the size of the whole dataset.

The use of batches assume that the data can be processed sequentially and that after applying a clustering algorithm to a batch, the result can be merged with the results from previous batches. This processing assumes that data is available sequentially as in a data stream and that the batch is complete after observing an amount of data that fits in memory.

4.1.4 Approximation strategies

These strategies assume that some computations can be saved or approximated with reduced or null impact on the final result. The actual approximation strategy is algorithm dependent, but usually the most costly part of clustering algorithms corresponds to distance computation among instances or among instances and prototypes. This circumstance focus these strategies particularly on hierarchical, prototype based and some density based algorithms, because they use distances to decide how to assign examples to partitions.

For example, some of these algorithms are iterative and the decision about what partition is assigned to an example does not change after a few iterations. If this can be determined at an early stage, all these distance computations can be avoided in successive iterations.

This strategy is usually combined with a summarization strategy where groups of examples are reduced to a point that is used to decide if the decision can be performed using only that point or the distances to all the examples have to be computed.

4.1.5 Divide and conquer strategies

This is a general strategy applied in multiple domains. The principle is that data can be divided in multiple independent datasets and that the clustering results can be then merged on a final model. This strategy rely sometimes on a hierarchical scheme to reduce the computational cost of merging all the independent models. Some strategies assume that each independent clustering represent a view of the model, being the merge a consensus of partitions. The approach can also result on almost independent models that have to be joined, in this case the problem to solve is how to merge the parts of the models that represent the same clusters.

4.2 Algorithms

All these scalability strategies have been implemented in several algorithms that represent the full range of different approaches to clustering. Usually more than one strategy is combined in an algorithm to take advantage of the cost reduction and scalability properties. In this section, a review of a representative set of algorithms and the use of these strategies is presented.

4.2.1 Scalable hierarchical clustering

The main drawback of hierarchical clustering is its high computational cost (time $O(n^2)$, space $O(n^2)$) that makes it impractical for large datasets. The proposal in [38] divides the clustering process in two steps. First a one pass clustering algorithm is applied to the dataset, resulting in a set of cluster summaries that reduce the size of the dataset. This new dataset fits in memory and can be processed using a single link hierarchical clustering algorithm.

For the one-pass clustering step, the leader algorithm is used. This algorithm has as parameter (d), the maximum distance between example and cluster prototype. The processing of each example follows the rule, if the nearest existing prototype is closer than d, it is included in that cluster and its prototype recomputed, otherwise, a new cluster with the example is created. The value of the parameter is assumed to be known or can be estimated from a sample of the dataset. The time complexity of this algorithm is $O(nk)$ being k the number of clusters obtained using the parameter d.

The first phase of the proposed methodology applies the leader algorithm to the dataset using as a parameter half the estimated distance between clusters (h). For the second stage, the centers of the obtained clusters are merged using the single-link algorithm until the distance among clusters is larger than h.

The clustering obtained this way is not identical to the resulting from the application of the single-link algorithm to the entire dataset. To obtain the same partition, an additional process is performed. During the merging process, the clusters that have pairs of examples at a distance less than h are also merged. For doing this, only the examples of the clusters that are at a distance less than $2h$ have to be examined. The overall complexity of all three phases is $O(nk)$, that corresponds to the complexity of the first step. The single-link is applied only to the cluster obtained by the first phase, reducing its time complexity to $O(k^2)$, being thus dominated by the time of the leader algorithm.

4.2.2 Rough-DBSCAN

In [43] a two steps algorithm is presented. The first step applies a one pass strategy using the leader algorithm, just like the algorithm in the previous section. The application of this algorithm results in an approximation of the different densities of the dataset. This densities are used in the second step, that consists in a variation of the density based algorithm DBSCAN.

This method uses a theoretical result that bounds the maximum number of leaders obtained by the leader algorithm. Given a radius τ and a closed and bounded region of space determined by the values of the features of the dataset, the maximum number of leaders k is bounded by:

$$k \leq \frac{V_S}{V_{\tau/2}}$$

being V_S the volume of the region S and $V_{\tau/2}$ the volume of a sphere of radius $\tau/2$. This number is independent of the number of examples in the dataset and the data distribution.

For the first step, given a radius τ, the result of the leader algorithm is a list of leaders ($\mathcal{L}$), their followers and the count of their followers. The second step applies the DBSCAN algorithm to the set of leaders given an ϵ and a $MinPts$ parameters.

The count of followers is used to estimate the count of examples around a leader. Different estimations can be derived from this count. First it is defined $\mathcal{L}_l$ as the set of leaders at a distance less or equal than ϵ to the leader l:

$$\mathcal{L}_l = \{l_j \in \mathcal{L} \mid \|l_j - l\| \leq \epsilon\}$$

The measure $roughcard(N_\epsilon(l, \mathcal{D}))$ is defined as:

$$roughcard(N_\epsilon(l, \mathcal{D})) = \sum_{l_i \in \mathcal{L}_l} count(l_i)$$

approximating the number of examples less than a distance ϵ to a leader. Alternate counts can be derived as upper and lower bounds of this count using $\epsilon + \tau$ (upper) or $\epsilon - \tau$ (lower) as distance.

From this counts it can be determined if a leader is dense or not. Dense leaders are substituted by their followers, non dense leaders are discarded as outliers. The final result of the algorithm is the partition of the dataset according to the partition of the leaders.

The computational complexity of this algorithm is for the first step $O(nk)$, being k the number of leaders, that does not depend on the number of examples n, but on the radius τ and the volume of the region that contains the examples. For the second step, the complexity of the DBSCAN algorithm is $O(k^2)$, given that the number of leaders will be small for large datasets, the cost is dominated by the cost of the first step.

4.2.3 CURE

CURE [25] is a hierarchical agglomerative clustering algorithm. The main difference with the classical hierarchical algorithms is that it uses a set of examples to represent the clusters, allowing for non spherical clusters to be represented. It also uses a parameter that shrinks the representatives towards the mean of the cluster, reducing the effect of outliers and smoothing the shape of the clusters. Its computational cost is $O(n^2 \log(n))$

The strategy used by this algorithm to attain scalability combines a divide an conquer and a sampling strategy. The dataset is first reduced by using only a sample of the data. Chernoff bounds are used to compute the minimum size of the sample so it represents all clusters and approximates adequately their shapes.

In the case that the minimum size of the sample does not fit in memory a divide and conquer strategy is used. The sample is divided in a set of disjoint batches of the same size and clustered until a certain number of clusters is achieved or the distance among clusters is less than an specified parameter. This step has the effect of a pre-clustering of the data. The clusters representatives from each batch are merged and the same algorithm is applied until the desired number of clusters is achieved. A representation of this strategy appears in figure 4.1. Once the clusters are obtained all the dataset is labeled according to the nearest cluster. The complexity of the algorithm is $O(\frac{n^2}{p} \log(\frac{n}{p}))$, being n the size of the sample and p the number of batches used.

4.2.4 BIRCH

BIRCH [50] is a multi stage clustering algorithm that bases its scalability in a first stage that incrementally builds a pre-clustering of the dataset. The first stage combines a one pass strategy

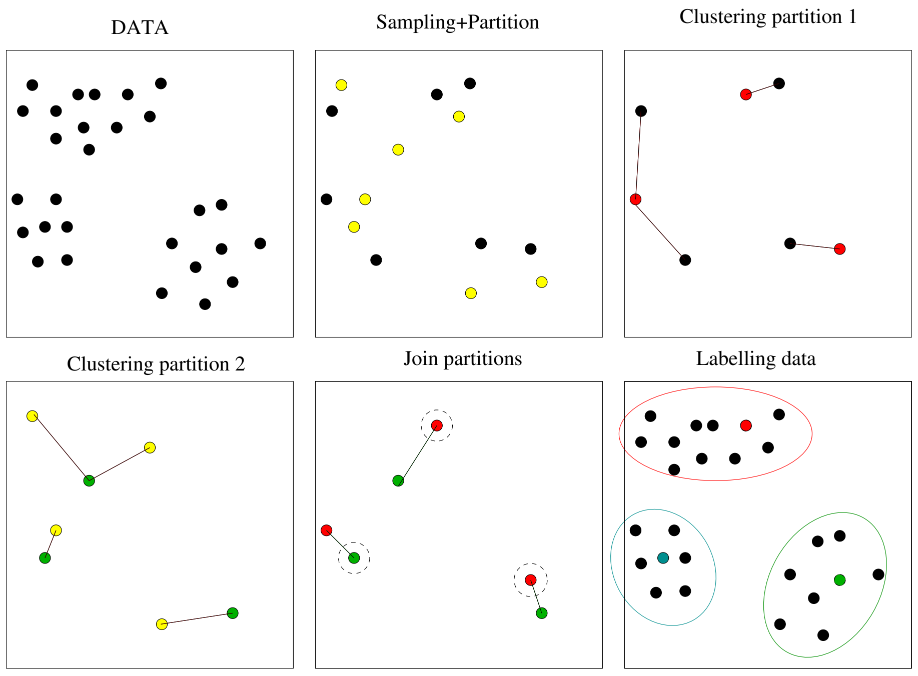


Figure 4.1: CURE

and a summarization strategy that reduces the actual size of the dataset to a size that fits in memory.

The scalability strategy relies on a data structure named Clustering Feature tree (CF-tree) that stores information that summarizes the characteristics of a cluster. Specifically, the information in a node is the number of examples, the sum of the examples values and the sum of their square values. From these values other quantities about the individual clusters can be computed, for instance, the centroid, the radius of the cluster, its diameter and quantities relative to pairs of clusters, as the inter-cluster distance or the variance increase.

A CF-tree (figure 4.2) is a balanced n-ary tree that contains information that represents probabilistic prototypes. Leaves of the tree can contain as much as l prototypes and their radius can not be more than t. Each non terminal node has a fixed branching factor (b), each element is a prototype that summarizes its subtree. The choice of these parameters is crucial, because it determines the actual available space for the first phase. In the case of selecting wrong parameters, the CF-tree can be dynamically compressed by changing the parameters values (basically t). In fact, t determines the granularity of the final groups.

The first phase of BIRCH inserts sequentially the examples in the CF-tree to obtain a set of clusters that summarizes the data. For each instance, the tree is traversed following the branch of the nearest prototype of each level, until a leave is reached. Once there, the nearest prototype from the leave to the example is determined. The example could be introduced in this prototype or a new prototype could be created, depending whether the distance is greater or not than the value of the parameter t. If the current leave has not space for the new prototype (already contains l prototypes), the algorithm proceeds to create a new terminal node and to distribute the prototypes among the current node and the new leaf. The distribution is performed choosing the two most different prototypes and dividing the rest using their proximity to these two prototypes. This division will create a new node in the ascendant node. If the new node exceeds the capacity of the father, it will be split and the process will continue upwards until the root of the tree is reached if necessary. Additional merge operations after completing this process could be performed to compact the tree.

For the next phase, the resulting prototypes from the leaves of the CF tree represent a coarse vision of the dataset. These prototypes are used as the input of a clustering algorithm. In the

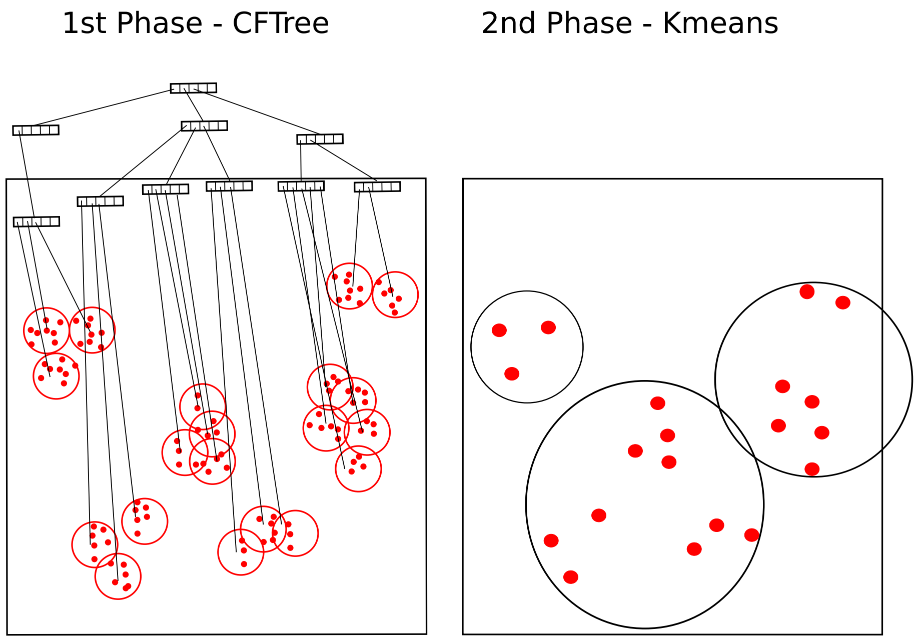


Figure 4.2: BIRCH algorithm

original algorithm, single link hierarchical clustering is applied, but also K-means clustering could be used. The last phase involves labeling the whole dataset using the centroids obtained by this clustering algorithm. Additional scans of the data can be performed to refine the clusters and detect outliers.

The actual computational cost of the first phase of the algorithm depends on the chosen parameters. Chosen a threshold t, considering that s is the maximum number of leaves that the CF-tree can contain, also that the height of the tree is $\log_b(s)$ and that at each level b nodes have to be considered, the temporal cost is $O(nb \log_b(s))$. The temporal cost of clustering the leaves of the tree depends on the algorithm used, for hierarchical clustering it is $O(s^2)$. Labeling the dataset has a cost $O(nk)$, being k the number of clusters.

4.2.5 OptiGrid

OptiGrid [29] presents an algorithm that divides the space of examples in an adaptive multi-dimensional grid that determines dense regions. The scalability strategy is based on recursive divide and conquer. The computation of one level of the grid determines how to divide the space on independent datasets. These partitions can be divided further until no more partitions are possible.

The main element of the algorithm is the computation of a set of low dimensional projections of the data that are used to determine the dense areas of examples. These projections can be computed using PCA or other dimensionality reduction algorithms and can be fixed for all the iterations. For a projection, a fixed number of orthogonal cutting planes are determined from the maxima and minima of the density function computed using kernel density estimation or other density estimation method. These cutting planes are used to compute a grid. The dense cells of the grid are considered clusters at the current level and are recursively partitioned until no new cutting planes can be determined given a quality threshold. A detailed implementation is presented in algorithm 4.1

For the computational complexity of this method. If the projections are fixed for all the computations, the first step can be obtained separately of the algorithm and is added to the total cost. The actual cost of computing the projections depends on the method used. Assuming axis parallel projections the cost for obtaining k projections for N examples is $O(Nk)$, $O(Ndk)$

Algorithm 4.1 OptiGrid algorithm

```
Given: number of projections k, number of cutting planes q, min cutting quality
        min_c_q, data set X
Compute a set of projections $P = \{P_1, ..., P_k\}$
Project the dataset X wrt the projections $\{P_1(X), ..., P_k(X)\}$
BestCuts ← ∅, Cut ← ∅
for $i \in 1..k$ do
    Cut ← ComputeCuts($P_i(X)$)
    for c in Cut do
        if CutScore(c)> min_c_q then BestCuts.append(c)
    end
end
if BestCuts.isEmpty() then return X as a cluster
BestCuts ← KeepQBestCuts(BestCuts,q)
Build the grid for the q cutting planes
Assign the examples in X to the cells of the grid
Determine the dense cells of the grid and add them to the set of clusters $C$
foreach cluster $cl \in C$ do
    apply OptiGrid to cl
end
```

otherwise, being d the number of dimensions. Computing the cutting planes for k projections can be obtained also in $O(Nk)$. Assigning the examples to the grid depends on the size of the grid and the insertion time for the data structure used to store the grid. For q cutting planes and assuming a logarithmic insertion time structure, the cost of assigning the examples has a cost of $O(Nq\min(q, \log(N)))$ considering axis parallel projections and $O(Nqd\min(q, \log(N)))$ otherwise. The number of recursions of the algorithm is bound by the number of clusters in the dataset that is a constant. Considering that q is also a constant, this gives a total complexity that is bounded by $O(Nd\log(N))$

4.2.6 Scalable K-means

This early algorithm for clustering scalability presented in [8] combines a sampling strategy and a summarization strategy. The main purpose of this algorithm is to provide an on-line and anytime version of the K-means algorithm that works with a pre-specified amount of memory.

The algorithm repeats the following cycle until convergence:

1. Obtain a sample of the dataset that fits in the available memory
2. Update the current model using K-means
3. Classify the examples as:
    (a) Examples needed to compute the model
    (b) Examples that can be discarded
    (c) Examples that can be compressed using a set of sufficient statistics as fine grain prototypes

The discarding and compressing of part of the new examples allows to reduce the amount of data needed to maintain the model each iteration.

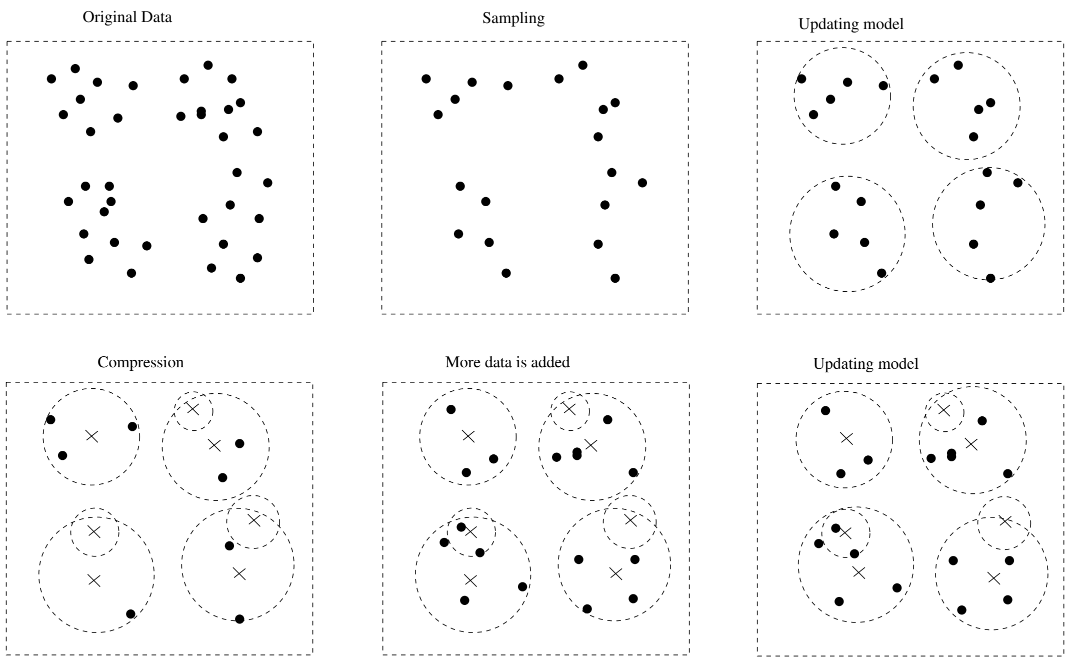


Figure 4.3: Scalable K-means

The algorithm divides the compression of data in two differentiated strategies. The first one is called primary compression, that aims to detect those examples that can be discarded. Two criteria are used for this compression, the first one determines those examples that are closer to the cluster centroid than a threshold. These examples are not probably going to change their assignment in the future. The second one consist in perturbing the centroid around a confidence interval of its values. If an example does not change its current cluster assignment, it is considered that future modifications of the centroid will still include the example.

The second strategy is called secondary compression, that aims to detect those examples that can not be discarded but form a compact subcluster. In this case, all the examples that form these compact subclusters are summarized using a set of sufficient statistics. The values used to compute that sufficient statistics are the same used by BIRCH to summarize the dataset.

The algorithm used for updating the model is a variation of the K-means algorithm that is able to treat single instances and also summaries. The temporal cost of the algorithm depends on the number of iterations needed until convergence as in the original K-means, so the computational complexity is $O(kni)$, being k the number of clusters, n the size of the sample in memory, and i the total number of iterations performed by all the updates.

4.2.7 STREAM LSEARCH

The STREAM LSEARCH algorithm [24] assumes that data arrives as a stream, so holds the property of only examining the data once. The algorithm process the data in batches obtaining a clustering for each batch and merging the clusters when there is not space to store them. This merging is performed in a hierarchical fashion. The strategy of the algorithm is then a combination of one-pass strategy plus batch and summarization strategies.

The basis of the whole clustering scheme is a clustering algorithm that solves the facility location (FL) problem. This algorithm reduces a sequential batch of the data to at most $2k$ clusters, that summarize the data. These clusters are used as the input for the hierarchical merging process. The computational cost of the whole algorithm relies on the cost of this clustering algorithm. This algorithm finds a set of between k and $2k$ clusters that optimizes the FL problem using a binary search strategy. An initial randomized procedure computes

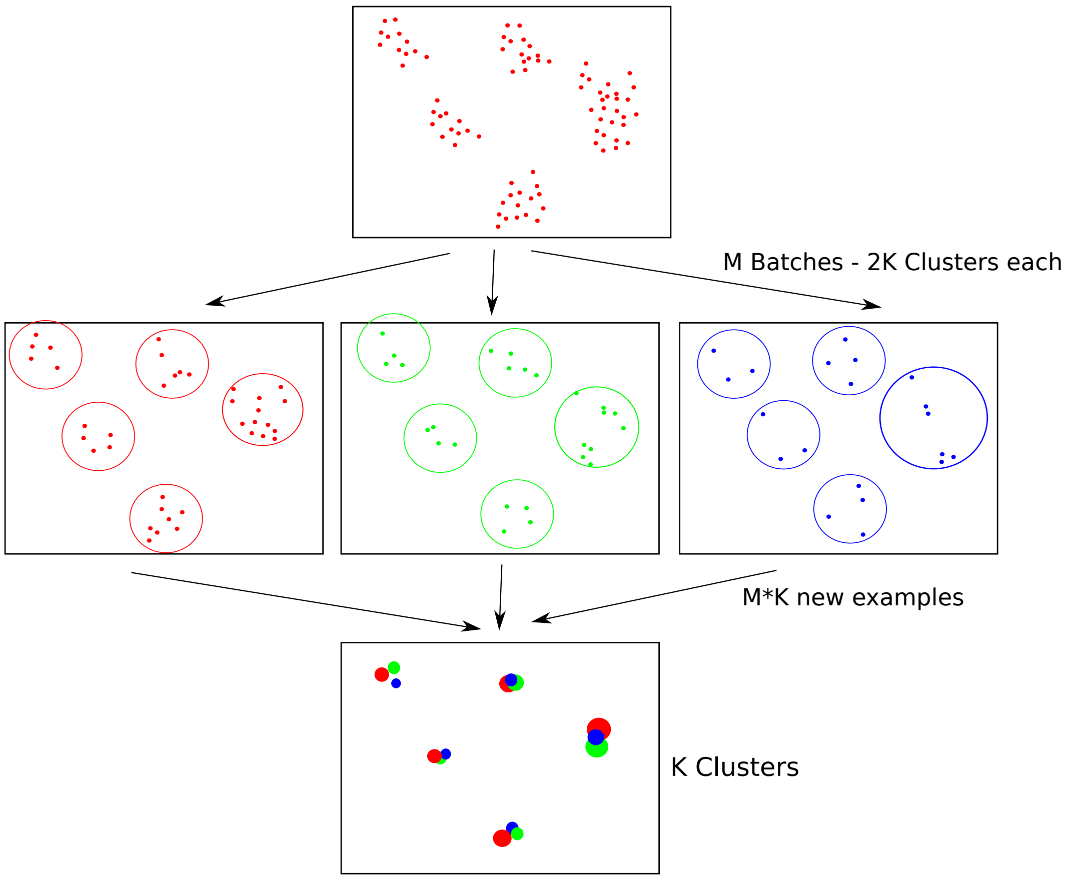


Figure 4.4: STREAM LSEARCH

the clusters used as initial solution. The cost of this algorithm is $O(nm + nk\log(k))$ being m the number of clusters of the initial solution, n the number of examples and k the number of clusters.

The full algorithm can be outlined as:

1. Input the first m points; use the base clustering algorithm to reduce these to at most $2k$ cluster centroids. The number of examples at each cluster will act as the weight of the cluster.

2. Repeat the above procedure until $\frac{m^2}{2k}$ examples have been processed so we have m centroids

3. Reduce them to $2k$ second level centroids

4. Apply the same criteria for each existing level so after having m centroids at level i then $2k$ centroid at level $i + 1$ are computed

5. After seen all the sequence (or at any time) reduce the $2k$ centroids at top level to k centroids

The number of centroids to cluster is reduced geometrically with the number of levels, so the main cost of the algorithm relies on the first level. This makes the time complexity of the algorithm $O(nk\log(nk))$, while needing only $O(m)$ space.

4.2.8 Mini batch K-means

Mini Batch K-means [41] uses a sampling strategy to reduce the space and time that K-means algorithm needs. The idea is to use small bootstrapped samples of the dataset of a fixed size

Algorithm 4.2 Mini Batch K-Means algorithm

```
Given: k, mini-batch size b, iterations t, data set X
Initialize each c ∈ C with an x picked randomly from X
v ← 0
for $i \leftarrow 1$ to $t$ do
    M ← b examples picked randomly from X
    for $x \in M$ do
        d[x] ← f(C,x)
    end
    for $x \in M$ do
        c ← d[x]
        v[c] ← v[c] + 1
        $\eta \leftarrow \frac{1}{v[c]}$
        c ← (1-$\eta$)c+$\eta$x
    end
end
```

that can be fit in memory. Each iteration, the sample is used to update the clusters. This procedure is repeated until the convergence of the clusters is detected or a specific number of iterations is reached.

Each mini batch of data updates the cluster prototypes using a convex combination of the attribute values of the prototypes and the examples. A learning rate that decreases each iteration is applied for the combination. This learning rate is the inverse of number of examples that have been assigned to a cluster during the process. The effect of new examples is reduced each iteration, so convergence can be detected when no changes in the clusters occur during several consecutive iterations. A detailed implementation is presented in algorithm 4.2.

The mayor drawback of this algorithm is that the quality of the clustering depends on the size of the batches. For very large datasets, the actual size for a batch that can be fit in memory can be very small compared with the total size of the dataset. The mayor advantage is the simplicity of the approach. This same strategy is also used for scaling up other algorithms as for example backpropagation in artificial neural networks.

The complexity of the algorithm depends on the number of iterations needed for convergence (i), the size of the samples (n), and the number of clusters (k) so it is bound by $O(kni)$.

4.2.9 Canopy clustering

Canopy clustering [37] uses a divide and conquer and an approximation strategies to reduce the computational cost. It also uses a two phases clustering algorithm, that is implemented using the well known mapreduce paradigm for concurrent programming.

The first stage divides the whole dataset in a set of overlapping batches called *canopies*. The computation of these batches depends on a cheap approximate distance that determines the neighborhood of a central point given two distance thresholds. The smaller distance (T_2) determines the examples that will belong exclusively to a canopy. The larger distance (T_1) determines the examples that can be shared with other canopies. The values of these two distance thresholds can be manually determined or computed using crossvalidation.

To actually reduce the computational cost of distance computation the distance function used in this first phase should be cheap to compute. The idea is to obtain an approximation of the densities in dataset. The specific distance depends on the characteristics of the attributes

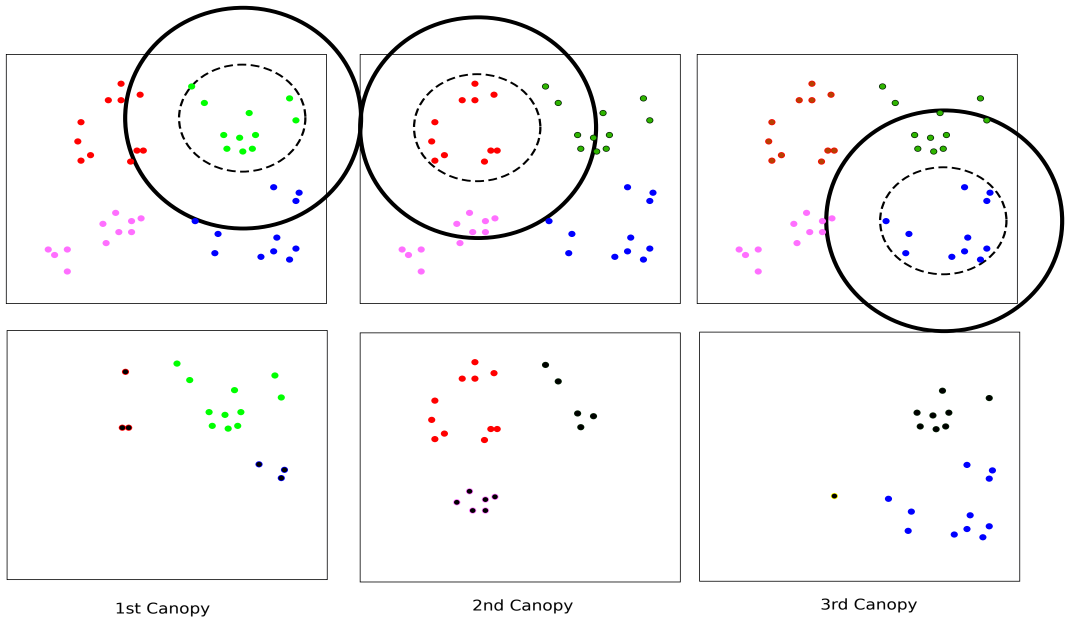


Figure 4.5: Canopy clustering

in the dataset, but it is usually simple to obtain such a function by value discretization or using locality sensitive hashing.

The computation of the canopies proceeds as follows: One example is randomly picked as the center of a canopy from the dataset, all the examples that are at a distance less than T_2 are assigned to this canopy and can not be used as centers in the future iterations. All the examples that are at a distance less than T_1 are included in the canopy but can be used as centers in the future. The process is repeated until all the examples have been assigned to a canopy. In figure 4.5 can be seen a representation of this process.

The second stage of the algorithm consist in clustering all the canopies separately. For this process, different algorithms can be used, for example agglomerative clustering, expectation maximization (EM) for gaussian mixtures or K-means. Also different strategies can be used for applying these algorithms. For example, for K-means or EM the number of prototypes for a canopy can be fixed at the beginning, using only the examples inside the canopy to compute them, saving this way many distance computations. Other alternative is to decide the number of prototypes globally, so they can move among canopies and be computed not only using the examples inside a canopy, but also using the means of the nearest canopies.

These different alternatives make difficult to give a unique computational complexity for all the process. For the first stage, the data has to be divided in canopies, this computational cost depends on the parameters used. The method used for obtaining the canopies is similar to the one used by the leader algorithm, this means that equivalently as was shown in 4.2.2, the number of partitions obtained does not depend on the total number of examples (n), but on the volume defined by the attributes and the value of parameter T_2. Being k this number of canopies, the computational cost is bounded by $O(nk)$. The cost of the second stage depends on the specific algorithm used, but the number of distance computations needed for a canopy will be reduced in a factor $\frac{n}{k}$, so for example if single link hierarchical clustering is applied the total computational cost of applying this algorithm to k canopies will be $O(\frac{n}{k}^2)$.

4.2.10 Indexed K-means

The proposal in [31] relies in an approximation strategy. This strategy is applied to the K-means algorithm. One of the computations that have most impact in the cost of this algorithm is that,

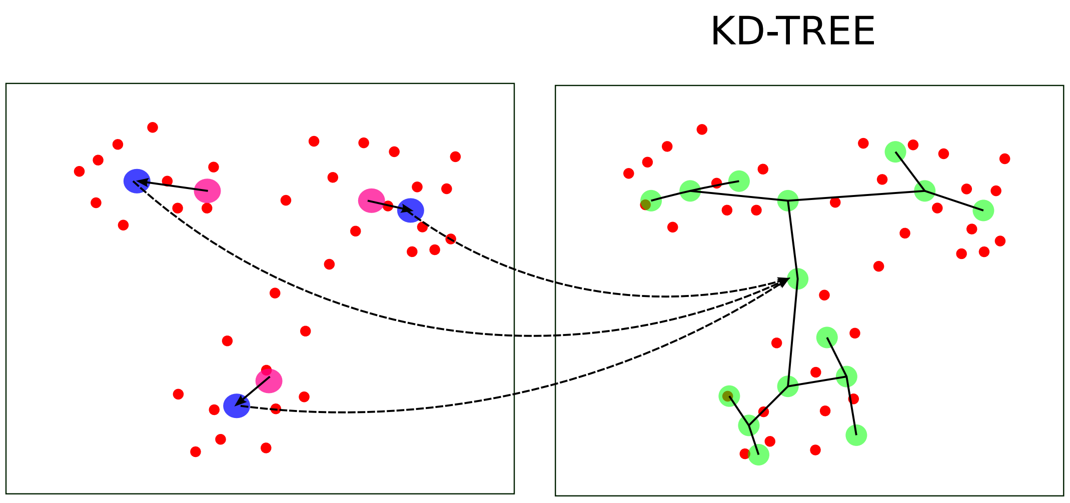


Figure 4.6: Indexed K-means

each iteration all the distances from the examples to the prototypes have to be computed. One observation about the usual behavior of the algorithms is that, after some iterations, most of the examples are not going to change their cluster assignment for the remaining iterations, so computing their distances increases the cost, without having an impact in the decisions of the algorithm.

The idea is to reduce the number of distance computations by storing the dataset in an intelligent data structure that allows to determine how to assign them to the cluster prototypes. This data structure is a kd-tree, a binary search tree that splits the data along axis parallel cuts. Each level can be represented by the centroid of all the examples assigned to each one of the two partitions.

In this proposal, the K-means algorithm is modified to work with this structure. First, a kd-tree is built using all the examples. Then, instead of computing the distance from each example to the prototypes and assigning them to the closest one, the prototypes are inserted in this kd-tree. At each level, the prototypes are assigned to the branch that has the closest centroid. When a branch of the tree has only one prototype assigned, all the examples in that branch can be assigned directly to that prototype, avoiding further distance computations. When a leave of the kd-tree is reached and still there is more that one prototype, the distances among the examples and the prototypes are computed and the assignments are decided by the closest prototype as in the standard K-means algorithm. A representation of this algorithm can be seen in figure 4.6.

The actual performance depends on how separated are the clusters in the data and the granularity of the kd-tree. The more separated the clusters are, the less distance computations have to be performed, as the prototypes will be assigned quickly to only one branch near to the root of the kd-tree.

The time computational cost in the worst case scenario is the same as K-means, as in this case all the prototypes will be assigned to all branches, so all distance computations will be performed. The more favorable case will be when the clusters are well separated and the number of levels in the kd-tree is logarithmic respect to the dataset size (n), this cost will depend also on the volume enclosed in the leaves of the kd-tree and the number of dimensions (d). The computational cost for each iteration is bound by $\log(2^d k \log(n))$.

The major problem of this algorithm is that as the dimensionality increases, the benefit of the kd-tree structure degrades to a lineal search. This is a direct effect of the curse of the dimensionality and the experiments show that for a number of dimensions larger than 20 there are no time savings.

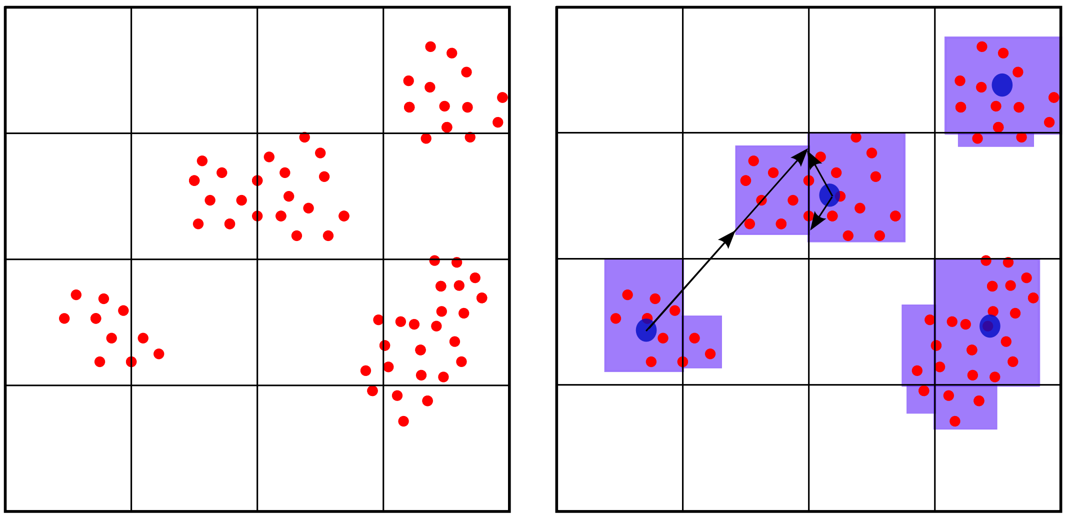


Figure 4.7: Quantized K-means

4.2.11 Quantized K-means

The proposal in [47] relies on an approximation strategy combined with a summarization strategy. The idea is to approximate the space of the examples by assigning the data to a multidimensional histogram. The bins of the histograms can be seen as summaries. This reduces the distance computations by considering all the examples inside a bin of the histogram as a unique point.

The quantization of the space of attributes is obtained by fixing the number of bins for each dimension to $\rho = \lfloor \log_m(n) \rfloor$, being m the number of dimensions and n the number of examples. The size of a bin is $\lambda_l = \frac{\overline{p_l} - \underline{p_l}}{\rho}$, being $\overline{p_l}$ and $\underline{p_l}$ the maximum and minimum value of the dimension l. All examples are assigned to a unique bin depending on the values of their attributes.

From this bins, a set of initial prototypes are computed for initializing a variation of the K-means algorithm. The computation of the initial prototypes uses the assumption that the bins with a higher count of examples are probably in the areas of more density of the space of examples. A max-heap is used to obtain these highly dense bins. Iteratively, the bin with the larger count is extracted from the max-heap and all the bins that are neighbors of this bin are considered. If the count of the bin is larger than its neighbors, it is included in the list of prototypes. All neighbor cells are marked, so they are not used as prototypes. This procedure is repeated until k initial bins are selected. The centroids of these bins are used as initial prototypes.

For the cluster assignment procedure two distance functions are considered involving the distance from a prototype to a bin. The minimum distance from a prototype to a bin is computed as the distance to the nearest corner of the bin. The maximum distance from a prototype to a bin is computed as the distance to the farthest corner of the bin. In figure 4.7, the quantization of the dataset and these distances are represented.

Each iteration of the algorithm first computes the maximum distance from each bin to the prototypes and then it keeps the minimum of these distances as $\overline{d}(b_i, s_*)$. Then, for each prototype the minimum distance to all the bins is computed and the prototypes that are at a distance less than $\overline{d}(b_i, s_*)$ are assigned to the bins.

If only one prototype is assigned to a bin, then all its examples are assigned to the prototype without more distance computations. If there is more than one prototype assigned, the distance among the examples and the prototypes are computed and the examples are assigned to the nearest one. After the assignment of the examples to prototypes, the prototypes are recomputed as the centroid of all the examples.

Further computational improvement can be obtained by calculating the actual bounds of the bins, using the maximum and minimum values of the attributes of the examples inside a bin. This allows to obtain a more precise maximum and minimum distances from prototypes to bins, reducing the number of prototypes that are assigned to a bin.

It is difficult to calculate the actual complexity of the algorithm because it depends on the quantization of the dataset and how separated the clusters are. The initialization step that assigns examples to bins is $O(n)$. The maximum number of bins is bounded by the number of examples n, so at each iteration in the worst case scenario $O(kn)$ computations have to be performed. In the case that the data presents well separated clusters, a large number of bins will be empty, reducing the actual number of computations.

5 Semisupervised Clustering

- Sometimes we have available some information about the dataset we are analyzing unsupervisedly
- Could be interesting to incorporate this information to the clustering process in order to:
  - Bias the search of the algorithm toward the solutions more consistent with our knowledge
  - Improve the quality of the result reducing the algorithm natural bias (predictivity/stability)
- The information that we have available can be of different kinds:
  - Sets of labeled instances
  - Constrains among certain instances: Instances that have to be in the same group/instances that can not belong to the same group
  - General information about the properties that the instances of a group have to hold
- Usually the use will depend on the model we can obtain
  1. Begin with a prior model that changes how the search is performed
  2. Bias the search, pruning the models that are not consistent with the supervised knowledge
  3. Modify the similarity among instances to match the constraints imposed by the prior knowledge

5.1 Biasing using labeled examples

- Assuming that we have some labeled examples, these can be used to obtain an initial model
- We only have to know what examples belong to clusters, the actual clusters are not needed
- We can begin from this model the clustering process
- This means that we decide the starting point of the search using the supervised information

- This initial model changes the search and the final model (bias)
- This differs from semi-supervised learning from a supervised perspective, were the labels of some of the examples are known

Basu, Banerjee, Mooney, "**Semi supervised clustering by seeding**", ICML 2002

- Algorithm based on K-means (spherical clusters based on prototypes)
- The usual initialization of K-means is by selecting randomly the initial prototype (there are other alternatives)
- Two proposals:
  - Use the labeled examples to build the initial prototypes (seeding)
  - Use the labeled examples to build the initial prototypes and constrain the model so the labeled examples are always in the initial clusters (seed and constraint)
- The initial prototypes give an initial probability distribution for the clustering

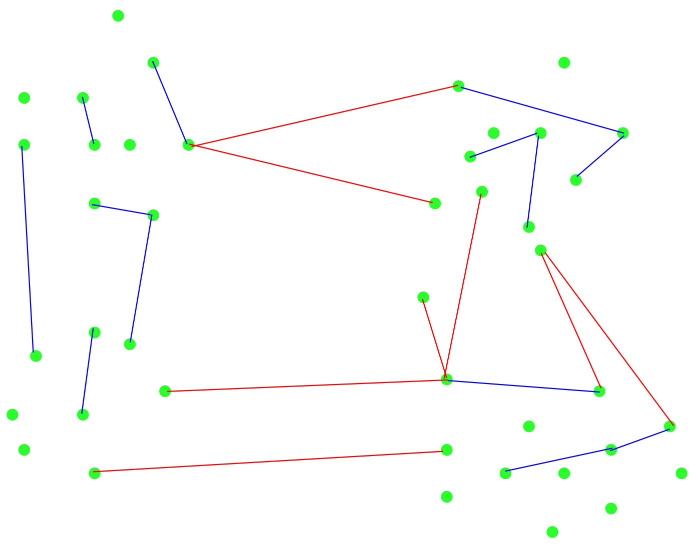
```
Algorithm: Seeded-KMeans
Input: The dataset $\mathcal{X}$, the number of clusters $K$, a set $\mathcal{S}$ of labeled instances (k groups)
Output: A partition of $\mathcal{X}$ in $K$ groups
begin
    Compute $K$ initial prototypes ($\mu_i$) using the labeled instances
    repeat
        Assign each example from $\mathcal{X}$ to their nearest prototype $\mu_i$
        Recompute the prototype $\mu_i$ with the examples assigned
    until Convergence
end
```

```
Algorithm: Constrained-KMeans
Input: The dataset $\mathcal{X}$, the number of clusters $K$, a set $\mathcal{S}$ of labeled instances (k groups)
Output: A partition of $\mathcal{X}$ in $K$ groups
begin
    Compute $K$ initial prototypes ($\mu_i$) using the labeled instances
    repeat
        Maintain the examples from $\mathcal{S}$ in their initial classes
        Assign each example from $\mathcal{X}$ to their nearest prototype $\mu_i$
        Recompute the prototype $\mu_i$ with the examples assigned
    until Convergence
end
```

5.2 Biasing using instance constraints

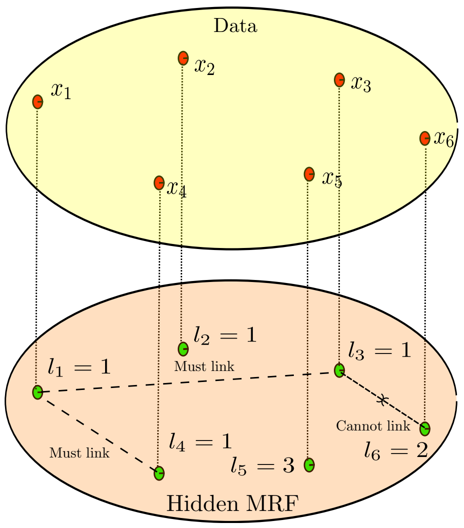
- To have labeled examples means that the number of clusters and something about the characteristic of the data is known
- Sometimes it is easier to have information about if two examples have to be in the same or different clusters

- This information can be expressed by means of constraints among examples: *must links* and *cannot links*
- This information can be used to bias the search and only look for models that maintain these constraints



Basu, Bilenko, Mooney, "**A probabilistic framework for semi-supervised clustering**", ICML 2002

- Algorithm based on K-means (spherical clusters based on prototypes)
- A set of must-link cannot-link constraints is defined over a subset of examples
- The quality function of the K-means algorithm is modified to bias the search
- A hidden markov random field is defined using the constraints
- The labels of the examples can be used to define a markov random field
- The must-links and cannot-links define the dependence among the variables
- The clustering of the examples has to maximize the probability of the hidden markov random field



- A new objective function for the K-Means algorithm is needed
- The main idea is to introduce a penalty term to the objective function that:
  - Penalizes the clustering that puts examples with must-links in different clusters
  - Penalizes the clustering that puts examples with cannot-links in the same cluster
- This penalty has to be proportional to the distance among the instances
- The new objective function is:

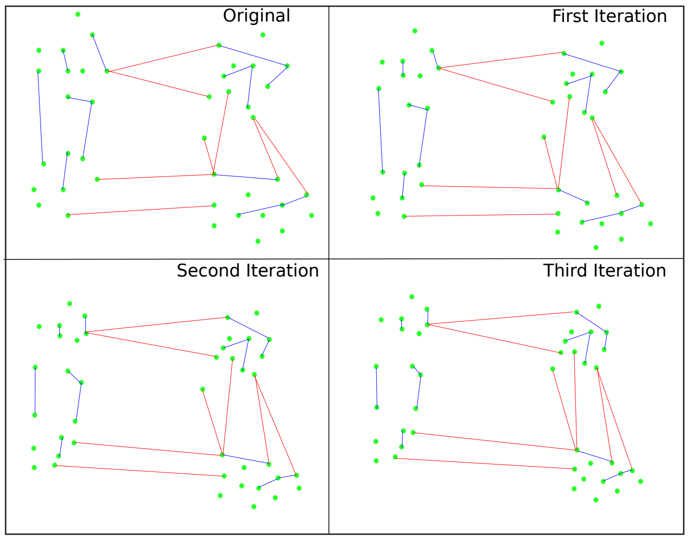
$$\begin{aligned}\mathcal{J}_{obj} \;=\; & \sum_{x_i \in \mathcal{X}} D(x_i, \mu_{l_i}) + \sum_{(x_i,x_j)\in\mathcal{M},[l_i \neq l_j]} w_{ij}\ \varphi_D(x_i,x_j) + \\ & \sum_{(x_i,x_j)\in\mathcal{C},[l_i = l_j]} \bar{w}_{ij}\ (\varphi_{Dmax} - \varphi_D(x_i,x_j))\end{aligned}$$

  Where D is the distortion function, $\mathcal{M}$ and $\mathcal{C}$ are the sets of must and cannot links, w_{ij} and $\bar{w}_{ij}$ are the weight of the penalty for violating the constraints and φ_D is an increasing function of the distance between two examples
- The specific function depends on the distortion measure selected
- The initialization of the process begins by analyzing the constraints and inferring a set of examples to obtain the initial prototypes
- The constraints conform a graph that divide a part of the examples in different groups (connected components of the graph)
- With these instances K prototypes are computed and used to initialize the algorithm
- The algorithm minimizes the objective function using an expectation and an minimization step

```
Algorithm: HMRF-KMeans
Input: The dataset $\mathcal{X}$, the number of clusters $K$, a set of must and cannot links, a
       distance function $D$ and a set of weights for violating the constraints
Output: A partition of $\mathcal{X}$ in $K$ groups
begin
    Compute $K$ initial prototypes ($\mu_i$) using constraints
    repeat
        E-step: Reassign the labels of the examples using the prototypes ($\mu_i$) to
        minimize $\mathcal{J}_{obj}$
        M-step: Given the cluster labels recalculate cluster centroids to minimize $\mathcal{J}_{obj}$
    until Convergence
end
```

- The constraints can also be seen as an indication of the inadequacy of the distance measure between two instances
  - A must-link violation means that the distance function assigns a similarity less than the desired similarity
  - A cannot-link violation means that distance function assigns a similarity larger than the desired similarity

- The previous algorithm can be modified to introduce weights to modify the distances among instances with constraints (an additional maximization step is needed)
- Other approaches learn the more adequate distance function to fulfill the constraints
- The constraints are used as a guide to find a distance matrix that represents the relations among examples
- This problem can be defined as an optimization problem that optimizes the distances among examples with respect to the constraints
- This methods are related to kernel methods, the goal is to learn a Kernel matrix that represents a new space where the instances have appropriate distances
- There are several methods:
  - **Relevant Component Analysis** [Yeung, Chang (2006)] (optimization of linear combination of distance kernels generated by must and cannot links)
  - Optimization using Spectral Graphs for maintaining the constrains in the new space and the structure of the original space
  - Learning of Mahalanobis distances: separate/approach the different dimensions to match the constraints
  - Kernel Clustering (Kernel K-means) with kernel matrix learning via regularization
  - Probabilistic Clustering (Model Based) with constraints via EM (modify the probabilities matching the constrains)



6 Association Rules

There are applications where we are interested in the relations among the different variables that describe the data. The idea is that these relations can reveal some notion of causality through attribute correlation.

There are different models focused on discovering these kinds of relationships like bayesian networks for instance. Even clustering methods can be used for finding these relationships. In this chapter we will focus on a more simple model, but that is in essence related to the techniques from these methods.

The idea is to lose focus on the examples in the data and to use them only for assess variable relationships. The methods that we are going to describe focus specifically on discrete valued variables, and are usually described for binary data, but can be easily applied to multivalued discrete data using binarization as preprocess.

A *binary* dataset is a collection of examples where each row is described by a binary attribute. For instance:

	A	B	C	D	...
T1	1	0	0	1	...
T2	0	1	1	1	...
T3	1	0	1	0	...
T4	0	0	1	0	...

The usual meaning for the variable is that an specific event occurs or not for an example. The typical domain does not allow for having many attributes that occur for an example, for instance, individual purchase transactions from a supermarket. This makes easy to represent the data as a sparse matrix that only stores the occurrences of the attributes for each example. For instance, the database:

TID	Items
1	Bread, Chips, Beer, Yogourt, Eggs
2	Flour, Beer, Eggs, Butter,
3	Bread, Ham, Eggs, Butter, Milk
4	Flour, Eggs, Butter, Chocolate
5	Beer, Chips, Bacon, Eggs

defines individual purchases of items by clients. This chapter will consider that each example is independent from the rest. In the next chapter we will add a temporal link to the data considering sequences of transactions from the same user. For know we will consider the link among variables and extract what is called *association rules*.

6.1 Definitions

From a binary/transactions database we will define a association rule as the representation of the coocurence of events in the database. For instance, in the previous example database we can find that the following associations appear several times:

$$\{\text{Flour}\} \rightarrow \{\text{Eggs}\}$$
$$\{\text{Beer}\} \rightarrow \{\text{Chips}\}$$
$$\{\text{Bacon}\} \rightarrow \{\text{Eggs}\}$$

We are going to be interested on the associations that appear with an specific frequency.

Formally, we will define R as the set of attributes that a transaction of the database can have. We define also X as a subset of attributes from R. We say that X is a **pattern** from the database if there is any row where all the attributes of X are present (its value equals 1).

We define the *support* (*frequency*) of a pattern X as the function:

$$fr(X) = \frac{|M(X,r)|}{|r|}$$

where $|M(X,r)|$ is the number of times that X appears in the DB and $|r|$ is the size of the database. Given a minimum support ($min\_sup \in [0,1]$), we say that the pattern X is frequent (*frequent itemset*) if:

$$fr(X) \geq min\_sup$$

$\mathcal{F}(r, min\_sup)$ is the set of patterns that are frequent in :

$$\mathcal{F}(r, min\_sup) = \{X \subseteq R / fr(X) \geq min\_sup\}$$

. Given a database with R attributes and X and Y attribute subsets, with $X \cap Y = \emptyset$, an *a*ssociation rule is the expression:

$$X \Rightarrow Y$$

. The function $conf(X \Rightarrow Y)$ is the *confidence* of an association rule, computed as:

$$conf(X \Rightarrow Y) = \frac{fr(X \cup Y)}{fr(X)}$$

We consider a minimum value of confidence ($min\_conf \in [0,1]$). Given a minimum confidence and a minimum support, an association rule $(X \Rightarrow Y)$ exists in a DB if:

$$(fr(X \cup Y) \geq min\_sup) \wedge (conf(X \Rightarrow Y) \geq min\_conf)$$

There is a trivial approach for obtaining all the association rules that have a support and confidence higher than given thresholds, it is enough to find all frequent subsets from R. This means to explore for all possible patters X with $X \subseteq R$) and subpatterns Y where $Y \subseteq X$ the association rule $(X - Y) \Rightarrow Y$. Obviously, there are $2^{|R|}$ possible candidates, making this approach unpractical. Figure 6.1 represent all possible subsets considering for attributes.

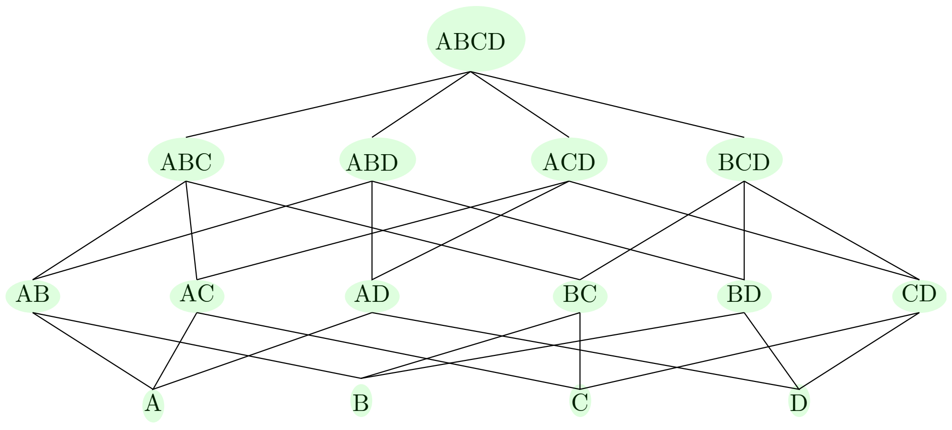


Figure 6.1: Space of possible association rules with four attributes

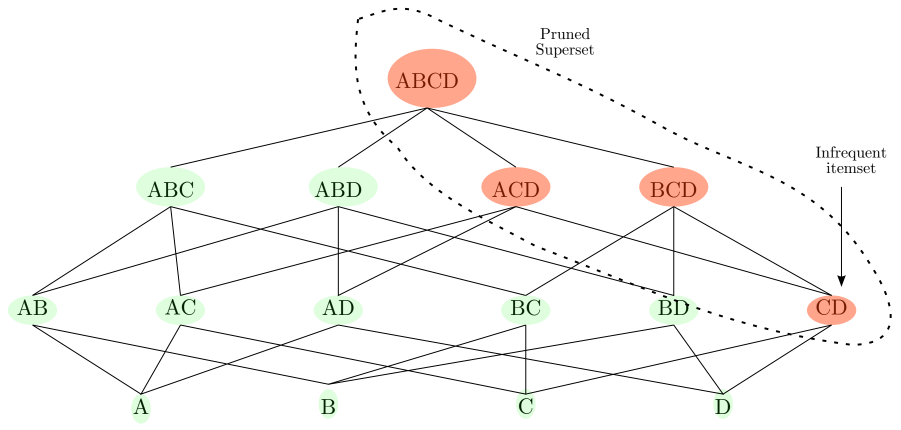


Figure 6.2: Effect of prunning itemsets containing non frequent itemsets

6.2 The Apriori algorithm

We need an approach that prunes the search space, reducing the set of candidates to a more practical size. Given X and Y with $Y \subseteq X$, the following property holds, if $fr(Y) \geq fr(X)$, if X is frequent, Y also is frequent. In other words, if any subset Y from X it is not frequent then X is not frequent. This is known as the *anti-monotone* property of support.

A feasible exploration approach is to begin with all the frequent sets of size 1 and to increase its size iteratively, checking the anti-monotone property in order to prune the candidates that include infrequent itemsets. Figure 6.2 represents the effect of prunning al subsets that contain the itemset `CD` considering that is not frequent.

The *apriori* algorithm was the first one developed for the exploration of the space of association rules. It defines the set of candidates to evaluate each iteration in the following way, being $\mathcal{F}_l(r, min\_sup)$ the set of frequent sets from R of size l, and given a set of patterns of length l, the only frequent set candidates of length $l+1$ will be those which all subsets are in the frequent sets of length l. Formally:

$$C(\mathcal{F}_{l+1}(r)) = \{X \subseteq R/|X| = l+1 \wedge \forall_Y\ (Y \subseteq X \wedge |Y| = l \Rightarrow Y \in \mathcal{F}_l(r))\}$$

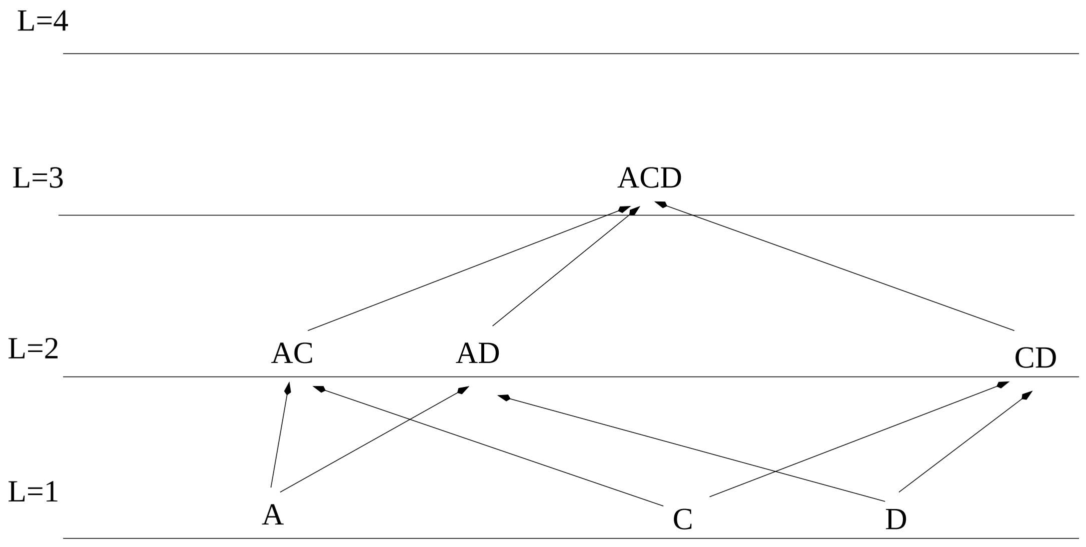
The computation of association rules can be done iteratively starting with the smallest frequent subsets until no more candidates are obtained. Algorithm 6.1 presents an implementation.

For example, given four attributes $\{A, B, C, D\}$, and a database of transactions we have a set of possible candidates (see figure 6.1). If we compute the pattern candidates of length 1,

Algorithm 6.1 Apriori algorithm frror frequent itemsets exploration

```
Algorithm: Apriori (R,min_sup,min_conf)
C,CCan,CTemp:set of frequent subsets
RAs:Set of association rules, L:integer
L=1
CCan=Frequent_sets_1(R,fr_min)
RAs=$\emptyset$
while $CCan \neq \emptyset$ do
    L=L+1
    CTemp=Candidate_sets(L,R,CCan) = $C(\mathcal{F}_{l+1}(r))$
    C=Frequent_sets(CTemp,min_sup)
    RAs=RAs $\cup$ Confidence_rules(C,min_conf)
    CCan=C
end
```

we will find that some attributes will have a frequency larger than a specific support threshold. Assume that these attributes are $\{A, C, D\}$, the next graph shows the space of candidates after prunning candidates that include the non frequent item B.



The patterns of length 2 that are not prunned will be the possible candidates for frequent itemsets of this length. If we assume that all these patterns are frequent we will have that there is only one pattern left as candidate for frequent patterns of length three, so the algorithms stops at this pattern length.

Something that has to be noticed is that the specific value used as *min_sup* has effect on the computational cost of the search because it limits what is going to be explored. If it is too high, only a few patterns will appear (we could miss interesting rare occurrences), if it is too low the computational cost will be too high (too many associations will appear). The extreme is when the value is 0, in this case all items are frequent, so all the possible combinations of items will be generated. Unfortunately the threshold value can not be known beforehand.

Sometimes, multiple minimum supports could be used, different for different groups of items, to explore the interesting associations. This could reduce the combinatorial explosion due to very frequent items and focus on less frequent but more interesting ones. These thresholds are domain specific and some previous analysis of the data should be performed to guess these values.

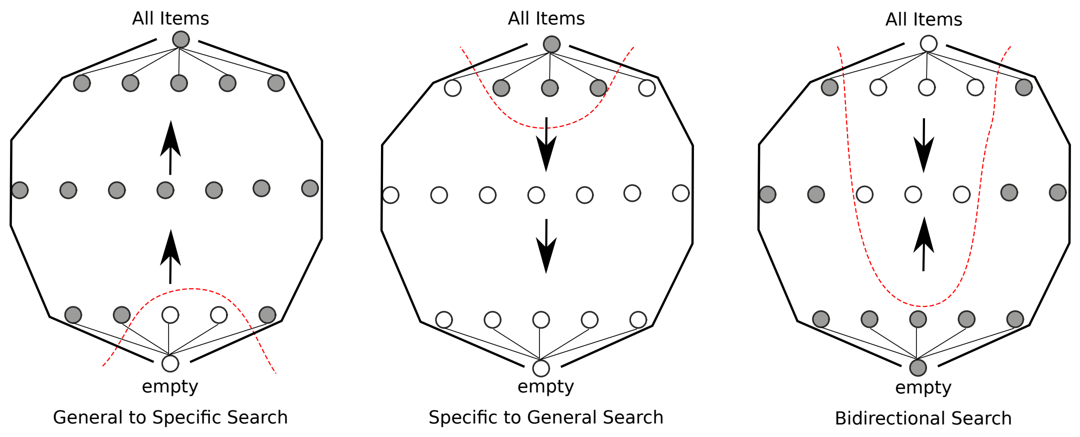


Figure 6.3: Different search strategies for the apriori algorithm

Other elements of the problem also have an influence on the computational cost, such as the number of different items, the more items there are the more space is needed to count its support, the size of the database, because the algorithm has to perform multiple passes to count the support, and the average transaction width that affects the maximum length of frequent itemsets.

Some specialized algorithms have been developed to deal with these problems in some domains, for example, in bioinformatics, it is usual to have long frequent itemsets.

The number of possible frequent itemsets obtained by the apriori algorithm is one of its main problems. This can be improved by targeting subsets of itemsets that have specific properties. There are two kinds of itemsets that are used for reducing the number of candidates:

- *Maximal frequent itemsets*, defined as frequent itemset for which none of its immediate supersets are frequent
- *Closed frequent itemsets:*, defined as frequent itemset for which none of its immediate supersets have the same support

This is the relationship among these subsets and the whole set of frequent itemsets.

$$\text{Maximal frequent itemsets} \subseteq \text{Closed frequent itemsets} \subseteq \text{Frequent itemsets}$$

The actual reduction on the number of final item sets will depend on the domain and the support threshold, but in practice the number of candidates are sometimes orders of magnitude smaller.

There are variations of the way the itemsets are explored by the apriori algorithm that can reduce the computational cost depending on the characteristics of the items. The described algorithm explores the lattice of itemsets from general-to-specific, but the items can be explored from specific-to-general when the maximum itemset size is large or the transactions are dense. In this case an item has to be decomposed in all its subsets if it is not frequent and exploration stops for a candidate when it is maximal frequent. A didirectional search can also be applied using the information from one direction to prune candidates from the other direction. See figure 6.3.

Other methods for reducing the computational cost include defining equivalence classes for the items so several items are counted as the same, or using for example prefixes (or suffixes) as equivalence classes when it makes sense in the domain.

Also the way the itemsets are explored can be changed from a breadth first search (as apriori has been defined) that can be memory demanding, to a depth first search, that requires less memory for the exploration. This strategy is exploited by algorithms targeting maximal frequent itemsets ot that use an specific-to-general exploration of the itemsets. The purpose is to detect earlier maximal itemsets, so more prunning can be applied to the search.

6.2.1 Example

To illustrate how the apriori algorithm works we will extract the frequent itemsets from the following dataset of transactions:

	A	B	C	D	E	F
1	1	1	0	0	1	0
2	1	0	1	0	1	1
3	0	1	1	1	0	1
4	1	1	1	0	0	0
5	0	1	0	0	1	1
6	1	0	1	1	0	0
7	1	1	0	0	1	0
8	1	1	0	0	0	0
9	1	0	1	1	0	1
10	1	1	1	0	1	1
11	1	1	0	0	1	0
12	1	1	0	1	1	0
13	1	1	1	1	1	1
14	0	1	1	0	0	0
15	1	1	0	1	0	0
16	1	1	0	0	1	0
17	0	1	0	0	1	0
18	0	1	1	0	1	0
19	1	0	0	0	1	0
20	1	1	0	0	1	0

fr([A])=15/20
fr([B])=16/20
fr([C])= 9/20
fr([D])= 6/20
fr([E])=13/20
fr([F])= 6/20

First we have to give the minimum support threshold, for example we can decide arbitrarily to use a value of 0.25. In this case there is no information from the domain that helps us to decide the value, but obviously a deeper study of the data is necessary to make this decision. In this case we can observe that all columns have larger frequency, thus, the set of candidates of length 1 is:

F1= {[A],[B],[C],[D],[E],[F]}

From this we can compute the candidates for frequent sets of length 2 (all combinations):

C(F1)= {[A,B],[A,C],[A,D],[A,E],[A,F],
[B,C],[B,D],[B,E],[B,F],
[C,D],[C,E],[C,F],
[D,E],[D,F],
[E,F]}

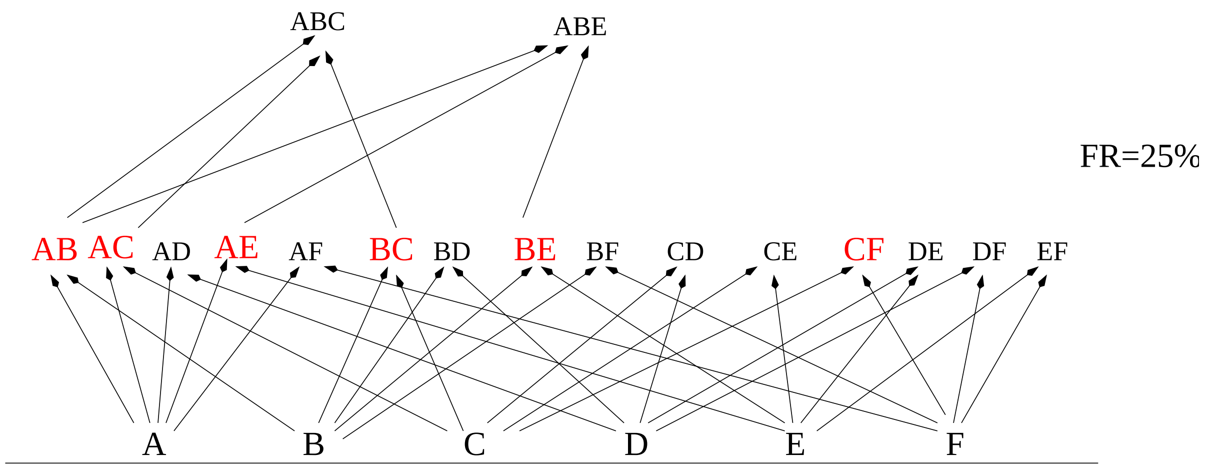
From the data table we can compute the frequency of each candidate:

fr([A,B])= 11/20	fr([B,C])= 6/20	fr([C,D])= 4/20	fr([D,E])= 2/20
fr([A,C])= 6/20	fr([B,D])= 4/20	fr([C,E])= 4/20	fr([D,F])= 2/20
fr([A,D])= 4/20	fr([B,E])= 11/20	fr([C,F])= 5/20	fr([E,F])= 4/20
fr([A,E])= 10/20	fr([B,F])= 4/20		
fr([A,F])= 4/20			

Of the candidates, the ones that have a support greater than 25% are:

F2={[A,B],[A,C],[A,E],[B,C],[B,E],[C,F]}

In the lattice exploration we would have:



If we evaluate the confidence of each one of the association rules we have

conf(A → B)= 11/15	conf(A → E)= 10/15	conf(B → E)= 11/16
conf(B → A)= 11/16	conf(E → A)= 10/13	conf(E → B)= 11/13
conf(A → C)= 6/15	conf(B → C)= 6/16	conf(C → F)= 5/9
conf(C → A)= 6/9	conf(C → B)= 6/9	conf(F → C)= 5/13

Choosing also arbitrarily a confidence threshold of 2/3, the rules that are significant are:

$$\{A \rightarrow B,\ B \rightarrow A,\ C \rightarrow A,\ A \rightarrow E,\ E \rightarrow A,\ C \rightarrow B,\ B \rightarrow E,\ E \rightarrow B\}$$

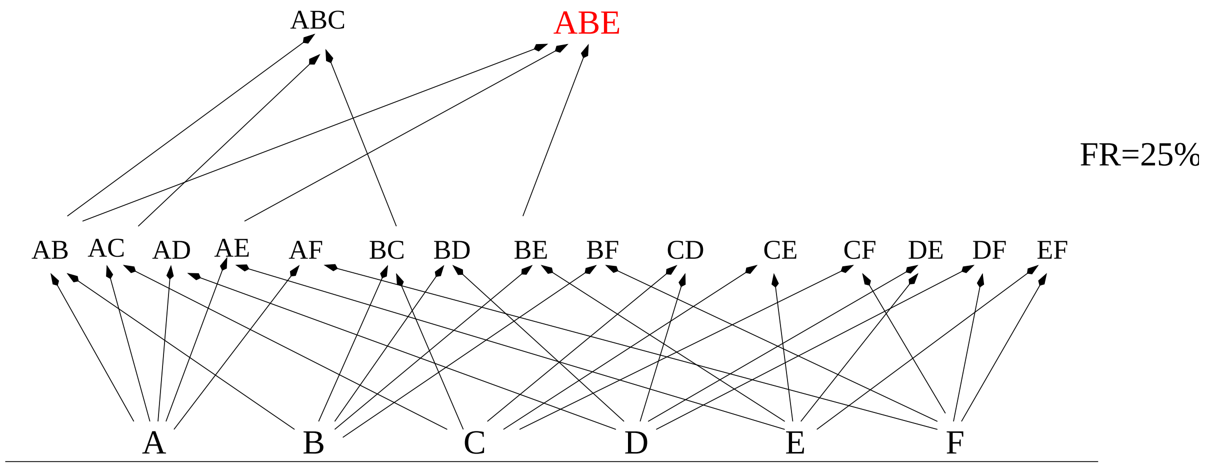
From the set F2, we can compute the set of candidates for F3

C(F2)={[A,B,C],[A,B,E]}

fr([A,B,C])= 3/20   fr([A,B,E])= 8/20

Of the candidates, the ones that have support greater than 25% are:

F3={[A,B,E]}



The confidence of their corresponding association rules are:

$$\begin{aligned}&\text{conf}(A \wedge B \rightarrow E)= 8/11\\&\text{conf}(A \wedge E \rightarrow B)= 8/10\\&\text{conf}(B \wedge E \rightarrow A)= 8/11\end{aligned}$$

If we choose as confidence value 2/3 the rules that are significant:

$$\{A \wedge B\rightarrow E,\ A \wedge E \rightarrow B,\ B \wedge E \rightarrow A\}$$

From all the itemsets discovered:

fr([A,B])= 11/20	fr([B,C])= 6/20
fr([A,C])= 6/20	fr([B,E])= 11/20
fr([A,E])= 10/20	fr([C,F])= 5/20
fr([A,B,E])= 8/20	

{[A,C], [B,C], [C,F], [A,B,E]} are maximal frequent
{[A,B],[B,E],[A,E]} are closed frequent

Now we choose a minimum support of 50%. The columns that have larger frequency are only A, B and E. Therefore:

$$F1= \{[A],[B],[E]\}$$

From this we can compute the support set candidates:

$$C(F1)=\{[A,B],[A,E],[B,E]\}$$

All have a support greater than 50%, therefore

$$F2=\{[A,B],[A,E],[B,E]\}$$

And the candidates are:

$$C(F2)=\{[A,B,E]\}$$

That also have a support greater than 50%, thus:

$$F3=\{[A,B,E]\}$$

- Despite of prunning and search strategies for Association Rules candidate generation is expensive
- Other strategies allow to extract association rules using specialized data structures

6.3 The FP-Grow algorithm

Han, Pei, Yin, Mao **Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach** (2004)

- The problem of the *apriori* approach is that the number of candidates to explore in order to find long patterns can be very large
- This approach tries to obtain patterns from transaction databases without candidate generation

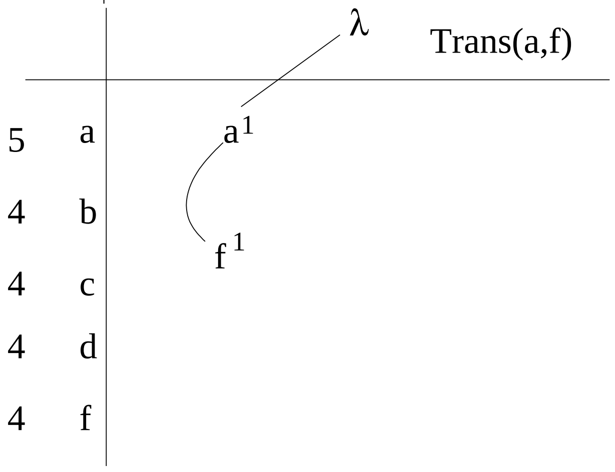
- It is based on a specialized data structure (**FP-Tree**) that summarizes the frequency of the patterns in the DB
- The patterns are explored incrementally adding prefixes without candidate generation
- The goal of this structure is to avoid to query the DB to compute the frequency of patterns
- It assumes that there is an order among the elements of a transaction
- This order allows to obtain common prefixes from the transactions
- The transactions with common prefixes are merged in the structure maintaining the frequency of each subprefix

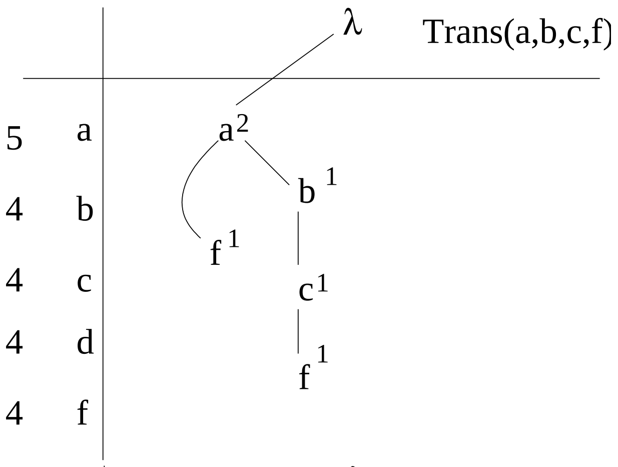
1. Compute the frequency of each individual item in the DB
2. Create the tree root (empty prefix)
3. For each transaction from the DB
   (a) Pick the more frequent items
   (b) Order the items by their original frequency
   (c) Iterate for each item, inserting it in the tree
      - If the node has a descendant equal to the actual item increase its frequency
      - Otherwise, a new node is created

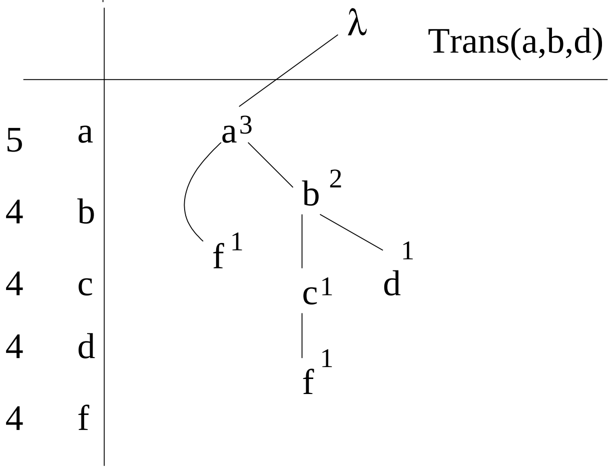
BD	Transactions
1	a, e, f
2	a, c, b, f, g
3	b, a, e, d
4	d, e, a, b, c
5	c, d, f, b
6	c, f, a, d

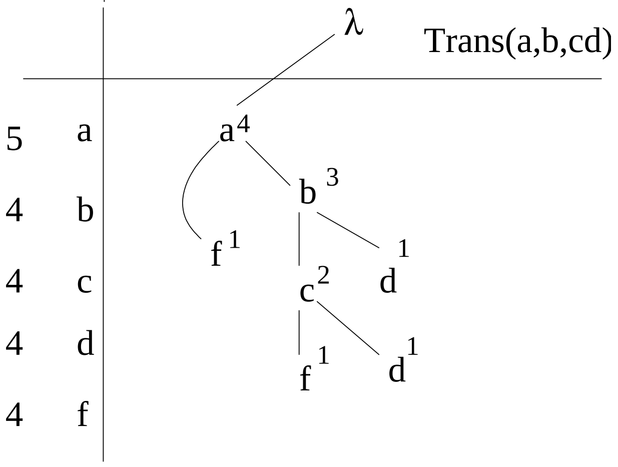
Item	frequency
a	5
b	4
c	4
d	4
e	3
f	4
g	1

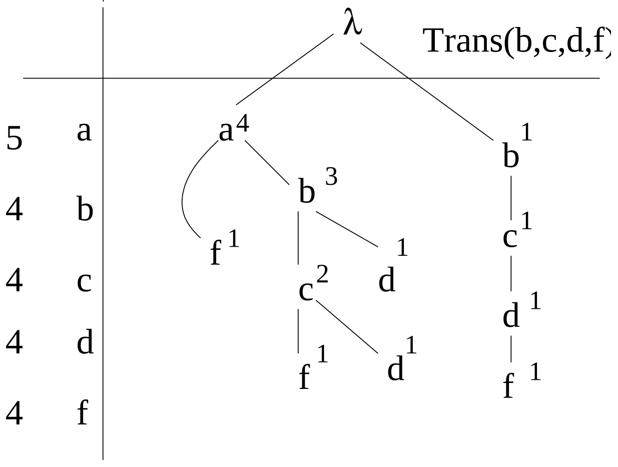
BD	Transactions (fr=4)
1	a, f
2	a, c, b, f
3	a, b, d
4	a, b, c, d
5	b, c, d
6	a, c, d, f

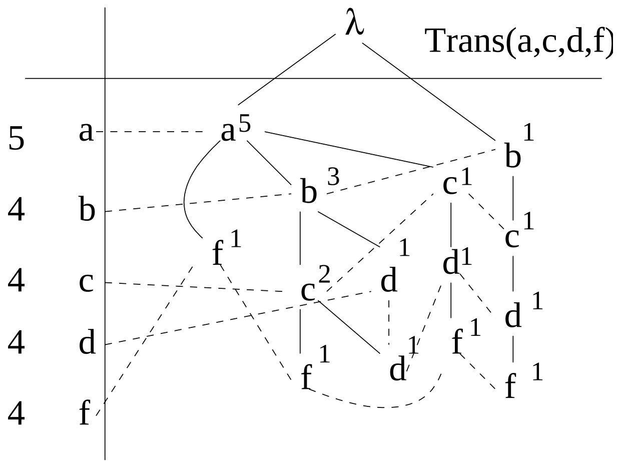



λ
Trans(a,b,c,f)
5 a
4 b
4 c
4 d
4 f
a2
b 1
f 1
c1
f 1


λ
Trans(a,b,d)
5 a
4 b
4 c
4 d
4 f
a3
b 2
f 1
1 d
c1
f 1


λ
Trans(a,b,cd)
5 a
4 b
4 c
4 d
4 f
a4
b 3
f 1
1 d
c2
f 1
d1

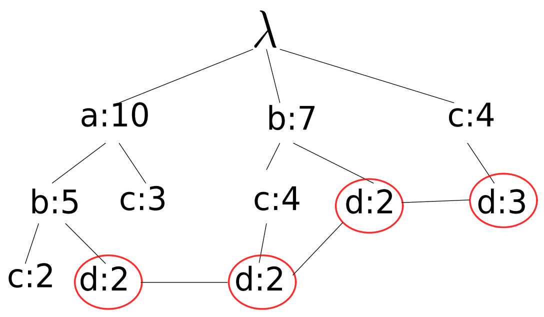

λ
Trans(b,c,d,f)
5 a
4 b
4 c
4 d
4 f
a4
b 3
f 1
1 d
c2
f 1
d1
b 1
c1
d 1
f 1



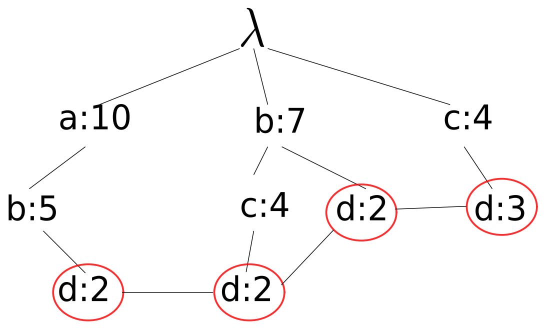
- From the FP-tree we can obtain all frequent patterns using its properties and a bottom-up strategy:
  - For any given frequent item a_i we can obtain all the frequent patterns that contain the item following the links of a_i in the tree
  - To compute the frequent patterns with a_i as suffix only the prefix subpaths of the nodes a_i have to be used and their frequency corresponds to the frequency of node a_i
- The frequent patterns can be obtained from a recursive traversal of the paths in the FP-tree and the combination of subpatterns

1. Given an item, select all the paths that contain the item (**prefix paths**)
2. Convert all the paths that contain the item into a *conditional* FP-tree:
   (a) Update the counts along the path using the frequency of the selected item
   (b) Truncate the paths removing the nodes for the item
   (c) Eliminate from the paths the items that are no longer frequent (if any)
3. For all the items previous in order that are frequent in the conditional FP-tree
   (a) Count the prefix as frequent
   (b) Recursivelly find the frequent items for that prefix

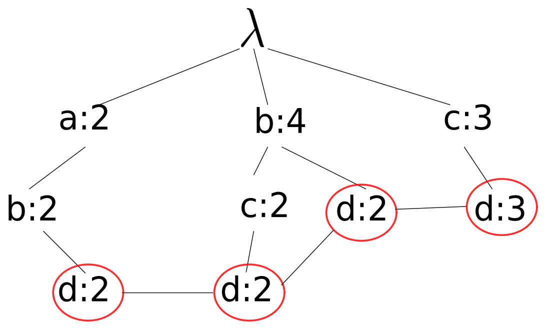
Extract patterns with suffix d



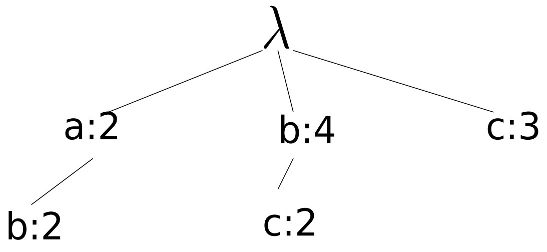
Keep only the paths that contain d (minsupport = 2)



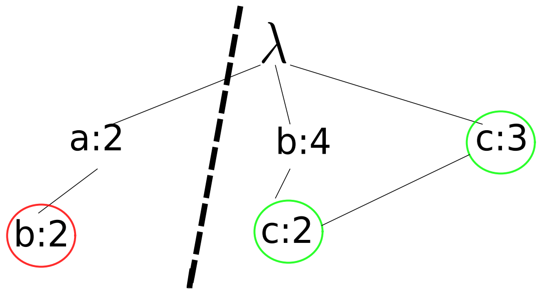
Generate the **conditional FP-tree** for d (updating the counts)



Prune the path eliminating d (is no longer needed)



Solve the problem for the predecessors of d, in this case b and c (we are looking if bd and cd are frequent)



If we continue the algorithm, the patterns extracted would be {[d], [bd], [cd], [abd], [bcd]}

6.4 Measure for association rules interestingness

- Given a rule, its interestingness can be computed from a contingency table:

	Y	$\neg Y$	
X	f_{11}	f_{10}	f_{1+}
$\neg X$	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	$\|r\|$

- f_{11}; support of X and Y

- f_{10}; support of X and $\neg Y$
- f_{01}; support of $\neg X$ and Y
- f_{01}; support of $\neg X$ and $\neg Y$
- Sometimes the *confidence* of a rule does not represent well its interestingness

	Y	$\neg Y$	
X	15	5	20
$\neg X$	75	5	80
	90	10	100

- $conf(X \rightarrow Y) = 0.75$ but $conf(\neg X \rightarrow Y) = 0.9357$
- Other measures are based on probabilistic independence and correlation

Lift/Interest factor	$N \times \frac{f_{11}}{f_{1+} \times f_{+1}}$
All Confidence	$\min(\frac{f_{11}}{f_{1+}}, \frac{f_{11}}{f_{+1}})$
Max Confidence	$\max(\frac{f_{11}}{f_{1+}}, \frac{f_{11}}{f_{+1}})$
Kulczynski	$\frac{1}{2}(\frac{f_{11}}{f_{1+}} + \frac{f_{11}}{f_{+1}})$
Cosine Measure	$\frac{f_{11}}{\sqrt{f_{1+}f_{+1}}}$

f_{11}	f_{01}	f_{10}	f_{00}	Lift	All C	Max C	Kulc	Cos
10000	1000	1000	100000	9.26	0.91	0.91	0.91	0.91
10000	1000	1000	100	1	0.91	0.91	0.91	0.91
100	1000	1000	100000	8.44	0.09	0.09	0.09	0.09
1000	1000	1000	100000	25.75	0.5	0.5	0.5	0.5
1000	100	10000	100000	9.18	0.09	0.91	0.5	0.29
1000	10	100000	100000	1.07	0.01	0.99	0.5	0.10

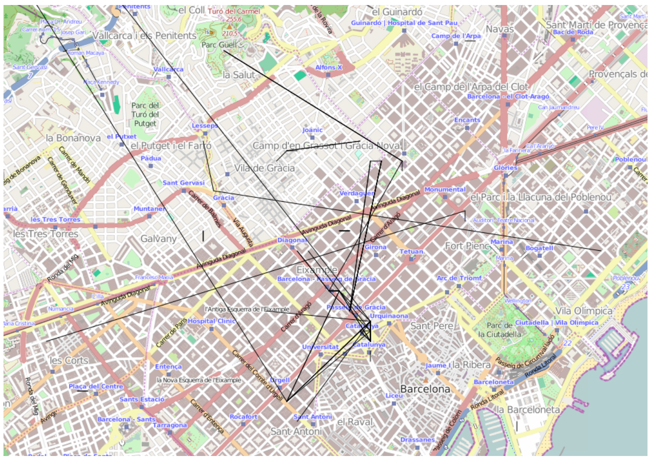
- There are plenty other measures in the literature:
  - ϕ/Correlation analysis
  - Odds ratio
  - Kappa
  - Jaccard
  - Mutual information, conviction, Gini index, certain factor, ...
- Properties that a measure must hold
  - If A and B are statistically independent then $M(A, B) = 0$
  - $M(A, B)$ increases monotonically with $P(A, B)$ when $P(A)$ a $P(B)$ remain unchanged

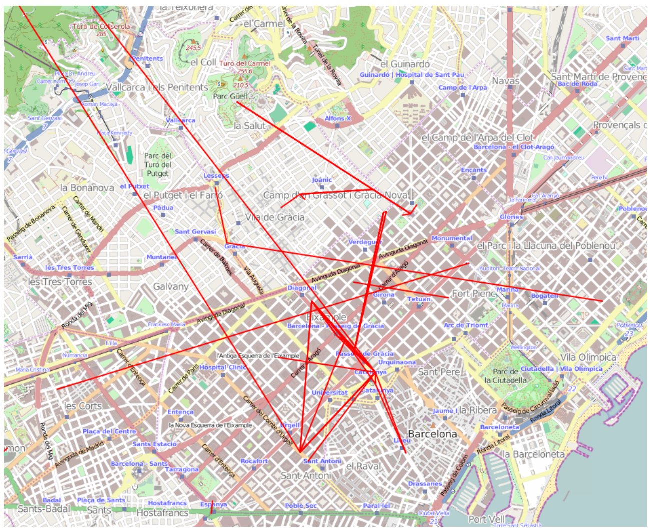
  - $M(A, B)$ decreases monotonically with $P(A)$ (or $P(B)$) when $P(A, B)$ and when $P(B)$ (or $P(A)$) remain unchanged

- Other properties
  - Symmetry ($M(A, B) = M(B, A)$)
  - Invariant to row/column scaling
  - Invariant to inversions ($f_{11} \leftrightarrow f_{00}$, $f_{10} \leftrightarrow f_{01}$)
  - Null addition invariant (not affected if f_{00} is increased)

- The dataset consists on tweets with a time stamp and longitude and latitude coordinates
- Data has been collected for several months (2.5 million tweets) in the Barcelona area ($30Km \times 30Km$)
- The goal is to perform spatio-temporal analysis of the behavior of people in a geographical area
- We are interested in patterns that show connections among different parts of the city
- One approach is to consider a tweet as an event performed by a user
- The events of a user in a single day can be seen as a transaction (market basket)
- The attributes of a transaction are the time and the position of the user
- To discover city connections can be solved by discovering frequent itemsets
- Considering all possible values for position and time makes impossible to discover frequent patterns
- In order to obtain a suitable representation for frequent itemsets discovery we need to discretize the attributes
- For time:
  - Consider different meaningful hour groups (morning, evening, night, ...)
  - Groups of hours
- For coordinates:
  - Equally spaced grid (granularity?)
  - Clustering (what clustering algorithm?)
- We pick to discretize time in intervals of 8 hours (0-7, 8-16, 13-23)
- Two possibilities for coordinates:
  - An equally spaced 300×300 grid ($100m \times 100m$)
  - Clusters obtained by the leader algorithm (radius $100m$)
- The first option makes that a transaction has a determined number of attributes ($300 \times 300 \times 3$), but no all the possibilities will appear

- For second the option the number of attributes will depend on the densities in the coordinates
- Generate the database of transactions
- Discard the transactions that have only value for 1 attribute
- Decide for a value for the support parameters
- Use FP-Grow because of the size of the dataset and the number of possible attributes
- Only present maximal itemsets to reduce the number or redundant patterns



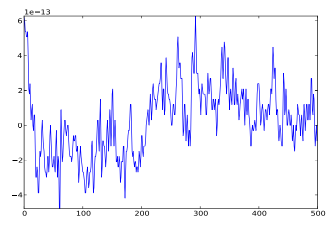


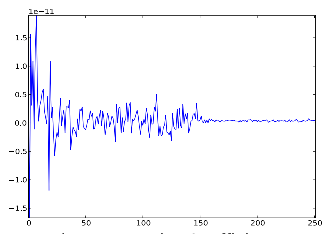
7 Structured Datasets

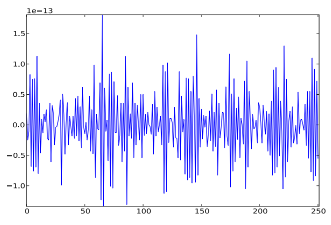
- There are some domains where patterns are more complex
- In these domains instances are related to each other
- Mining these relationships is more interesting than mining instances individually
- For example:
  - Temporal domains
  - Relational databases
  - Structured instances (trees, graphs)
- Usually the methods used in this kind of domains are specific
- These domains suppose some kind of sequential relationship among instances (usually temporal)
- We can have a unique sequence or a set of sequences
- This kind of data can not be analyzed using classical techniques from time series analysis (ARIMA modeling, Kalman filter, ...)
- What makes different this kind of data?
  - Usually qualitative data
  - Very short series or a long series that has to be segmented
  - Interest in the relationships among series
  - Interest only in a part of the series (episodes, anomalies, novelty, ...)
- Customer transactions over the time: Sets of transactions ordered by purchase time
- Web Data: Browsing activity of users in a web site
- Sensor data: History of events generated by a sensor (alarms, anomalies)
- Genome sequences: DNA/Protein sequences

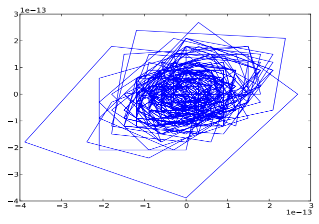
7.1 Clustering for temporal series

- **Clustering of temporal series:** Clustering algorithms applied to a set of short series
  - Representation of the series, representation of the groups
  - New distance measures (scale invariant, shape distances, ...)
  - How to segment a unique series in a set of series? what parts are interesting?
- Segments (windows) of the series (overlapping/non overlapping)
  - Width of the window (series segmentation)
- Feature extraction: Generate informative features from the series windows:
  - Extreme points (maximum, minima, inflection points)
  - Frequency domain features (Fourier Transform, Wavelets, ...)
  - Chaos theory (time-lag transformation)
  - Probabilistic models (Hidden Markov Models)

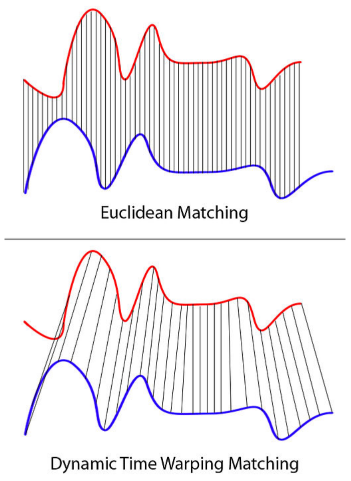

Original Series

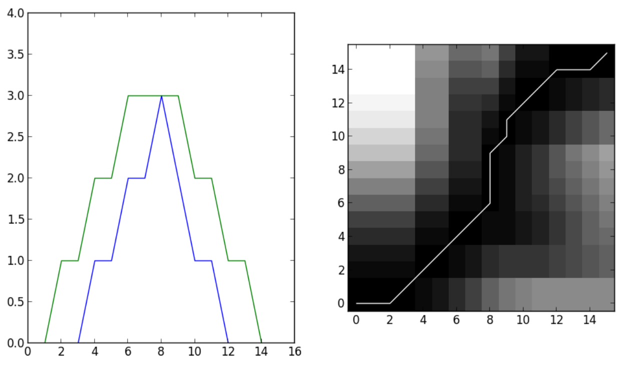

Discrete Fourier Coefficients


Discrete Wavelet Coefficients


1-Lag Series

- Usual distance functions ignore time dynamic
  - Euclidean, hamming, ...
- Patterns in series contain noise, time/amplitude scaling, translations
  - Dynamic Time Warping (DTW)
  - Longest Common Subsequence (LCSS)
  - Edit Distance with Real Penalty (ERP)
  - Edit Distance on Real Sequence (EDR)
  - Spatial Assembly Distance (SpADe)





7.2 Frequent sequences

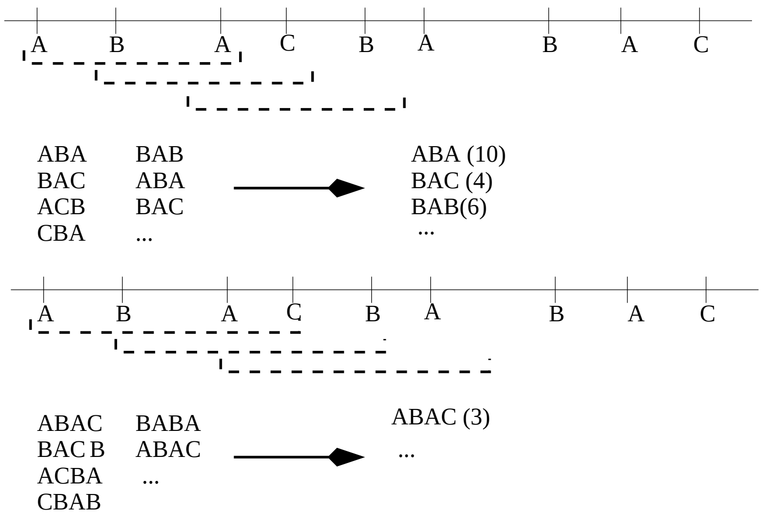
Frequent sequence patterns: Discovery of patterns inside the sequence

- The series does not need to have a temporal relationship (eg.: DNA sequences)
- The values can be continuous, discrete or structured (transaction data)
- Goals:
  - Discovery of subseries that reveal some causality of events
  - Discovery of abnormal/novel patterns (deviation from normal behavior)
  - Discovery of frequent patterns

Mannila, Toivonen, Verkamo **Discovery of frequent episodes in event sequences** (1997)

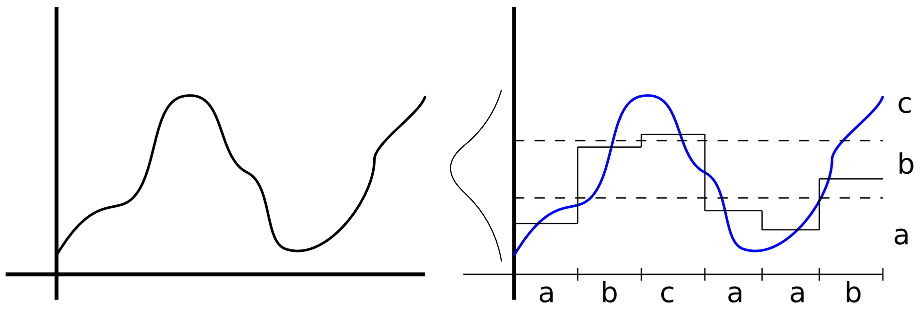
- Qualitative data (events), temporal relationship
- The goal is to discover if close events are causally related
- The result is a set of frequent episodes of different length
- The algorithm discover the patterns using a sliding window of a fixed size
- The size of the window is increased in successive iterations until no further patterns are discovered

- To reduce time complexity uses the property that frequent pattens have to contain smaller frequent patterns (the same than in association rules)



Keogh, Lonardi, Chiu **Finding Surprising Patterns in a Time Series Database In Linear Time and Space** (2002)

- Quantitative data, continuous time series
- The goal is to discover the most frequent patterns given a window length
- The trivial algorithm is $O(N^2)$
- To reduce computational complexity the windows are transformed to a simplified representation (Symbolic Aggregate Approximation/SAX)
- Distance over new representation is a lower bound of distance in original space



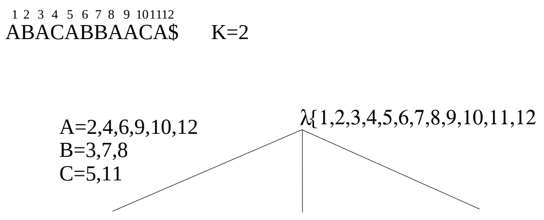
- A window of a specified length is used to segment the sequence (overlapped windows)
- The transformed sequence is stored in multiple hash tables using locality sensitive hashing
- Search for similar sequences:
  - Sequences appearing in the same or contiguous buckets in different hash tables are compared
  - First, compute the distance in the transformed series (cheap), if it is less than a threshold, then compute the exact distance
  - If sequences are near, store the sequences as candidate frequent motifs
- Return the candidates that appear more than a number of times

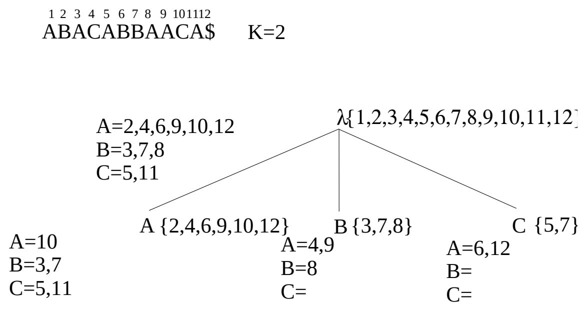
Vilo **Discovering Frequent Patterns from Strings** (1998)

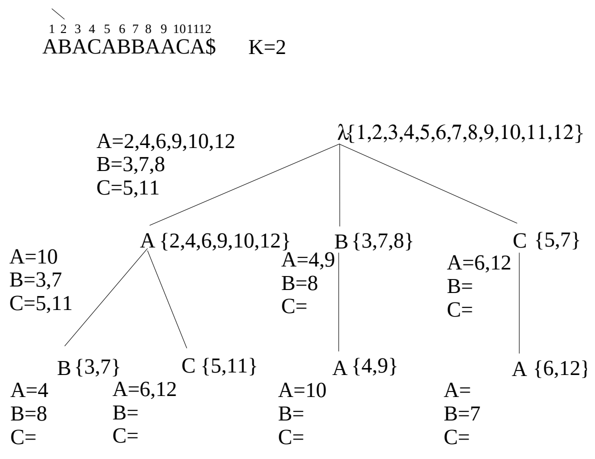
- The sequence is a string of characters (no assumptions about the relationship among the elements of the sequence)
- We look for frequent patterns of any length (complete, with equivalent classes, with wildcards)
- These algorithms use specialized data structures to obtain the patterns within a reasonable computational cost (for example suffix-trees)
- The construction of these structures are sometimes a preprocess (later we extract the patterns) or given a threshold frequency we obtain only the patterns we are looking for

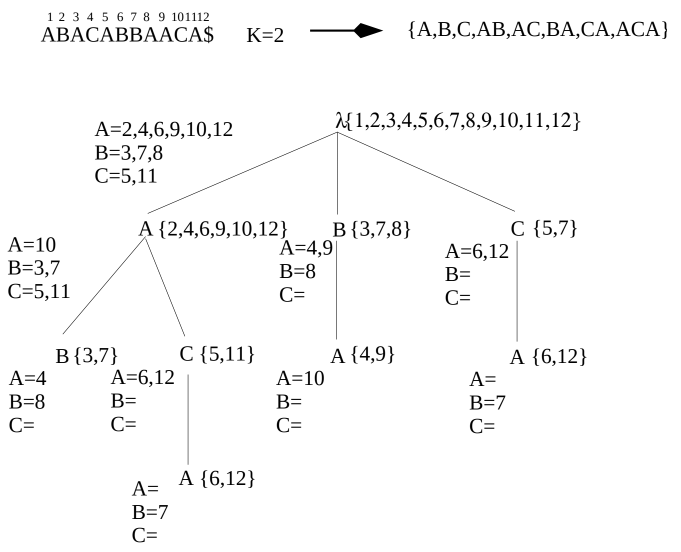
A suffix-trie is obtained from the sequence, a given (k) parameter sets the minimum frequency of the patterns. The structure is created with the empty string.

1. For each character that belongs to the alphabet of the sequence
   (a) Compute a list with the next position to the occurrence of the character
   (b) If the character appears more than k times we create a new descendant in the trie
2. Recursive call for each descendant created




1 2 3 4 5 6 7 8 9 101112
ABACABBAACA$
K=2
λ{1,2,3,4,5,6,7,8,9,10,11,12}
A=2,4,6,9,10,12
B=3,7,8
C=5,11
A {2,4,6,9,10,12}
B {3,7,8}
C {5,7}
A=10
B=3,7
C=5,11
A=4,9
B=8
C=
A=6,12
B=
C=


1 2 3 4 5 6 7 8 9 101112
ABACABBAACA$
K=2
λ{1,2,3,4,5,6,7,8,9,10,11,12}
A=2,4,6,9,10,12
B=3,7,8
C=5,11
A {2,4,6,9,10,12}
B {3,7,8}
C {5,7}
A=10
B=3,7
C=5,11
A=4,9
B=8
C=
A=6,12
B=
C=
B {3,7}
C {5,11}
A {4,9}
A {6,12}
A=4
B=8
C=
A=6,12
B=
C=
A=10
B=
C=
A=
B=7
C=



- Sometimes the events in the sequence are sets of elements
- The definition of sequence has to be adapted:
  - Given an ordered list of elements:

$$s = < e_1 e_2 e_3 ... >$$

  - Each element contains a collection of items

$$e_i = \{i_1, i_2, ..., i_k\}$$

- Also the definition of subsequence:
  - A sequence $< a_1, a_2, ..., a_n >$ is contained in another sequence $< b_1, b_2, ..., b_m >$ $(m \geq n)$ if exists integers $i_1 < i_2 < ...i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$,$a_n \subseteq b_{in}$

Agrawal, Srikant, **Mining Sequential Patterns** (1995)
Srikant, Agrawal, **Mining Sequential Patterns: Generalizations and Improvements** (1996)

- An apriori approach to the mining of sequential patterns
- A similar monotonic property exists among a sequence and its subsequences:
  - Given a sequential pattern s all its subsequences are also sequential patterns
  - Given a sequential pattern s if s' is a subsequence then $support(s) \leq support(s')$
- This means that generating all the frequent sequences for a database can be done using a sequential approach
- Each iteration the sequences of length $k+1$ are generated by merging the frequent sequences of length k

- For the generation of candidates all items in each transaction are listed alphabetically
- Each iteration the database is scanned to test for the candidates and compute their support
- Sequences not frequent are pruned

Candidate generation: Given s_1 and s_2, frequent sequences of length k identical except for the last element

- The last element of s_1 and s_2 contain only one item. Assuming that $s_1 = sx$ and $s_2 = sy$ where s is the maximum common prefix of s_1 and s_2 and x, y two items, then are candidates of length $k+1$ $s(xy)$, sxy and syx

$$(ab)(ac)ab + (ab)(ac)ac \Rightarrow (ab)(ac)a(bc), (ab)(ac)abc, (ab)(ac)acb$$

- If the last element contains more than item, and in alphabetical order these last itemsets are identical except for the last item. Assuming that $s_1 = s(x_1 \cdots x_{m-1}x_m)$ and $s_2 = s(x_1 \cdots x_{m-1}x_{m+1})$ where s is the maximum common prefix of s_1 and s_2, then is a candidate $s(x_1 \cdots x_{m-1}x_m x_{m+1})$

$$(ab)(ac)(ab) + (ab)(ac)(ac) \Rightarrow (ab)(ac)(abc)$$

Candidate generation:

- If the last element of s_2 contains one item, and the second last element of s_2 is identical to the last element of s_1 except for one item that is the last in the last element of s_1 in alphabetical order. Assuming that $s_1 = s(x_1 \cdots x_{m-1}x_m)$ and $s_2 = s(x_1 \cdots x_{m-1}y)$ where s is the maximum common prefix of s_1 and s_2, then is a candidate $s(x_1 \cdots x_{m-1}x_m)y$

$$(ab)(abc) + (ab)(ab)a \Rightarrow (ab)(abc)a$$

Data streams: Modeling an on-line continuous series of data

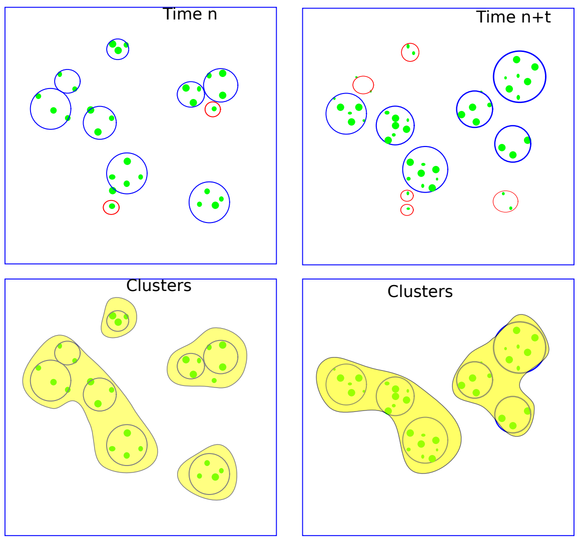
- Each item of the series is an instance (one value, a vector of values, structured data)
  - For instance, sensory data (one or multiple synchronized data), stream of documents (twitter/news)
- Data is generated from a set of clusters (stable or changing over time)
  - For example, states from a process or semantic topics
- The data is processed incrementally (model changes with time)
  - Only the current model
  - Periodic snapshots
- Different goals: Model the domain, detect anomalies/novelty/bursts, change (Concept drift)

Aggarwal, Han, Wang, Yu, **On Clustering Massive Data Streams: A Summarization Paradigm** Data Streams - Models and Algorithms, Springer, 2007, 31, 9-38

- On-line phase:
  - Maintains/creates microclusters
  - Number of microclusters is larger than the actual number of clusters of the dataset (approximate the densities)
  - When new data arrives, it is incorporated to a microcluster or generates new microclusters
  - The number of microclusters is fixed, so microclusters are merged to maintain the number
  - Periodically the microclusters are stored
- Off-line phase:
  - Given a time window the stored clusters are used to compute the microclusters inside the time frame
  - A k-means algorithm is used to compute the clusters for the time window

Cao, Ester, Qian, Zhou, **Density-Based Clustering over an Evolving Data Stream with Noise** Proceedings of the Sixth SIAM International Conference on Data Mining, 2006

- On-line phase:
  - Core-micro-clusters (a weighted sum of close points)
  - The weight of a point fades exponentially with time (damping model)
  - New examples are merged and mc are classified as:
    * core-mc, sets of points with weight over a threshold
    * potential-mc
    * outlier-mc, sets of points with weight below a threshold
  - outlier-mc dissapear with time
- Off-line phase: Modified version of DBSCAN

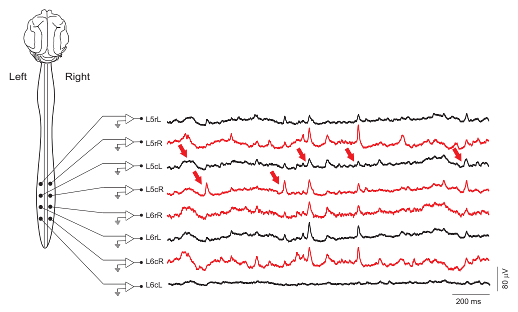


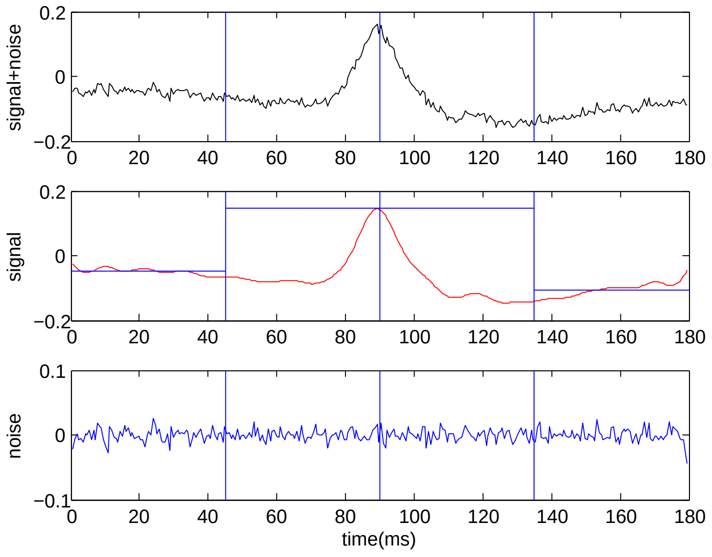
- Recorded signals over the lumbar vertebrae of an anesthetized cat (from L4 to L7)
- Different experimental conditions

- Study of the peaks generated spontaneously by groups of neurons

- Study of the synchronizations of the peaks in the different recording points

- Study of the sequential pattern behavior



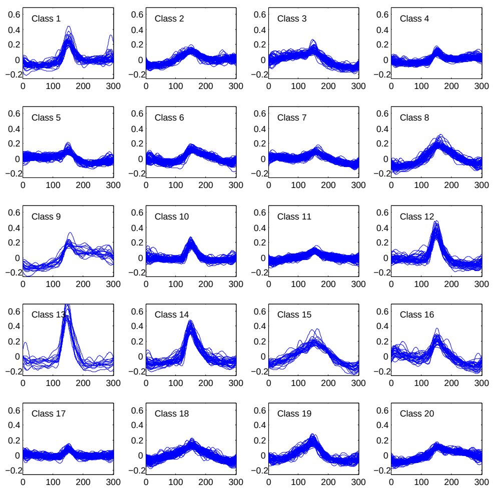


- Goal: Determine an structure for the peaks

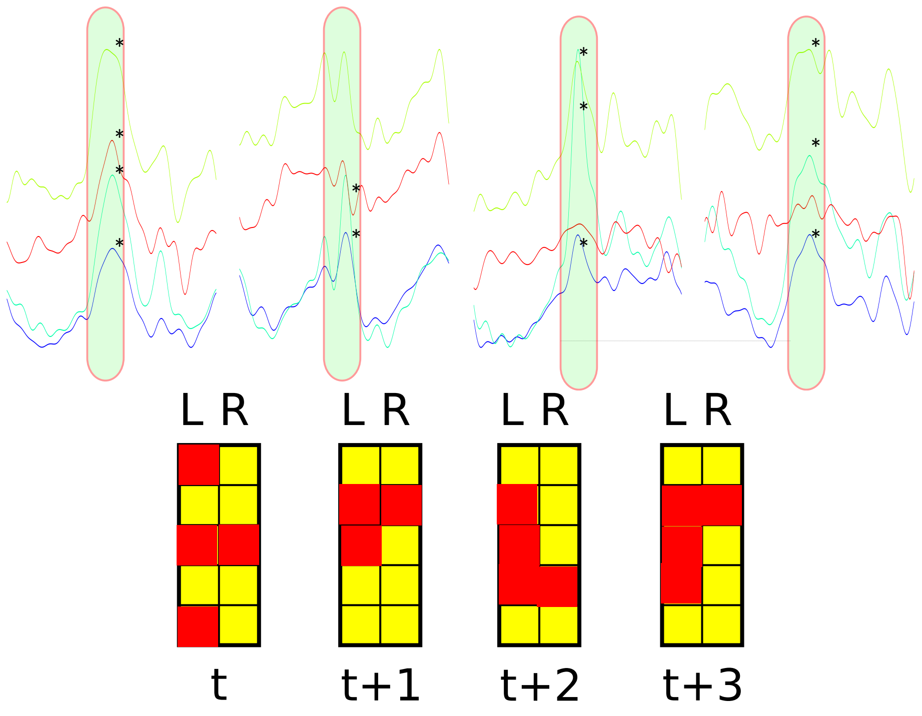
- Process

  - Identify the peaks in the signals (peak finding algorithm)
  - The peaks extracted from each measuring point will be the first dataset
  - Clean/Transform the data to visualize the structure
  - Cluster the datasets to obtain patterns of peaks (k-means $\Longrightarrow$ determine the number of clusters)

- Results: A tentative dictionary of peaks shapes



- Goal: Find patterns in the synchronizations of the peaks
- Process
  - Determine the synchronizations in the signal (synchronization algorithm)
  - Generate a database of transactions from the synchronizations
  - Apply a frequent transactions algorithm to the transactions database
- Results: A set of frequent sequences of synchronization

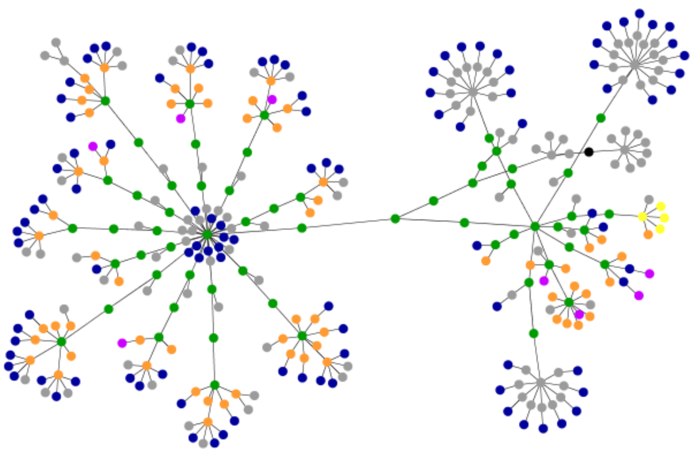


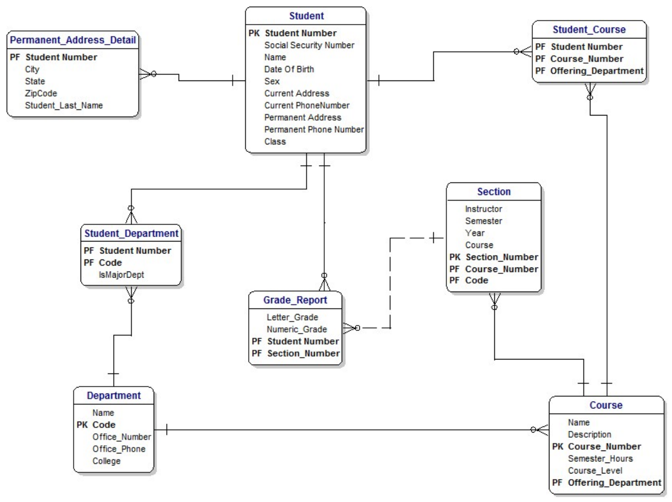
```
L4ci1  --> L5cd5 : 24
L4ci1  --> L6ci2 : 24
L4ci1  --> L6rd5 : 25
L4ci2-------------------------------------
L4ci2  --> L5ci8 : 22
L4ci2  --> L6rd5 : 25
L4ci2  --> L6rd8 : 23
L4ci5-------------------------------------
```

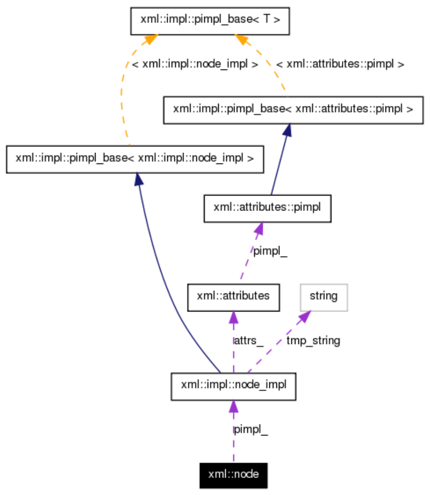
```
L4ci5  --> L5cd11 : 21
L4ci9-------------------------------------
L4ci9  --> L5cd11 : 21
L4ci9  --> L6cd4 : 21
L4ci9  --> L6rd5 : 26
```

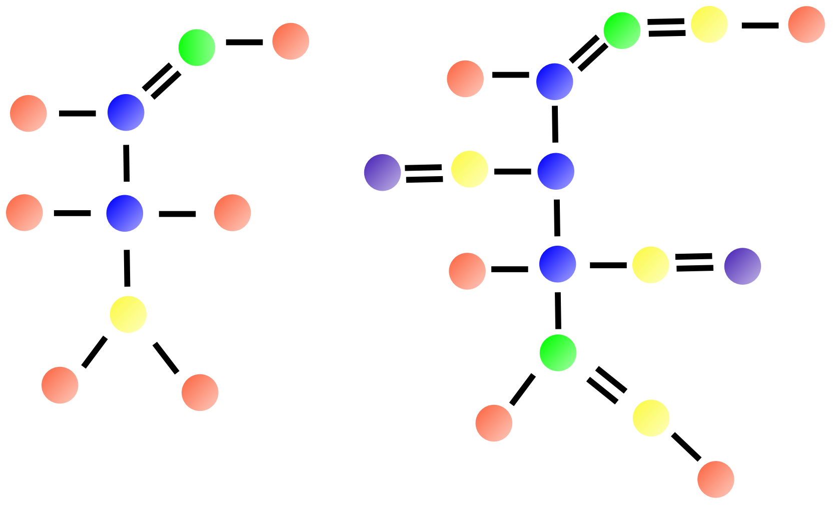
- There is a lot of information that has a relational structure
- Methods and models used for unstructured data are not expressive enough
- Sometimes structure can be flattened, but lots of interesting information is lost
  - Relational database ⇒ unique merged table
  - Attributes representing relations ⇒ inapplicable attributes
  - Graph data ⇒ strings based on graph traversal algorithms
  - Documents ⇒ bag of words

7.3 Graph mining






xml::impl::pimpl_base< T >
< xml::impl::node_impl >
< xml::attributes::pimpl >
xml::impl::pimpl_base< xml::attributes::pimpl >
xml::impl::pimpl_base< xml::impl::node_impl >
xml::attributes::pimpl
pimpl_
xml::attributes
string
attrs_
tmp_string
xml::impl::node_impl
pimpl_
xml::node



- All these types of data have in common that can be represented using graphs and trees
- Algorithms that use these data structures as input are needed
- Historically we can find different approaches to the discovery of patterns in graphs/trees:
  - Inductive logic programming: Structure is represented using logic formulas
  - Graph algorithms
    - Classic algorithms for detecting dense subgraphs (cliques)
    - Graph isomorphism algorithms
    - Graph partitioning algorithms
- Most of the problems used to discover structures in graphs are NP-Hard
  - Graph partitioning (Not for bi partitioning)
  - Graph isomorphism
- Other approaches for mining graphs
- Two different problems:
  - Mining large graphs (only one structure) ⇒ Partitioning, dense subgraphs
  - Mining sets of graphs ⇒ common substructures
- Some information can be described as a large graph (several instances connected by different types of relations)

  - For example: Social networks, Web pages,

- We are interested in discovering interesting substructures by:
  - Dividing the graph in subgraphs (k-way partitioning, node clustering)
  - Extracting dense substructures
- The simplest partitioning of a graph is to divide the graph in two subgraphs
- We assume that edges have values as labels (similarity, distance, ...)
- This problem is the *minimum cut-problem*:

  "Given a graph, divide the set of nodes in two groups so the cost of the edges connecting the nodes between the groups is minimum"
- This problem is related to the *maximum flow problem* that can be solved in polynomial time
- The general problem is NP-hard
- It can be solved approximately by local search algorithms (hill climbing)
- Kerninghan-Lin Algorithm:
  1. Start with a random cut of the graph
  2. Interchange a pair nodes from different partitions that reduces the cut
  3. Iterate until convergence
- There are different variations of this algorithm that changes the strategy for selecting the pair of nodes to interchange

Clustering algorithms can be adapted to obtain a graph partition

- K-means and K-medoids variations:
  - Nodes of the graphs as prototypes
  - Objective functions to define node membership to clusters (geodesic distance)
  - Network structure indices
- Spectral Clustering
  - Define the Laplacian matrix from the graph
  - Perform the eigendecomposition
  - The largest Eigenvalues determine the number of clusters
- Girvan-Newman Algorithm (Social Networks)
  - Is based on the concept of *edge betweenness centrality*:

$$B(e) = \frac{NumConstrainedPaths(e, i, j)}{NumShortPahts(i, j)}$$

  - *Random Walk Betweeness*: Compute how often a random walk starting on node i passes through node j
  - Detects bridges among dense components (edges that are not in the shortest paths between pairs of nodes)
  - Algorithm:
    1. Rank edges by $B(e)$
    2. Delete edge with the highest score
    3. Iterate until a specific criteria holds (eg. number of components)

- Not always a complete partitioning of the graph is necessary
- Dense subgraphs can represent interesting behaviors
- Different types of (pseudo)dense subgraphs:
  - Clique: All nodes are connected
  - Quasi-clique: Almost all nodes are connected (minimum density or minimum degree)
  - K-core: Every node connects to at least k nodes
  - K-plex: Each node is missing no more than k-1 edges
  - ...
- Different goals: Minimum size, all or the best ranked, overlapping or not

Algorithms:

- Exact enumeration: Exhaustive search (NP-Hard)
- Heuristic enumeration:
  - Shingles: Use of hash functions to identify vertices with common neighbours
  - GRASP Algorithm: Local randomized search (γ-cliques)
- Bounded approximation
- Some information can be described as a collection graph
  - For example: XML documents, chemical molecules
- We can use two approaches:
  - Cluster the graphs for common patterns and summarization
  - Find frequent substructures
- We look at a graph as a complex object
- We have to adapt the elements of clustering algorithms to this objects:
  - Distance measures to compare graphs
  - Summarization of graphs as prototypes
- This algorithm partitions a set of XML documents in K classes

- The summarization and similarity are based on frequent substructures
- For each partition the frequent substructures of size l are computed and used as prototype
- The number of substructures of the prototypes in a graph is used as similarity
- The more expensive part is the computation of the substructures (cost is reduced by linearizing the graphs)

Algorithm: XProj

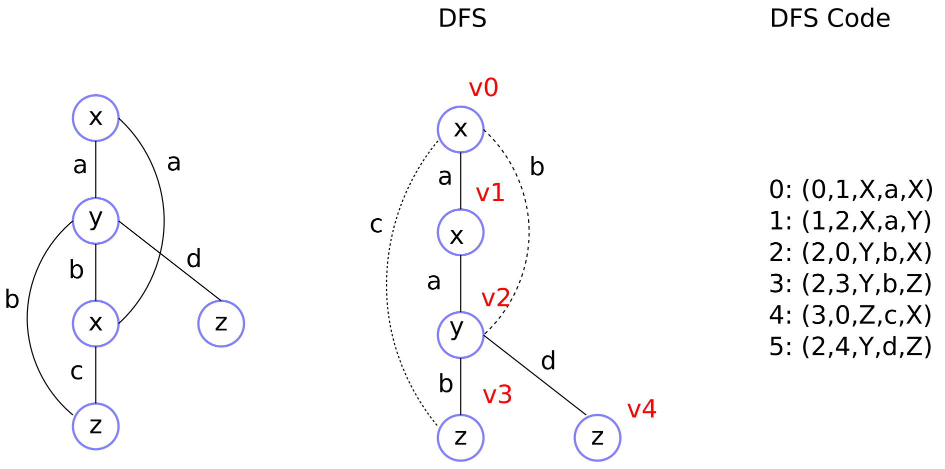
```
Partition the documents in K random subsets
Compute the prototypes (frequent substructures length $l$) $\mathcal{S}_k$
repeat
    Assign each document to the most similar prototype ($\mathcal{S}_k$)
    /* let $\mathcal{M}_1, \ldots \mathcal{M}_k$ be the new partitions */
    foreach $\mathcal{M}_i$ do
        Recompute the most frequent substructures of length $l$ of $\mathcal{M}_i$
        if frequency of substructures $> min_{sup}$ then
            Modify the prototype
        end
    end
until convergence criteria
```

- We look for frequent graphs/trees in a database
- Usually the structures are transformed to some kind of canonical representation (adjacency matrix, tree traversal)
- This representation gives a unique code for each different graph
- We can discover patterns in this code related to the patterns that appear in the original graphs
- Some of the approaches use the same properties used in association rules increasing the size of the patterns until no pattern is found

Yan, Han **gSpan: Graph-Based Substructure Pattern Mining** Proceedings of the IEEE International Conference on Data Mining 2002

- The graphs have labels in their edges and their vertices
- A canonical representation is used to reduce the cost to compute graph isomorphism
- This representation is based on the tree obtained by the depth first search of the graph and a lexicographical order among labels
- This representation transforms a graph into a string that contains the labels of the graph
- With this representation we can have an ordering over all the graphs that can be used to explore all possible subgraphs
- We assume an initial vertex (v_0) and an order among vertices ($i = 0 \ldots n$), many DFS traversals of a graph can obtained

- For a DFS traversal of a graph we define the forward edges (those with $i < j$, the DFS tree of the graph) and the backward edges (those with $i > j$)
- We define a linear order among edges, given $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$
  - if $i_i = i_2$ and $j_1 < j_2$ then $e_1 \prec_T e_2$; if $i_1 < j_1$ and $j_1 = i_2$ then $e_1 \prec_T e_2$; if $e_1 \prec_T e_2$ and $e_2 \prec_T e_3$ then $e_1 \prec_T e_3$
- Given a DFS tree for a graph and edge sequence can be defined based on $\prec_T$, this is the DFS code of the graph
- Given the DFS codes for a graph the linear order and a linear order for its labels $\prec_L$, a linear order among codes can be defined
- The *Minimum DFS code* is the **canonical label** for G



0: (0,1,X,a,X)
1: (1,2,X,a,Y)
2: (2,0,Y,b,X)
3: (2,3,Y,b,Z)
4: (3,0,Z,c,X)
5: (2,4,Y,d,Z)

Initialization (D, $MinSup$)

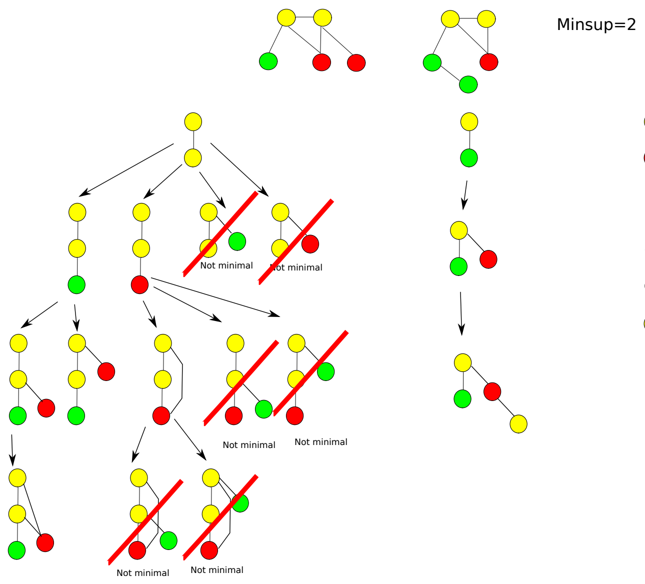
```
1. sort labels of the vertices and edges in $D$ by frequency
2. remove infrequent vertices and edges
3. $S_0$=code of all frequent graphs with single edge
4. sort $S_0$ in DFS lexicographic order
5. $S = S_0$
6. for each code $s$ in $S_0$
   (a) gSpan($s$,$D$,$MinSup$,$S$)
   (b) $D = D - s$
   (c) if $|D| < MinSup$ then return
```

gSpan(s,D,$MinSup$,S)

```
1. if $s! = mincode(s)$ then return
2. insert $s$ into $S$
3. $C = \emptyset$
4. scan $D$
```

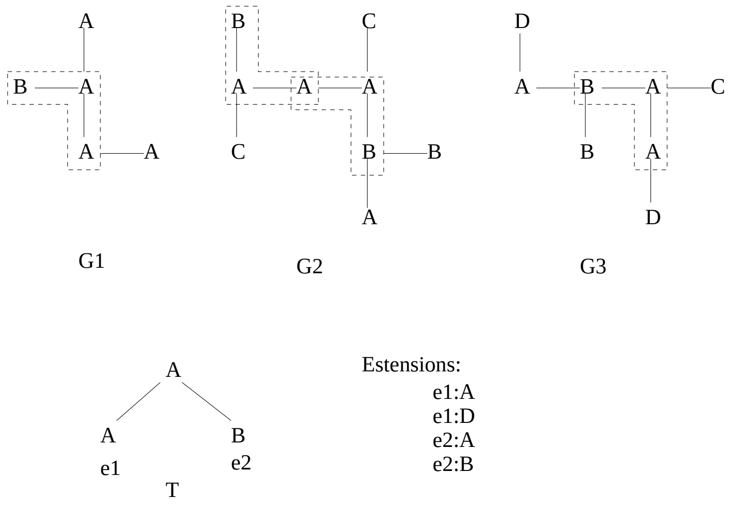
- find every edge e such that s can be right-most extended to frequent $s * e$
- insert $s * e$ into C;

5. sort C in DFS lexicographic order
6. for each $s * e$ in C do
   - gSpan($s * e$,D,$MinSup$,S)



Rückert, Kramer **Frequent Free Tree Discovery in Graph Data** Proceedings of the 2004 ACM symposium on Applied computing

- Mining of frequent free trees inside graph databases
- A free tree is an connected acyclic labeled graph (can be seen as a rootless tree)
- A canonical representation for a free tree is calculated, using as tree root the most centered node and establishing an order for its labels
- A level traversal of the tree defines the representation of a graph
- This representation reduces the problem of graph isomorphism
- The mining is performed level wise starting with a free tree with a node
- Each step the candidate trees are extended with a node



Bibliography

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, volume 27,2 of *ACM SIGMOD Record*, pages 94–105, New York, June 1–4 1998. ACM Press.

[2] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMod-99)*, volume 28,2 of *SIGMOD Record*, pages 49–60, New York, June 1–3 1999. ACM Press.

[3] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesus M. Perez, and Iñigo Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243 – 256, 2013.

[4] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 1027–1035. SIAM, 2007.

[5] Adil M. Bagirov, Julien Ugon, and Dean Webb. Fast modified global k-means algorithm for incremental cluster construction. *Pattern Recognition*, 44(4):866 – 876, 2011.

[6] Asa Ben-Hur, David Horn, Hava T. Siegelmann, and Vladimir Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

[7] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM, 2001.

[8] Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors, *KDD*, pages 9–15. AAAI Press, 1998.

[9] Christopher JC Burges. Dimension reduction: A guided tour. *Foundations and Trends in Machine Learning*, 2(4):275–365, 2009.

[10] F. Camastra and A. Verri. A novel kernel method for clustering. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(5):801–804, May 2005.

[11] Feng Chen and Daniel B. Neill. Human rights event detection from heterogeneous social media graphs. *Big Data*, 3(1):34–40, 2015.

[12] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random structures and algorithms*, 22(1):60–65, 2003.

[13] M. Dash, K. Choi, P. Scheuermann, and H. Liu. Feature Selection for Clustering - A Filter Solution. In *ICDM*, pages 115–122, 2002.

[14] Michel M Deza and Elena Deza. *Encyclopedia of Distances*. Springer, 2013.

[15] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *KDD*, pages 551–556. ACM, 2004.

[16] David L Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[17] R. Dubes and A Jain. *Algorithms for Clustering Data*. PHI Series in Computer Science. Prentice Hall, 1988.

[18] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*, volume November. John Wily & Sons, Inc., New York, second edition, 2000.

[19] Charles Elkan. Using the triangle inequality to accelerate k-means. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 147–153. AAAI Press, 2003.

[20] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jia Wei Han, and Usama Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, page 226. AAAI Press, 1996.

[21] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine learning*, 2(2):139–172, 1987.

[22] Frey and Dueck. Clustering by passing messages between data points. *SCIENCE: Science*, 315, 2007.

[23] John H Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. *Artificial intelligence*, 40(1):11–61, 1989.

[24] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *Knowledge and Data Engineering, IEEE Transactions on*, 15(3):515 – 528, may-june 2003.

[25] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An efficient clustering algorithm for large databases. *Inf. Syst*, 26(1):35–58, 2001.

[26] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, July 2001.

[27] X. He, D. Cai, and P. Niyogi. Laplacian Score for Feature Selection. In *NIPS*, 2005.

[28] Alexander Hinneburg and Daniel A Keim. An efficient approach to clustering in large multimedia databases with noise. In *KDD*, volume 98, pages 58–65, 1998.

[29] Alexander Hinneburg, Daniel A Keim, et al. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *VLDB*, volume 99, pages 506–517. Citeseer, 1999.

[30] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004. 10.1023/B:AIRE.0000045502.10941.a9.

[31] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881 –892, jul 2002.

[32] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[33] R. Kohavi. Wrappers for Feature Subset Selection. *Art. Intel.*, 97:273–324, 1997.

[34] Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296. ACM, 2006.

[35] A. C. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, February 2003.

[36] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[37] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. *KDD*, pages ?–?, 2000.

[38] Bidyut Kr. Patra, Sukumar Nandi, and P. Viswanath. A distance based clustering method for arbitrary shaped clusters in large datasets. *Pattern Recognition*, 44(12):2862–2870, 2011.

[39] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[40] B. Scholkopf, A. Smola, and K. R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[41] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178. ACM, 2010.

[42] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[43] P. Viswanath and V. Suresh Babu. Rough-dbscan: A fast hybrid density based clustering method for large data sets. *Pattern Recognition Letters*, 30(16):1477 – 1488, 2009.

[44] Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A statistical information grid approach to spatial data mining. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Twenty-Third International Conference on Very Large Data Bases*, pages 186–195, Athens, Greece, 1997. Morgan Kaufmann.

[45] Y. Wang and Y. Zhang. Non-negative matrix factorization: a comprehensive review. *Knowledge and Data Engineering, IEEE Transactions on*, 25(6):1336–1353, 2013.

[46] L. Wolf and A. Shashua. Feature Selection for Unsupervised and Supervised Inference. *Journal of Machine Learning Research*, 6:1855–1887, 2005.

[47] Zhiwen Yu and Hau-San Wong. Quantization-based clustering algorithm. *Pattern Recognition*, 43(8):2698 – 2711, 2010.

[48] H. Zeng and Yiu ming Cheung. A new feature selection method for Gaussian mixture clustering. *Pattern Recognition*, 42(2):243–250, February 2009.

[49] Kai Zhang, Ivor W. Tsang, and James T. Kwok. Maximum margin clustering made practical. *IEEE Transactions on Neural Networks*, 20(4):583–596, 2009.

[50] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Min. Knowl. Discov*, 1(2):141–182, 1997.

[51] Zhen-yue Zhang and Hong-yuan Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *Journal of Shanghai University (English Edition)*, 8(4):406–424, 2004.