Unsupervised Machine Learning and Data Mining

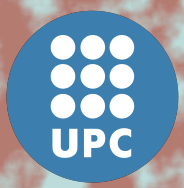
(Course Slides)

Javier Béjar

AMLT

Master in Artificial Intelligence

Course 2016/2017 Fall Semester



Contents

1 Introduction to Knowledge Discovery 1

2 Data Preprocessing 19

3 Data Clustering 57

4 Cluster Validation 113

5 Clustering of Large Datasets 137

6 Semisupervised Clustering 155

7 Association Rules 169

8 Mining Structured Data 197

Preface

These are the slides for the first half of the course *Advanced Machine Learning Techniques* from the master on Artificial Intelligence of the Barcelona Computer Science School (Facultat d'Informàtica de Barcelona), Technical University of Catalonia (UPC, BarcelonaTech).

This slides are used in class to present the topics of the course and have been prepared using the papers and book references that you can find in the course website http://www.cs.upc.edu/~bejar/amlt/amlt.html).

This document is a complement so you can prepare the classes and use it as a reference, but it is not a substitute for the classes or the material from the webpage of the course.

Javier Béjar
Barcelona, 2016

1

Introduction to Knowledge Discovery

Knowledge Discovery in Databases

Javier Béjar

CS - MIA

AMLT - 2016/2017

Outline

1. Knowledge Discovery in Databases
   - Introduction
   - Definitions of KDD
2. The KDD process
   - Steps of KDD
   - Discovery goals
   - Mining Methodologies
3. Applications
4. Tools
5. Challenges

1 Knowledge Discovery in Databases
2 The KDD process
3 Applications
4 Tools
5 Challenges

Knowledge Discovery in Databases

- Practical application of the methodologies from machine learning/statistics to large amounts of data
- The main problem addressed is the impossible task of manually analyzing (make sense of) all the data we are systematically collecting
- These methodologies are useful for automating/helping the process of analysis/discovery
- The final goal is to extract (semi)automatically actionable/useful knowledge

"We are drowning in information and starving for knowledge"

Knowledge Discovery in Databases

- The high point of KDD starts around late 1990s
- Many companies show their interest in obtaining the (possibly) valuable information stored in their databases (purchase transactions, e-commerce, web data, ...)
- The goal is to obtain information that can lead to better commercial strategies and practices from a better understanding of the consumers preferences and their behaviour
- Many companies are putting a lot of effort on the development/use of this kind of technology (analysis and tools)
- Several buzz words have appeared: Business Intelligence, Business Analytics, Predictive Analytics, *Data Science*, Big Data ...

Knowledge Discovery in Databases

- Not only business data are in need of these kinds of techniques
- Analyzing scientific data has supposed an important impulse
  - Space probes
  - Remote sensors on satellites
  - Astronomical observations (big array observatories)
  - Large scientific experiments (LHC, ITER)
  - Genome Project, microarray data $\Rightarrow$ Bioinformatics
  - Neuroscience (Human Brain Project)
- Data grows faster that the ability to analyze it

KDD: Machine learning

- **Inductive machine learning**: Discovery of patterns/models from data
- Supervised discovery/Unsupervised discovery
- Unstructured/Structured representations
- Logic representations/Probabilistical representations
- Scalability

KDD: Statistics/Data Analysis

- **Statistical Data Modeling**: Fitting of probability models to data
- Supervised/Unsupervised modeling
- Structured models
- Probabilistic representation/interpretation of data
- Scalability
- Statistical Machine Learning

KDD: Databases/Algorithmics/Visualization

- Data access:
  - SQL vs NoSQL
  - Distributed file systems
  - Redundancy/Fault Tolerance/Parallelism
- Databases for structured data: Transactions, Graphs, Time sequences
- Distributed processing paradigms/scalabilility: MapReduce, Hadoop, Spark, ..
- Data visualization: from data cubes to structured data representation

KDD definitions

"It is the search of valuable information in great volumes of data"

"It is the explorations and analysis, by automatic or semiautomatic tools, of great volumes of data in order to discover patterns and rules"

"It is the **nontrivial** *process of identifying* **valid**, **novel**, *potentially* **useful**, *and ultimately* **understandable patterns** *in data"*

Elements of KDD

Pattern: Any representation formalism capable to describe the common characteristics of a group if instances

Valid: A pattern is valid if it is able to predict the behaviour of new information with a degree of *certainty*

Novelty: It is novel any knowledge that it is not know respect the domain knowledge and any previous discovered knowledge

Useful: New knowledge is useful if it allows to perform actions that yield some benefit given a established criteria

Understandable: The knowledge discovered must be analyzed by an expert in the domain, in consequence the interpretability of the result is important

1 Knowledge Discovery in Databases

2 The KDD process

3 Applications

4 Tools

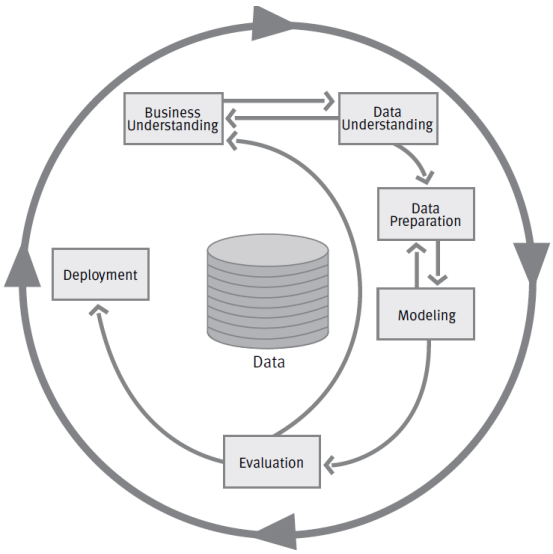
5 Challenges

KDD as a process

- The actual discovery of patterns is only one part of a more complex process
- Raw data in not always ready for processing (80/20 project effort)
- Some general methodologies have been defined for the whole process (CRISP-DM or SEMMA)
- These methodologies address KDD as an engineering process, despite being business oriented are general enough to be applied on any data discovery domain

The KDD process (I)

- Steps of the Knowledge Discovery in DB process
  1. Domain study
  2. Creating the dataset
  3. Data preprocessing
  4. Dimensionality reduction
  5. Selection of the discovery goal
  6. Selection of the adequate methodologies
  7. Data Mining
  8. Result assessment and interpretation
  9. Using the knowledge



The KDD process (II)

1. Study of the domain
   Gather information about the domain. Characteristics, goal of the discovering process (attributes, representative examples, types of patterns, sources of data)

2. Creating the dataset
   From the information of the previous step it is decided what source of data will be used. It has to be decided what attributes will describe the data and what examples are needed for the goals of the discovery process

The KDD process (III)

3. Data preprocessing and cleaning
   It has to be studied the circumstances that affect the quality of the data
   - Outliers
   - Noise (does it exists?, does it present any pattern?, can it be reduced?)
   - Missing values
   - Discretization of continuous values

The KDD process (V)

4. Data reduction and projection
   We have to study what attributes are relevant to our goal (depending on the task some techniques can be used to measure the relevance of the attributes) and the number of examples that are needed. Not all the data mining algorithms are scalable
   - Instance selection (do we need all the examples? sampling techniques)
   - Attribute selection (what is really relevant?)

The KDD process (VI) - Attribute selection

- It is very important to use methods for attribute selection:
  - Reduces the dimensionality of the data (curse of dimensionality)
  - Eliminates/Reduces irrelevant and redundant information
  - The result of the process is easier to interpret
- Attribute selection techniques:
  - Mathematical/Statistical techniques: Principal component analysis (PCA), projection pursuit, Multidimensional scaling
  - Heuristics for attribute relevance evaluation (ranking of attributes, search in the space of subsets of attributes)

The KDD process (VII)

5. Selecting the discovery goal
   The characteristics of the data, the domain and the aim of the project determines what kind of analysis are feasible or possible (group partitioning, summarization, classification, discovery of attribute relations, ...)

6. Selecting the adequate methodologies
   The goal and the characteristics of the data determines the more adequate methodologies

The KDD process (VIII)

7. Applying the methodologies (Data Mining)
   The different parameters of the chosen methodologies has to be adjusted by experimentation and analysis in order to obtain the best possible results

8. Interpreting the results
   From the knowledge of the domain (expert) it will be assessed the relevance and importance of the result. This interpretation step could suppose feedback for the previous steps, it is possible that some adjustments are needed or some previous decisions have to be changed

9. Incorporating the new knowledge
   The new knowledge is used to perform the intended task goal of the discovery process

Goals of the KDD process

There are different goals that can be pursued as the result of the discovery process, among them:

Classification: We need models that allow to discriminate instances that belong to a previously known set of groups (the model could or could not be interpretable)

Clustering/Partitioning/Segmentation: We need to discover models that clusters the data into groups with common characteristics (a characterizations of the groups is desirable)

Regression: We look for models that predicts the behaviour of continuous variables as a function of others

Goals of the KDD process

Summarization: We look for a compact description that summarizes the characteristics of the data

Causal dependence: We need models that reveal the causal dependence among the variables and assess the strength of this dependence

Structure dependence: We need models that reveal patterns among the relations that describe the structure of the data

Change: We need models that discover patterns in data that has temporal or spatial dependence

Methodologies for KDD

There are a lot of methodologies that can be applied in the discovery process, the more usual are:

- **Decision trees, decision rules:**
  - Usually are interpretable models
  - Can be used for: Classification, regression, and summarization
  - trees: C4.5, CART, QUEST, rules: RIPPER, CN2, ..
- **Classifiers, Regression:**
  - Low interpretability but good accuracy
  - Can be used for: Classification and regression
  - Statistical regression, function approximation, Neural networks, Support Vector Machines, k-NN, Local Weighted Regression, ...

Methodologies for KDD

- **Clustering:**
  - Its goal is to partition datasets or discover groups
  - Can be used for: Clustering, summarization
  - Statistical Clustering, Unsupervised Machine learning, Unsupervised Neural networks (Self-Organizing Maps)
- **Dependency models (attribute dependence, temporal dependence, graph substructures)**
  - Its goal is to obtain models (some interpretables) of the dependence relations (structural, causal temporal) among attributes/instances
  - Can be used for: causal dependence discovery, temporal change, substructure discovery
  - Bayesian networks, association rules, Markov models, graph algorithms, ...

1 Knowledge Discovery in Databases

2 The KDD process

3 Applications

4 Tools

5 Challenges

Applications

- Business:
  - Costumer segmentation, costumer profiling, costumer transaction data, customer churn
  - Fraud detection
  - Control/analysis of industrial processes
  - e-commerce, on-line recommendation
  - Financial data (stock market analysis)
- WEB mining
  - Text mining, document search/organization
  - Social networks analysis
  - User behavior

Applications

- Scientific applications:
  - Medicine (patient data, MRI scans, ECG, EEG, ...)
  - Pharmacology (Drug discovery, screening, in-silicon testing)
  - Astronomy (astronomical bodies identification)
  - Genetics (gen identification, DNA microarrays, bioinformatics)
  - Satellite/Probe data (meteorology, astronomy, geological, ...)
  - Large scientific experiments (CERN LHC, ITER)

1 Knowledge Discovery in Databases
2 The KDD process
3 Applications
4 Tools
5 Challenges

Tools for KDD

- There are a lot of tools available for KDD
- Some tools were developed at universities (C5.0, CART/MARS) and have become a commercial product, others still remain open source (Weka, R, scikit-learn)
- Big fish eats little fish (C5.0 $\rightarrow$ Clementine $\rightarrow$ SPSS-clementine $\rightarrow$ IBM DBMiner)
- Data analysis software companies incorporate KDD techniques inside classical data analysis tools (SPSS, SAS)
- Companies selling databases add KDD tools as an added value (IBM DB2 (intelligent Miner), SQL Server, Oracle)
- Machine Learning as a Service (Amazon, Microsoft, Google, IBM Watson, Big ML, ...)

Tools for the course

- Python
  - General programming language, easy to learn
  - numpy, scipy, pandas
  - scikit-learn (`http://scikit-learn.org`)
  - Data preprocessing, Clustering Algorithms, Association Rules, ...
- R (`http://cran.r-project.org/`)
  - Statistic analysis oriented language, more steep learning curve
  - Many packages
  - Data preprocessing, Clustering Algorithms, Association Rules, ...

1 Knowledge Discovery in Databases

2 The KDD process

3 Applications

4 Tools

5 Challenges

Open problems

- Scalability (More data, more attributes)
- Overfitting (Patterns with low interest)
- Statistical significance of the results
- Methods for temporal data/relational data/structured data
- Methods for data cleaning (Missing data and noise)
- Pattern comprehensibility
- Use of domain knowledge
- Integration with other techniques (OLAP, DataWarehousing, Business Intelligence, Intelligent Decision Support Systems)
- Privacy

2

Data Preprocessing

Data Preprocessing

Javier Béjar

CS - MAI

AMLT - 2016/2017

Outline

1. Introduction
   - Data Representation
2. Data Preprocessing
   - Outliers
   - Missing Values
   - Normalization
   - Discretization
3. Dimensionality Reduction
   - The curse of dimensionality
   - Component Analysis
   - Multidimensional Scaling
   - Application: Wheelchair control
   - Attribute Selection
4. Similarity functions

1 Introduction

2 Data Preprocessing

3 Dimensionality Reduction

4 Similarity functions

Data representation

- Unstructured datasets:
  - Examples described by a flat set of attributes: attribute-value matrix
- Structured datasets:
  - Individual examples described by attributes but also having relations among them: sequences (time, spatial, ...), trees, graphs
  - Sets of structured examples (sequences, graphs, trees)

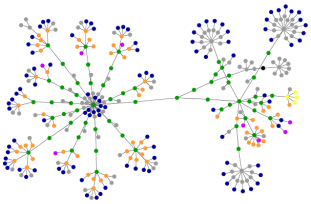
Unstructured data

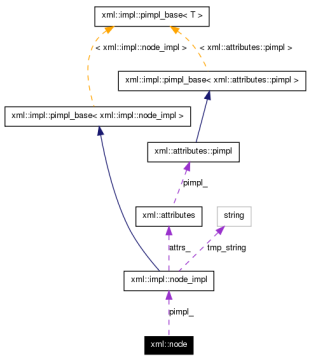
- Only one table of observations
- Each example represents an instance of the problem
- Each instance is represented by a set of attributes (discrete, continuous)

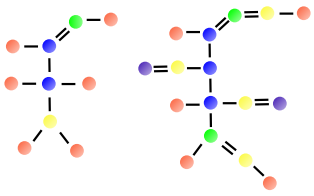
A	B	C	···
1	3.1	a	···
1	5.7	b	···
0	-2.2	b	···
1	-9.0	c	···
0	0.3	d	···
1	2.1	a	···
⋮	⋮	⋮	⋱

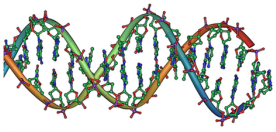
Structured data

- One sequential relation among instances (Time, Strings)
  - Several instances with internal structure (eg: sequences of events)
  - Subsequences of unstructured instances (eg: sequences of complex transactions)
  - One big instance (eg: time series)
- Several relations among instances (graphs, trees)
  - Several instances with internal structure (eg: XML documents)
  - One big instance (eg: social network)









Data Streams

- Endless sequence of data (eg: sensor data)
  - Several streams synchronized
  - Unstructured instances
  - Structured instances
- Static/Dynamic model

Data representation

- Most of unsupervised learning algorithms are specifically fitted for unstructured data
- The data representation used is equivalent to a database table (attribute-value pairs)
- More specialized algorithms have been developed to process structured data: Graph clustering, Sequence mining, Frequent substructures
- The representation of these types of data is sometimes algorithm dependent

1 Introduction

2 Data Preprocessing

3 Dimensionality Reduction

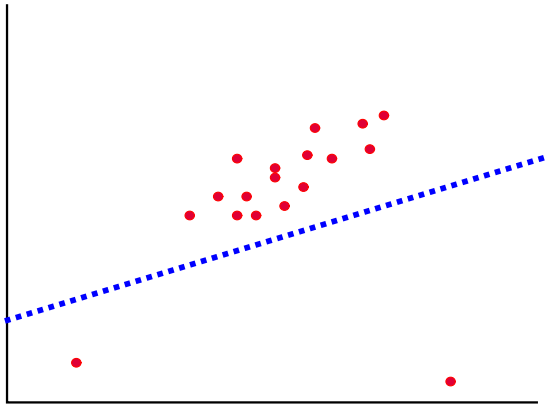
4 Similarity functions

Data preprocessing

- Usually raw data is not directly adequate for analysis
- The usual reasons:
  - The quality of the data (noise/missing values/outliers)
  - The dimensionality of the data (too many attributes/too many examples)
- The first step of any data analysis task is to assess the quality of the data
- The techniques used for data preprocessing are usually oriented to unstructured data

Outliers

- Outliers are examples that have values very different from the rest of the dataset
- They can be considered as examples with erroneous values
- It could have an important impact on the results of some algorithms



Outliers

- The different values can happen in all or only a few of attributes
- The usual way to correct this circumstance is to eliminate the examples
- If the exceptional values are only in a few attributes these could be treated as missing values

Outliers Detection

Parametric methods

- Assuming a probabilistic distribution for the attributes (eg: Univariate Gaussian, Gaussian Mixture Model), test for the probability of the examples, those with low probability can be labeled as outliers
- Using the upper and lower quantiles of the distribution as decision criteria, discard the observations that are at a specific distance above or below the quantiles

Outliers Detection

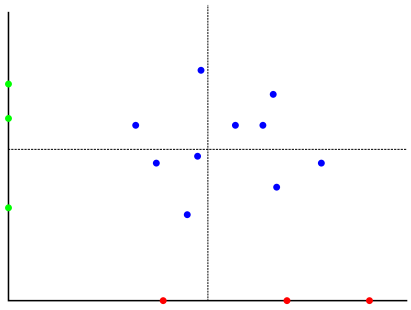
Non parametric methods

- Generate a histogram of the attributes (binning) and discard the examples that are in bins that have a low number of examples
- Perform kernel density estimation (KDE) and label as outliers examples that belong to low density areas
- Using a proximity criteria, test for abnormalities in the distribution of the distances to the k-nearest neighbors

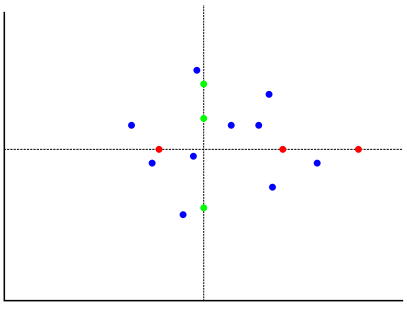
Missing values

- Missing values appear because of errors or omissions during the gathering of the data
- They can be substituted to increase the quality of the dataset (value imputation)
  - Global constant for all the values
  - Mean or mode of the attribute (global central tendency)
  - Mean or mode of the attribute but only of the k nearest examples (local central tendency)
  - Learn a model for the data (regression, bayesian) and use it to predict the values
- Problem: we are modifying the statistical distribution of the data)

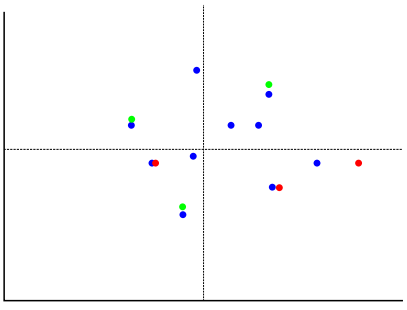
Missing values



Missing Values



Mean substitution



1-neighbor substitution

Normalization

Normalizations are applied to quantitative attributes in order to eliminate the effect of having different scale measures

- **Range normalization:** Transform all the values of the attribute to a preestablished scale (eg: [0,1], [-1,1])

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

- **Distribution normalization:** We transform the data in order to obtain certain specific statistical distribution with preestablished parameters (Usually a Gaussian distribution with mean 0 and standard deviation 1)

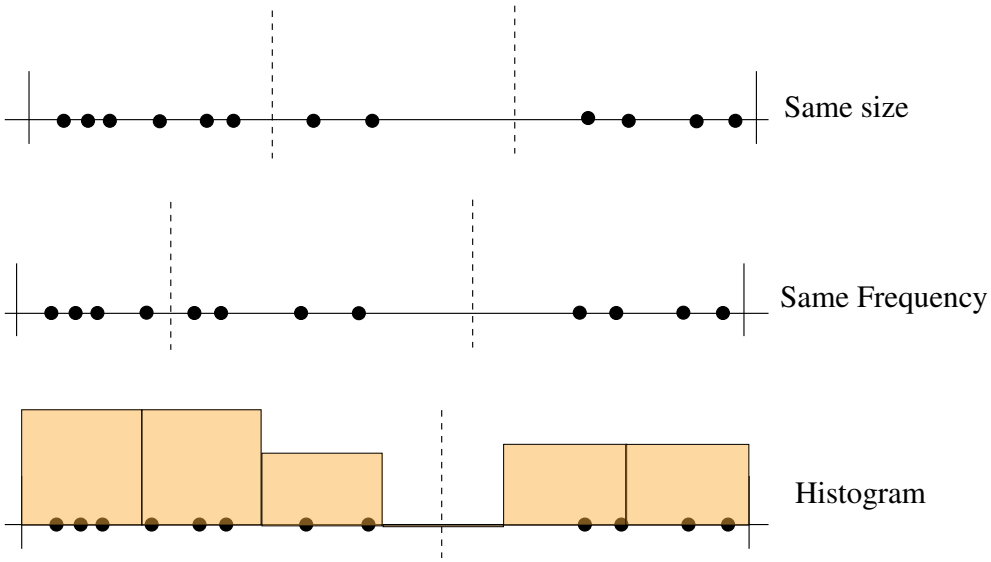
$$\frac{x - \mu_x}{\sigma_x}$$

Discretization

Discretization allows to transform quantitative attributes to qualitative attributes

- **Equal size bins:** We choose the number of values that we need and divide the interval in equal bins
- **Equal frequency bins:** We choose the number of values that we need, but we create the intervals so each bean has the same number of examples (the size of the intervals will be different)
- **Distribution approximation:** We calculate a histogram of the data and we fit a function. The intervals are where the function has its minima
- **Other techniques:** We can apply entropy based measures, MDL or even clustering

Discretization



Python Notebooks

These two Python Notebooks show some examples of the effect of missing values imputation and data discretization and normalization

- Missing Values Notebook (click here to go to the url)
- Preprocessing Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

1 Introduction

2 Data Preprocessing

3 Dimensionality Reduction

4 Similarity functions

The curse of dimensionality

- There are two problems that come from the dimensionality of a dataset
  - The computational cost of processing the data (scalability of the algorithms)
  - The quality of the data (more probability of bad data)
- There are two elements that define the dimensionality of a dataset
  - The number of examples
  - The number of attributes
- Usually the problem of having too many examples can be solved using sampling.
- Attribute reduction has different solutions

Reducing attributes

- Usually the number of attributes of the dataset has an impact on the performance of the algorithms:
  - Because their poor scalability (cost is a function of the number of attributes)
  - Because the inability to cope with irrelevant/noisy/redundant attributes
- There are two main methodologies to reduce the number of attributes of a dataset
  - Transforming the data to a space of less dimensions preserving somewhat the original data (dimensionality reduction)
  - Eliminating the attributes that are not relevant for the goal task (feature subset selection)

Dimensionality reduction

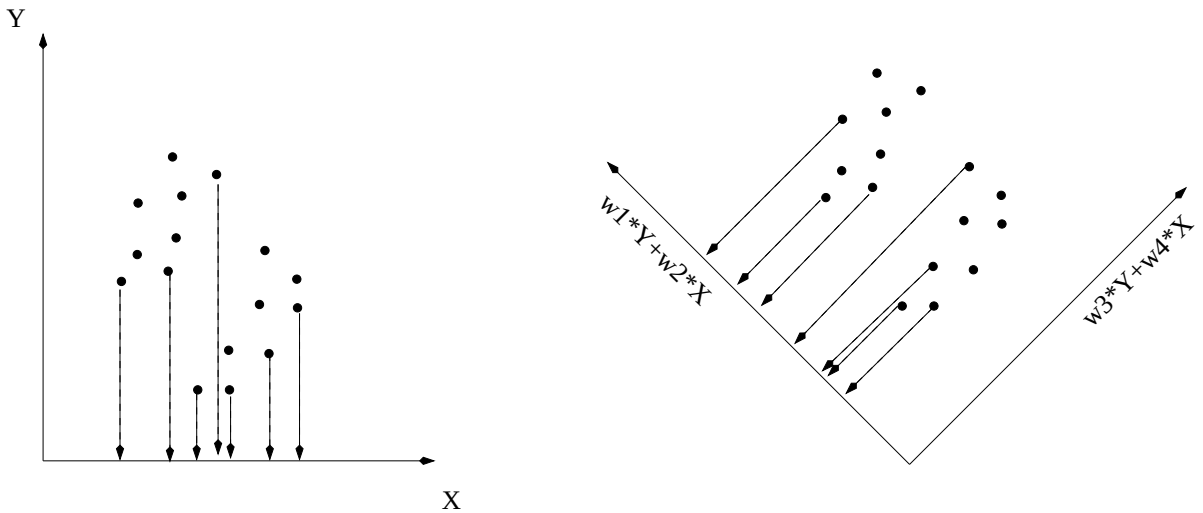
- We are looking for a new dataset that preserves the information of the original dataset but has less attributes
- Many techniques have been developed for this purpose
  - Projection to a space that preserve the statistical model of the data (PCA, ICA)
  - Projection to a space that preserves distances among the data (Multidimensional scaling, random projection, nonlinear scaling)

Component analysis

- **Principal Component Analysis:** Data is projected on a set of orthogonal dimensions (components) that are linear combination of the original attributes. The components are uncorrelated and are ordered by the information they have. We assume gaussian distribution. Global variance is preserved.
- **Independent Component Analysis:** Transforms the dataset projecting the data to a set of variables statistically independent (all statistical momentums are independent). We assume non gaussian data.

Principal Component Analysis

The model that we want is a projection matrix where the dimensions are orthogonal (linearly independent) and preserve data variance



Principal Component Analysis

- Principal components are an ordered set of vectors that are 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:

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

Principal Component Analysis

- Optimizing partially for μ and λ_i:

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

- We can obtain the matrix V_q by minimizing:

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

Principal Component Analysis

- 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

Kernel PCA

- PCA is a linear transformation, this means that if data is linearly separable, the reduced dataset will be 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, ...)

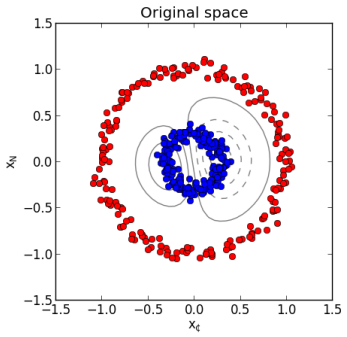
Kernel PCA

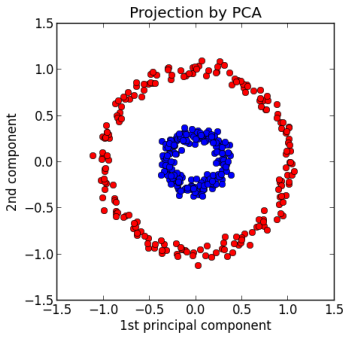
- The computation of the components is equivalent to PCA but performing the eigen decomposition of the covariance matrix computed for the transformed examples

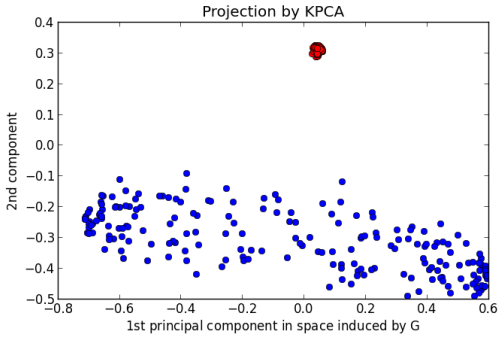
$$C = \frac{1}{M}\sum_{j=1}^{M}\Phi(x_j)\Phi(x_j)^T$$

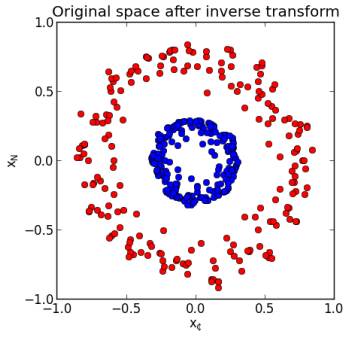
- Pro: Helps to discover patterns that are non linearly separable in the original space
- Con: Does not give a weight/importance for the new components

Kernel PCA









Sparse PCA

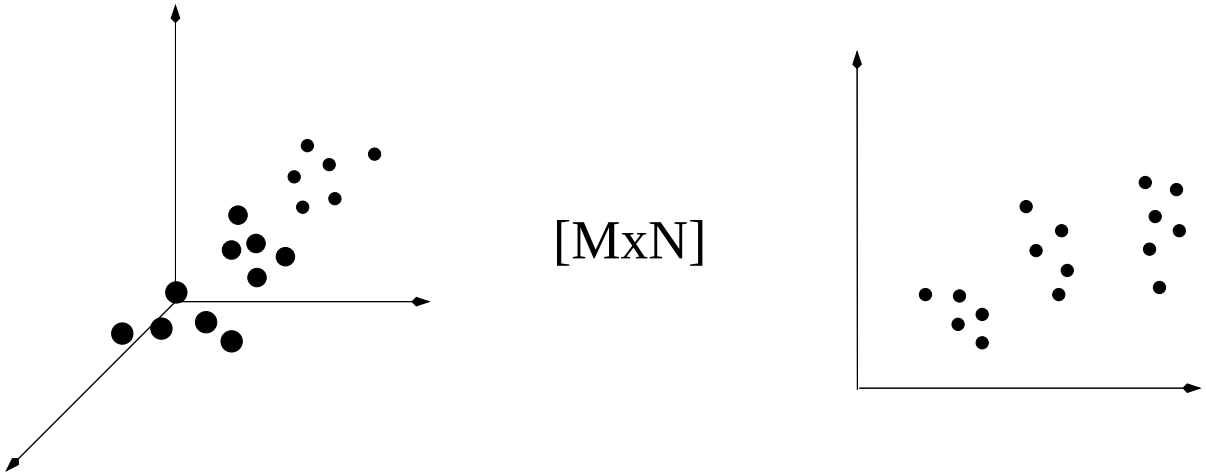
- PCA transforms data to a space of the same dimensionality (all eigenvalues are non zero)
- An alternative to PCA is to solve the minimization problem posed by the reconstruction error using regularization
- A penalization term is added to the objective function proportional to the norm of the eigenvalues matrix

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

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

Multidimensional Scaling

A transformation matrix transforms a dataset from M dimensions to N dimensions preserving pairwise data distances



Multidimensional Scaling

- **Multidimensional Scaling:** Projects a dataset to a space with less dimensions preserving the pair distances among the data
- A projection matrix is obtained by optimizing a function of the pairwise distances (stress function)
- This means that the actual attributes are not used in the transformation
- There are different objective functions that can be used (least squares, Sammong mapping, classical scaling, ...).
- The optimization problem is solved by gradient descent

Multidimensional Scaling

- Least Squares Multidimensional Scaling (MDS)
- The distorsion is defined as 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 problem is defined as:

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

Multidimensional Scaling

- Several optimization strategies can be used to solve this problem
- If the distance matrix is euclidean it can be solved using 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:

$$Z^{(k+1)} = Z^{(k)} - \alpha \nabla S_D(Z)$$

Multidimensional Scaling - Other functions

- Sammong Mapping (emphasis on smaller distances)

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

- Classical Scaling (similarity instead of distance)

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

- Non metric MDS (assumes a ranking among the distances, non euclidean space)

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

Random Projection

- A random transformation matrix is generated:
  - Rectangular matrix $N \times d$
  - Columns must have unit length
  - Elements are generated from a gaussian distribution (not strictly necessary)
- A matrix generated this way is almost orthogonal (close to a PCA transformation)
- The projection will preserve the relative distances among examples
- The effectiveness usually depends on the number of dimensions (this can be estimated from the number of examples)

Nonnegative Matrix Factorization (NMF)

- NMF performs an approximation of a matrix in the product of two matrices

$$V = WH$$

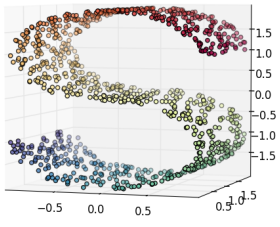
- Similar to PCA, the eigen decomposition transforms the original data matrix in the product of two matrices
- The main difference is that the values of the matrices are constrained to be positive
- The formulation assumes that the data is a sum of unknown positive latent variables
- The positiveness assumption helps to interpret the result
  - Eg.: In text mining, a document is an aggregation of topics
- There are many variants and algorithms

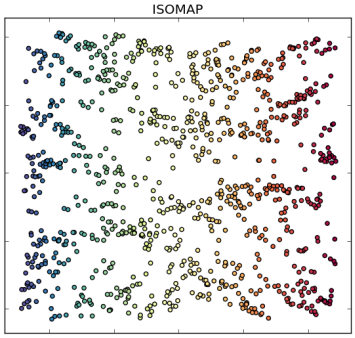
Nonlinear scaling

- The previous methods perform a linear transformation between the original space and the final space
- For some datasets this kind of transformation is not enough to maintain the information of the original data
- Nonlinear transformations methods:
  - ISOMAP
  - Local Linear Embedding
  - Local MDS

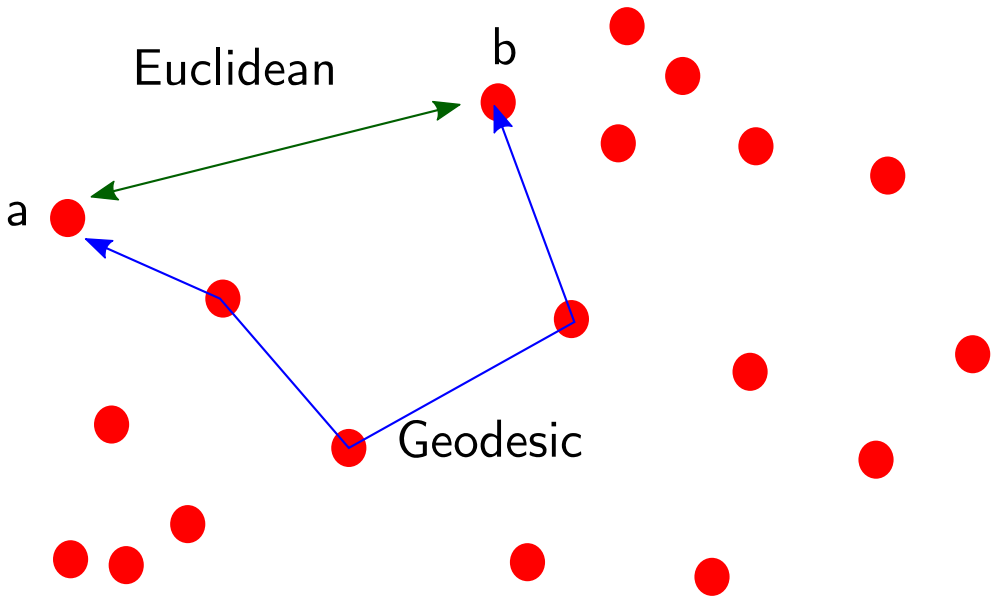
ISOMAP

- Assumes a low dimensional dataset embedded in a larger number of dimensions
- The geodesic distance is used instead of the euclidean distance
- It is assumed that the relation of an instance with its immediate neighbors is more representative of the structure of the data
- The transformation generates a new space that preserves neighborhood relationships





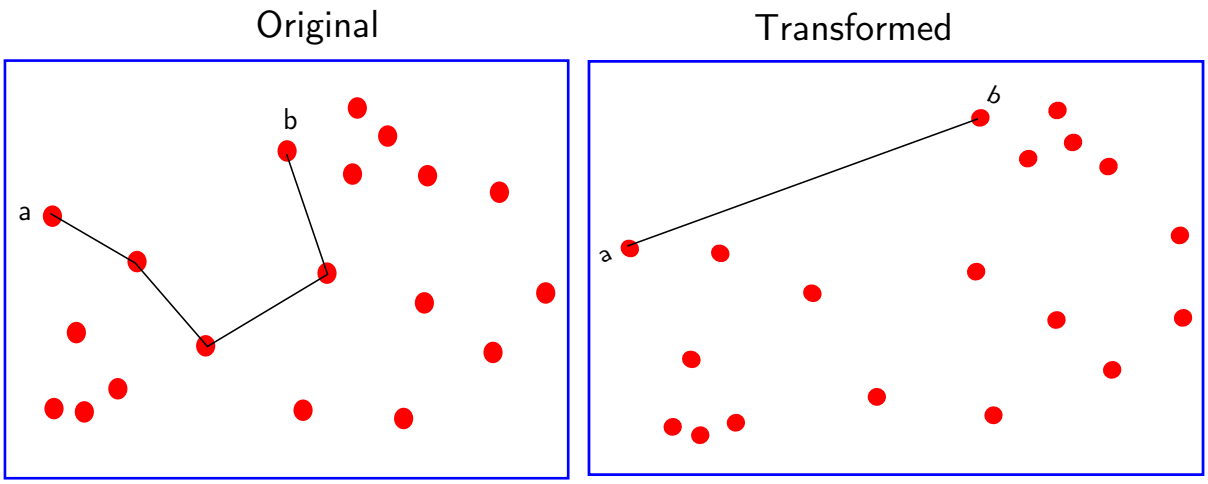
ISOMAP



ISOMAP

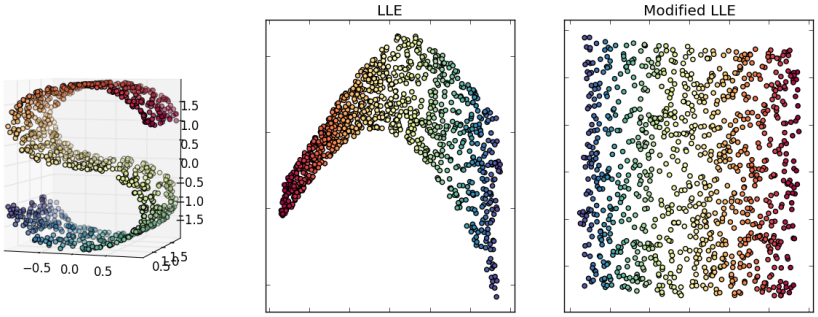
- Algorithm:
  1. For each data point find its k closest neighbors (points at minimal euclidean distance)
  2. Build a graph where each point has an edge to its closest neighbors
  3. Approximate the geodesic distance for each pair of points by the shortest path in the graph
  4. Apply a MDS algorithm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

ISOMAP Example



Local linear embedding

- Performs a transformation that preserves local structure
- Assumes that each instance can be reconstructed by a linear combination of its neighbors (weights)
- From this weights a new set of data points that preserve the reconstruction is computed for a lower set of dimensions
- Different variants of the algorithm exist



Local linear embedding

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

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

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

- Performs a transformation that preserves locality of closer points and puts farther away non neighbor points
- Given a set of pairs of points $\mathcal{N}$ where a pair (i, i') belong to the set if i is among the K neighbors of i' or viceversa
- Minimize the function:

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

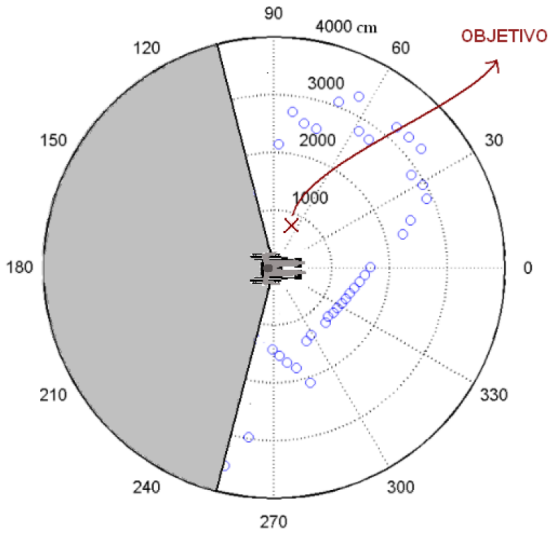
- The parameters τ controls how much the non neighbors are scattered

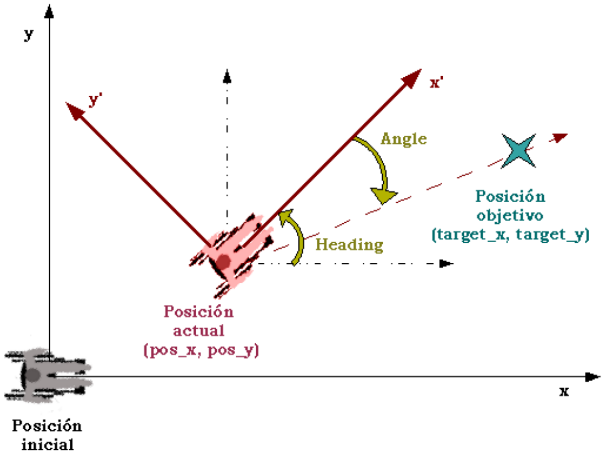
Wheel chair control characterization

- Wheelchair with shared control (patient/computer)
- Recorded trajectories of several patients in different situations
  - Angle/distance to the goal, Angle/distance to the nearest obstacle from around the chair (210 degrees)
- Characterization about how the computer helps the patients with different handicaps
- Is there any structure in the trajectory data?

Wheel chair control characterization

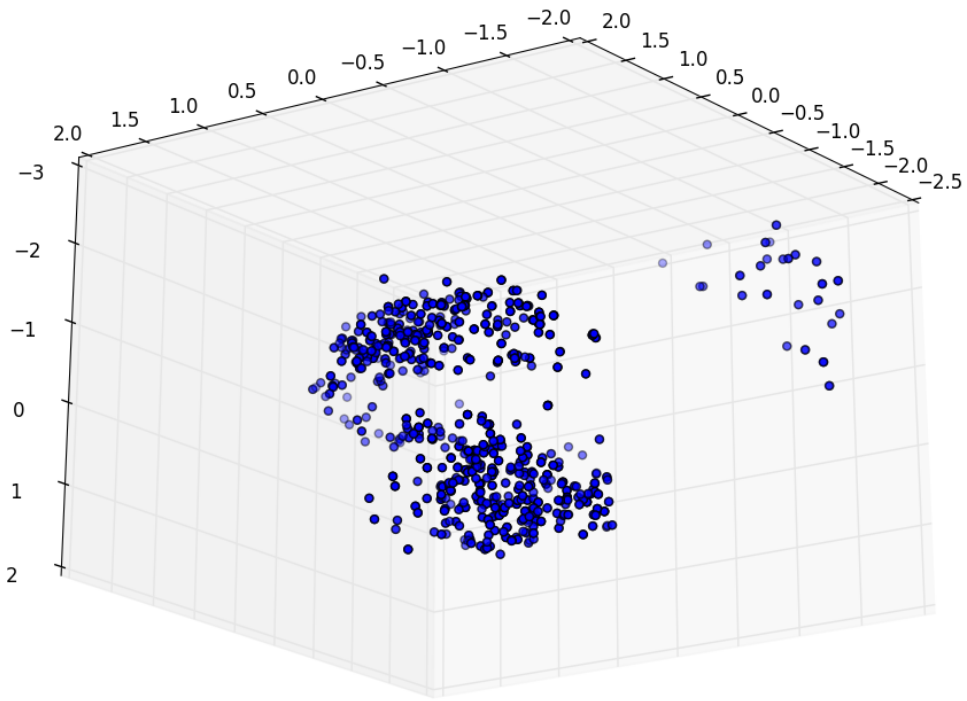
Wheel chair control characterization





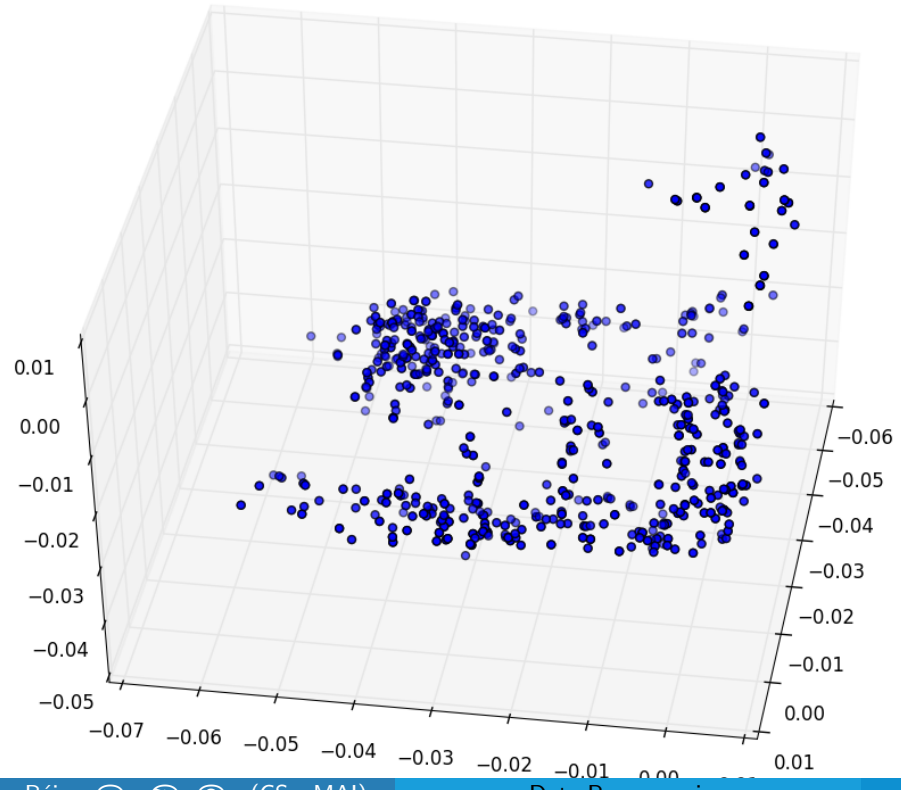
Wheel chair control characterization

PCA 88 $\Longrightarrow$ 3 dimensions



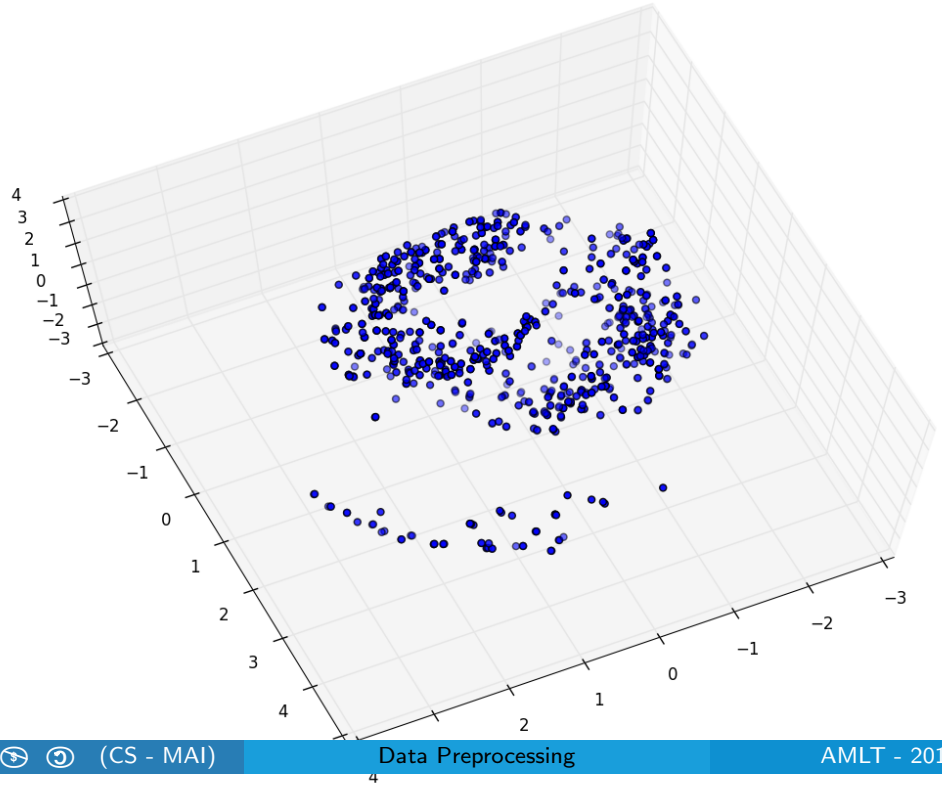
Wheel chair control characterization

SparsePCA 88 $\Longrightarrow$ 3 dimensions



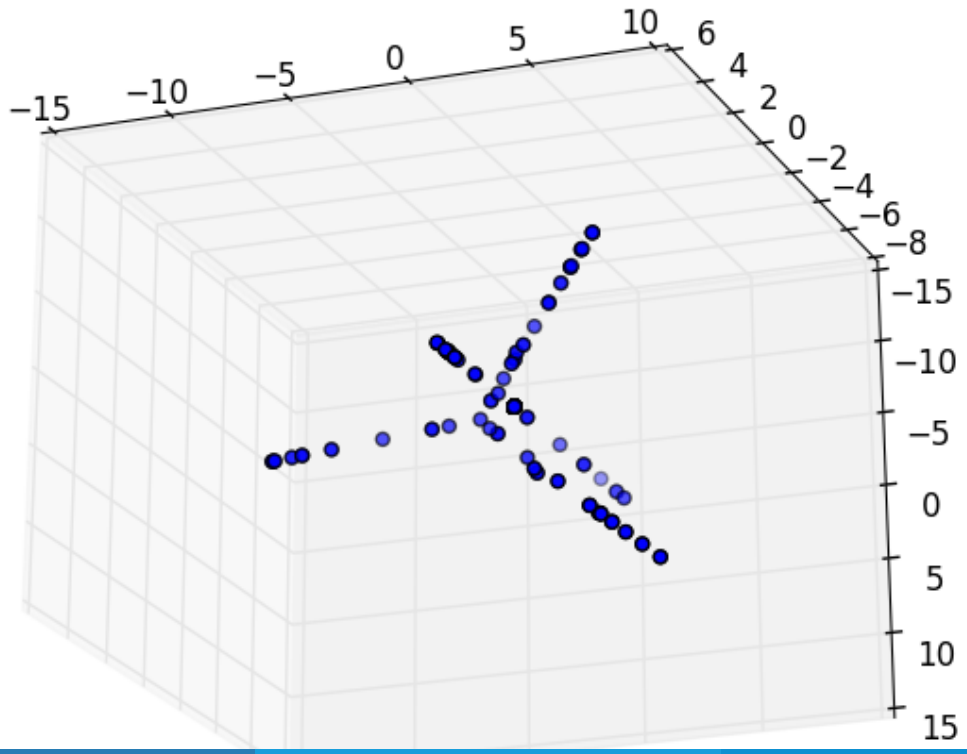
Wheel chair control characterization

MDS 88 $\Longrightarrow$ 3 dimensions



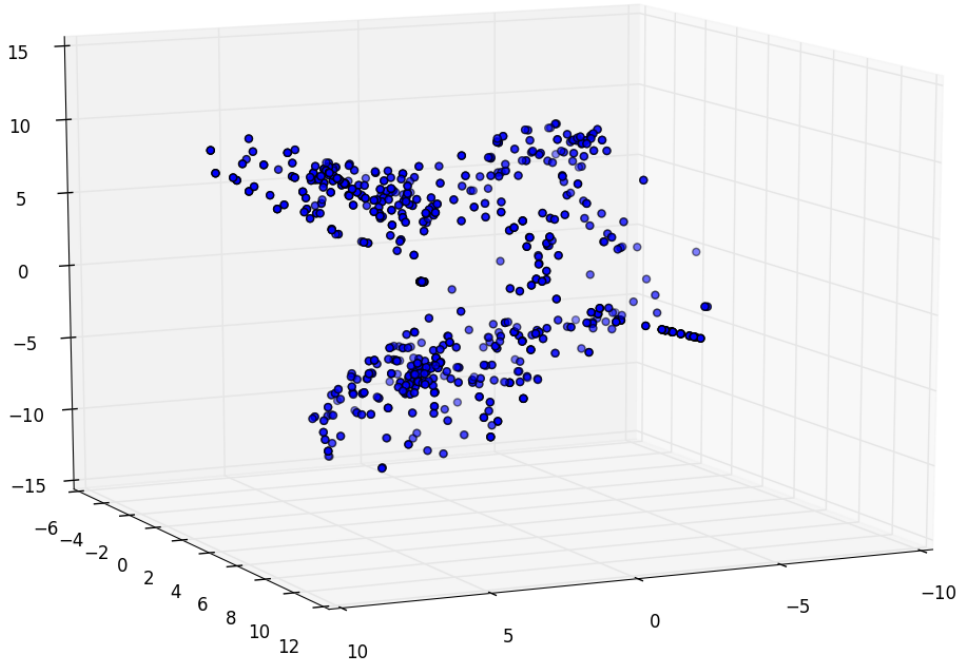
Wheel chair control characterization

ISOMAP (k-n=3) 88 $\Longrightarrow$ 3 dimensions



Wheel chair control characterization

ISOMAP (k-n=10) 88 $\Longrightarrow$ 3 dimensions



Unsupervised Attribute Selection

- The goal is to eliminate from the dataset all the redundant or irrelevant attributes
- The original attributes are preserved
- The methods for Unsupervised Attribute Selection are less developed than in Supervised Attribute Selection
- The problem is that an attribute can be relevant or not depending on the goal of the discovery process
- There are mainly two techniques for attribute selection: Wrapping and Filtering

Attribute selection

- **Wrapping:**
  - A model to evaluate the relevance of subsets of attributes
  - In supervised learning this is easy, in unsupervised learning it is very difficult
  - Results depend on the chosen model and on how well this models capture the actual structure of the data
- **Filtering:**
  - A measure assess the relevance of each attribute
  - This kind of measures are difficult to obtain for unsupervised tasks
  - 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)

Unsupervised feature selection

- The filter methods order the attributes using measures of the structure of the data
  - Measures of properties of the spatial structure of the data (Entropy, PCA, laplacian matrix)
  - Measures of the relevance of the attributes respect the inherent structure of the data
  - Measures of attribute correlation
- The wrapper methods are more diverse
  - Clustering algorithms that compute weights for the attributes
  - Clustering algorithms with an objective function that penalizes the size of the model
  - Consensus clustering

Laplacian Score

- The Laplacian Score is a filter method that ranks the features respect to their ability of preserving the natural structure of the data.
- This method uses the spectral matrix of the graph computed from the near neighbors of the examples
- Similarity is usually computed using a gaussian kernel (edges not present have a value of 0)

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

And all edges not present have a value of 0.

Laplacian Score

- The Laplacian matrix is computed from the similarity matrix S and the degree matrix D as

$$L = S - D$$

- The score first computes for each attribute r and their values f_r the transformation $\tilde{f}_r$ as:

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

- and then the score L_r is 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

Python Notebooks

These two Python Notebooks show some examples dimensionality reduction and feature selection

- Dimensionality reduction and feature selection Notebook (click here to go to the url)
- Linear and non linear dimensionality reduction Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

1 Introduction

2 Data Preprocessing

3 Dimensionality Reduction

4 Similarity functions

Similarity/Distance functions

- Unsupervised algorithms need similarity/distance to compare examples
- This comparison will be obtained using functions of the attributes of the examples
- Usually we suppose that the examples are embedded in a N-dimensional space where it can be defined a similarity/distance
- There are domains where this assumption is not true so some other kind of functions will be needed to represent instances relationships

Properties of Similarity/Distance functions

- The properties of a similarity function are:
  1. $s(p,q) = 1 \iff p = q$
  2. $\forall p, q \; s(p,q) = s(q,p)$
- The properties of a distance function are:
  1. $\forall p, q \; d(p,q) \geq 0 \;\; and \;\; \forall p, q \; d(p,q) = 0 \iff p = q$
  2. $\forall p, q \; d(p,q) = d(q,p)$
  3. $\forall p, q, r \; d(p,r) \leq d(q,p) + d(p,r)$

Distance functions

- **Minkowski metrics** (Manhattan, Euclidean)

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

- **Mahalanobis distance**

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

where φ is the matrix covariance of the attributes

Distance functions

- **Chebyshev Distance**

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

- **Camberra distance**

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

Similarity functions

- **Cosine similarity**

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

- **Pearson correlation measure**

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

Similarity functions

Binary data

- **Coincidence coefficient**

$$s(i,k) = \frac{a_{00} - a_{11}}{d}$$

- **Jaccard coefficient**

$$s(i,k) = \frac{a_{11}}{a_{00} + a_{01} + a_{10}}$$

Python Notebooks

This Python Notebook shows examples of using different distance functions

- Distance functions Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

Python Code

- In the code from the repository inside subdirectory `DimReduction` you have the Authors python program
- The code uses the datasets in the directory `authors` from the datasets zipfile
  - `Auth1` has fragments of books that are novels or philosophy works
  - `Auth2` has fragments of books written in English and books translated to English
- The code transforms the text to attribute vectors and applies different dimensionality reduction algorithms
- Modifying the code you can process one of the datasets and choose how the text is transformed into vectors

3

Data Clustering

Unsupervised Learning

Javier Béjar

CS - MAI

AMLT - 2016/2017

Outline

1. Unsupervised Learning
   - Introduction
2. Hierarchical Algorithms
   - Statistical Algorithms
   - Concept Formation Algorithms
3. Partitional algorithms
   - Model/Prototype Clustering
   - Density/Grid Clustering
   - Graph Clustering
   - Other Approaches
4. Applications

1 Unsupervised Learning

2 Hierarchical Algorithms

3 Partitional algorithms

4 Applications

Unsupervised Learning

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

Unsupervised Learning

- Learning by the discovery of predefined structures
  - For example: probability distributions/models using parametric or non parametric estimation
- It is assumed that the data is embedded in a N-dimensional space that has a similarity/dissimilarity function defined
- Bias:
  - Examples are more related to the nearest examples than to the farthest
  - Look for compact groups that are maximally separated from each other
- Areas related: Statistics, machine learning, graph theory, fuzzy theory, physics

Unsupervised Learning

- Goals:
  - Summarization: To obtain representations that describe an unlabeled dataset
  - Understanding: To discover the concepts inside the data
- These task are difficult because the discovery process is biased by context
  - Different answers can be valid depending of the discovery goal or the domain
  - There are few criteria to validate the results
- Representation of the clusters: Unstructured (partitions) or relational (hierarchies)

Unsupervised Learning Algorithms

Strategies

Hierarchical algorithms

- Examples are organized as a binary tree
- Based on a relationship among examples computed from similarities/dissimilarity functions
- No explicit division in groups, has to be chosen a posteriori

Partitional algorithms

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

1 Unsupervised Learning

2 Hierarchical Algorithms

3 Partitional algorithms

4 Applications

Hierarchical algorithms

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

Hierarchical algorithms

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

Computational cost

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

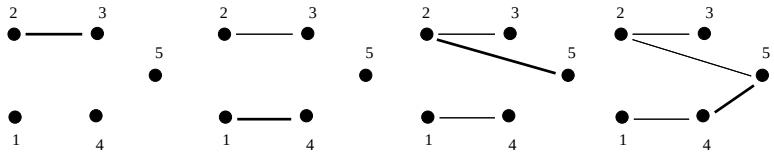
Agglomerative Graph Algorithm

Algorithm: Agglomerative graph algorithm

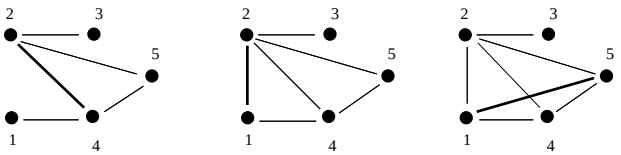
Compute distance/similarity matrix
repeat
  Find the pair of examples with smallest similarity
  Add an edge to the graph corresponding to this pair
  if *Agglomeration criteria holds* **then**
    Merge the clusters the pair belongs to
  end
until *Only one cluster exists*

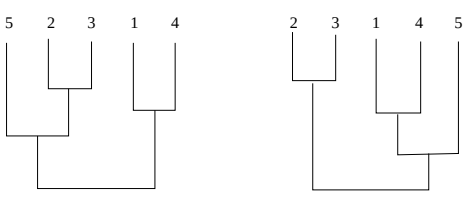
- Single linkage = The new edge is between to disconnected graphs
- Complete linkage = The new edge creates a clique with all the nodes of both subgraphs

Hierarchical algorithms - Graphs



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





Single Link Complete Link

Agglomerative Johnson algorithm

Algorithm: Agglomerative Johnson algorithm

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*

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

Hierarchical algorithms - Matrices

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

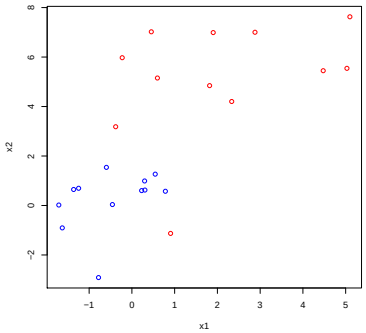
	2,3	4	5
1	7	2	7
2,3		7.5	6
4			4

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

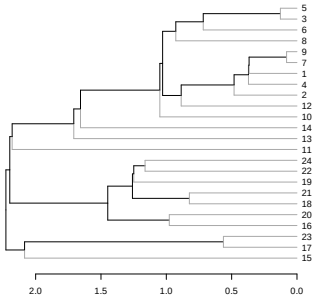
	1,4,5
2,3	6.725

Hierarchical algorithms - Example

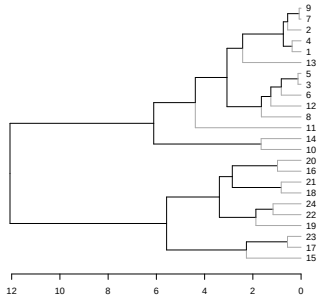
Data



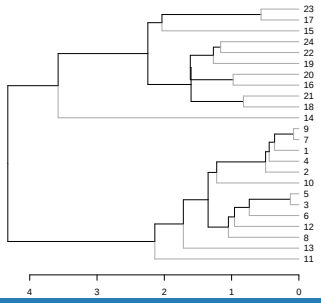
Single Link



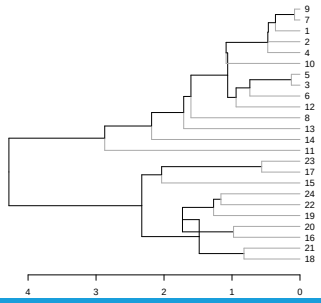
Complete Link



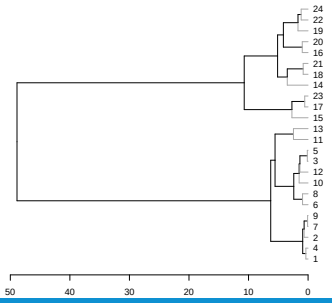
Median



Centroid



Ward



Hierarchical algorithms - Problems

- A partition of the data is not given, it has to be decided a posteriori
- Some undesirable and strange behaviours could appear (chaining, inversions, breaking large clusters)
- Some have problems with different sized clusters and convex shapes
- Dendrogram is not a practical representation for large amounts of data
- Its computational cost is high for large datasets
  - Time is $O(n^2)$ in the best case

Python Notebooks

This Python Notebook shows examples of using different hierarchical clustering algorithms

- Hierarchical Clustering Algorithms Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

Other hierarchical algorithms

Concept Formation

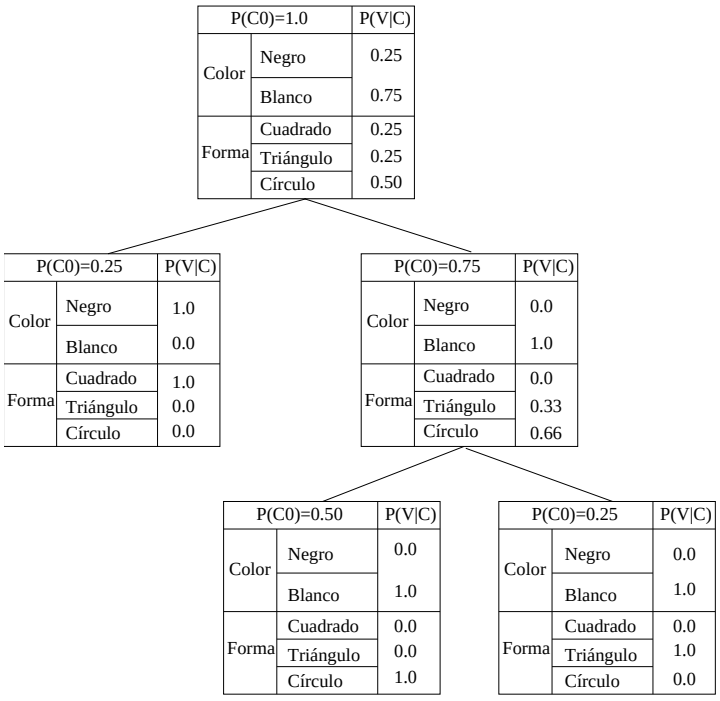
- Learning has an incremental nature (experience is acquired from continuous observation, not at once)
- Concepts are learned with their relationships (polithetic hierarchies of concepts)
- Search in the space of hierarchies
- An objective function measures the utility of the learned structure
- The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

Concept Formation - COBWEB

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

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

Probabilistic hierarchy



COBWEB

Category utility (CU)

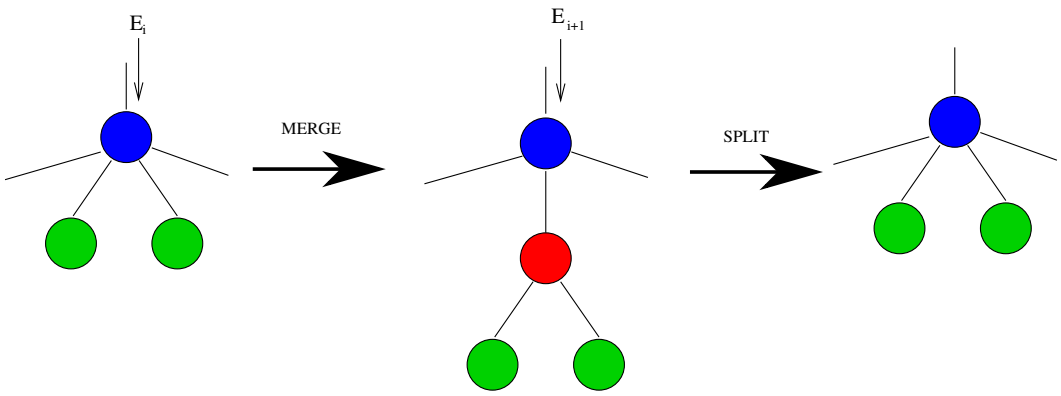
- Category utility balances:
  - Intra class similarity: $P(A_i = V_{ij}|C_k)$
  - Inter class similarity: $P(C_k|A_i = V_{ij})$
- It measures the difference between a partition of the data and no partition at all
- For qualitative attributes and a set of k categories $\{C_1, \ldots C_k\}$ is defined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

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

(see the derivation on the paper)

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
- **Divide:** A concept is substituted by its children



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

1 Unsupervised Learning
2 Hierarchical Algorithms
3 Partitional algorithms
4 Applications

Partitional algorithms

The computational cost to find the optimal partition of N objects in K groups is NP-hard

- 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

K-means

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

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

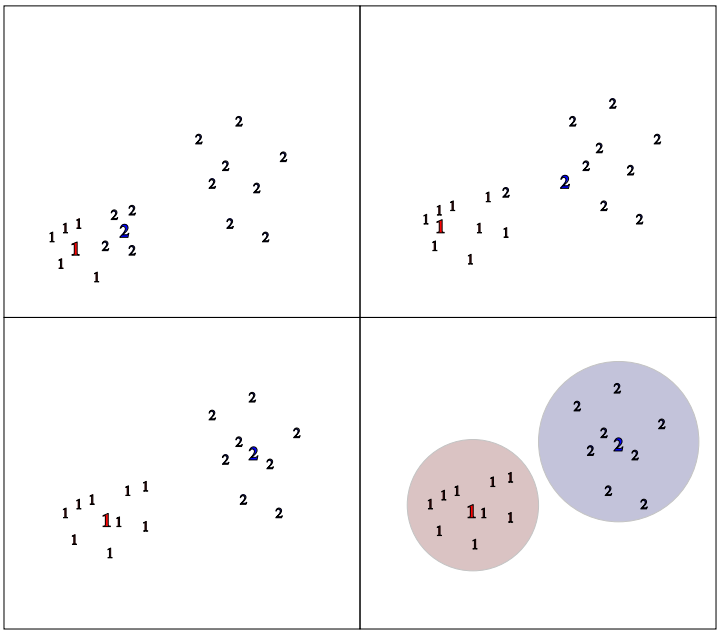
- Optimization by a Hill Climbing search algorithm
- The algorithm converges to a local minima

K-means

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* $< \epsilon$

K-means



K-means

Practical problems

- The algorithm is sensitive to the initialization (to run the algorithm from several random initializations is a common practice, with the additional computational cost)
- Sensitive to clusters with different sizes/densities and outliers
- To find the value of k is not an easy problem (experimentation with different values is needed)
- A solution is found even if the classes are not hyperspherical (some classes could be split/merged)
- No guarantee about the quality of the solution
- The spatial complexity makes it not suitable for large datasets

K-means++

Initialization Strategies

- K-means++ modifies the initialization strategy of the algorithm
- The idea is to try to maximize distance among initial centers
- Algorithm:
  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

Bisecting K-means

- Bisecting K-means iteratively splits a cluster in two until the desired number of clusters is obtained
- Pro: Reduces the effect of the initialization/a hierarchy is obtained/can be used to determine K
- Con: Different criteria can be used to decide which cluster to split (the largest, the one with largest variance, ...)
- Algorithm:
  1. Choose a number of partitions
  2. Apply K-means to the dataset with k=2
  3. Evaluate the quality of the current partition
  4. Pick the cluster to 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

Global K-means

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

Other K-means variants

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

K-medoids

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

K-medoids - PAM

Partitioning Around Medoids (PAM):

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

Incremental algorithms: Leader Algorithm

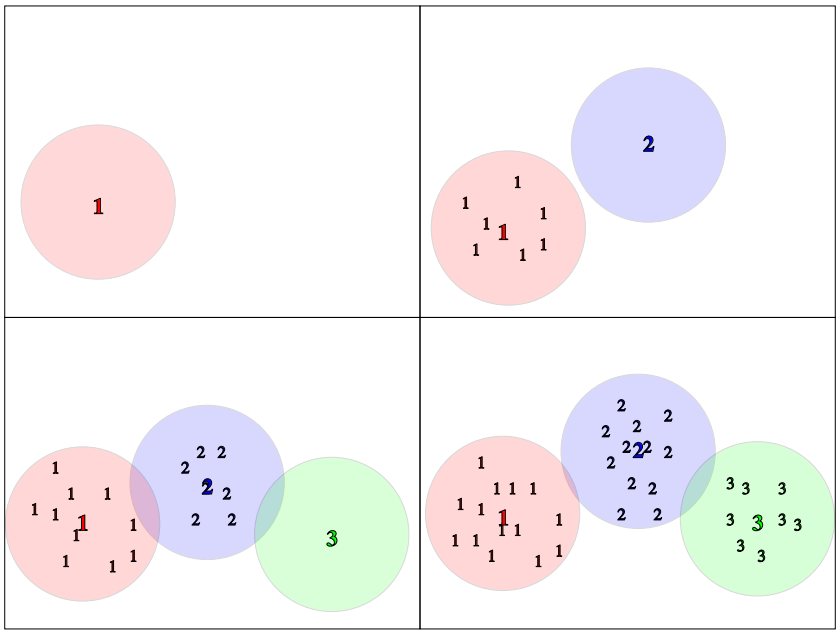
- The previous algorithms need all data from the beginning
- An incremental strategy is needed when data comes as a stream:
  - A distance/similarity threshold (D) determines the extent of a prototype
  - Inside the threshold: Incremental updating of the model (prototype)
  - Outside the threshold: A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

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

Leader Algorithm



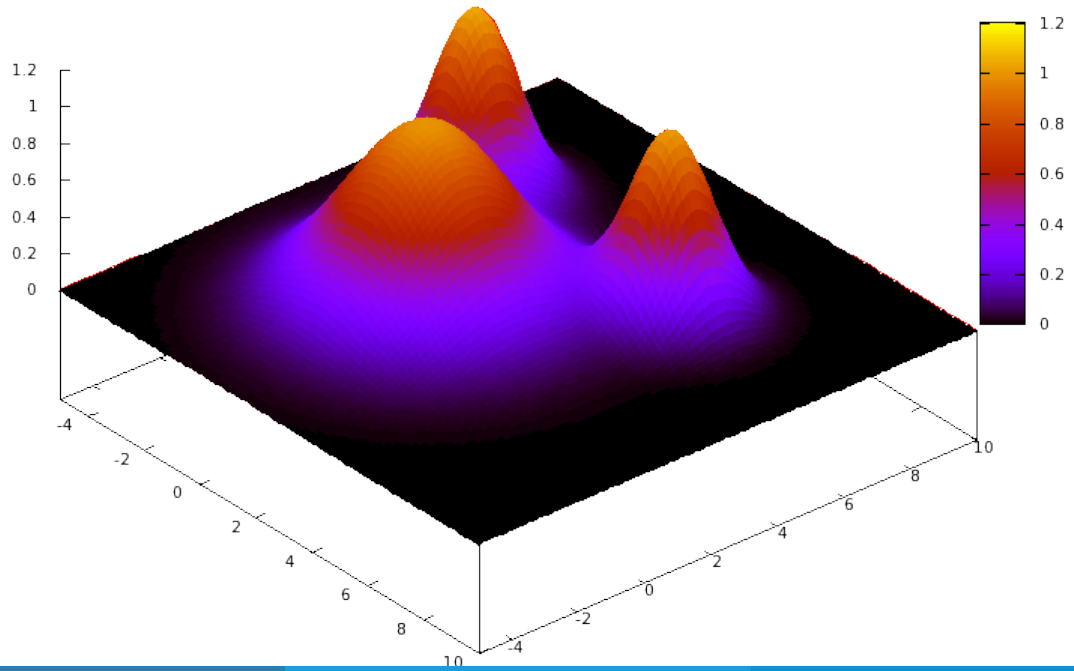
Mixture Decomposition - EM algorithm

- We assume that the data are drawn from a mixture of probability distribution functions (usually Gaussian), we are looking for the parameters of the distributions that explain better the data
- The model of the data is:

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

  Being K the number of clusters and $\sum_{i=1}^{K} P(w_i) = 1$
- Each instance has a probability to belong to a class

Mixture Decomposition - EM algorithm



Mixture Decomposition - EM algorithm

- The goal is to estimate the parameters of the distribution that describes each class (e.g.: means and standard deviations)
- The algorithm maximizes the likelihood of the distribution respect the dataset
- It performs iteratively two steps:
  - **Expectation:** We calculate a function that assigns a degree of membership to all the instances to any of the K probability distributions
  - **Maximization:** We re-estimate the parameters of the distributions to maximize the membership

EM Algorithm (K Gaussian)

- For the Gaussian case:

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

Being $\overrightarrow{\mu}$ the vectors of means and Σ the covariance matrices

EM Algorithm (K Gaussian)

The computations depend on the assumptions that we make about the attributes (independent or not, same σ, ...)

- The attributes are independent: μ_i and σ_i have to be computed for each class ($O(k)$ parameters) (model: hyper spheres or ellipsoids parallel to coordinate axis)
- The attributes are not independent: μ_i, σ_i and σ_{ij} have to be computed for each class ($O(k^2)$ parameters) (model: hyper ellipsoids non parallel to coordinate axis)

EM Algorithm (K Gaussian)

- For the case of A independent attributes:

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

- The model to fit is

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

EM Algorithm (K Gaussian)

- The update of the parameters in the maximization step is:

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

EM Algorithm (K Gaussian)

- A set of K initial distributions is generated $N(\mu_i, \sigma_i)$, μ_i and σ_i are vectors corresponding to the mean and the variance of the attributes
- We repeat until convergence:
  1. **Expectation:** Compute the membership of each instance to each probability distribution. Usually we use the log likelihood function of the distribution
     - Each instance will have a weight depending of the probability assigned by the previous step $w_{x_{j,i}} = log(P(x_j|N(\mu_i, \sigma_i)))$
  2. **Maximization:** Recompute the parameters using the weights from the previous steps and obtain the new μ_i and σ_i for each distribution

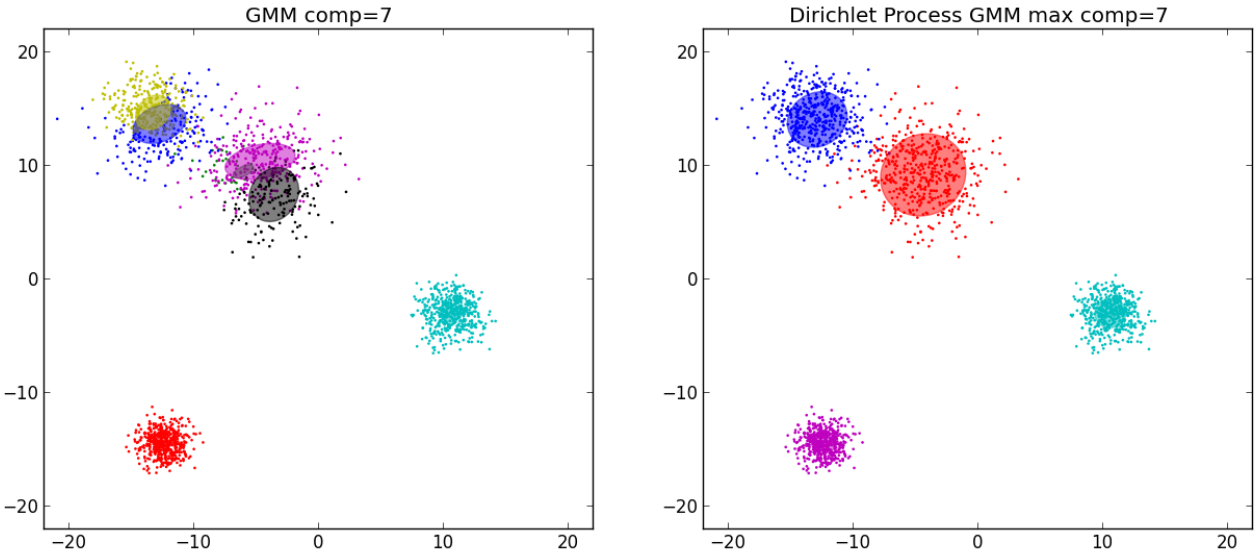
EM algorithm - Comments

- K-means is a particular case of this algorithm (hard partition)
- The main advantage is that we obtain a membership as a probability (soft assignments)
- Using different probability distribution we can find different kinds of structures.
- The problem is to derive the calculations for the iterative updating of the parameters

Dirichlet Process Mixture Model

- One of the problems of GMM is to decide a priori the number of components
- This problem can be solved using a mixture model that uses as a prior a Dirichlet distribution
- Dirichlet distribution assumes an unbound number of components
- A finite weight is distributed among all the components by the Dirichlet distribution
- The fitting of the model will decide what number of components better suits the data

Dirichlet Process Mixture Model



Fuzzy Clustering

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

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

  where $\sum_{k=1}^{K} \delta(C_k, x_i) = 1$ and b is a blending factor
- This is an advantage from other algorithms when the groups are overlapped

Fuzzy Clustering

- C-means is the most known fuzzy clustering algorithm, it is the fuzzy version of K-means
- The algorithm performs an iterative optimization of the objective function
- The updating of the cluster centers is computed as:

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

- And the updating of the memberships:

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

Fuzzy Clustering

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

Python Notebooks

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

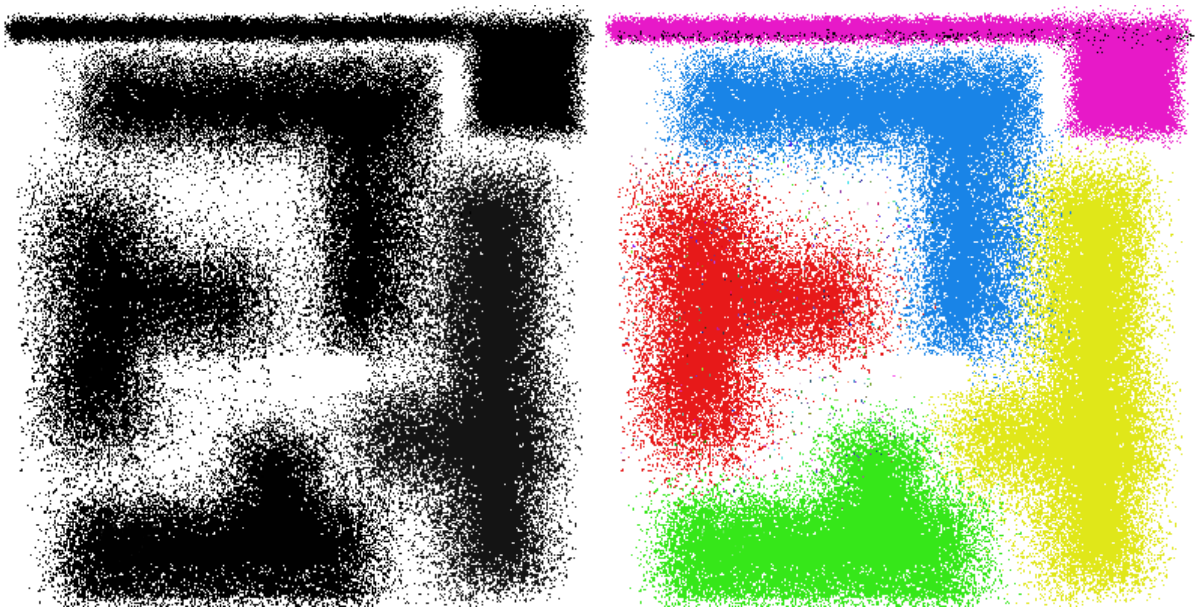
- Prototype Based Clustering Algorithms Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

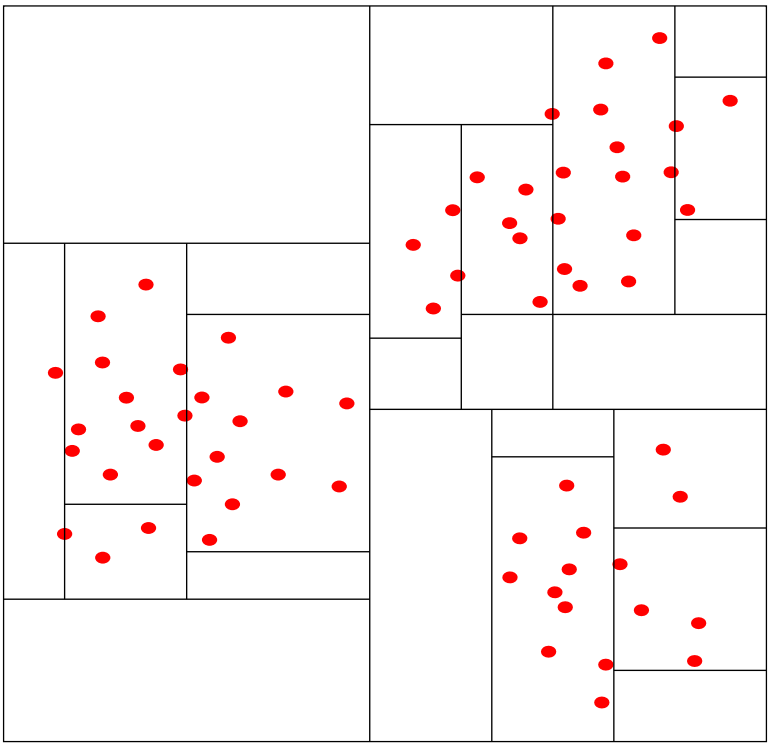
Density/Grid Based Clustering

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

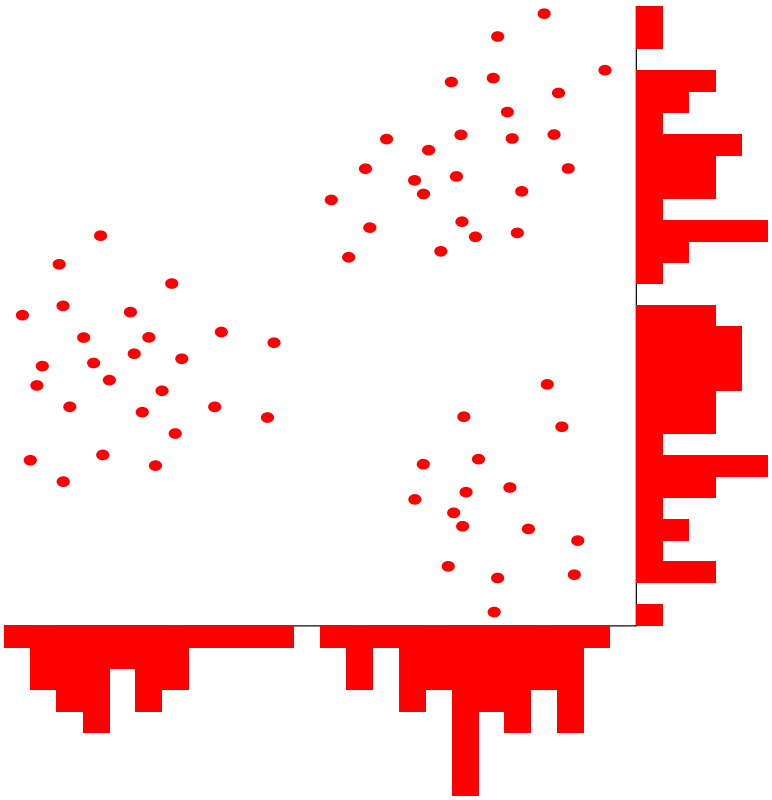
Density estimation



Space Partitioning



Multidimensional Histograms



DBSCAN/OPTICS

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

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

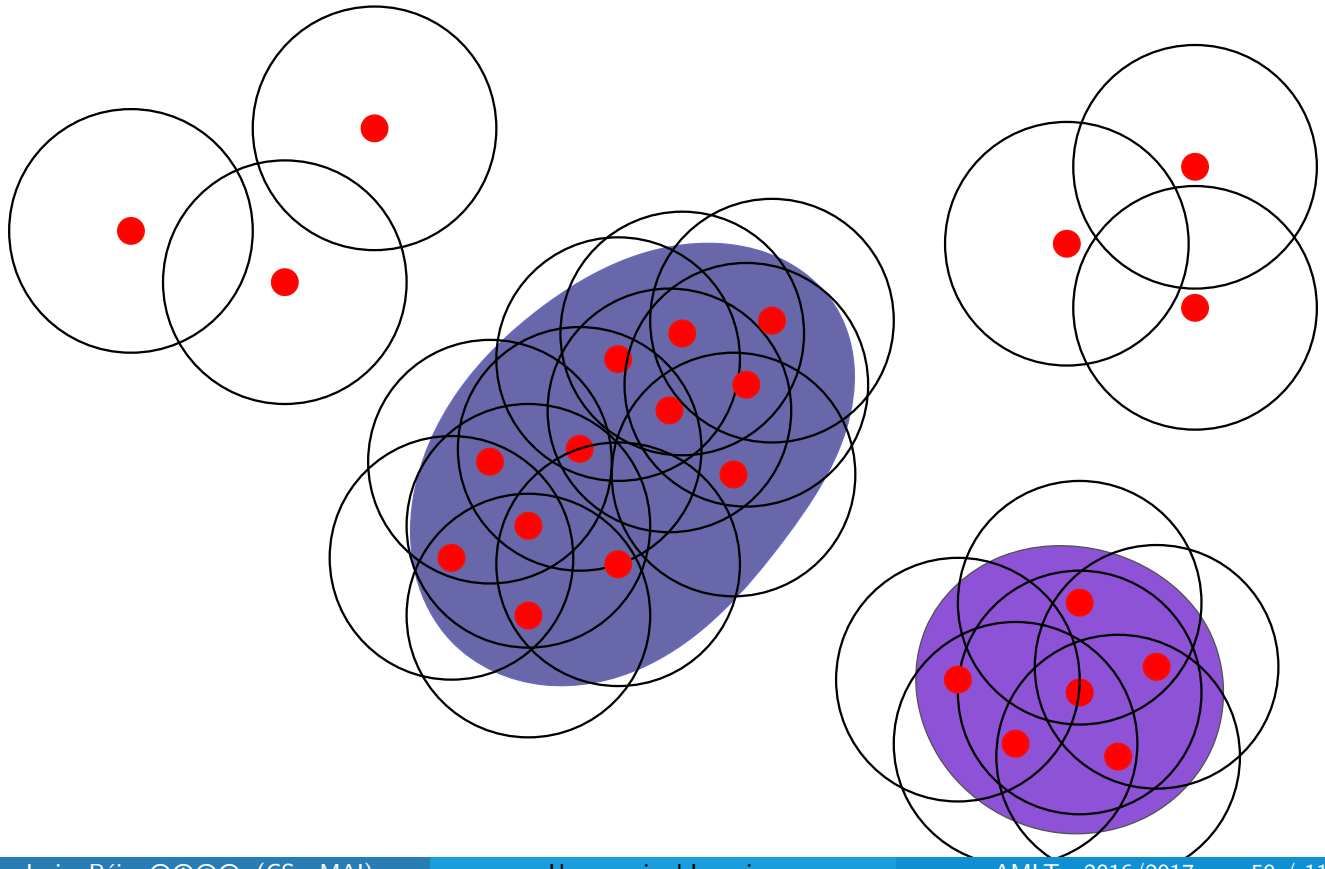
- Used in spatial databases, but can be applied to data with more dimensionality
- Based on finding areas of high density, it finds arbitrary shapes
- We define ε-**neighbourhood**, as the instances that are at a distance less than ε to a given instance

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

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

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

DBSCAN - Algorithm



DBSCAN - Density

- 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
- We say that p and q are **Density connected** if there is an instance o such that both p and q are **Density Reachable** from o

DBSCAN - Cluster Definition

Cluster

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

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

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

DBSCAN - Algorithm

Algorithm

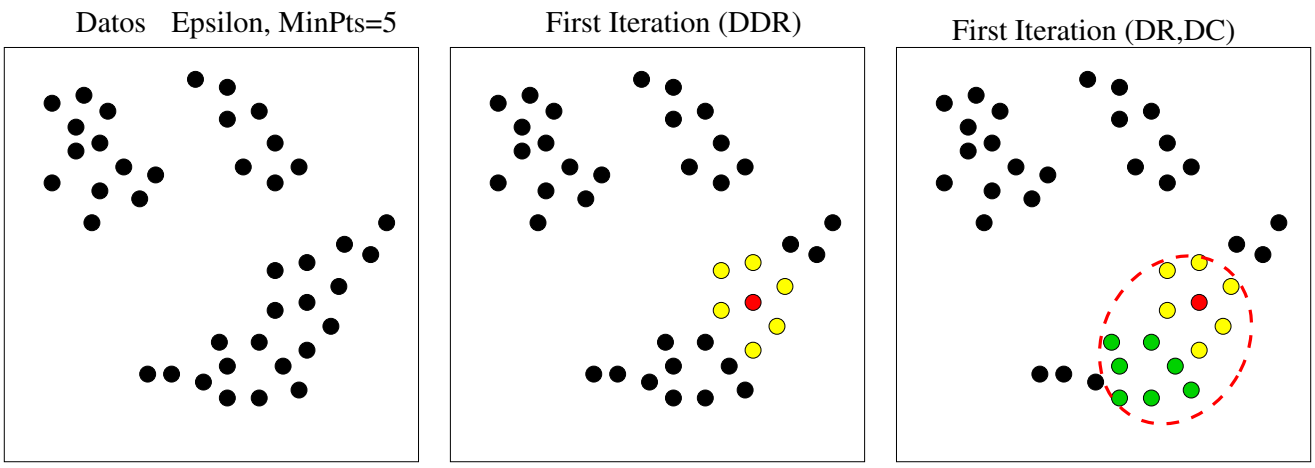
1. We start with an arbitrary instance and compute all density reachable instances with respect ε and $MinPts$.
2. If it is a core point we will obtain a group, otherwise, it is a border point and we will start from other unclassified instances

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

ε and $MinPts$ are set from the thinnest cluster from the dataset

The OPTICS algorithm defines a heuristic to find a good set of values for these parameters

DBSCAN - Algorithm



DENCLUE

Hinneburg, Keim **An Efficient Approach to Clustering in Large Multimedia Databases with noise** (1998)

- Clustering algorithm based on kernel density estimation
- Defines the influence of an example in a dataset as the sum of a kernel for all the data (for example a gaussian kernel)

$$f_B^y(x) = f_B(x, y)$$

- Defines the density function of a dataset as the sum of the influences of all examples of the dataset

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

DENCLUE

- We define the gradient of $f_B^D(x)$ as

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

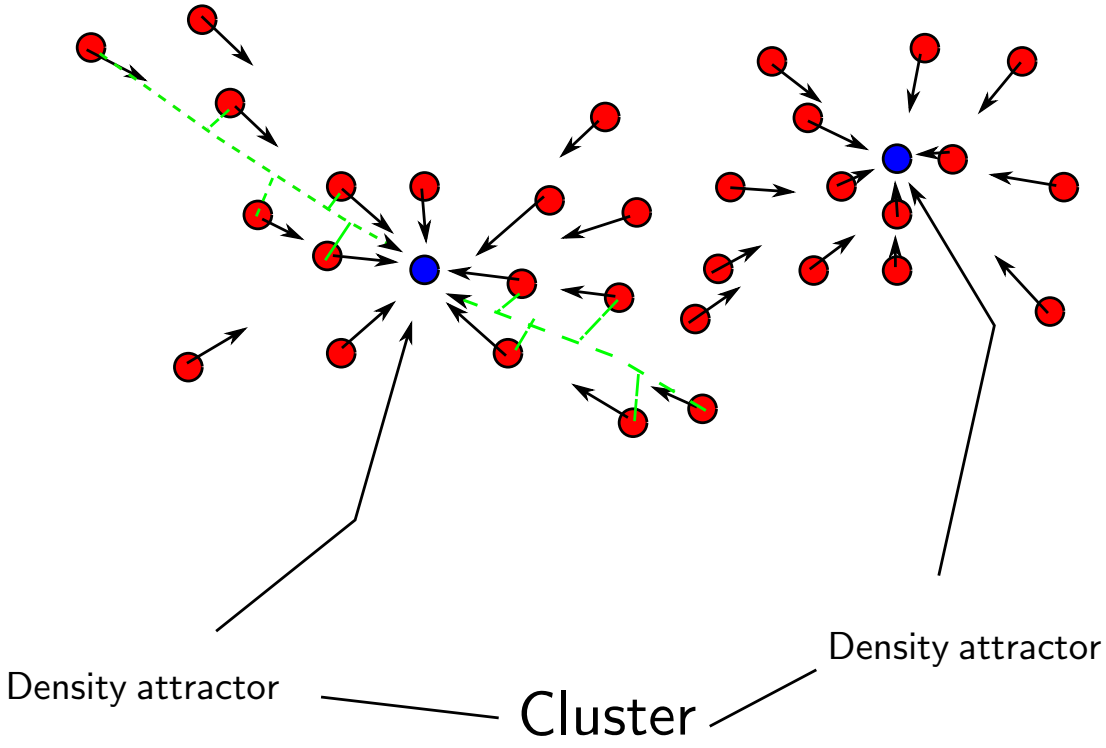
- A point x^* is a *density-attractor* iff is a local maximum of the density function f_B^D
- A point x is *density-attracted* to a density-attractor iff $\exists k \in N; d(x^k, x^*) \leq \epsilon$

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

DENCLUE

- We define a *Center-defined Cluster* (wrt to σ,ξ) for a density-attractor x^* as the subset of examples being density-attracted by x^* and with $f_B^D(x^*) > \xi$
- We define 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 $\forall p \in P : f_B^D(p) > \xi$
- Different algorithms can be reproduced with the right choice of σ and ξ parameters (DBSCAN, K-means, hierarchical clustering)

DENCLUE - example



DENCLUE - Algorithm

- For efficiency reasons the density functions are only computed for the neighbours of each point
- 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 this hypercubes given a threshold, only the highly populated ones and their neigbours 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.

CLIQUE

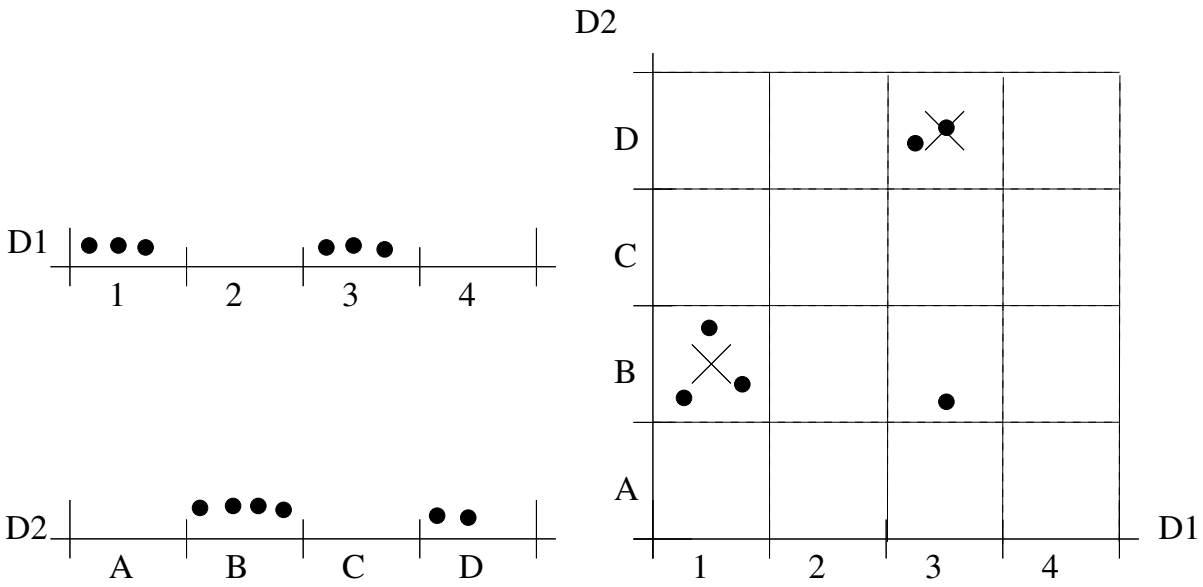
Agrawal, Gehrke, Gunopulos, Raghavan **Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications** (1998)

- It generates CNF descriptions from the groups that discovers
- The goal is to find a space with less dimensions where the groups are easier to identify
- The algorithm is scalable and can be used with data of high dimensionality
- The goal is to obtain interpretability
- It is based on density estimation techniques
  - Each dimension of the dataset is divided using a grid
  - The bins with higher density than certain threshold are identified

CLIQUE - Algorithm

1. Identify the subspaces with clusters
   - It applies a bottom up strategy identifying first the bins of higher density in one dimension and combining them
   - 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

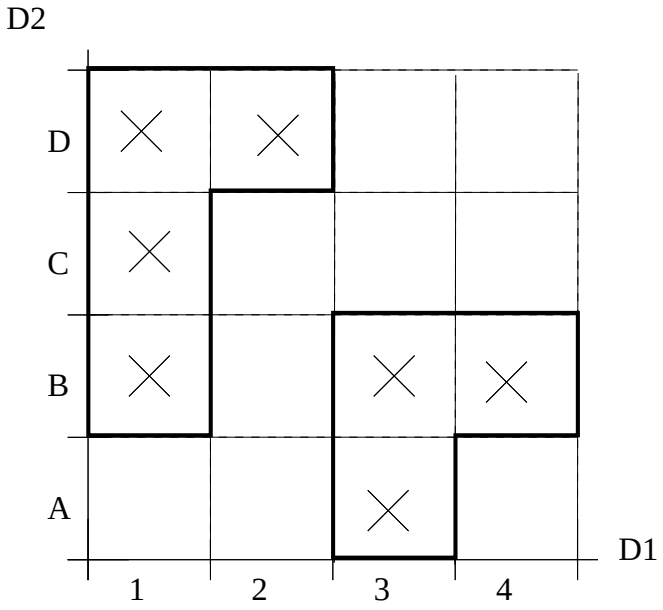
CLIQUE - Algorithm



CLIQUE - Algorithm

2. Identify the clusters
   - We receive the set of dense bins from the previous step
   - We look for contiguous bins and consider them in the same group
   - 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

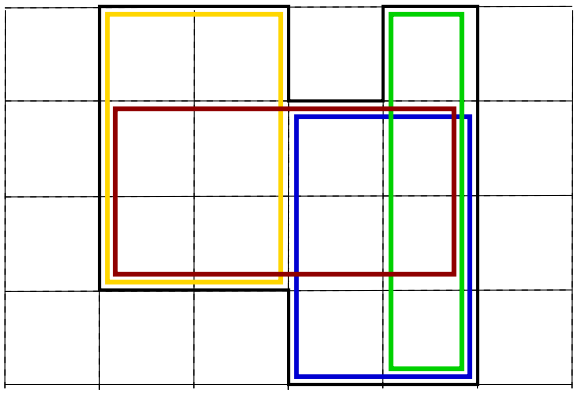
CLIQUE - Algorithm

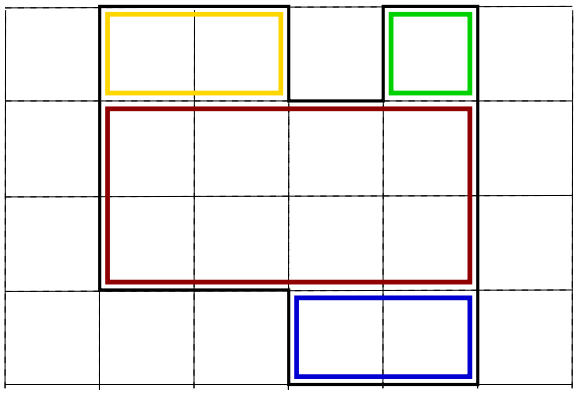


CLIQUE - Algorithm

3. Generate the minimal description of the clusters
   - We receive a set of connected components in k dimensions
   - The problem is to look for the minimal cover for each connected component (NP-hard)
   - We compute an approximation using rectangles that cover part of the group maximally and after this we reduce the redundancies

CLIQUE - Algorithm





Python Notebooks

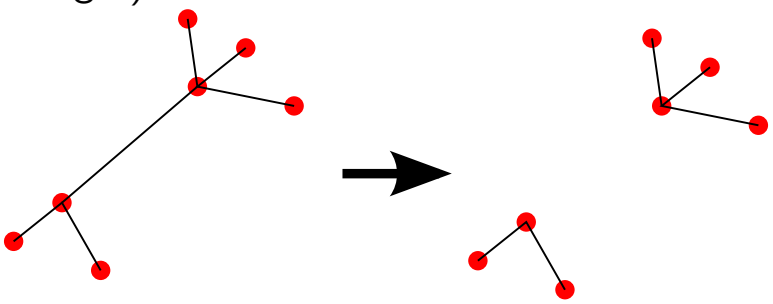
This Python Notebook compares Prototype Based and Density Based Clustering algorithm

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

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

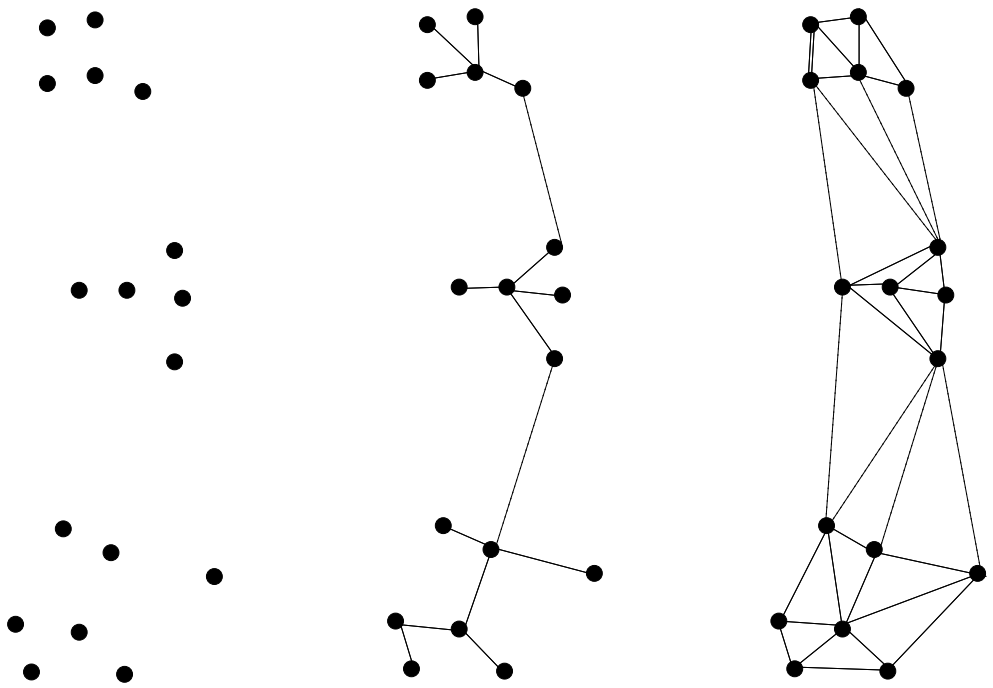
Based in graph theory

- Different kinds of graphs are defined with the examples (MST, Voronoi, Delanau, ...) based on their similarities/distances/neighborhood relations
- A consistency criteria for the edges of the graph is defined (delete inconsistent edges)



- The set of unconnected components defines the clusters
- Two advantages: we do not need to know the number of classes beforehand, we do not look for a specific model (any shape is possible)

Based in graph theory



Spectral Clustering

- Spectral graph theory defines properties that hold the eigenvalues and eigenvectors of the adjacency matrix or Laplacian matrix of a graph
- Spectral clustering uses spectral properties of the distance matrix
- The distance matrix represents a graph that connects the examples
  - Complete graph
  - Neighbourhood graph (different definitions)
- From the diagonalization of this matrix some clustering algorithms can be defined

Spectral Clustering (First approach)

- We start with the similarity matrix (W) of a dataset (complete or not)
- This matrix represents the similarity graph of the instances
- The degree of an edge is defined as:

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

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

Spectral Clustering (First approach)

- We can cluster a dataset following this steps:
  1. Compute the Laplace matrix from the similarity matrix
  2. Compute the first K eigenvalues of the Laplace matrix
  3. Use the eigenvectors as new datapoints
  4. Apply K-means as clustering algorithm
- We are embedding the dataset in a space with less dimensions

Spectral Clustering (Second approach)

- From the graph defined from the similarity matrix and its Laplacian matrix (see previous slides), it is possible to define the problem as a graph partitioning problem, so any graph partitioning algorithm can be used
- 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 :

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

Spectral Clustering (Second approach)

- Using directly the weights of the Laplacian not always gives good results, there are alternative objective functions that are used also to solve this problem

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

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

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

Other clustering approaches

- Several other approaches exists for partitioning data
  - Affinity Propagation clustering (message passing)
  - Self organizing maps (neural networks)

Affinity Propagation Clustering

- Affinity clustering is a message passing algorithm related to graph partitioning
- The algorithm establish 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 $s(i, k)$ (eg.: euclidean distance)
- The algorithm does not set the number of clusters, each example receives a value $r(k, k)$ that represents the preference for each point to be an exemplar

Affinity Propagation Clustering - Messages

- The examples pass two kind of messages
  - *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
- The algorithm is related to other message passing algorithms used for belief propagation in probabilistic graphical models

Affinity Propagation Clustering - Updating

- 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:

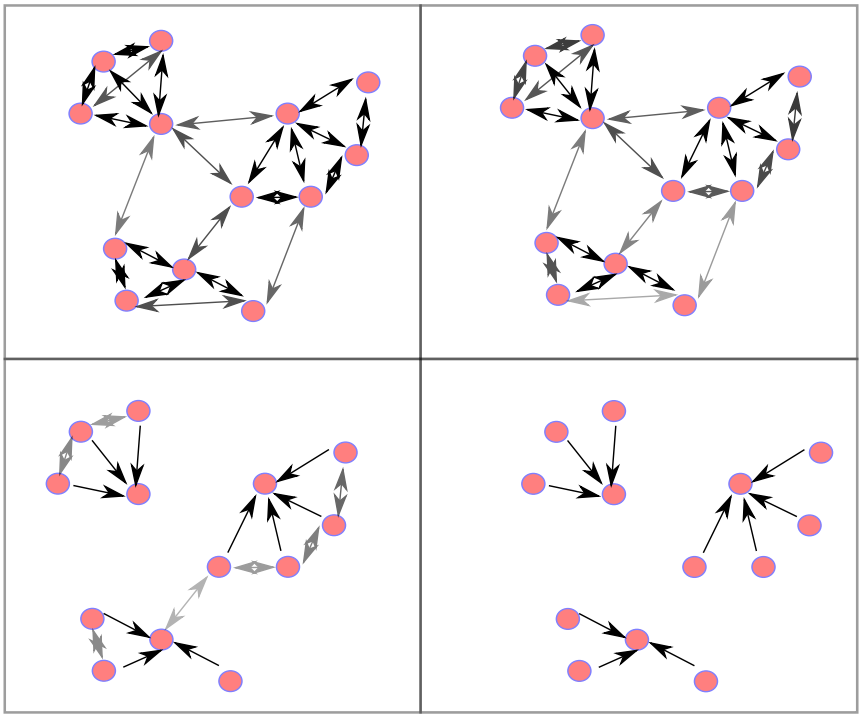
$$a(k,k) = \sum_{i' \neq k} max(0, r(i',k))\}$$

- 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.

Affinity Propagation Clustering - Algorithm

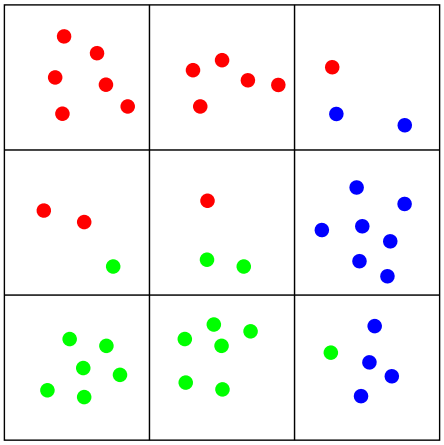
1. Update the responsibilities given the availabilities
2. Update the availabilities given the responsibilities
3. Compute the exemplars
4. Terminate if the exemplars do not change in a number of iterations

Affinity Propagation Clustering - Algorithm



Unsupervised Neural Networks

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



Self-Organizing Maps

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

Algorithm: Self-Organizing Map algorithm

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

Self-Organizing Maps

- During the iterations the mesh is transformed to be closer to the data, but maintaining the bidimensional relationship between prototypes
- The performance of the algorithm depends on the learning rate α, usually is decreased from 1 to 0 during the iterations
- The neighborhood of a prototype is defined by the adjacency of the cells and the distance of the prototypes
- The number of neighbors used in the update is decreased during the iterations from a predefined number to 1 (only the prototype nearest to the observation)
- Different variations of the algorithm give more weight depending on the distance of the prototypes

Python Notebooks

This Python Notebook has examples for Spectral and Affinity Propagation Clustering

- Spectral and Affinity Propagation Clustering Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

Python Code

- In the code from the repository inside subdirectory `Clustering` you have the python programs `HierarchicalAuthors`, `PartitionalAuthors` and `DensityBasedCity`
- The first and second ones use the `authors` dataset and allows to compare hierarchical clustering and partitional clustering with this data. You can observe what happens with both datasets and using different attributes
- The third one uses data from the `City` datasets that represents events in different cities (Tweets and Crime) showing results for a variety clustering algorithms. You can use data from different cities.

1 Unsupervised Learning

2 Hierarchical Algorithms

3 Partitional algorithms

4 Applications

Barcelona Twitter/Instagram Dataset

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

The data

- The dataset is composed of several hundreds of thousands of events during several months (represents less than 5 % of the actual events)
- Each event has a geographical position (latitude/longitude) and a time stamp (inside an area of 30×30 Km^2)
- For the purpose of the analysis (geoprofiles) the actual data is difficult to analyze:
  - The resolution of the coordinates is too fine, low probability of two events in the exact same place
- Clustering geographically the events can help to make sense of the data

Clustering of the events

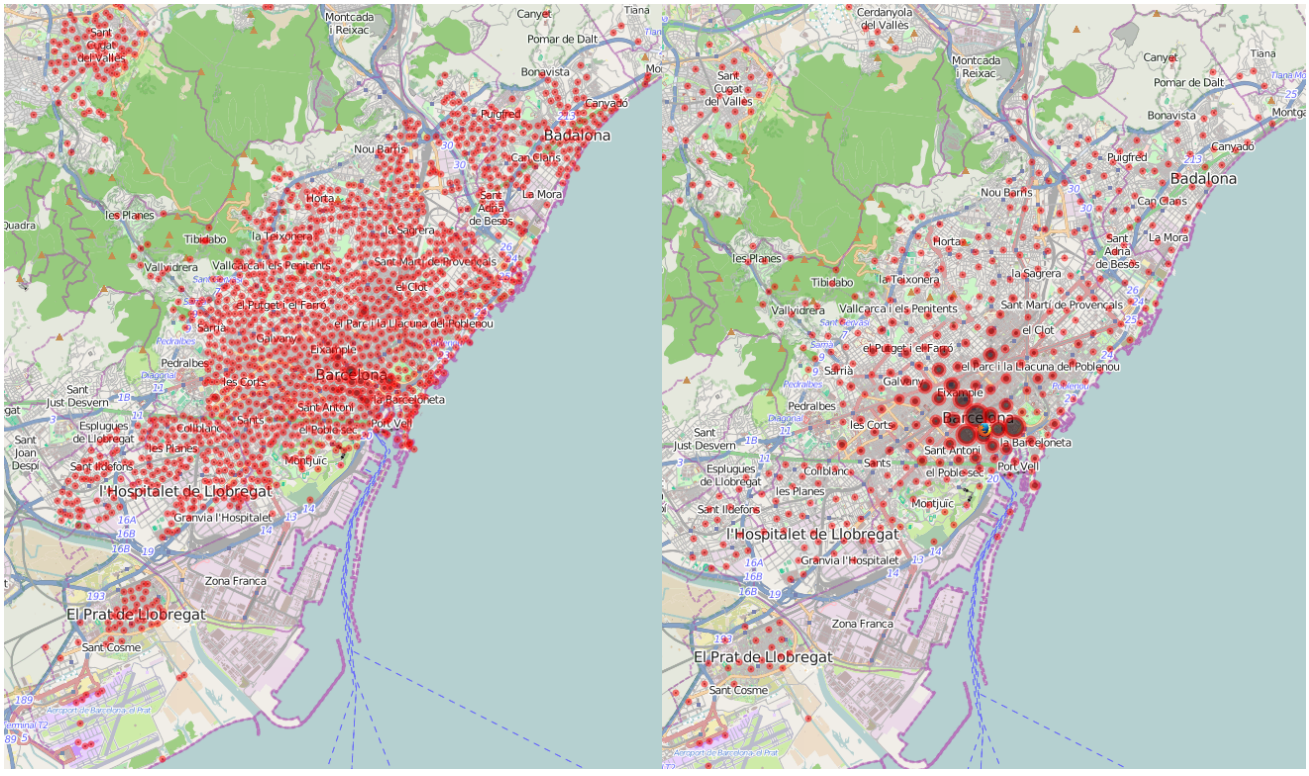
- Being the data geographically distributed there are few alternatives (clusters of arbitrary shapes)
- The more adequate methods seem:
  - Density based clustering
  - Grid based clustering
- The size of the dataset ($\sim$ 2.5 million events) could arise scalability issues, so a preliminar coarse grained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

Clustering of the events - DBSCAN

- The epsilon parameter in this case has an intuitive meaning (how close geographically the points have to be to be 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 tweets from almost everywhere, so we do not want to have just one cluster that connects everything
- Unfortunately the results are not so good, apart of taking a long time to obtain the results ($\sim$ 4 hours) only a few clusters are found and a large number of examples is discarded as noise

Clustering of the events - Leader

Leader algorithm (200m/500m radius)



Geographical profiles

- We want to find groups of users that have similar behavior (in place and time) along the period of data collection
- The clusters from the discretization can be the basis for the geographical profiles
- Additionally a discretization of time can help to have 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

Geographical profiles

- It is more difficult to choose what clustering algorithm is adequate
- We can explore different alternatives and analyze the results
- Some choices will depend on:
  - If the dataset generated is continuous or discrete
  - 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 (adaptative and non predefined shape)

K-means

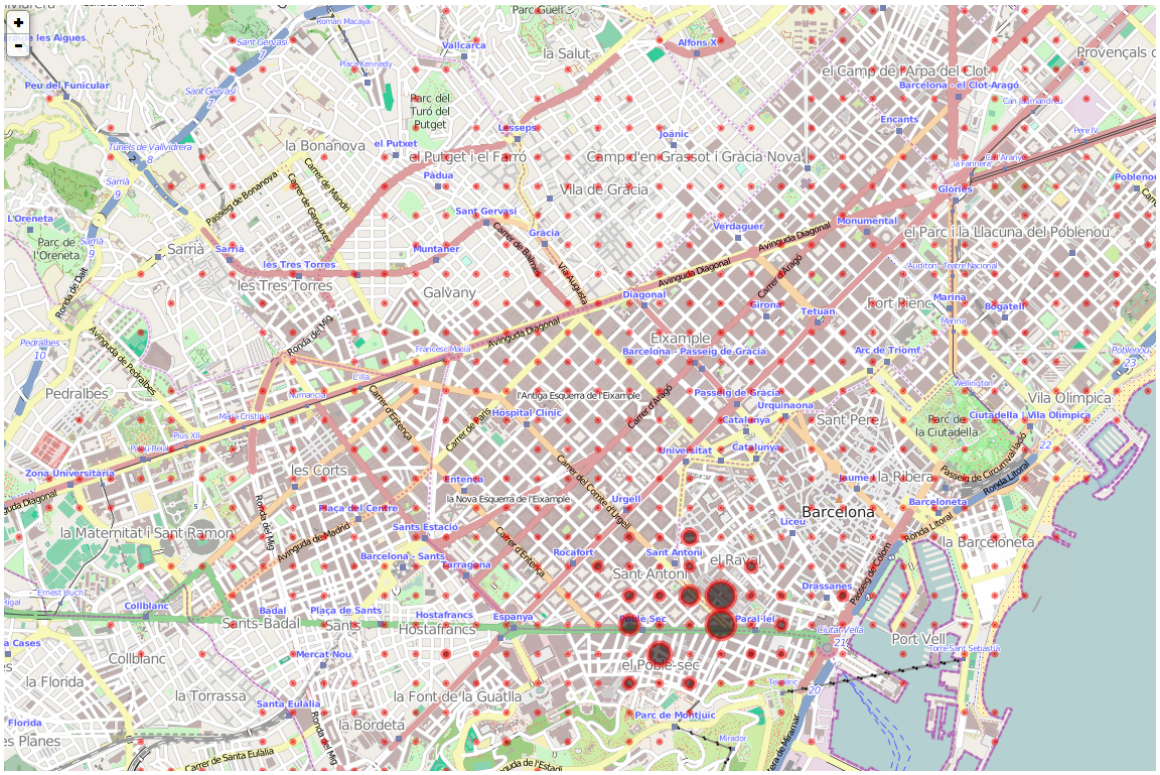
- Clustering results depend on the chosen representation
- Our assumption is that:
  - Several profiles should arise from the data
  - The number of people for each profile should be large
- A large value of k is used, very small clusters are bound to appear
- Most of clustering result in a very large cluster (most of the data) and a large number of very small ones
- The clustering that is closer to our assumptions is the one that uses binary representation normalized using TF-IDF
  - Relatively large number of clusters (around 25)
  - A large cluster, several medium sized clusters and a few small ones

K-means

Cluster example K=20

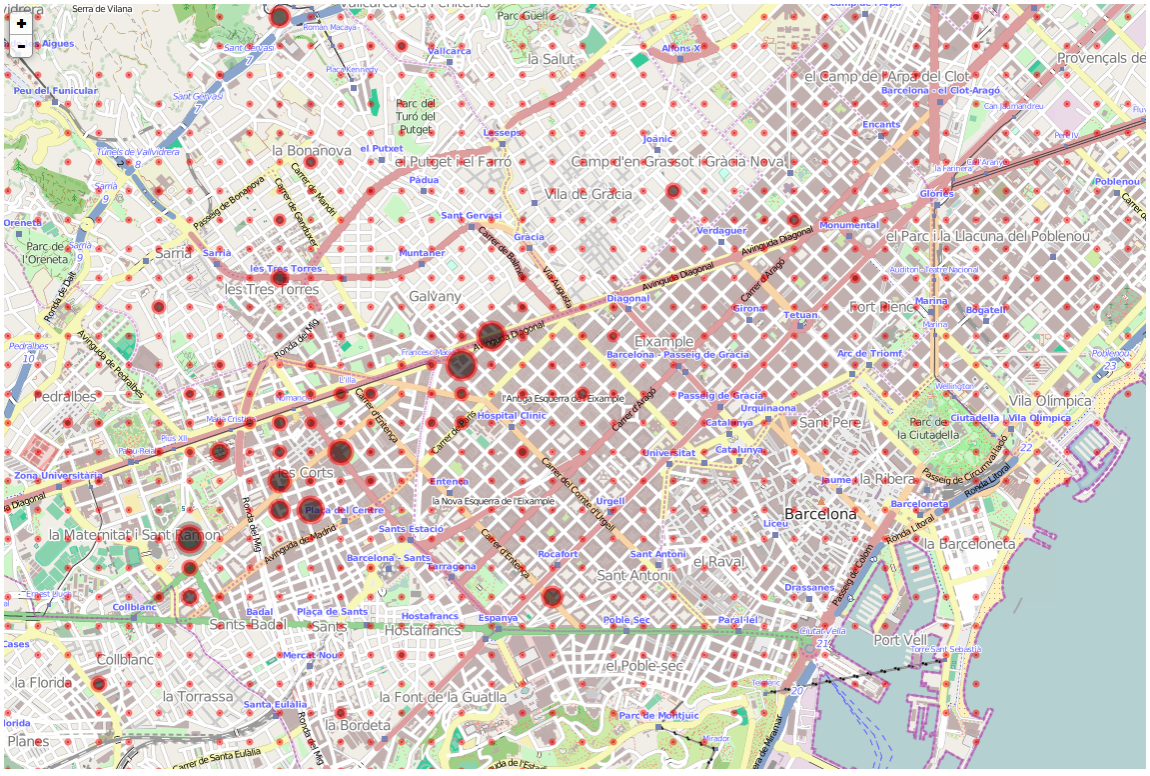
K-means

Cluster example K=20



K-means

Cluster example K=20

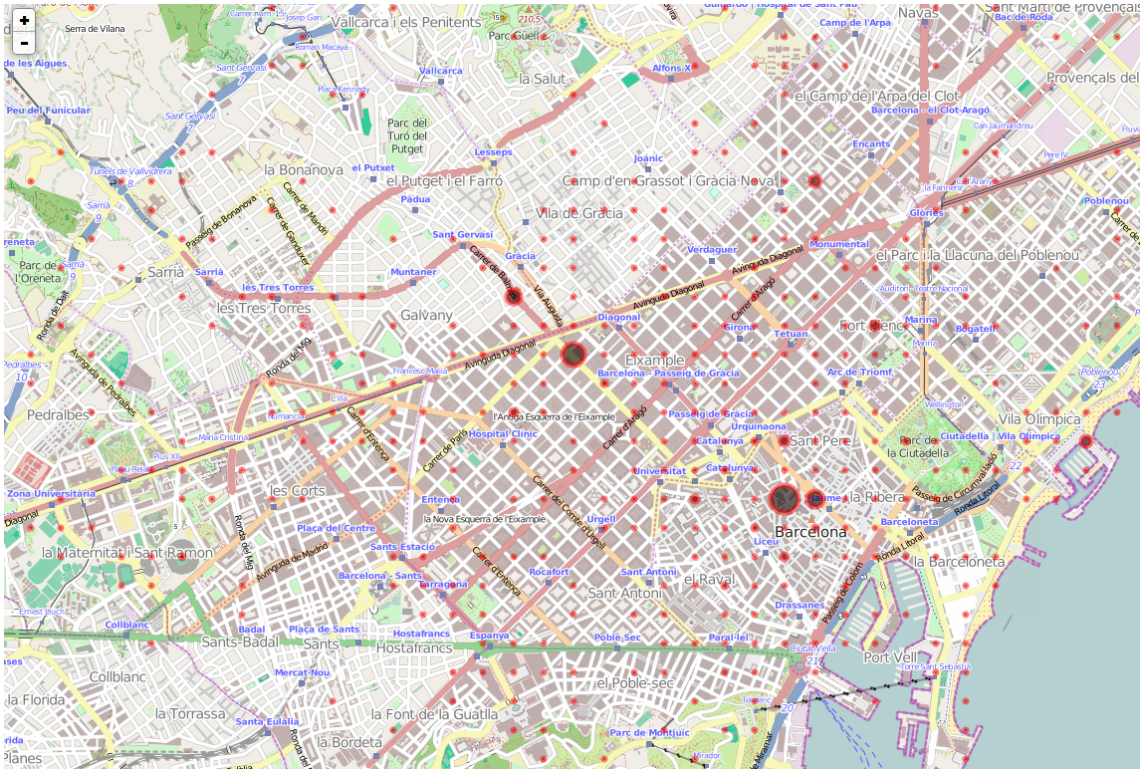


Affinity propagation

- The only parameter of the algorithm is the damping factor (explore the range of values)
- Like for K-means the results vary depending on the representation
- There are a slightly larger number of clusters and more distributed
- Also the clustering more close to our assumptions is the binary representation
- A larger number of clusters with more evenly distributed sizes

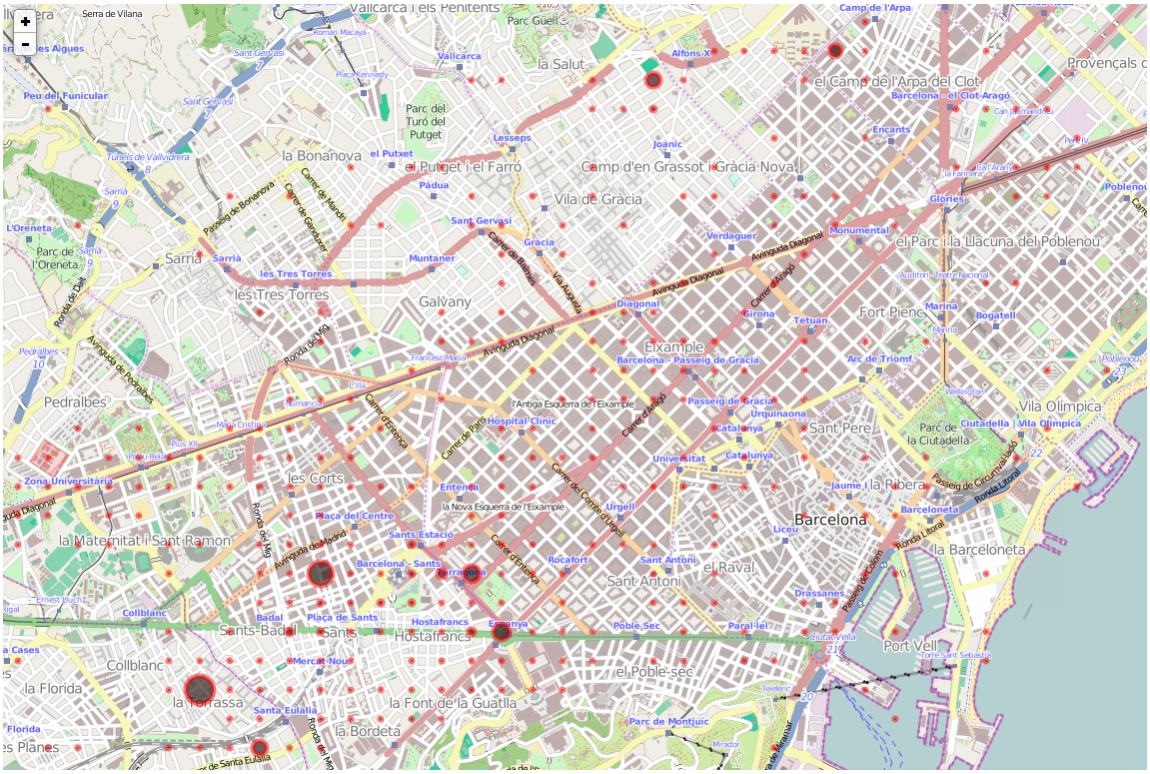
Affinity propagation

Cluster example



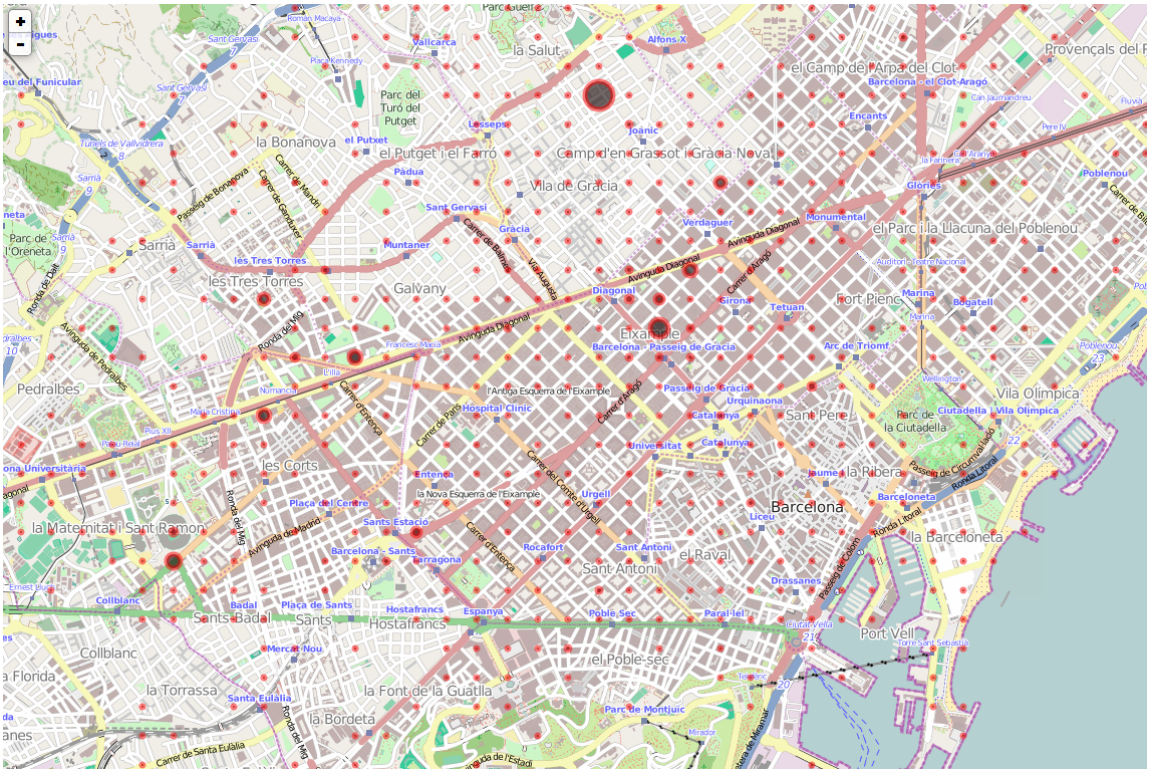
Affinity propagation

Cluster example



Affinity propagation

Cluster example



4

Cluster Validation

Clustering Evaluation/Model Assessment

Javier Béjar

CS - MAI

AMLT - 2016/2017

Outline

1. Cluster Evaluation
2. Internal criteria
3. External criteria
4. Number of clusters
5. Application
6. Cluster Visualization

1 Cluster Evaluation
2 Internal criteria
3 External criteria
4 Number of clusters
5 Application
6 Cluster Visualization

Model evaluation

- The evaluation of unsupervised learning is difficult
- There is no goal model to compare with
- The true result is unknown, it may depend on the context, the task to perform, ...
- Why do we want to evaluate them?
  - To avoid finding patterns in noise
  - To compare clustering algorithms
  - To compare different models/parameters

Model evaluation

What can be evaluated?

- Cluster tendency, there are clusters in the data?
- Compare the clusters to the true partition of the data
- Quality of the clusters without reference to external information
- Compare the results of different clustering algorithms
- Evaluate algorithm parameters
  - For instance, to determine the *correct* number of clusters

Model evaluation

Cluster Tendency

- Before clustering a dataset we can test if there are actually clusters
- We have to test the hypothesis of the existence of patterns in the data versus a dataset uniformly distributed (homogeneous distribution)
- **Hopkins Statistic**
  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 nearest neighbors in D ($d(q_i)$)
  3. Compute the quotient:

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

  4. If points are uniformly distributed the value of H will be around 0.5

Model evaluation

Cluster Quality criteria

- We can use different methodologies/criterion to evaluate the quality of a clustering:
  - Comparison with a model partition/labeled data (External criteria)
  - Quality measures based on the examples/quality of the partition (Internal criteria)
  - Comparison with other clusterings (Relative criteria)

1 Cluster Evaluation
2 Internal criteria
3 External criteria
4 Number of clusters
5 Application
6 Cluster Visualization

Internal criteria

- We can measure some properties that are expected in a good clustering
  - Compact groups
  - Well separated groups
- The indices are based on the model of the groups
- We can use indices based on the attributes values measuring the properties of a good clustering
- These indices are based on statistical properties of the attributes of the model
  - Values distribution
  - Distances distribution

Internal criteria - Indices

- Some of the indices correspond directly to the objective function optimized:
  - Quadratic error/Distorsion (k-means)

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

  - Log likelihood (Mixture of gaussians/EM)

Internal criteria - Indices

- For prototype based algorithms several measures can be use to compute quality indices
- Scatter matrices: interclass distance, intraclass distance, separation

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

Internal criteria - Indices

- Trace criteria (lower overall intracluster distance/higher overall intercluster distance)

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

- Determinant criteria (higher cross-intercluster distance)

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

Internal criteria - Indices

- Condorcet criteria (total interclass/intraclass distances)

$$\sum_{C_i \in C} \sum_{\substack{x_j, x_k \in C_i \\ x_j \neq x_k}} s(x_j, x_k) + \sum_{C_i \in C} \sum_{x_j \in C_i; x_k \notin C_i} s(x_j, x_k)$$

- Calinski-Harabasz index (interclass-intraclass distance ratio)

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

Internal criteria - Indices

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

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

where

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

Internal criteria - Indices

- 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

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

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

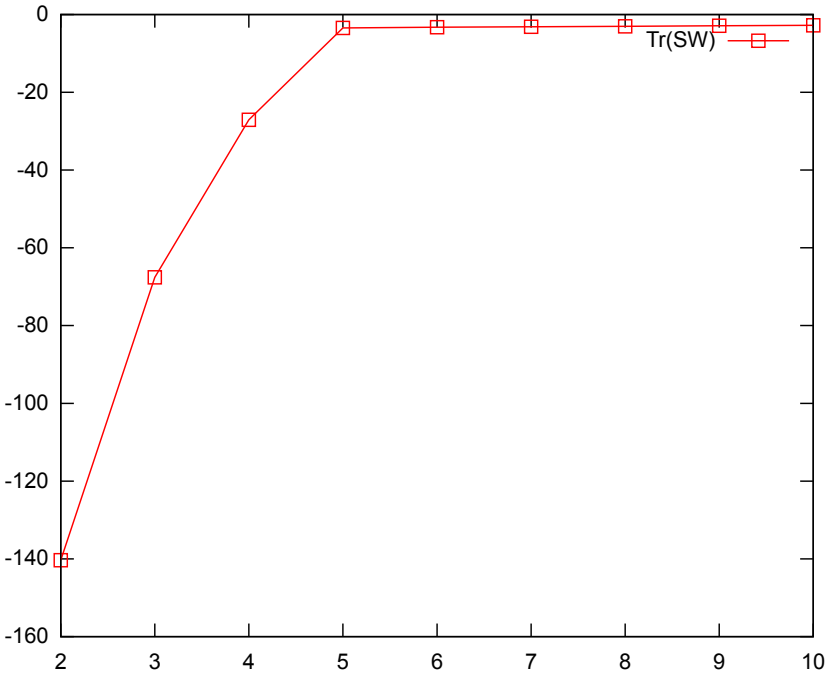
being $x_i \in C_j, H = \{h : 1 \leq h \leq K\}$

Internal criteria - Indices

- In the literature can be found more than 30 different indices
- Several studies and comparisons have been performed
- Recent studies (Arbelatiz et al, 2013) have exhaustively studied these indices, some have a performance significativelly better that others
- Some of the indices show a similar performance (not statistically different)
- The study concludes that Silhouette, Davies-Bouldin and Calinski Harabasz perform well in a wide range of situations

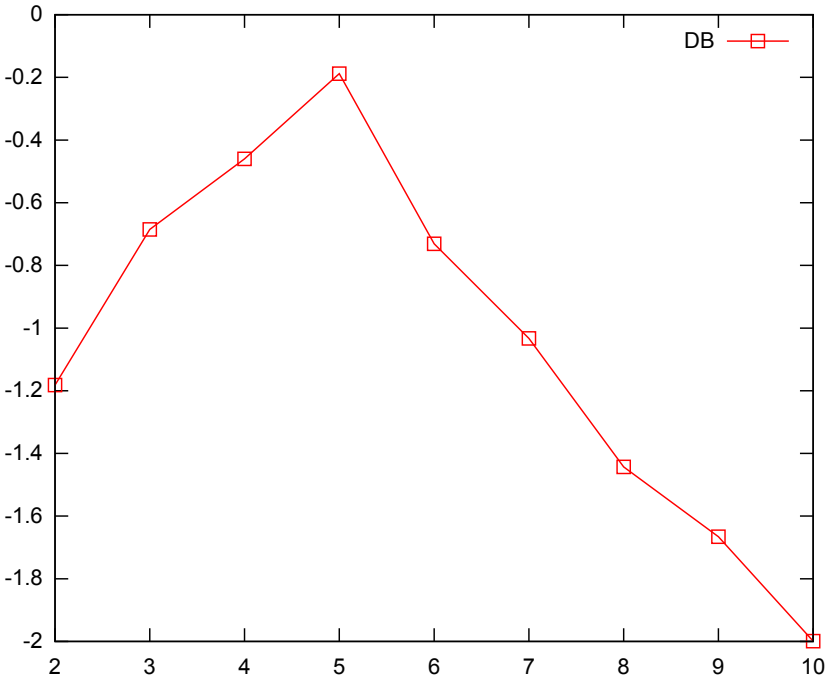
Internal criteria - Interclass distance

Data with 5 clusters



Internal criteria - Davies-Bouldin

Data with 5 clusters



1 Cluster Evaluation

2 Internal criteria

3 External criteria

4 Number of clusters

5 Application

6 Cluster Visualization

External criteria

- These indices measure if a clustering is similar to a model partition P
- It is equivalent to have a labeled dataset (ground thruth)
- If we do not have a model these criteria can be used to compare the results of using different parameters of different algorithms
  - For instance, can be used to assess the sensitivity to initialization
- The main advantage is that these indices are independent of the examples/cluster description
- These means that can be used to assess any clustering algorithm

External criteria - Indices

- 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:
  - 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)

External criteria - Indices

- Rand/Adjusted Rand statistic:

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

- Jaccard Coefficient:

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

- Folkes and Mallow index:

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

External criteria - Indices - Information Theory

- Defining 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 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

External criteria - Indices - Information Theory

- 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:

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

1 Cluster Evaluation

2 Internal criteria

3 External criteria

4 Number of clusters

5 Application

6 Cluster Visualization

Number of clusters

- A topic related to cluster validation is to decide if the number of clusters obtained is the correct one
- This point 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 sizes
- Usually internal criteria indices are used in this comparison
- A graphic of this indices for different number of clusters can show what number of clusters is more probable

Number of clusters - Indices

- Some of the internal validity indices can be used for this purpose:
  - Calinsky Harabasz index
  - Silhouette index
- Using the within class scatter matrix (S_W) other criteria can be defined:
  - Hartigan index:

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

  - Krzanowski Lai index:

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

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

Number of clusters - The Gap Statistic

- Assess the number of clusters comparing a clustering with the expected distribution of data given the null hypothesis (no clusters)
- Different clusters of the data increasing the number of clusters and compared to datasets (B) generated with a uniform distribution
- The Gap statistic:

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

- From the st. dev. (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:

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

Number of clusters - Cluster Stability

- The idea 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 their stability
- Using disjoint samples:
  - The dataset is divided in two disjoint samples that are clustered separately
  - Indices can be defined to assess stability, for example using the distribution of the number of neighbors that belong to the complementary sample
- Using non disjoint samples:
  - The dataset is divided in three disjoint samples (S_1,S_2, S_3)
  - Two clusterings are obtained from $S_1 \cup S_3$, $S_2 \cup S_3$
  - Indices can be defined about the coincidence of the common examples in both partitions

Python Notebooks

This Python Notebook has examples for Measures of Clustering Validation

- Clustering Validation Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

Python Code

- In the code from the repository inside subdirectory `Validation` you have the python program `ValidationAuthors`,
- The `authors` dataset is clustered with different algorithms (K-means, GMM, Spectral) and different validity indices are plotted for the number of clusters

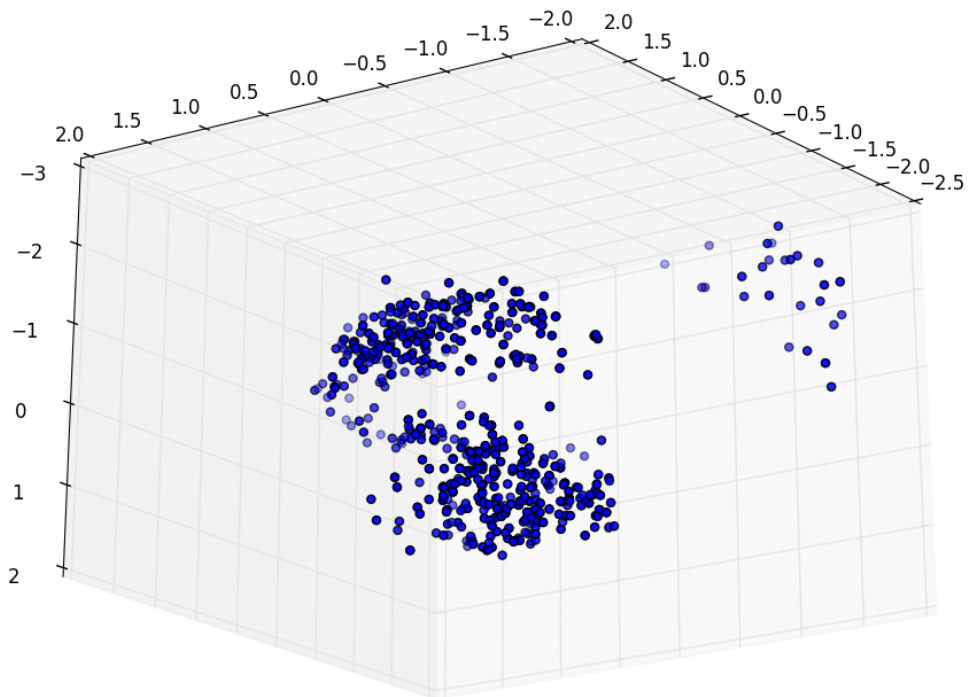
1 Cluster Evaluation
2 Internal criteria
3 External criteria
4 Number of clusters
5 Application
6 Cluster Visualization

Application: Wheelchair Control Dataset

- Wheelchair with shared control (patient/computer)
- Recorded trajectories of several patients in different situations
  - Angle/distance to the goal, Angle/distance to the nearest obstacle from around the chair (210 degrees)
- Characterization about how the computer helps the patients with different handicaps
- How many clusters characterize the trajectories?

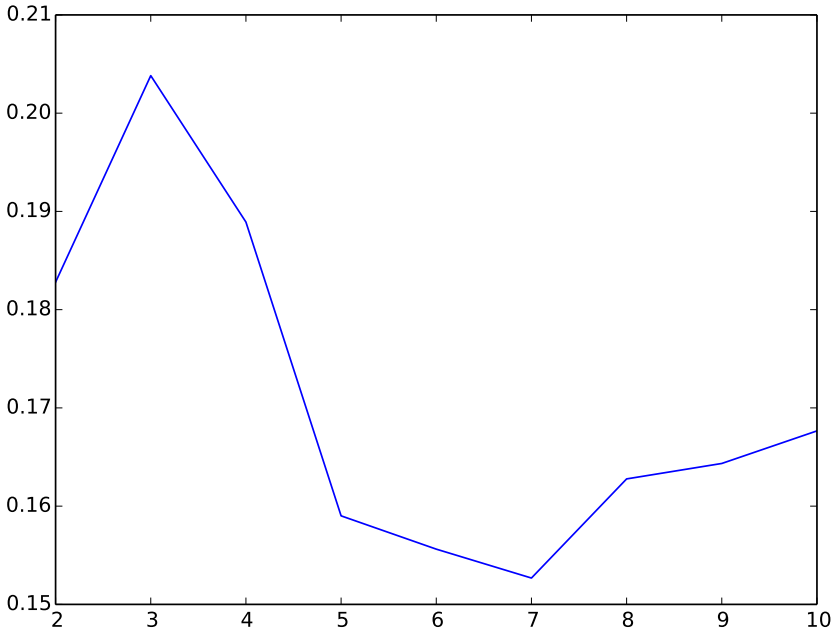
Wheel chair control characterization

PCA 88 $\Longrightarrow$ 3 dimensions



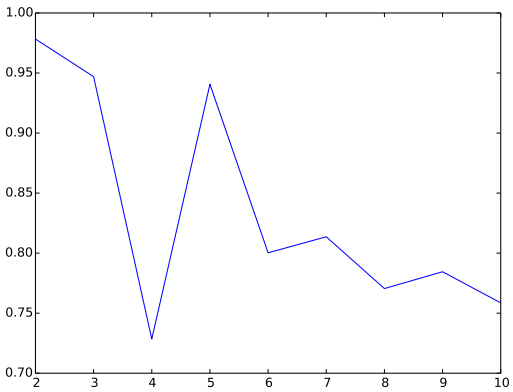
Wheel chair control characterization

Silhouette index for k-means

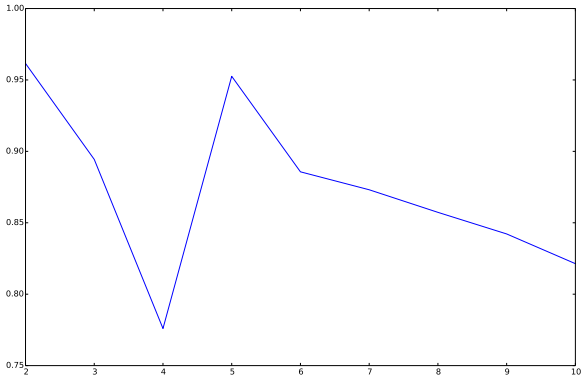


Wheel chair control characterization

Stability/Consensus ARAND and ANMI for k-means



ARAND



ANMI

1 Cluster Evaluation
2 Internal criteria
3 External criteria
4 Number of clusters
5 Application
6 Cluster Visualization

Cluster visualization

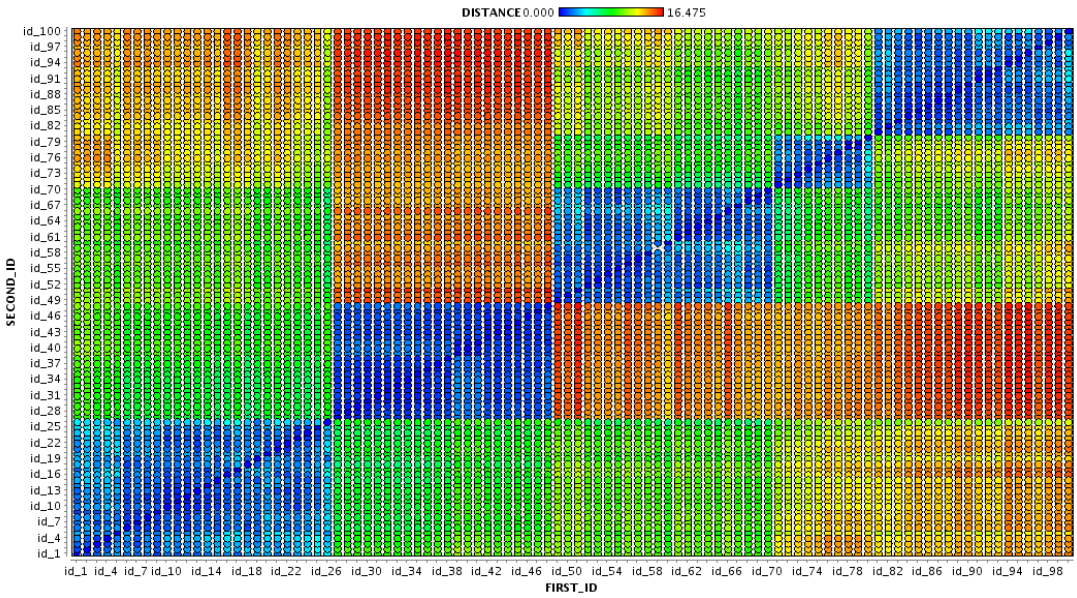
- Another method for model assessment is to visualize the data and look for clusters
- Dimensionality reduction
  - Project the dataset to 2 or 3 dimensions
  - The clusters in the new space could represent clusters in the original space
  - The confidence depends on the reconstruction error of the transformed data and that the transformation maintains the relations in the original space
- Distance matrix visualization
  - The distance matrix represents the examples relationships
  - Can be rearranged so the closer examples appear in adjacent columns
  - Patterns in the rearranged matrix can show cluster tendency
- Both methodologies are computationally expensive

Cluster visualization - Distance matrix

- There are several methods
- The simplest one is to use a hierarchical clustering algorithm and rearrange the matrix using a inorder traversal of the tree
- Results will depend on the algorithm used and the distance/similarity function
- Can be applied to quantitative and qualitative data
- See patterns in the distance matrix is not always guarantee of clusters in the data

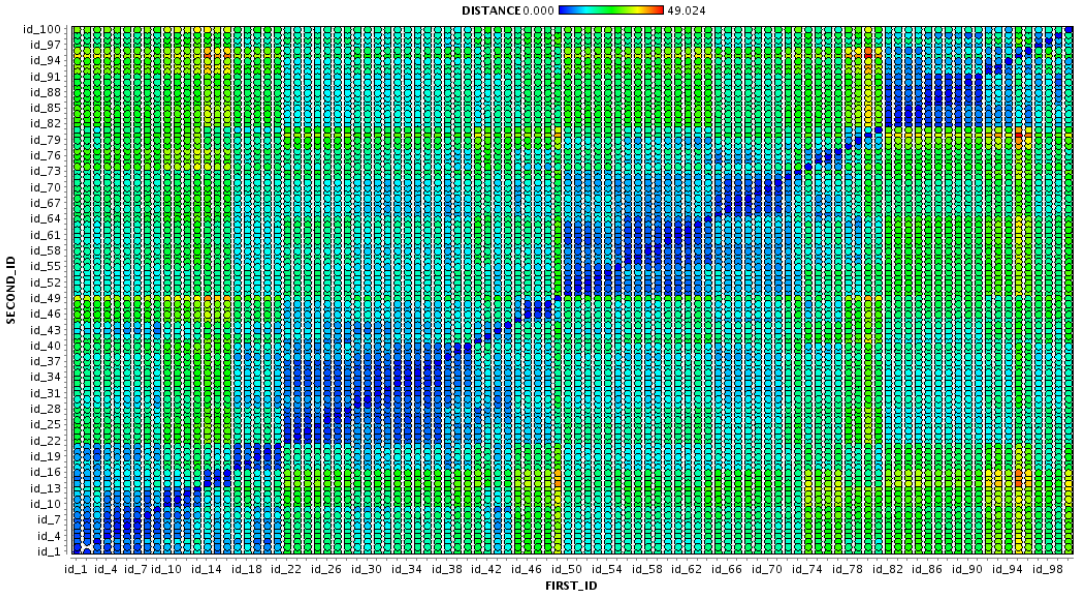
Cluster visualization - Distance matrix

Dataset with five well separated clusters



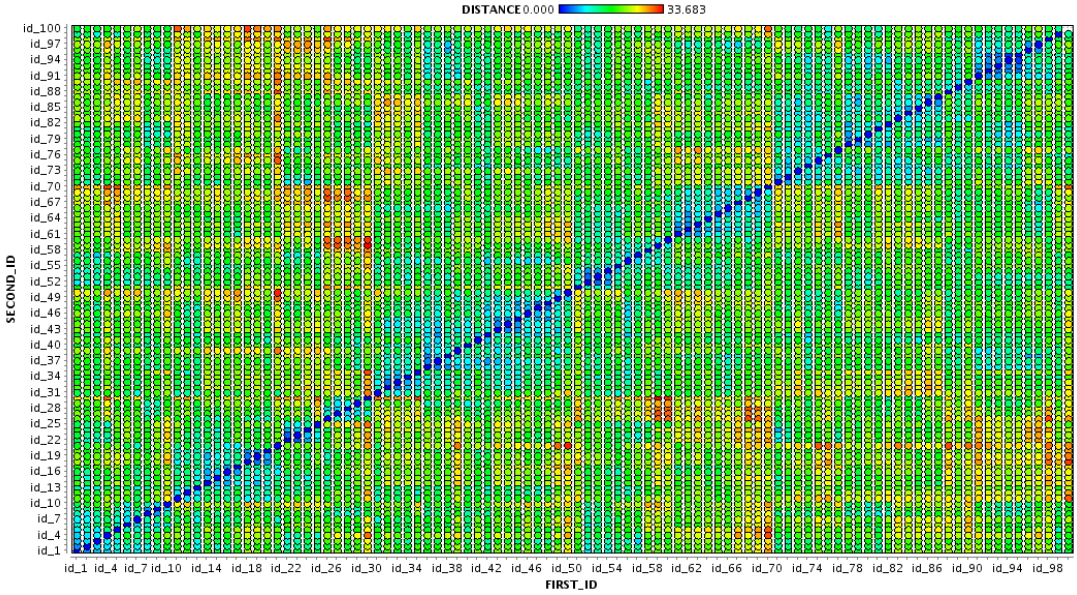
Cluster visualization - Distance matrix

Dataset with five noisy and overlapping clusters



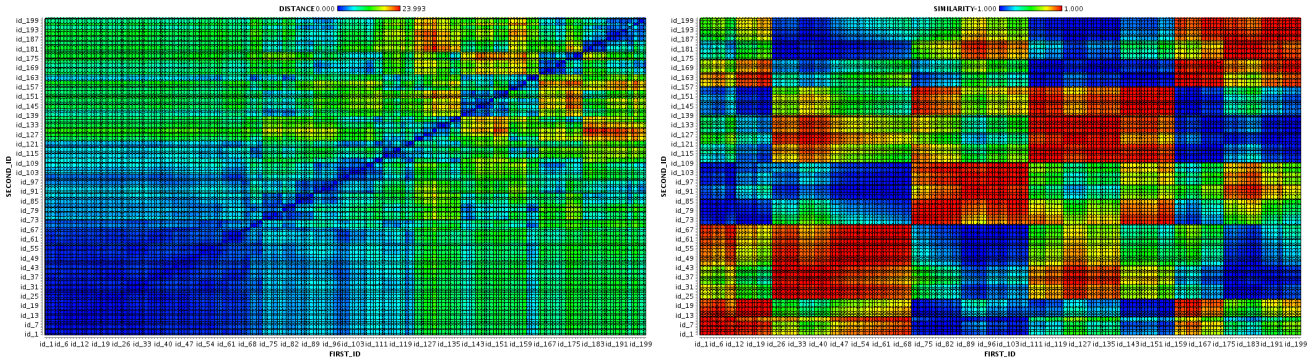
Cluster visualization - Distance matrix

Random Data



Cluster visualization - Distance matrix

Three rings dataset (euclidean distance, cosine similarity)



5

Clustering of Large Datasets

Clustering in KDD

Javier Béjar

CS - MIA

AMLT - 2016/2017

Outline

Clustering in KDD

- One of the main tasks in the KDD process is the analysis of data when we do not know its structure
- This task is very different from the task of prediction in which we know the goal function and we try to approximate it
- A great part of the KDD tasks are non supervised problems (KDNuggets poll, 2-3 most frequent task)
- Problems: Scalability, arbitrary cluster shapes, limited types of data, finding the correct parameters, ...
- There are some new algorithms that deal with these kind of problems

Strategies for cluster scalability

- One-pass
  - Process data as a stream
- Summarization/Data compression
  - Compress examples to fit more data in memory
- Sampling/Batch algorithms
  - Process a subset of the data and maintain/compute a global model
- Approximation
  - Avoid expensive computations by approximate estimation
- Paralelization/Distribution
  - Divide the task in several parts and merge models

One pass

- This strategy is based on incremental clustering algorithms
- They are cheap but order of processing affects greatly their quality
- Although they can be used as a preprocessing step
- Two steps algorithms
  1. A large number of clusters is generated using the one-pass algorithm
  2. A more accurate algorithm clusters the preprocessed data

Data Compression/Summarization

- Not all the data is necessary to discover the clusters
- Discard sets of examples and summarize by:
  - Sufficient statistics
  - Density approximations
- Discard data irrelevant for the model (do not affect the result)

Approximation

- Not using all the information available to make decisions
  - Using K-neighbours (data structures for computing k-neighbours)
- Preprocessing the data using a cheaper algorithm
  - Generate batches using approximate distances (eg: canopy clustering)
- Use approximate data structures
  - Use of hashing or approximate counts for distances and frequency computation

Batches/Sampling

- Process only data that fits in memory
- Obtain from the data set:
  - Samples (process only a subset of the dataset)
    - Determine the size of the sample so all the clusters are represented
  - Batches (process all the dataset)

Paralelization/Distribution/Divide&Conquer

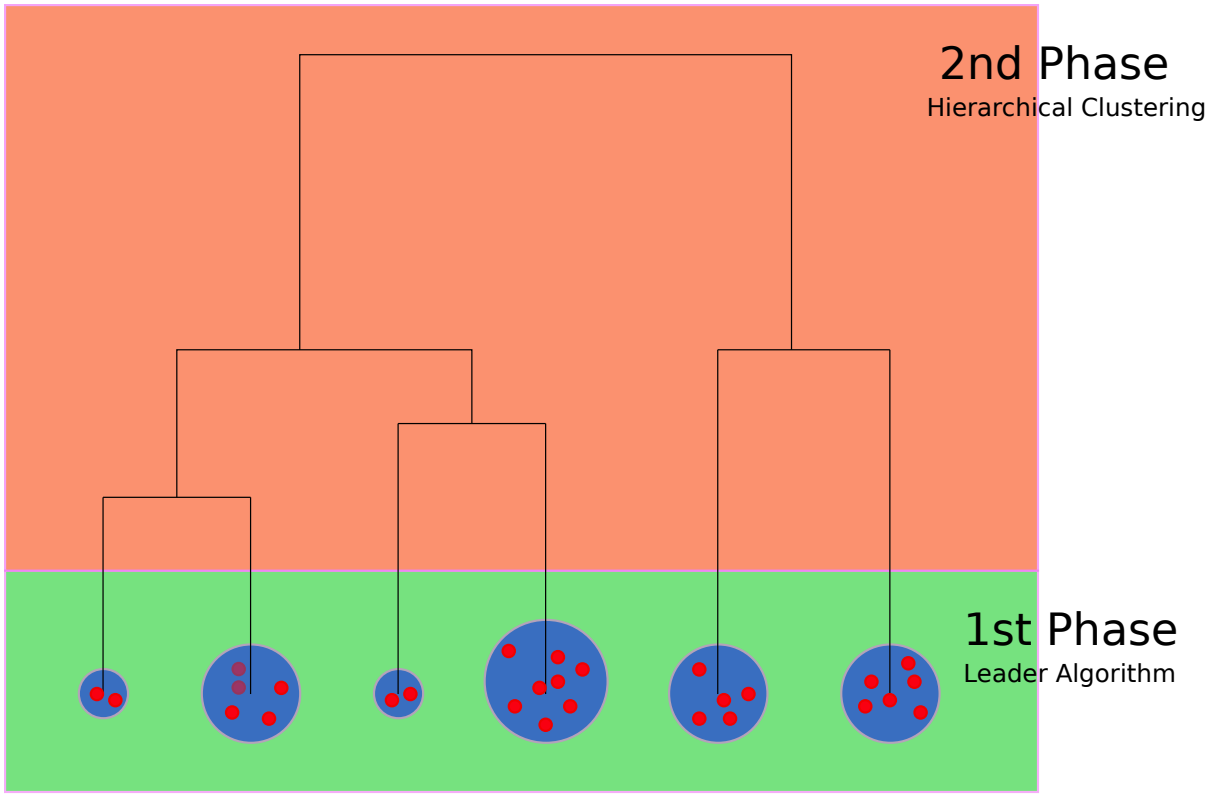
- Paralelization of clustering usually depends on the specific algorithm
- Some are difficult to parallelize (eg: hierarchical clustering)
- Some have specific parts that can be solved in parallel or by Divide&Conquer
  - Distance computations in k-means
  - Parameter estimation in EM algorithms
  - Grid density estimations
  - Space partitioning
- Batches and sampling are more general approaches
  - The problem is how to merge all the different partitions

Scalable Hierarchical Clustering

Patra, Nandi, Viswanath, **Distance based clustering method for arbitrary shaped clusters in large datasets** Pattern Recognition, 2011, 44, 2862-2870

- Strategy: One pass + Summarization
- The leader algorithm is used as a one pass summarization using Leader algorithm (many clusters)
- Single link is used to cluster the summaries
- Theoretical results guarantee the equivalence to SL at top levels
- Summarization makes the algorithm independent of the dataset size (depends on the radius used on the leader algorithm and the volume of the data)
- Complexity $O(c^2)$

One pass + Single Link



BIRCH

Zhang, Ramakrishnan, Livny **BIRCH: An Efficient Data Clustering Method for Very Large Databases** (1996)

- Strategy: One-pass + Summarization
- Hierarchical clustering with limited memory
- Incremental algorithm
- Based on probabilistic prototypes and distances
- We need two pass from the database
- Based on an specialized data structure named CF-tree (Clustering Feature Tree)

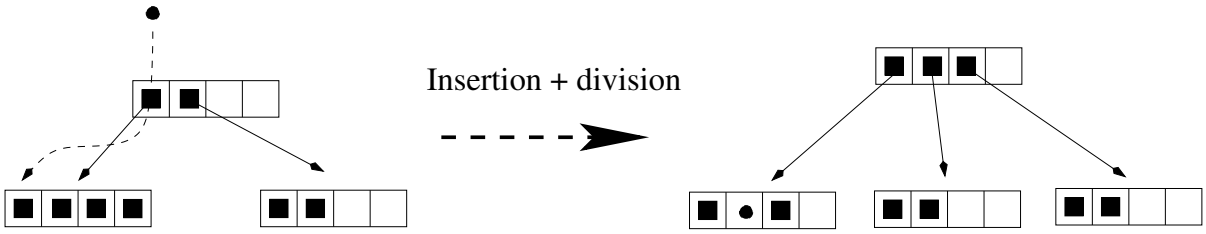
BIRCH (CF-tree)

- It is a balanced n-ary tree that contains groups represented by probabilistic prototypes
- Leaves can contain as much as L prototypes and its 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 the parameters is crucial because we could fill all available space during the process
- This can be solved by changing the parameters values (basically T) and recompressing the tree. In fact T determines the granularity of the final groups

BIRCH - Insertion algorithm

1. Traverse the tree until a leave is found and choose the nearest prototype
2. On this leave we could introduce the instance in an existing group or create a new prototype depending on if the distance is greater than parameter T or not
3. If the current leave has not space for the new prototype we proceed to create a new terminal node and we distribute the prototypes among the current node and the new node
4. The distribution is performed choosing the two most different prototypes and dividing the rest using their proximity to these two prototypes
5. This division will create a new node in the ascendant node, if the new node exceeds the capacity of the father we will split it and we will continue the process until the root of the tree is reached if necessary
6. Additionally we could perform merge operations to compact the tree and reduce space

BIRCH - Insertion Algorithm

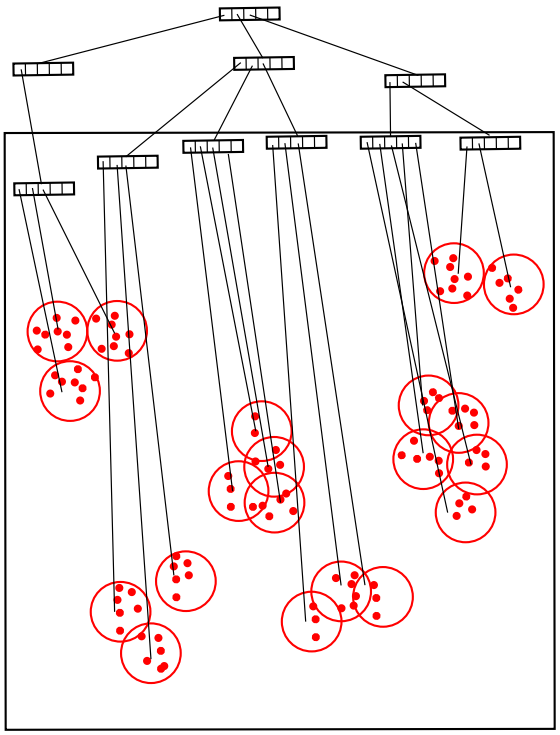


BIRCH - Clustering algorithm

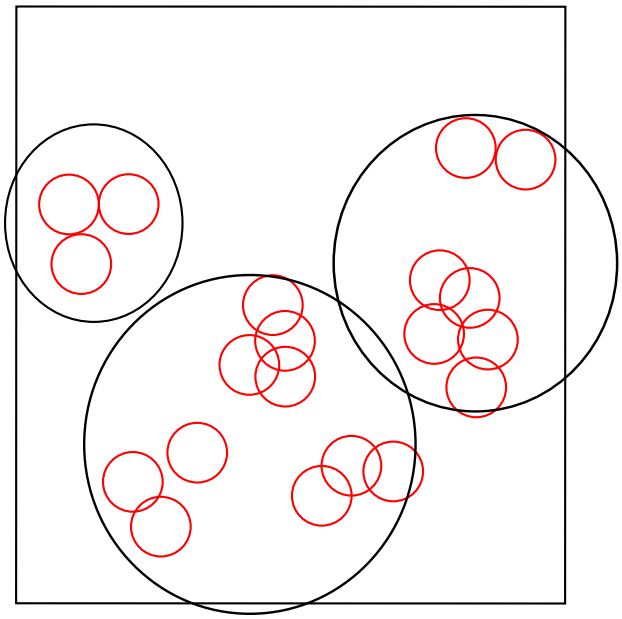
1. **Phase 1:** Construction of the CF-tree, we obtain a hierarchy that summarizes the database as a set of groups which granularity is defined by T
2. **Phase 2:** Optionally we modify the CF-tree in order to reduce its size by merging near groups and deleting outliers
3. **Phase 3:** We use the prototypes inside the leaves of the trees as new instances and we run a clustering algorithm with them (for instance K-means)
4. **Phase 4:** We refine the groups assigning the instances from the original database to the prototypes obtained in the previous phase

One pass + CFTREE (BIRCH)

1st Phase - CFTree



2nd Phase - Kmeans



Scalable K-means clustering

Bradley, Fayyad, Reina, **Scaling Clustering Algorithms to Large Databases**, Knowledge Discovery and Data Mining (1998)

- Strategy: Sampling + Summarization
- Clustering algorithms need to have all data in main memory to perform their task
- We try to obtain scalability looking for an algorithm that:
  - Only look at the data one time
  - To be anytime (always a result is available)
  - To be incremental (to cluster more data we do not need to start from scratch)
  - To be suspendable (we can restart the process from the actual state)
  - To use limited memory

Scalable K-means clustering (Algorithm)

- Obtain a sample that fits in memory
- Update the actual model
- Classify new instances as:
  - Necessary
  - Discardable (We keep their information as sufficient statistics)
  - Summarizable using data compression
- Decide if the model is stable or we keep clustering more data

Canopy Clustering

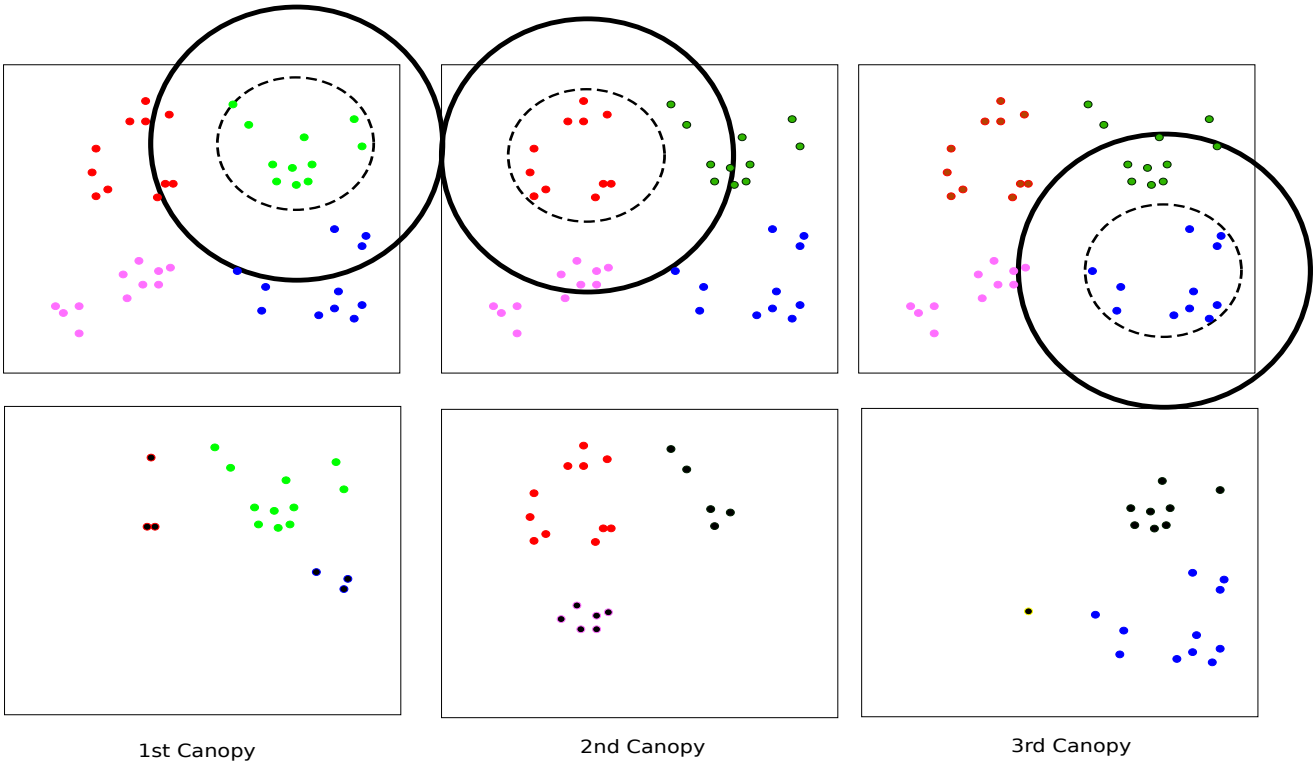
McCallum, Nigam, Ungar **Efficient clustering of high-dimensional data sets with application to reference matching** (2002)

- Strategy: Divide&Conquer + Approximation
- The approach is based on a two stages clustering
- The first stage can be seen as a preprocess determining the neighborhood of the densities of examples and reducing the number of distances that have to be computed on the second stage
- This first stage is the called *canopy clustering*, relies on a cheap distance and two parameters $T_1 > T_2$
- This parameters are used as two centered spheres that determine how to classify the examples.

Canopy Clustering - Algorithm

- Algorithm:
  1. One example is picked at random and the cheap distance from this example to the rest is computed
  2. All the examples that are at less than T2 are deleted and included in the canopy
  3. The points at less than T1 are added to the canopy of this examples without deleting them
  4. The procedure is repeated until the example list is empty
  5. Canopies can share examples
- After this procedure the examples can be clustered using different algorithms
- For agglomerative clustering only the distances among the examples in the canopies have to be computed

Canopy Clustering - Algorithm



Canopy Clustering - Algorithm

- For EM and K-means there are different methods:
  - Method 1: Prototypes are associated with canopies, only distances inside a canopy are computed and only the examples inside a canopy can influence a prototype. The number of prototypes per canopy have to be decided. The EM or K-means only use for each prototypes the examples of the associated canopies
  - Method 2: Use a set of prototypes covering the dataset and then distribute them in the canopies. We allow data in other canopies to influence the prototypes but only using the mean of their canopies.
  - Method 3: We allow to create or merge prototypes inside a canopy avoiding to have prototypes that cover examples that belong to more than one canopy.

Mini-batch K-means

Sculley, **Web-scale k-means clustering** Proceedings of the 19th international conference on World wide web, 2010, 1177-1178

- Strategy: Sampling
- Apply K-means to a sequence of bootstrap samples of the data
- Each iteration the samples are assigned to prototypes and the prototypes are updated with the new sample
- Each iteration the weight of the samples is reduced (learning rate)
- The quality of the results depends on the size of the batches
- Convergence is detected when prototypes are stable

Mini-batch K-means (algorithm)

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

CURE

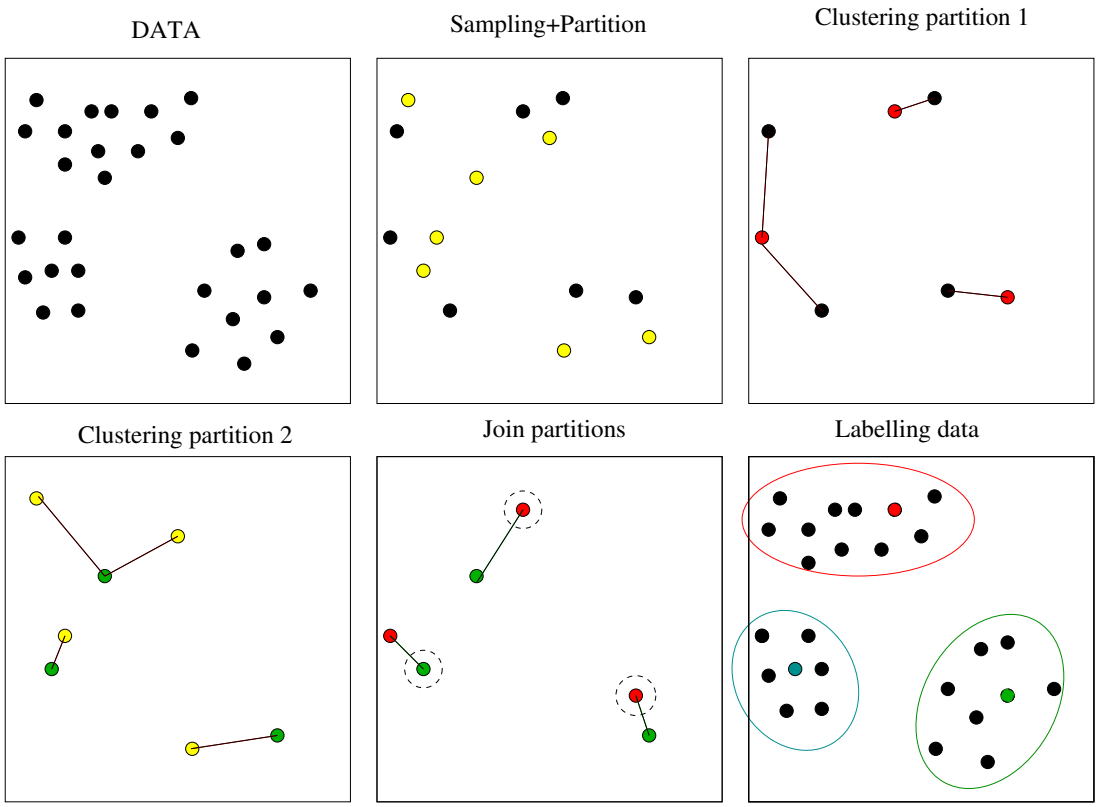
Guha, Rastogi, Shim **CURE: An efficient clustering algorithm for large databases** (1998)

- Strategy: Sampling + Divide&Conquer
- It uses hierarchical agglomerative clustering
- It is scalable and is capable of treating outliers
- Scalability is obtained using sampling techniques and partitioning the dataset
- It uses a set of representatives (c) for cluster instead of centroids, this allows to find non spherical groups
- The distance is computed as the nearest pair of representatives among groups
- The clustering algorithm is agglomerative and merges pairs of groups until k groups are obtained

CURE - Algorithm

1. Draws a random sample from the dataset
2. Partitions the sample in p groups
3. Executes the clustering algorithm on each partition
4. Deletes outliers
5. Runs the clustering algorithm on the union of all groups until it obtains k groups
6. Label the data accordingly to the similarity to the k groups

CURE - Algorithm



Rough-DBSCAN

Viswanath, Babu, **Rough-DBSCAN: A fast hybrid density based clustering method for large data sets** Pattern Recognition Letters, 2009, 30, 1477 - 1488

- Strategy; One-pass + Summarization
- Two stages algorithm:
  1. Preprocess using the leader algorithm
     - Determine the instances that belong to the higher densities and their neighbours
  2. Apply DBSCAN algorithm
     - Determine the densities for the selected instances
     - Approximate the values of the densities from their distances and the sizes of the neighbor
     - Assign the neighbors accordingly to the found densities

Python Notebooks

This Python Notebook has examples comparing K-means algorithm with two scalable algorithms Mini Batch K-means and BIRCH

- Clustering DM Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

6

Semisupervised Clustering

Semi-supervised Clustering

Javier Béjar

CS - MAI

AMLT - 2016/2017

Outline

Semi-supervised 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)

Semi-supervised Clustering

- 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

How to use supervised information

- 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

Semi supervised clustering using labeled examples

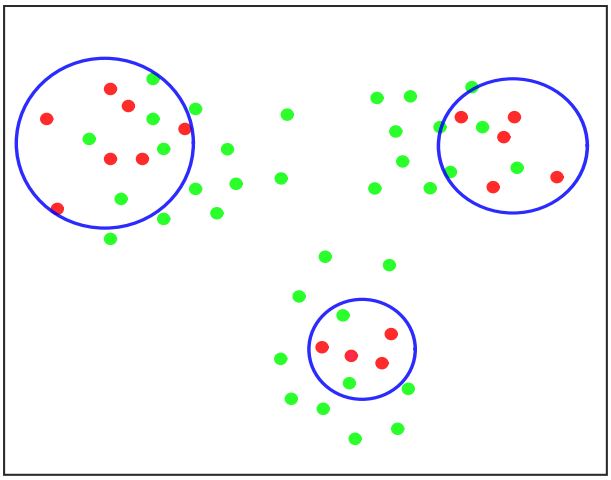
- Assuming that we have some labeled examples, these can be used to obtain an initial model
- We only have to know what examples belong to clusters, the actual clusters are not needed
- We can begin from this model the clustering process
- This means that we decide the starting point of the search using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

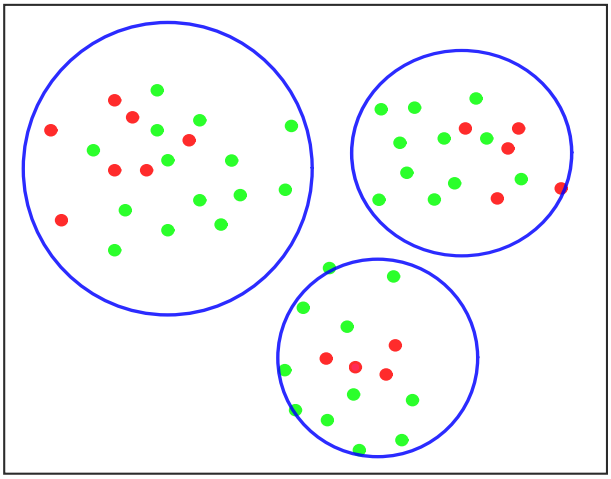
Semi supervised clustering using labeled examples

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

Semi supervised clustering using labeled examples





Seeded-KMeans

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 (μ_i) using the labeled instances

  repeat

    Assign each example from $\mathcal{X}$ to their nearest prototype μ_i

    Recompute the prototype μ_i with the examples assigned

  until *Convergence*

end

Constrained-KMeans

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 (μ_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 μ_i

    Recompute the prototype μ_i with the examples assigned

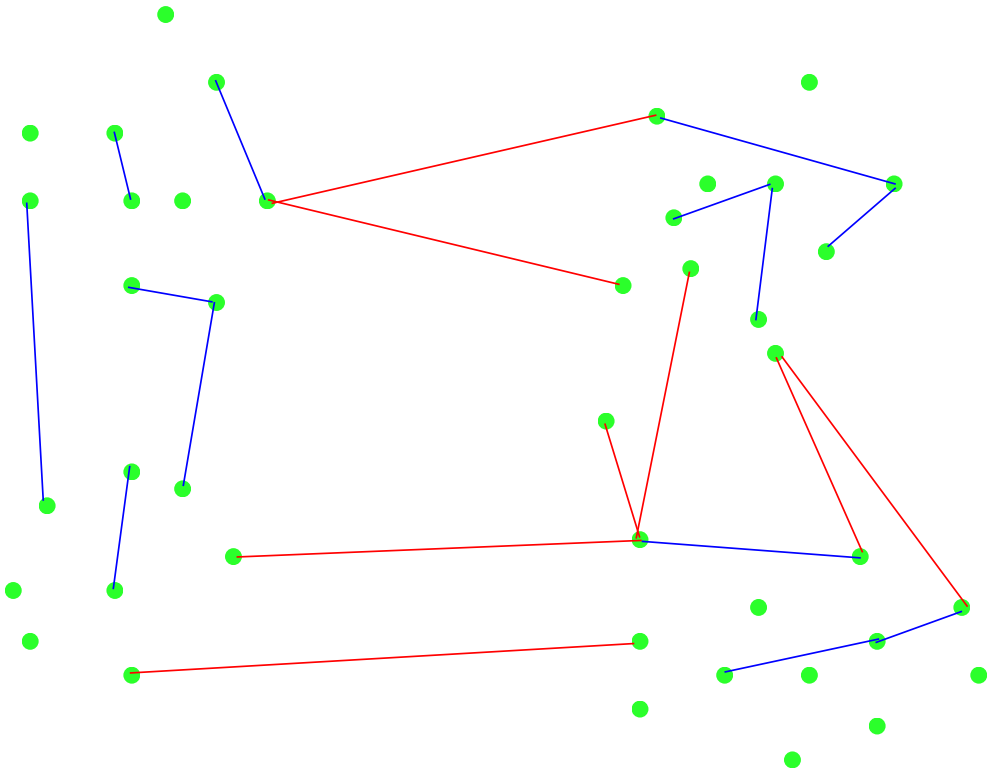
  until *Convergence*

end

Semi supervised clustering using 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

Semi supervised clustering using constraints



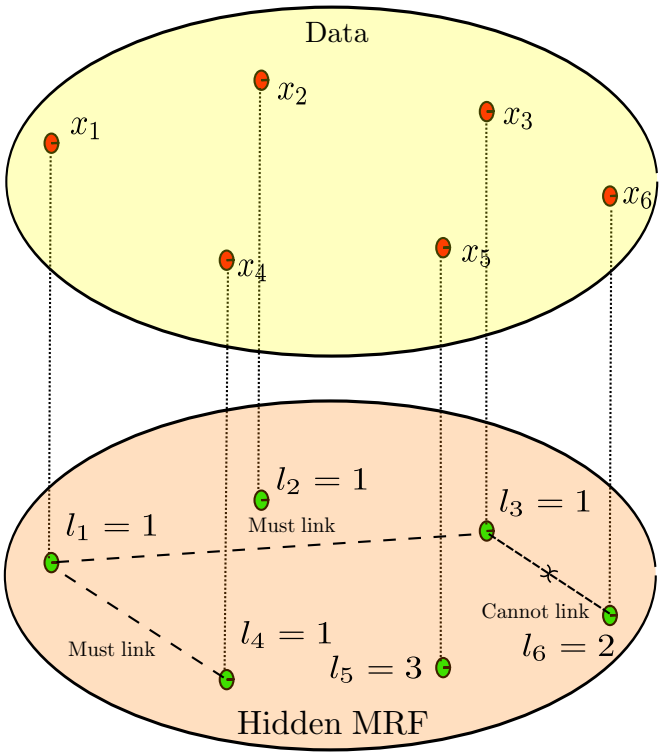
Semi supervised clustering using constraints

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

Semi supervised clustering using 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



Semi supervised clustering using constraints

- 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

Semi supervised clustering using constraints

- The new objective function is:

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

  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

HMRF-KMeans

- 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

HMRF-KMeans

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 (μ_i) using constraints

repeat

`E-step:` Reassign the labels of the examples using the prototypes (μ_i) to minimize $\mathcal{J}_{obj}$

`M-step:` Given the cluster labels recalculate cluster centroids to minimize $\mathcal{J}_{obj}$

until *Convergence*

end

Semi supervised clustering using constraints

- 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)

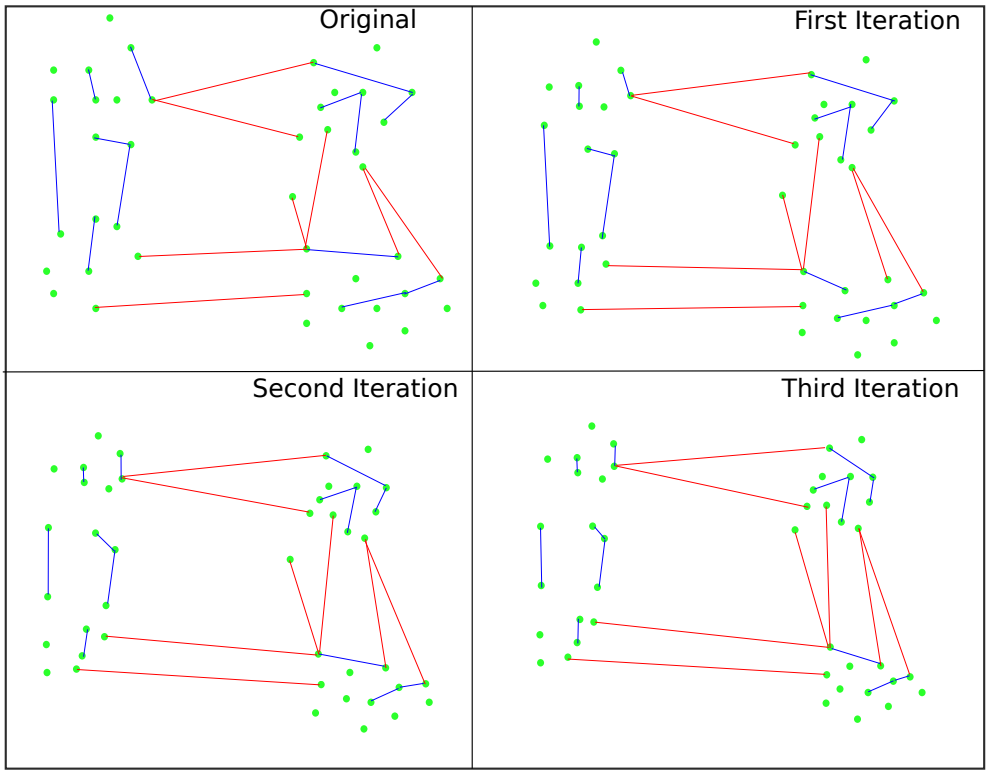
Semi supervised clustering with Distance Learning

- 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

Semi supervised clustering with Distance Learning

- 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)

Semi supervised clustering with Distance Learning



7

Association Rules

Association rules

Javier Béjar

CS - MAI

AMLT - 2016/2017

Outline

1. Association Rules
   - Introduction
   - Definitions
2. Apriori algorithm
   - Finding association rules
   - Algorithm
   - Computational Cost
3. FP-Growth
   - FP-tree
4. Measures of interestingness
5. Twitter - City Connections

1 Association Rules

2 Apriori algorithm

3 FP-Growth

4 Measures of interestingness

5 Twitter - City Connections

Association rules

- A **Binary** database is a database where each row is composed of binary attributes

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

- From this database frequent patterns of occurrence can be discovered

Association rules

- This kind of databases appear for example on transactional data (market basket analysis)
- An association rule represents coocurence of events in 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

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

Definitions (I)

- We define R as the set of attributes of the database
- We define X as a subset of attributes from R.
- We say that X is a **pattern** from DB if there is any row where all the attributes of X are 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 BD.

Definitions (II)

- 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 :

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

Definitions (III)

- Given a BD with R attributes and X and Y attribute subsets, with $X \cap Y = \emptyset$, an **association rule** is the expression:

$$X \Rightarrow Y$$

- $conf(X \Rightarrow Y)$ 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 ($min\_conf \in [0, 1]$)

1 Association Rules

2 Apriori algorithm

3 FP-Growth

4 Measures of interestingness

5 Twitter - City Connections

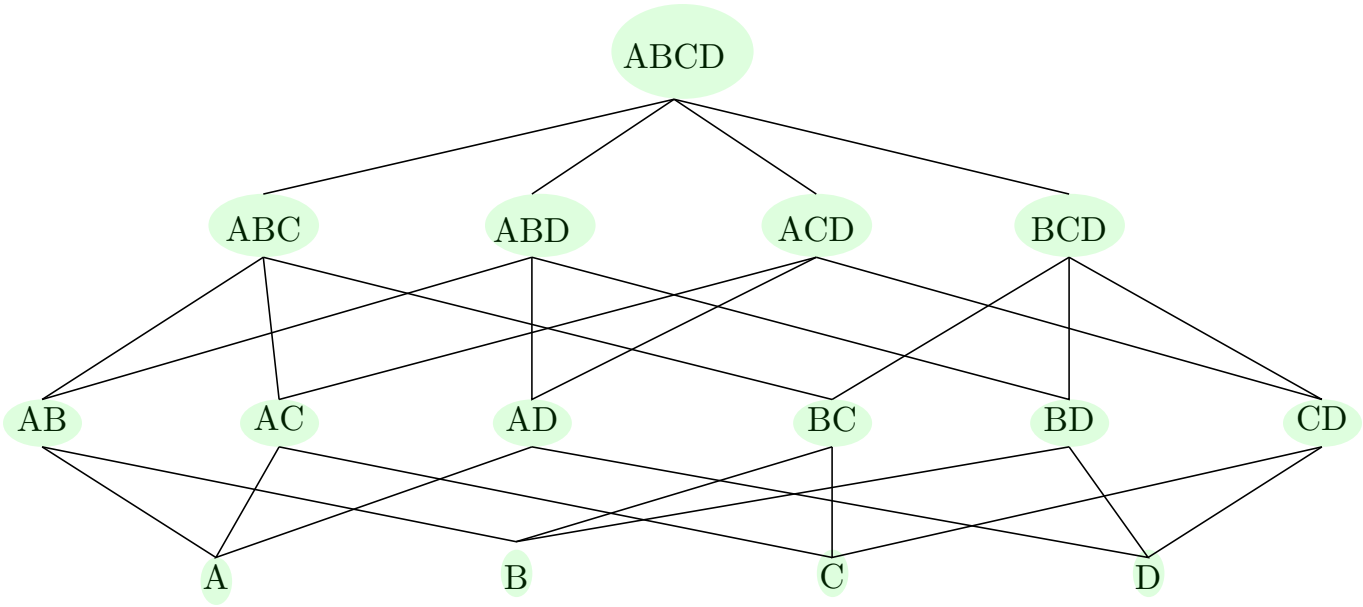
Finding association rules

- Given a minimum confidence and a minimum support, a association rule $(X \Rightarrow Y)$ exists in a DB if:

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

- Trivial approach:
  - It is enough to find all frequent subsets from R
  - We have to explore $\forall X \, \forall Y \, ((X \subseteq R) \wedge (Y \subseteq X))$ the association rule $(X - Y) \Rightarrow Y$
- There are $2^{|R|}$ candidates

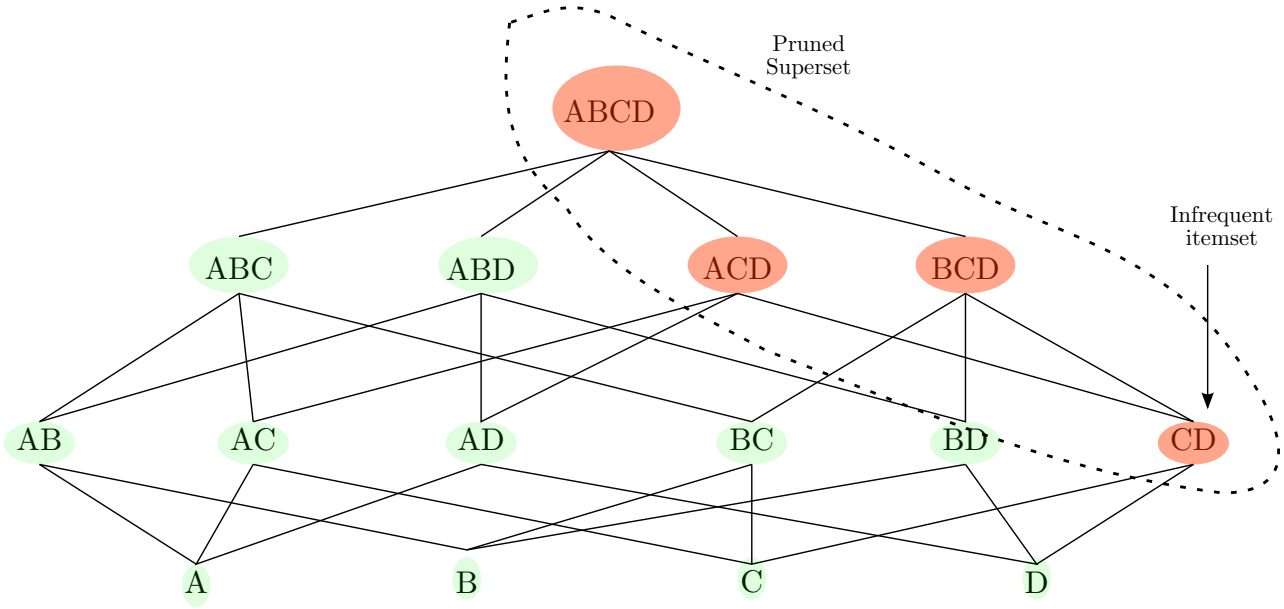
Space of itemsets



Properties of frequent sets

- We need ways to prune the search space
- Given X and Y with $Y \subseteq X$:
  - If $fr(Y) \geq fr(X)$, if X is frequent, Y also is frequent
  - If any subset Y from X is not frequent then X is not frequent
- This is known as the **anti-monotone** property of support
- A feasible exploration approach is
  - Find the frequent sets starting from size 1 and increasing its size
  - Prune the candidates including infrequent itemsets

Pruning of itemsets



Finding frequent sets

- $\mathcal{F}_l(r, min\_sup)$ is the set of frequent sets from R of size l.
- 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.

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

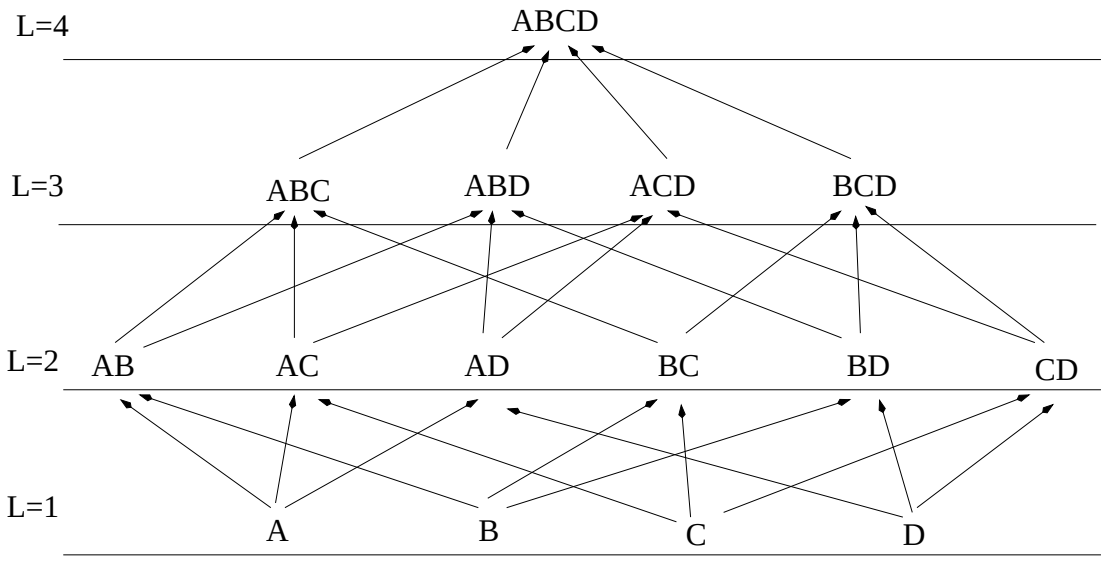
- The computation of association rules can be done iteratively starting with the smallest frequent subsets until no more candidates are obtained

Association Rules Algorithm

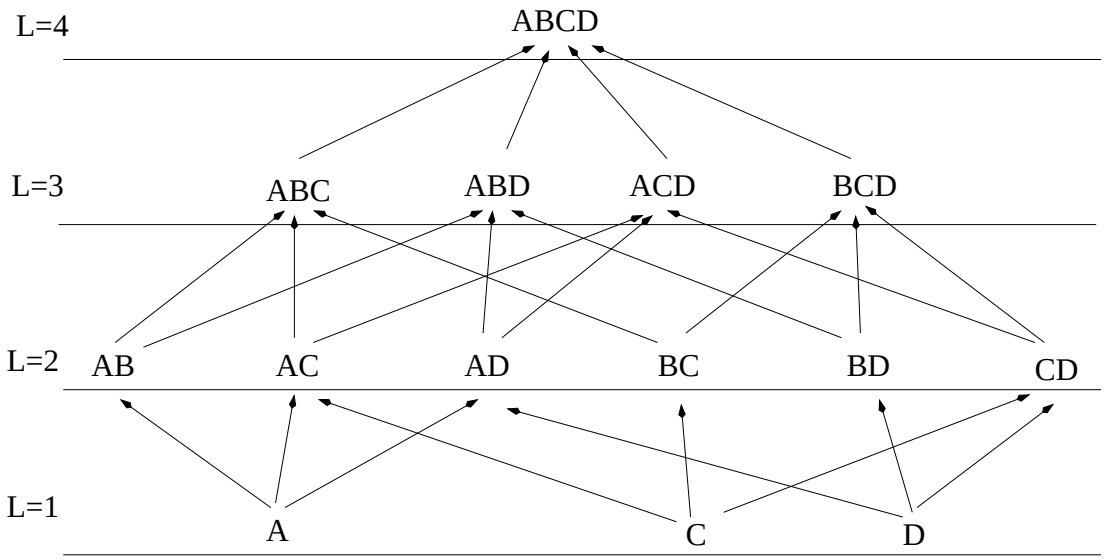
Algorithm: Apriori (R,min_sup,min_conf)

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

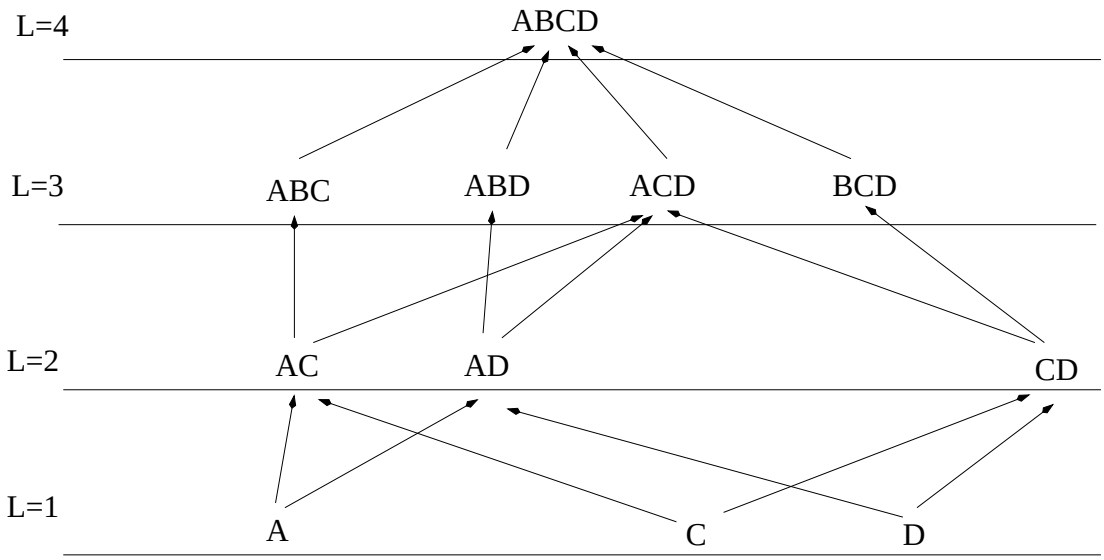
Initial set of itemsets



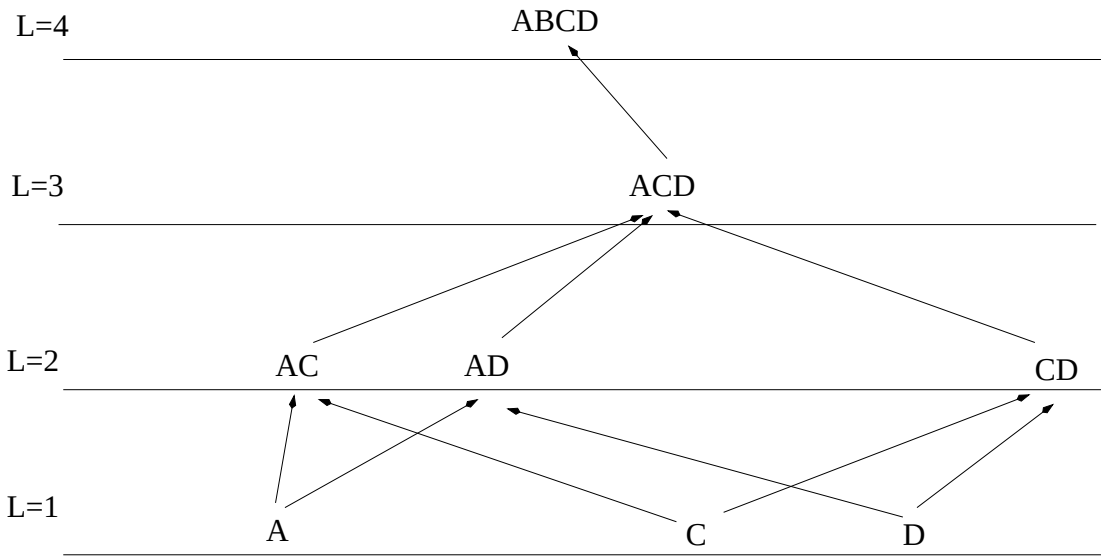
Itemsets size=1



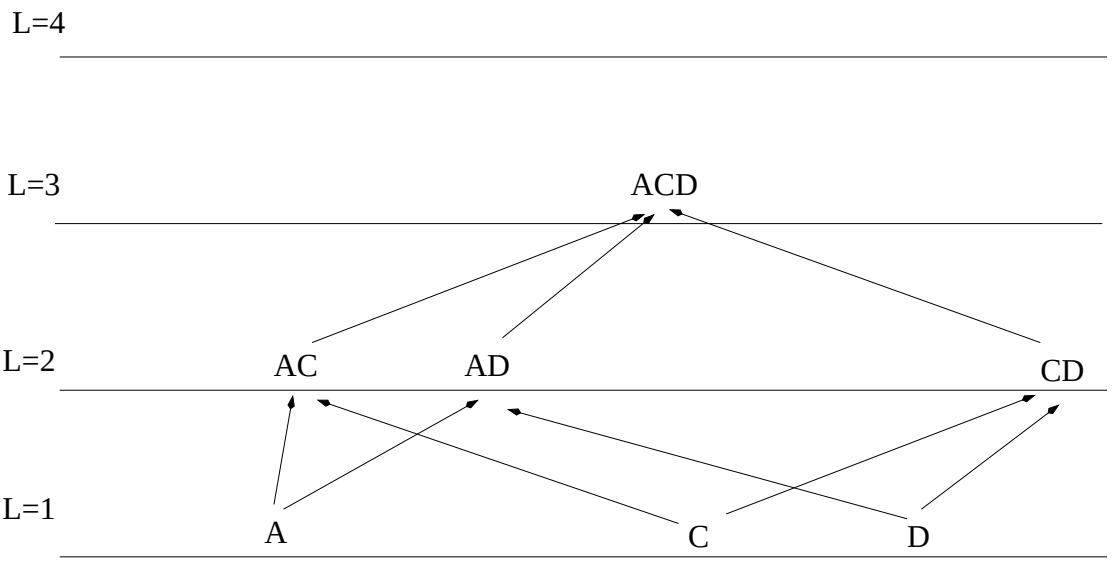
Itemsets size=2



Itemsets size=3



Final frequent itemsets



Effect of Support

- The specific value used as *min_sup* has effect on the computational cost of the search
  - 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)
- Unfortunately the threshold value can not be known beforehand
- Sometimes multiple minimum supports are needed (different for different groups of items)

Other factors affecting computational cost

- Number of different items
  - The more items there are the more space is needed to count its support
- Size of the database
  - The algorithm has to perform multiple passes to count the support
- Average transaction width
  - Affects the maximum length of frequent itemsets

Other pruning strategies

- Some itemsets are redundant because they have identical support as their supersets
- We could focus only on those itemsets, reducing the number of candidates
  - **Maximal frequent itemsets:** A frequent itemset for which none of its immediate supersets are frequent
  - **Closed frequent itemsets:** A frequent itemset for which none of its immediate supersets have the same support, and its support is larger than *minsupport*

Maximal frequent itemsets $\subseteq$ Closed frequent itemsets $\subseteq$ Frequent itemsets

1 Association Rules

2 Apriori algorithm

3 FP-Growth

4 Measures of interestingness

5 Twitter - City Connections

Other approaches

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

FP-Growth

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

FP-Tree

- 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

FP-Tree - Algorithm

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
   1. Pick the frequent items
   2. Order the items by their original frequency
   3. 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

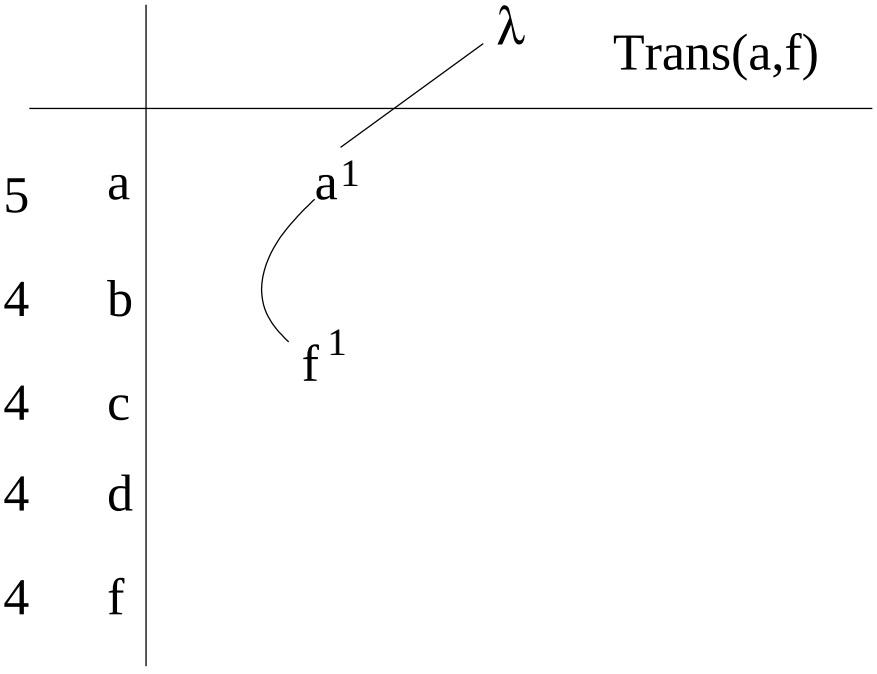
FP-Tree - Example

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

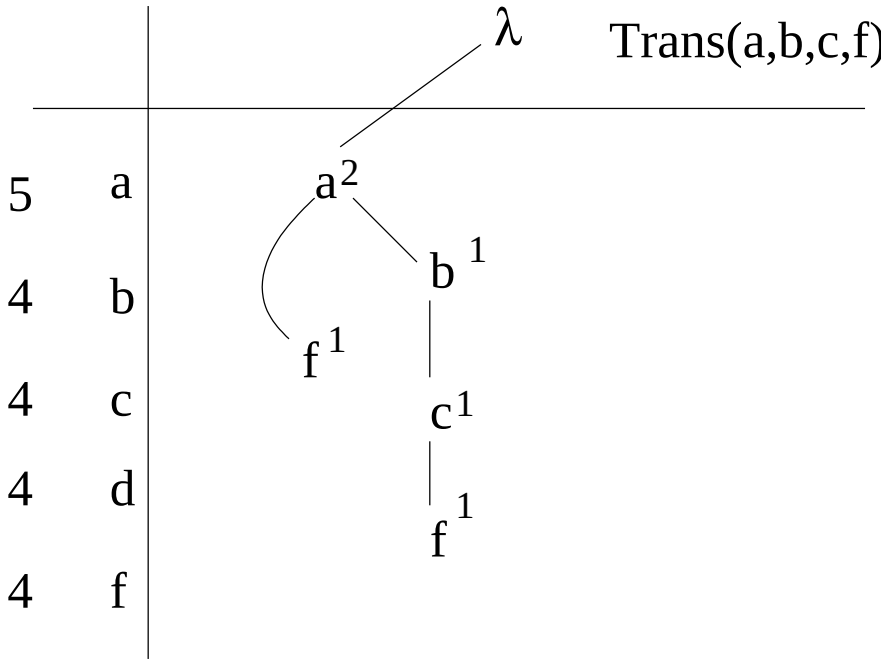
Item	frequency
a	5
b	4
c	4
d	4
e	3
f	4
g	1

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

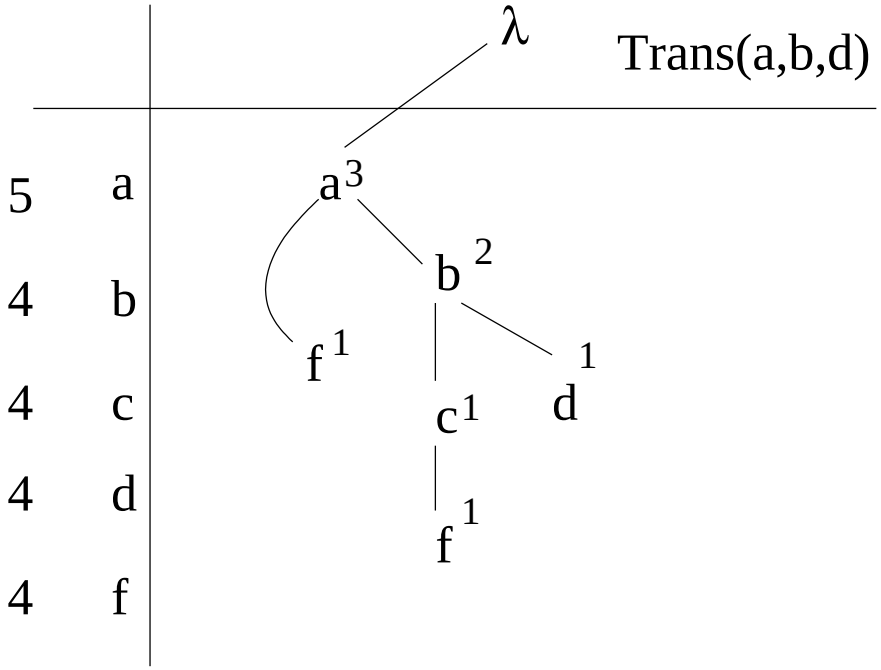
FP-Tree - Example



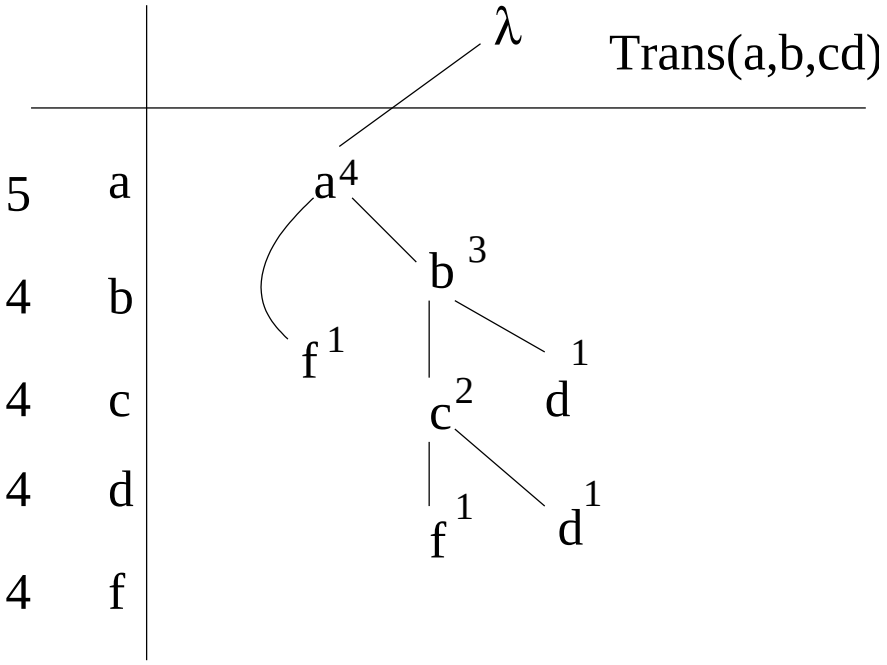
FP-Tree - Example



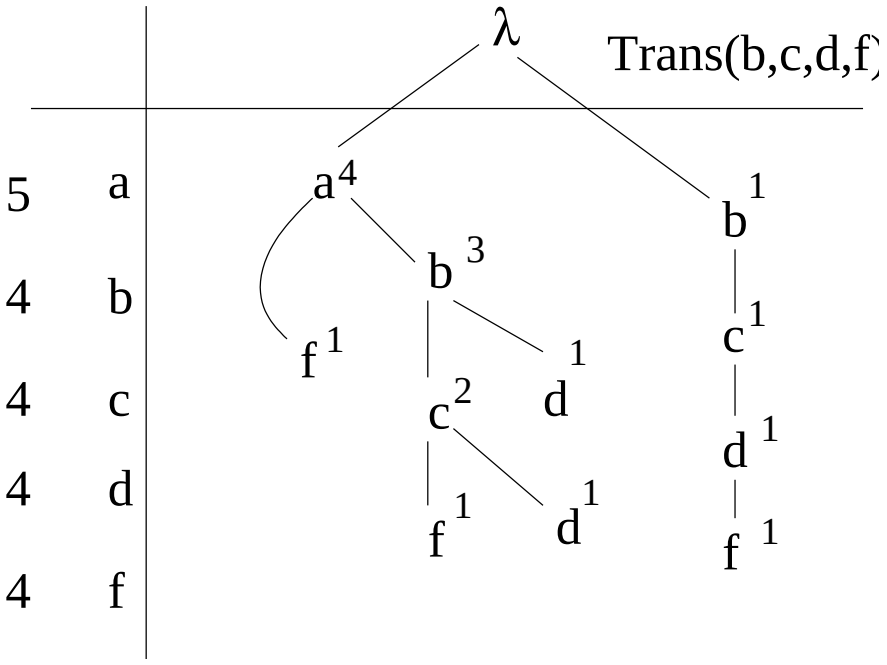
FP-Tree - Example



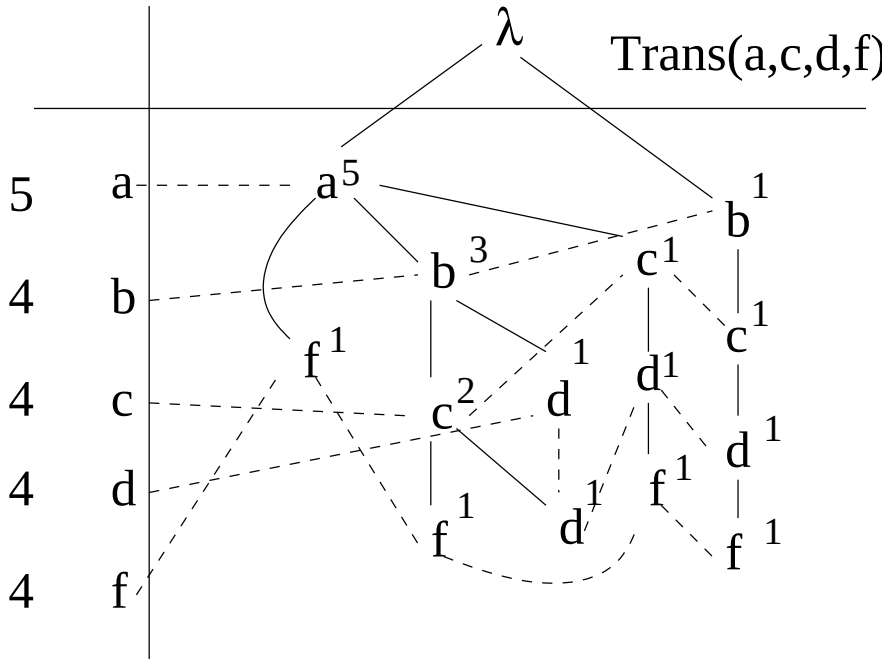
FP-Tree - Example



FP-Tree - Example



FP-Tree - Example

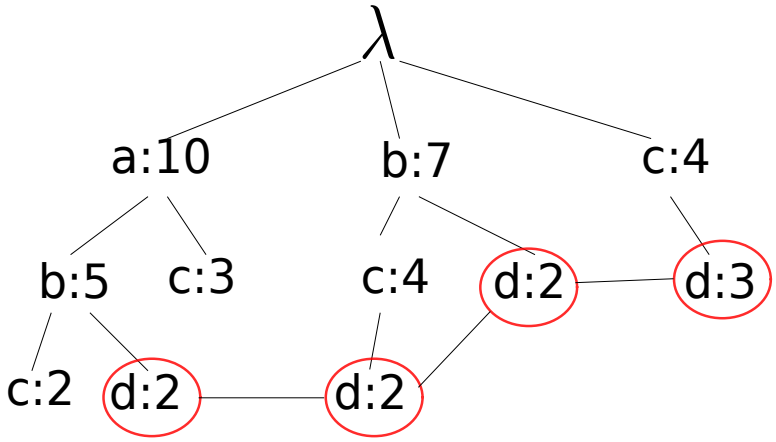


The FP-Growth Algorithm

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:
   1. Update the counts along the path using the frequency of the selected item
   2. Truncate the paths removing the nodes for the item
   3. 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
   1. Count the prefix as frequent
   2. Recursivelly find the frequent items for that prefix

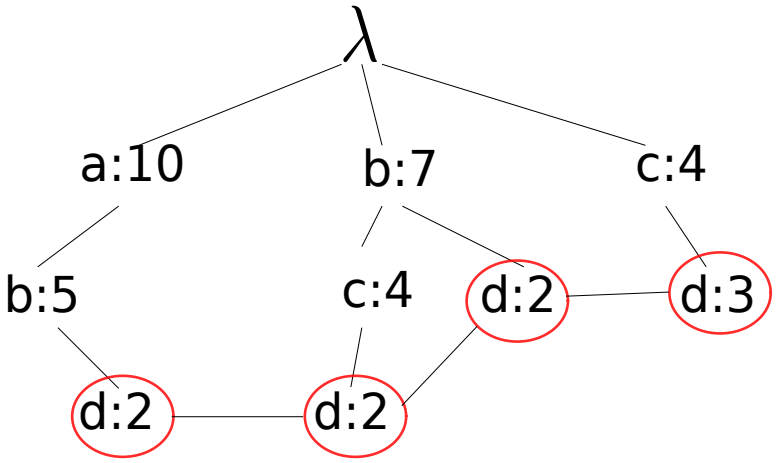
FP-Growth - Example

Extract patterns with suffix *d*



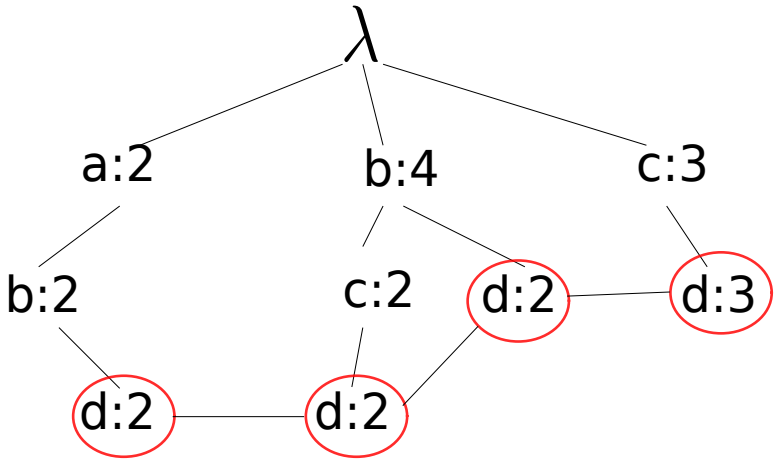
FP-Growth - Example

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



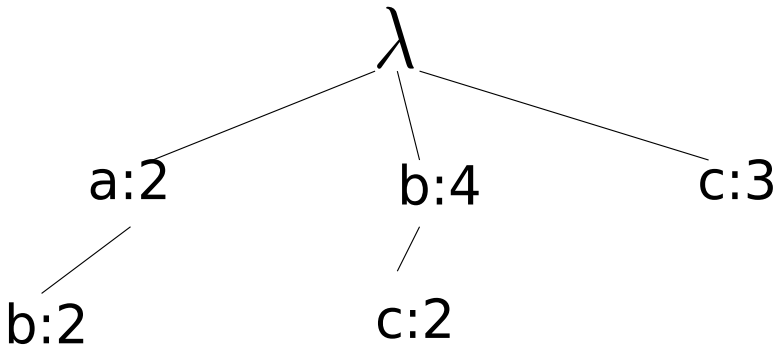
FP-Growth - Example

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



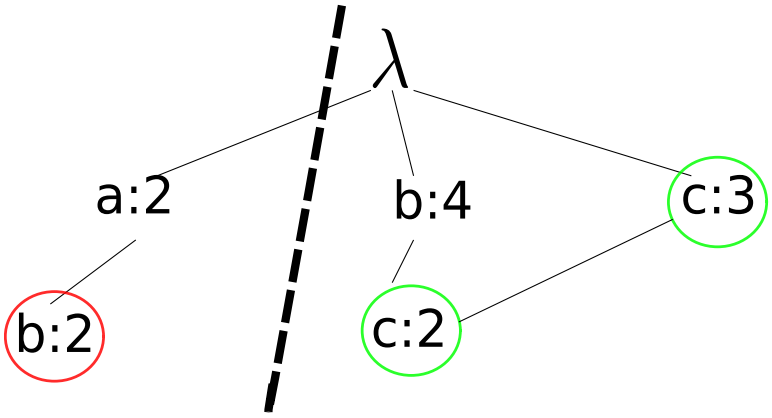
FP-Growth - Example

Prune the path eliminating d (is no longer needed)



FP-Growth - Example

Solve the problem for the predecesors 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]}

1 Association Rules
2 Apriori algorithm
3 FP-Growth
4 Measures of interestingness
5 Twitter - City Connections

Measures 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}	$\lvert r \rvert$

- 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 $\neg Y$

Measures for association rules interestingness

- Other measures are based on probabilistic independence and correlation

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

Properties of a measure of interestingness

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

Python Notebooks

This Python Notebook has examples using the Authors dataset for Apriori and FP-Growth

- Association Rules Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)

1 Association Rules

2 Apriori algorithm

3 FP-Growth

4 Measures of interestingness

5 Twitter - City Connections

Twitter - City Connections

- 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

Twitter - City Connections

- 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

Twitter - City Connections - Preprocess

- 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?)

Twitter - City Connections - Preprocess

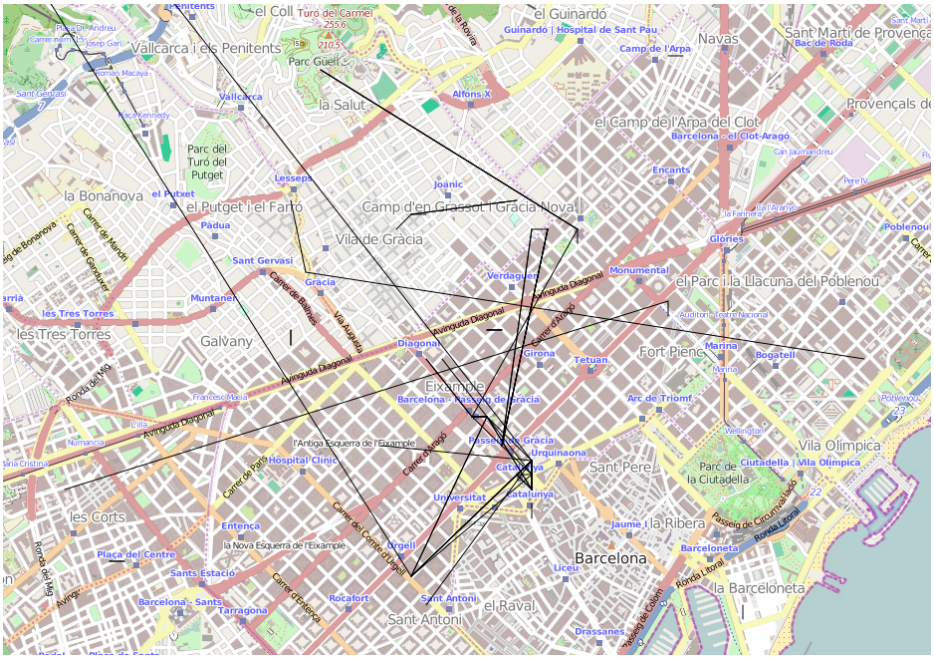
- We pick to discretize time in intervals of 8 hours (0-7, 8-16, 13-23)
- Two possibilities for coordinates:
  - An equally spaced 300×300 grid ($100m \times 100m$)
  - Clusters obtained by the leader algorithm (radius $100m$)
- The first option makes that a transaction has a determined number of attributes ($300 \times 300 \times 3$), but no all the possibilities will appear
- For second the option the number of attributes will depend on the densities in the coordinates

Twitter - City Connections - Process

- 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

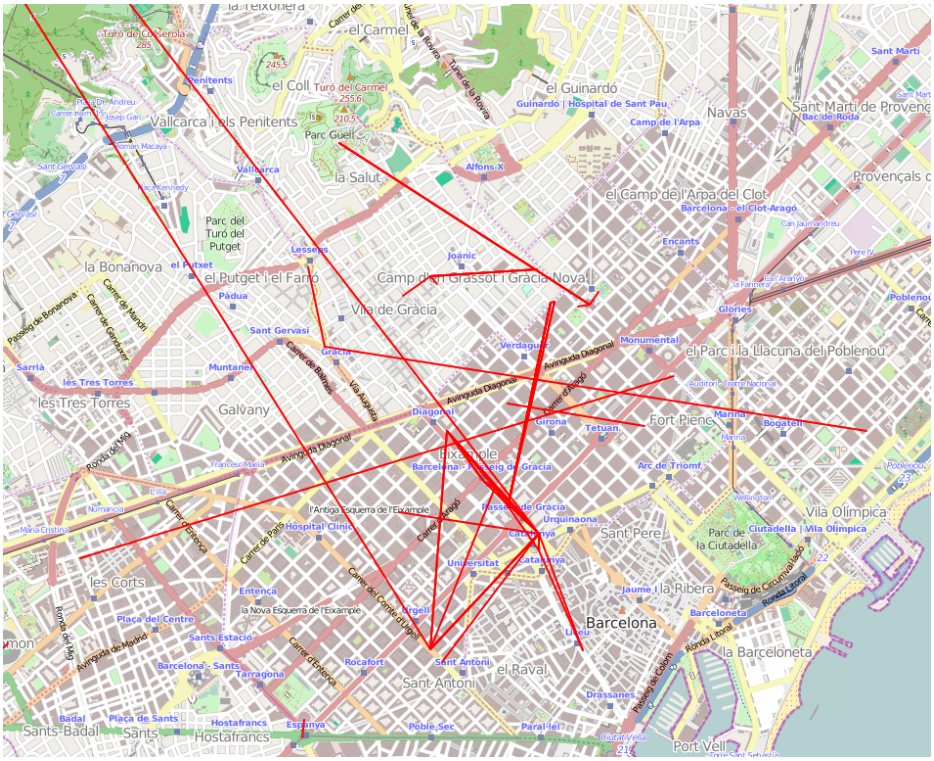
Twitter - City Connections

Grid Discretization 100m sup=50



Twitter - City Connections - results

Leader Clustering Discretization 100m sup=50



8

Mining Structured Data

Mining of structures

Javier Béjar

CS - MAI

AMLT - 2016/2017

Outline

1. Introduction
   - Motivation
2. Sequences
   - Introduction
   - Clustering Sequences
   - Frequent patterns in sequences
   - Frequent Strings
   - Frequent transaction sets
   - Clustering of Data Streams
3. Application
4. Graph mining
   - Introduction
   - Mining large graphs
   - Mining sets of graphs
     - Clustering of graphs
     - Mining frequent Graphs

1 Introduction

2 Sequences

3 Application

4 Graph mining

Mining of structures

- 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

1 Introduction

2 Sequences

3 Application

4 Graph mining

Mining of sequences

- 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, ...)

Examples of sequence data

- 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

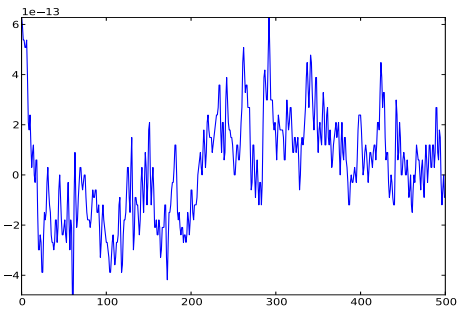
Mining of sequences - Clustering Sequences

- **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?

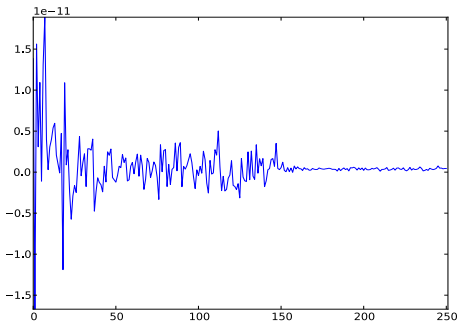
Clustering Sequences - Data representation

- 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)

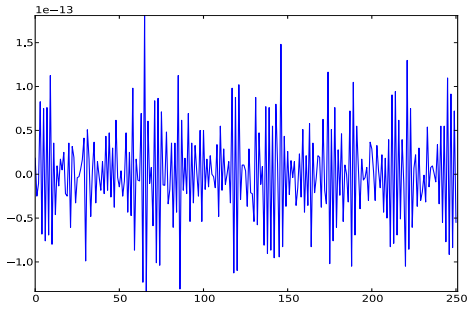
Clustering Sequences - Data representation



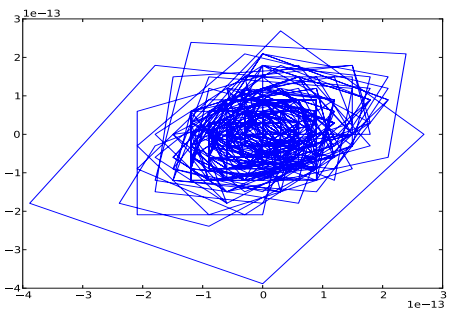
Original Series



Discrete Fourier Coefficients



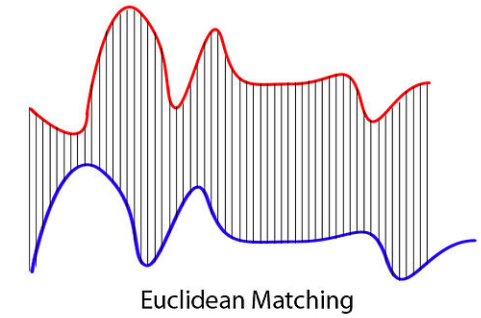
Discrete Wavelet Coefficients



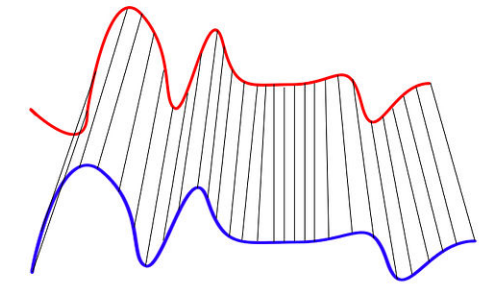
1-Lag Series

Clustering Sequences - Distance Functions

- 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)

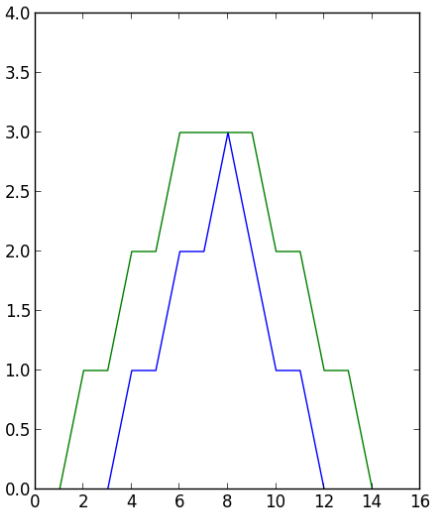


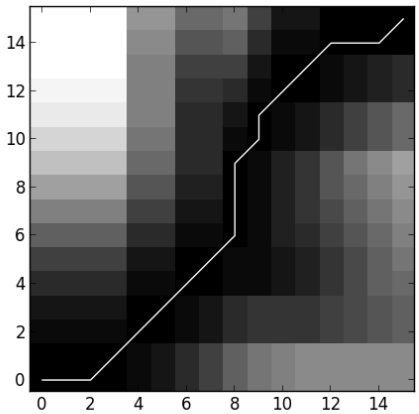
Euclidean Matching



Dynamic Time Warping Matching

Clustering Sequences - Distance Functions - DTW





Mining of sequences - Subpatterns

Frequent sequence patterns: Discovery of patterns inside the sequence

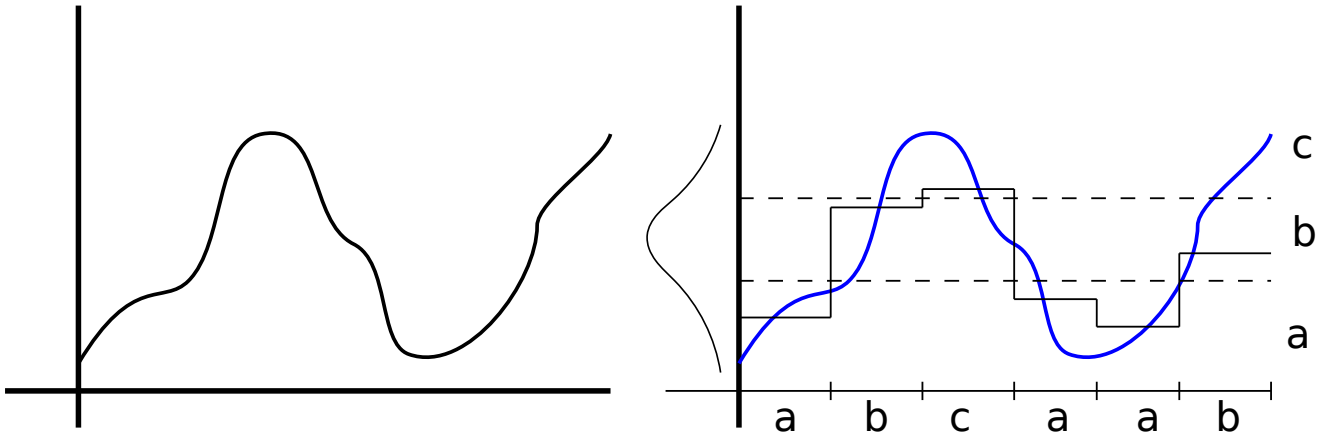
- The series does not need to have a temporal relationship (eg.: DNA sequences)
- The values can be continuous, discrete or structured (transaction data)
- Goals:
  - Discovery of subseries that reveal some causality of events
  - Discovery of abnormal/novel patterns (deviation from normal behavior)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

Frequent Motifs

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

Frequent Motifs - SAX



Frequent Motifs - Algorithm

- 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

Frequent Strings

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

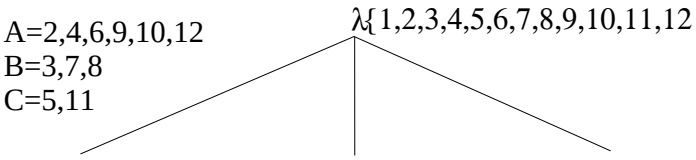
Frequent Strings - Algorithm

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
   1. Compute a list with the next position to the occurrence of the character
   2. If the character appears more than k times we create a new descendant in the trie
2. Recursive call for each descendant created

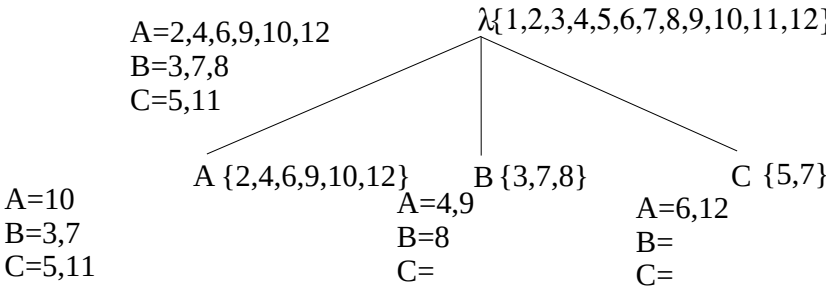
Frequent Strings - Example

1 2 3 4 5 6 7 8 9 101112
ABACABBAACA$ K=2



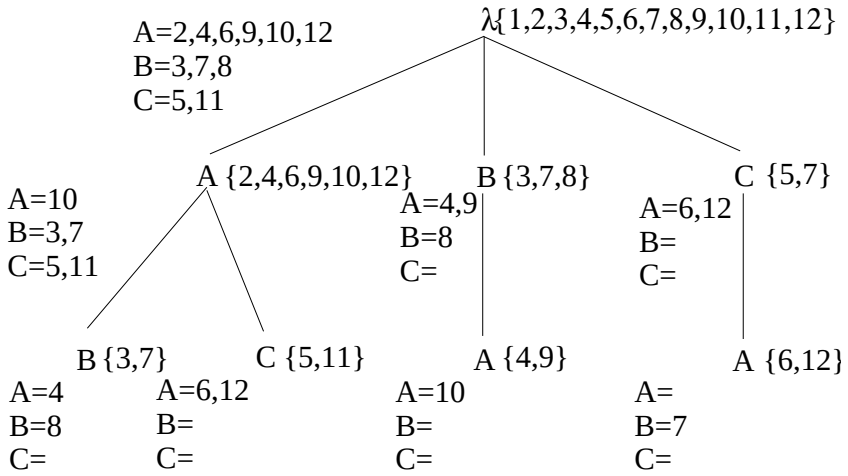
Frequent Strings - Example

1 2 3 4 5 6 7 8 9 101112
ABACABBAACA$ K=2



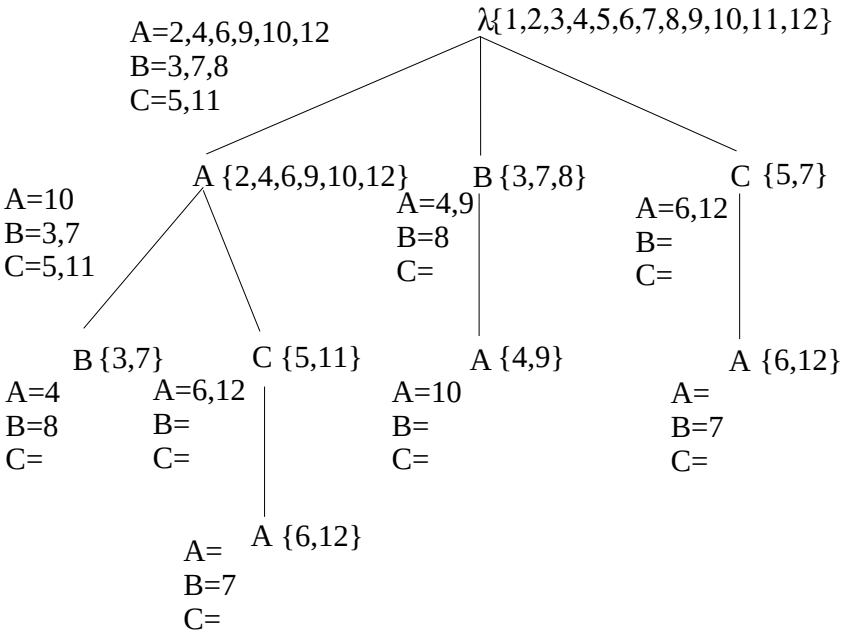
Frequent Strings - Example

1 2 3 4 5 6 7 8 9 101112
ABACABBAACA$ K=2



Frequent Strings - Example

1 2 3 4 5 6 7 8 9 101112
ABACABBAACA$ K=2 → {A,B,C,AB,AC,BA,CA,ACA}



Frequent Transactions

- 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 $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$,$a_n \subseteq b_{in}$

Generalized Sequence Pattern (GSP)

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

Generalized Sequence Pattern (GSP)

- 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

Generalized Sequence Pattern (GSP)

- Case 1:

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

- Case 2:

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

- Case 3:

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

Clustering of Data Streams

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)

Clustering of Data Streams - Micro clusters

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

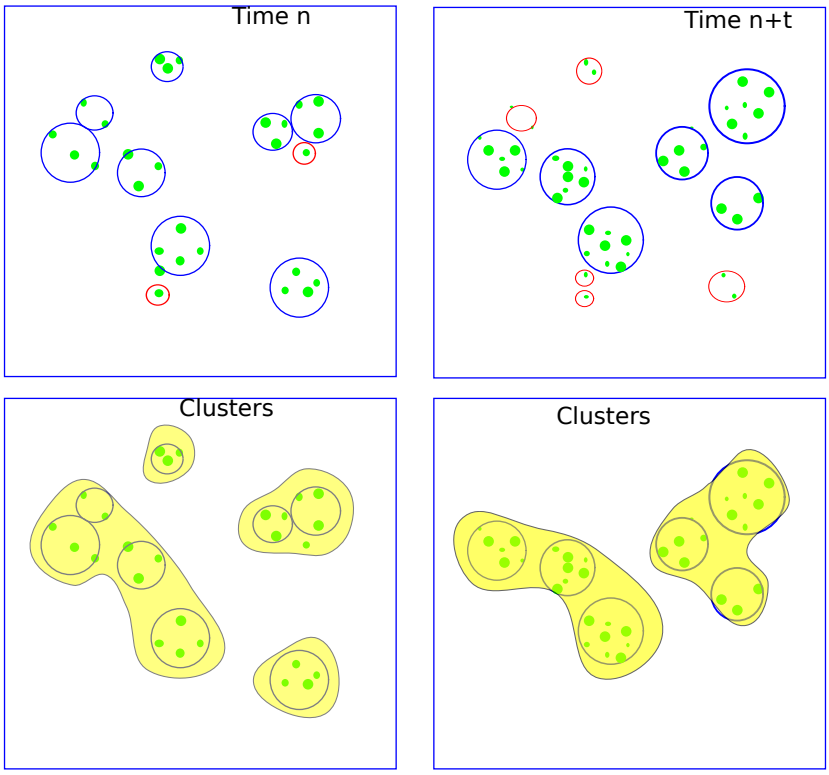
- On-line phase:
  - Maintains/creates microclusters
  - Number of microclusters is larger than the actual number of clusters of the dataset (approximate the densities)
  - When new data arrives, it is incorporated to a microcluster or generates new microclusters
  - The number of microclusters is fixed, 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

Clustering of Data Streams - Density Based

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

Clustering of Data Streams - Density Based



1 Introduction

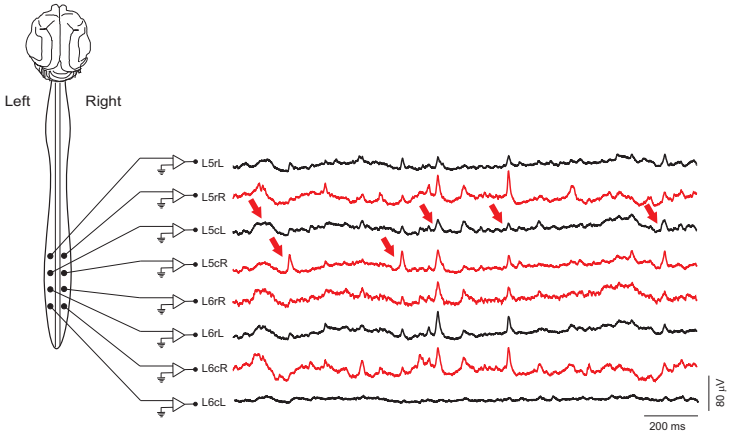
2 Sequences

3 Application

4 Graph mining

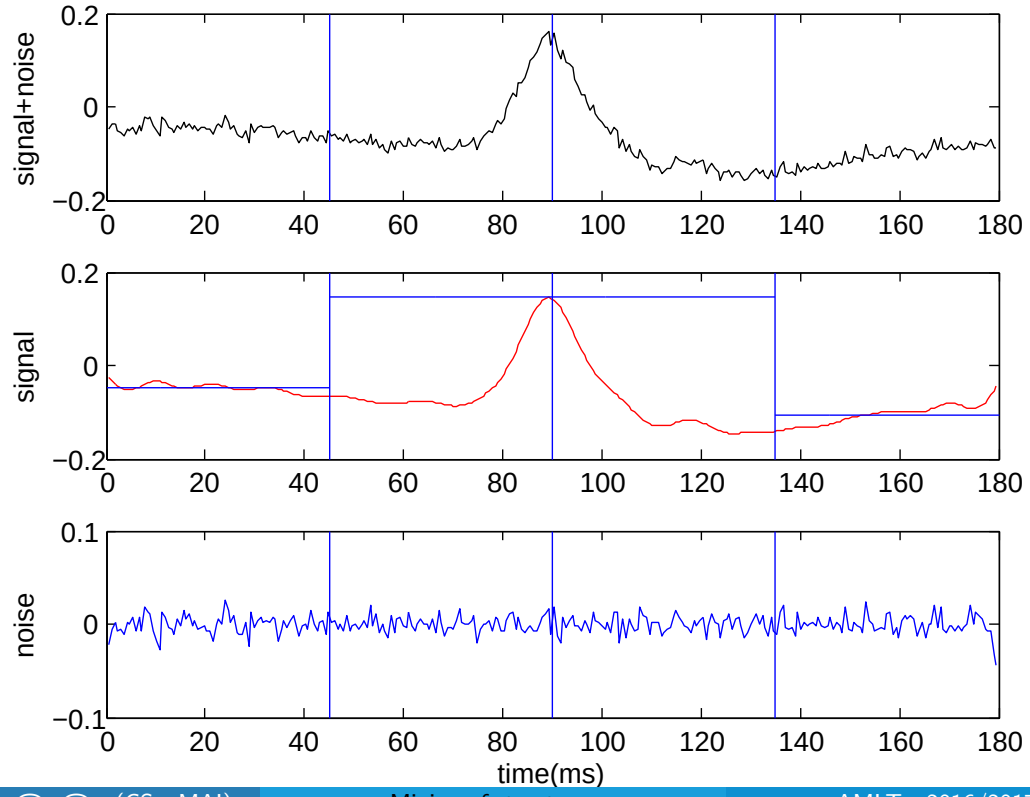
Spinal Cord Signals Analysis

- 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



Spinal Cord Signals Analysis

Peak Data



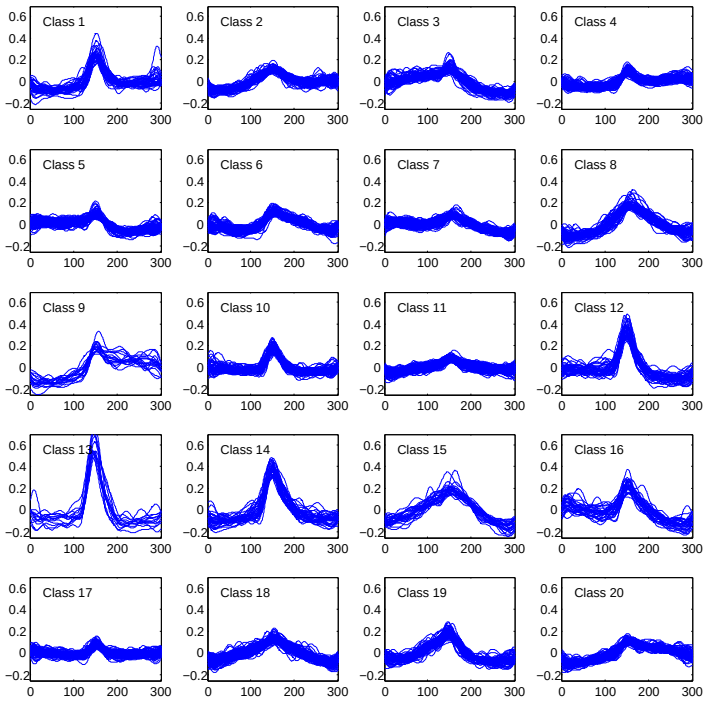
Spinal Cord Signals Analysis

Shapes dictionary

- 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

Spinal Cord Signals Analysis

Clusters



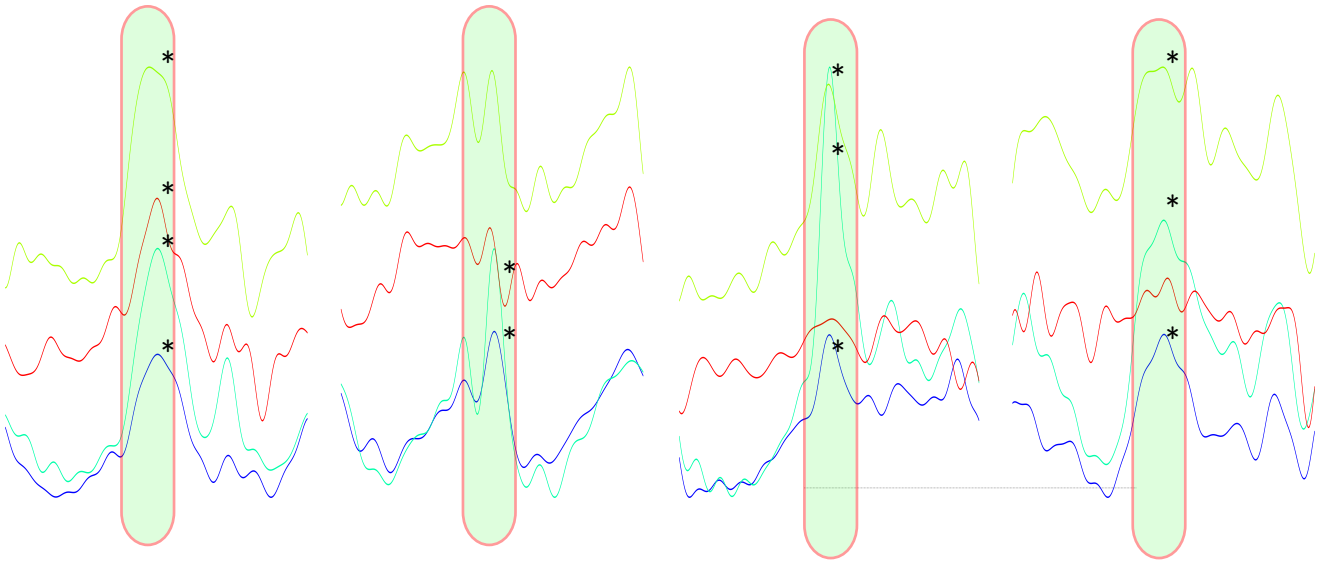
Spinal Cord Signals Analysis

Synchronization Patterns

- 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

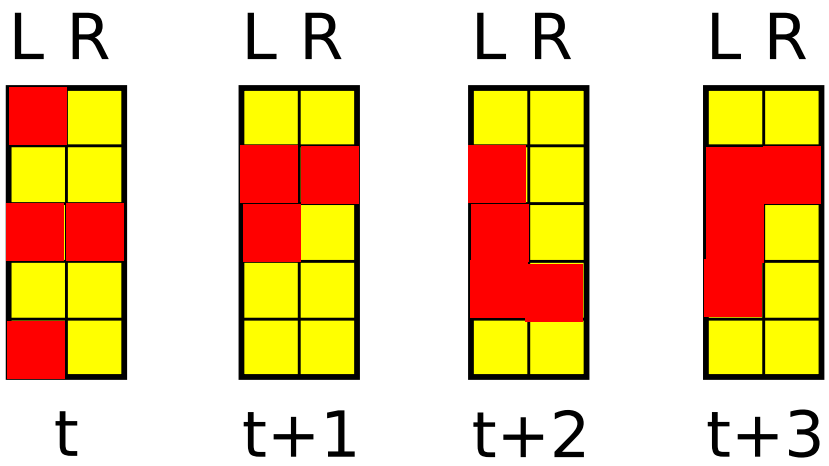
Spinal Cord Signals Analysis

Finding Synchronizations



Spinal Cord Signals Analysis

Synchronization sequences



1 Introduction

2 Sequences

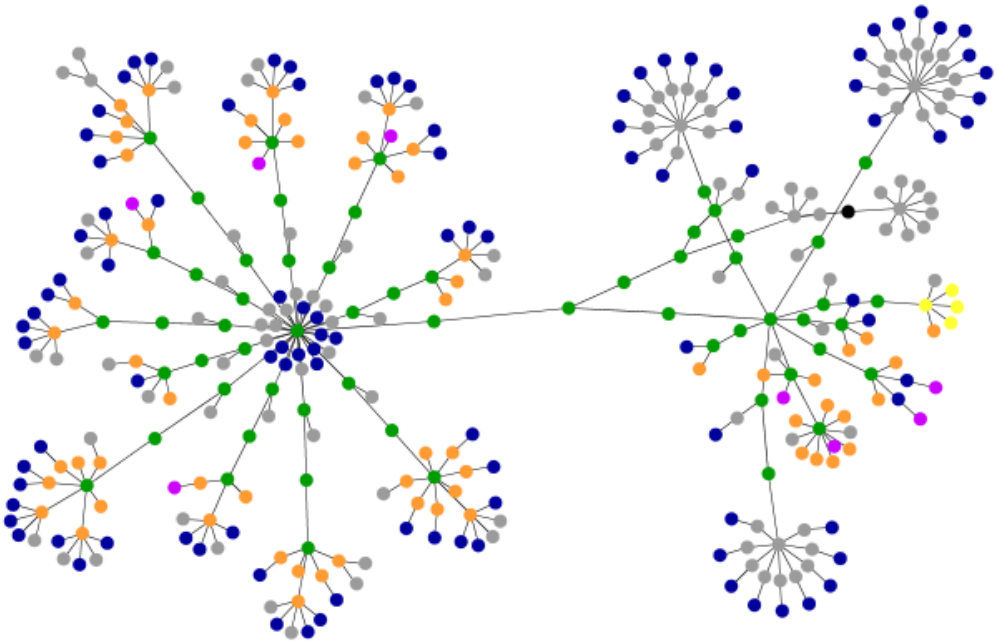
3 Application

4 Graph mining

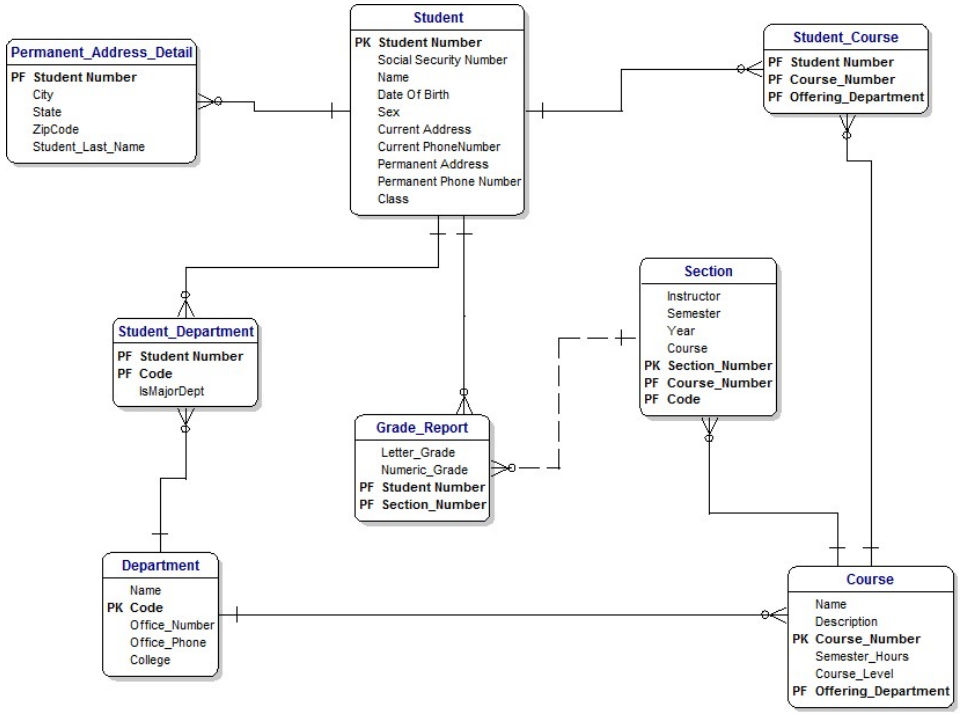
Mining of Structures

- 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

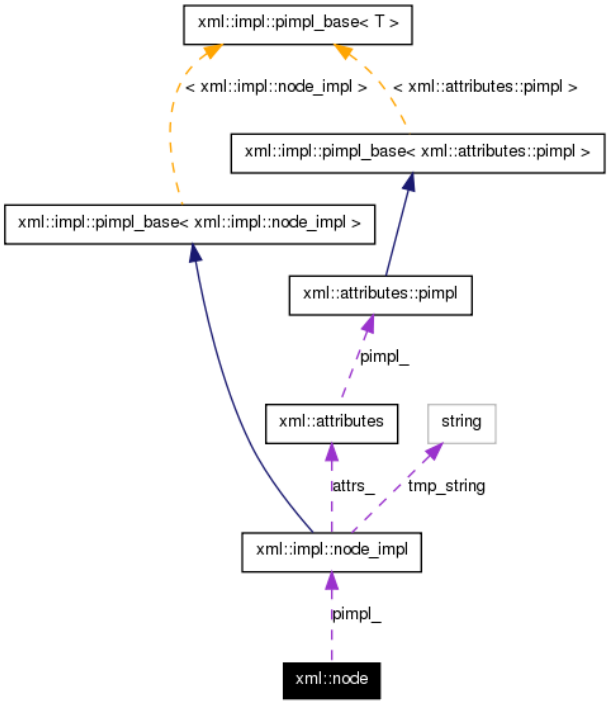
Mining of Structures (WWW/Social networks)



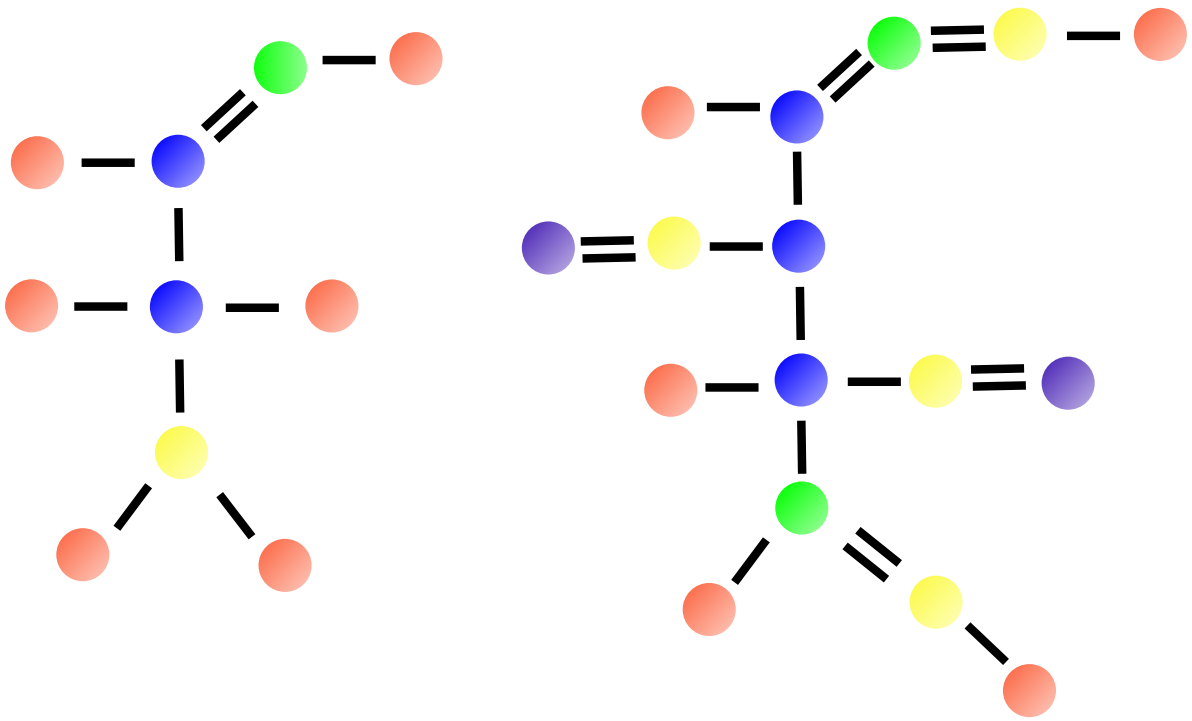
Mining of Structures (Relational databases/Ontologies)



Mining of Structures (XML documents/Text)



Mining of Structures (Chemical compounds/Gene interactions)



Mining of Structures

- 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

Mining of Structures: Computational issues

- 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) $\Rightarrow$ Partitioning, dense subgraphs
  - Mining sets of graphs $\Rightarrow$ common substructures

Mining Large Graphs

- 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

Graph partitioning (2-way)

- 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

Graph partitioning (k-way)

- 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

Graph partitioning (k-way) - Clustering

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

Graph partitioning (k-way) - Clustering

- 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)

Dense subgraphs

- 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

Mining Sets of Graphs

- 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

Clustering of Graphs

- 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

Clustering of Graphs - XProj

- 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)

Clustering of Graphs - XProj

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*

Mining frequent Graphs/Trees

- 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

gSpan

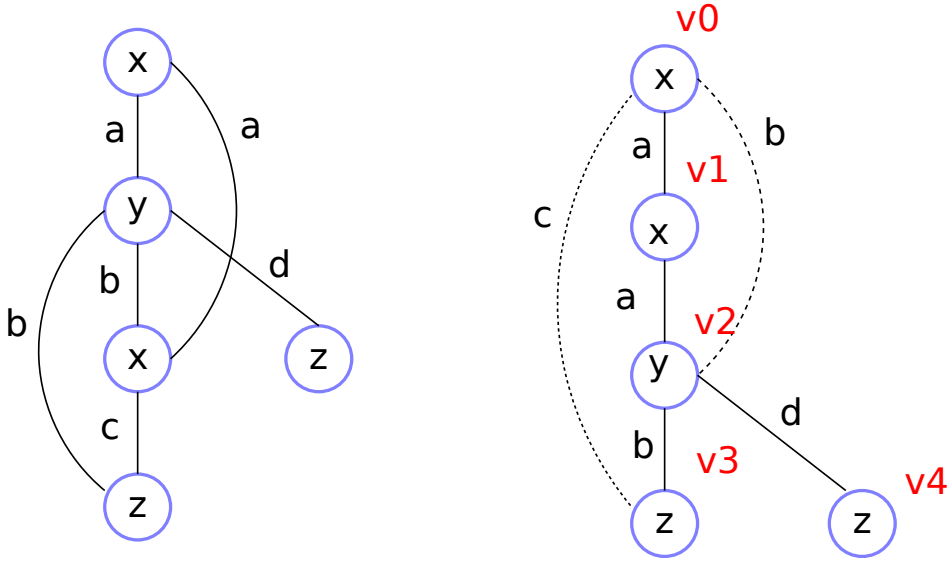
Yan, Han **gSpan: Graph-Based Substructure Pattern Mining**
Proceedings of the IEEE International Conference on Data Mining 2002

- 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

gSpan - DFS Lexicographic order

- 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

gSpan - DFS Lexicographic order



gSpan - Algorithm

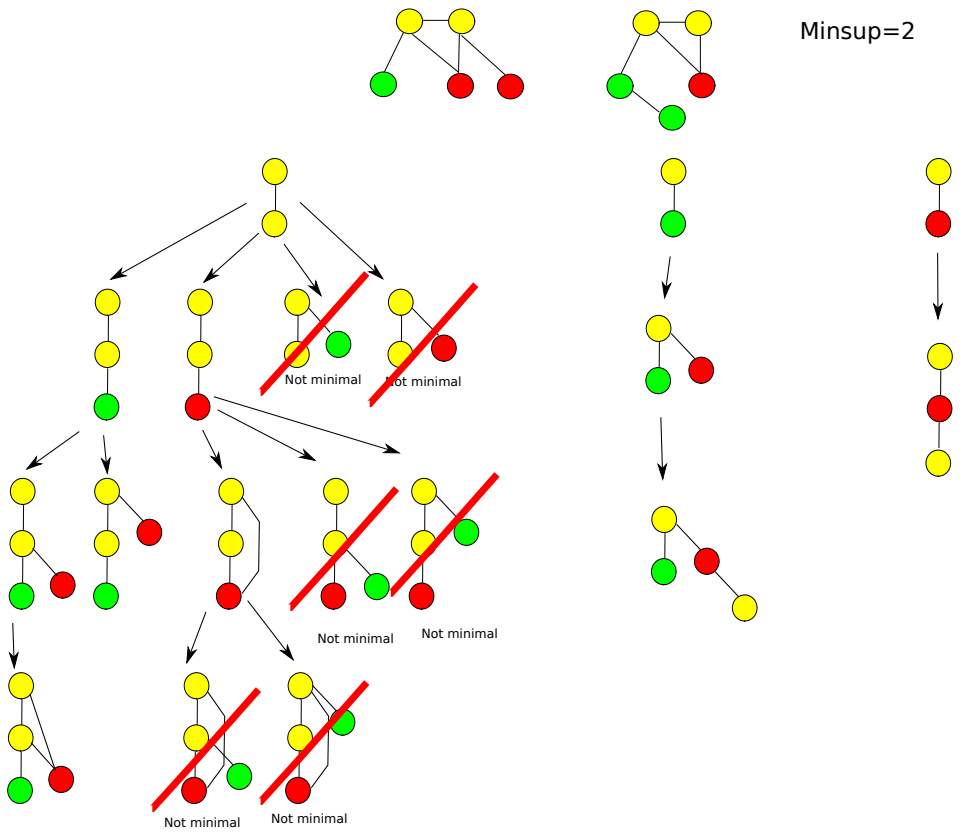
Initialization (D, $MinSup$)

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
   1. gSpan(s,D,$MinSup$,S)
   2. $D = D - s$
   3. 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)

gSpan - Example



Python Notebooks

This Python Notebook has examples of dense subgraphs discovery and comunity discovery using geolocation information from Twitter for London, Paris and Barcelona

- Dense Subgraphs Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)