Student's Handbook Guide to CMPUT 697 Introduction to KDD and Data Mining

A student's guide to Knowledge Discovery and Data Mining

This course will introduce principles of Knowledge Discovery and Data Mining with a focus on unsupervised data mining methods with a focus on multi-dimensional and spatial data. In the first part of the course, the instructor will provide background lectures, and in the second part, students will present research papers, and present at the end the results of their course project.

Disclaimer: Mining data using an actual pickaxe will lead to unsatisfactory results

Contents

Chapter 1: Introduction and Overview

Course works and evaluation

The two assignments are paper reviews 10% each. Written exam will before reading going over what was talked about in the lectures.

Introduction

The current situation consists of data-rich environment. There are an increasing amounts of data gathered with automated data collection tools. Data is often has temporal and spatial aspects. But it's also information-poor, and thus require powerful analysis methods and data management systems to turn the data into useful information.

Example of Analysis methods

For example a Brain tumour growth database. You have 3D tumours of patients overtime and the raw data consists of images. How to automatically recognize where the tumour is? How to retrieve "similar" studies, where the notion of similarity can be based on shape, location, growth pattern, etc.

Another example could be underwater observations. You have different type of data like temperature, oxygen concentration, etc. But how do we detect, extract, and store interesting data. Detect useful events, describing those events, etc.

KDD: Knowledge Discovery in Databases

Is a confluence of multiple discipline like ML, stats, visualization, etc. You have a process of extracting valid, previously unknown, and potentially useful knowledge in databases.

KDD Process

The process involves

1. Data cleaning and integration into data warehouse from different databases or information repositories
2. Selection, projections, transformation to get task-relevant data
3. Data-minding to find patterns
4. Visualization and evaluation to produce knowledge
5. Repeat from the beginning

Each consecutive step ablates or abstracts the raw or previous data into useful information or knowledge.

Data Mining

- A central step in the process
- Application of algorithms that find "patterns" in the data

Data Cleaning and Integration

Integration of data from different sources like mapping of attribute names, joining different tables. Elimination of inconsistencies, elimination of noise, imputation of missing values (if necessary and possible) some filling strategies could be default values, average value, or application specific computation.

Focusing on task-relevant data

You need to select relevant tuples or rows from the database tables. Projection or feature selection, selecting the relevant attributes of columns from the tables either manually or by some feature selection methods. Transformation i.e., normalization like mapping age from 18 to 82 to 0 to 100, discrtisation of numerical attributes like mapping 0 to 100 to low, medium, and high. You could also have computation of derived tuples or rows and derived attributes or columns like aggregation of sets of tuples or creating a new attributes based on other value(s).

Basic Data Minding Tasks

Find patterns based on:

- Clustering
- Outlier Detection
- Classification
- Association Rules
- Methods for special data types like spatial data, web data
- sequential patterns
- Trends and analysis of change

Clustering. Class labels are unknown you can group objects into sub-groups either by similarity or dissimilarity function like using distance or different paradigm for what constitutes clustering. Application includes:

- Exploratory data analysis
- Data reduction or summarization like
  - Customer profiling and segmentation
  - Document or images collection
  - Web access pattern

Outlier Detection. There are no class labels but we find objects that do not comply with the general behaviour of the data. There are different paradigms for outline detection.

Application include

- Data cleaning

- Fraud detection
- Rare event analysis

Related to one-class classification, where labels for a only a set of "inliers" or "normal" cases are available.

Classification. Class labels are known for a set of training examples. Find models, functions or rules based on attribute values of the training examples that

- describe and distinguish classes
- predict class membership for "new" objects

Applications:

- Classify gene expression values for tissue samples to predict disease type and suggest best possible treatment
- etc

Association Rules. Find frequent association in transaction databases. Frequently co-occurring items in the set of transaction. Rules with minimum confidence based on frequent item sets

Applications include

- Market-based analysis
- Catalog design
- ...
- Also used as a basis for clustering, and classifcation

Other methods. These include trends and evolution analysis where you have sequential patterns or regression analysis. Methods for special data types and applications.

Evaluation of Patterns

Interestingness of patterns. A pattern is interesting if its easily understood by humans, valid on new or test data with some degree of certainty, potentially useful, novel, or validates some hypothesis. Interestingness measures two kinds of aspect

- Objective: based on statistics and structures of pattern
- Subjective: based on user's belief in the data

How to produce only interesting pattern is still an open problem.

Visualization

Integration of visualization and data mining

- data visualization
- data mining result visualization
- data mining process visualization
- interactive visual data mining

There are different types of 2D and 3D plots, charts, and diagrams

- Box-plot
- Trees
- X-Y plots
- Parallel coordinates

Other Issues in Data mining

There are mining knowledge at multiple levels of abstraction. You have to incorporate background knowledge into the process. You could either use data mining query languages or use ad-hoc data mining or a combination of both. Handle noise and incomplete data. Efficiency and scalability of data mining algorithms. Parallel, distributed and incremental mining methods. Privacy preserving data mining. Etc, etc, etc.

Data Management

This includes high-level data models plus the query and data manipulation languages. Three-level architecture to provide data independence

1. External views
2. Logical level
3. Physical level

You can change aspects of the implementation of a database without affecting the applications using the data.

You also need to worry about

- Transactions
- Concurrency control
- Crash recovery

You need to protect data from being corrupted in a multi-user environment with potentially failing hardware and software.

Finally, you need to have an **Efficient Index Structures** to speed-up certain types of queries. In this course we will talk about spatial and multi-dimensional index structures to support similarity queries (range queries and knn queries).

Chapter 2: Clustering

Introduction to Clustering

Grouping a set of data objects into clusters

Cluster

A collection of data objects.
- Similar to one another with the same cluster
- Dissimilar to the object in other classes

Clustering is an **unsupervised classification** as there are no predefined classes

Typical usage include:

- As a stand-alone tool to get insight into data distribution
- As a preprocessing step for other algorithms

Measuring Similarity

To measure similarly, often a distance function *dist* is used the larger the distance the more dissimilar they are.

Properties of a distance function are:

- $dist(x, y) \geq 0$
- $dist(x, y) = 0 \iff x = y$
- $dist(x, y) = dist(y, x)$ (symmetry)
- if *dist* is a metric, which is often the case: $dist(x, z) \leq dist(x, y) + dist(y, z)$ (triangle inequality)

Definition of a distance function is highly application dependent. They may require standardization or normalization of attributes. Different definitions for Boolean, categorical, ordinal, interval-scaled, and ratio variable.

Nominal

Things like Boolean and categorical attributes that have arbitrary labels, and have no ordering.

Ordinal

Things like rating, preferences, anything that involves ordering that may or may not use numbers. The difference are not meaningful as different people will have different standards with regards to each label.

Interval-scale

Things like temperature in degrees C or Kelvin. These values are ordered, and are numerical i.e., not a category, it's continuous scale, but it has no natural 0. Ratio do not make any sense as the reference number is not consistent.

Ratio variable

Thing like height, etc. These values are ordered, and have all the properties like Interval-scale but has a neutral zero or a consistent reference point. Measuring things like in Kelvin. Therefore, both difference and ratios make sense.

Example of Distance Function

Here are a list of examples for different kinds of attributes.

Standardized numerical attributes

For standardized numerical attributes (objects described by d numerical attributes) $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$.

- General L_p-Metric otherwise called the Minkowski-Distance:

$$dist(x, y) = \sqrt[p]{\sum_{i=1}^{d}(x_i - y_i)^p}$$

- Euclidean Distance where $p = 2$:

$$dist(x, y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

- Manhattan-Distance where $p = 1$:

$$dist(x, y) = \sum_{i=i}^{d}|x_i - y_i|$$

- Maximum-Metric where $p = \infty$:

$$dist(x, y) = \max\{x_i - y_i || 1 \leq i \leq d\}$$

Comparing sets

Jaccard Distance

Basically is the number of similar elements as a proportion of the union between the two set.

$$dist(x, y) = \frac{|x \cup y| - |x \cap y|}{|x \cup y|} = 1 - \frac{|x \cap y|}{|x \cup y|}$$

To measure the Jaccard Similarity you take the complement of the Jaccard Distance or

$$sim(x, y) = \frac{|x \cap y|}{|x \cup y|}$$

Categorical attributes

For categorical attributes $x = x_1, \ldots, x_d$ and $y = (y_1, \ldots, y_d)$ the distance function would be:

$$dist(x, y) = \sum_{i=1}^{d}\delta(x_i, y_i) where \delta(x_i, y_i) = \begin{cases} 0 & if x_i = y_i \\ 1 & else \end{cases}$$

Basically this counts the number of differences in attributes between the two records.

Text documents

A document D is represented by a vector $r(D)$ of frequencies of the terms occurring in D for example.

$$r(D) =< w_1, \ldots, w_{|T|} > \quad w_i = f(t_i, D) \ for \ t_i \in T$$

where T is a "universe" of Terms and $f(t_i, D)$ is a function of the frequency of term t_i in document D (e.g., the log of the frequency). The distance between the two document D_1 and D_2 is defined by the cosine of the angle between the two vectors $x = r(D_1)$ and $y = r(D_2)$:

$$dist(x, y) = 1 - \frac{x \cdot y}{|x| \cdot |y|}$$

Mahalanobis Distance

Used to find the distance between two points $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ from the same distribution:

$$dist(x, y) = \sqrt{(x - y)^T \sum^{-1} (x - y)}$$

Where $\sum$ is the $d \times d$ covariance matrix of data set D

$$\sum_{ij} = \frac{1}{n - 1} \cdot \sum_{x \in D} [(x_i - \mu_i) \cdot (x_j - \mu_j)]$$

If the $\sum$ is the identity matrix then you have the Euclidean distance. If the $\sum$ is diagonal then you have a normalized Euclidean distance:

$$dist(x, y) = \sqrt{\sum_{i=1}^{d} \frac{(x_i - y_i)^2}{\sigma^2}}$$

Normalization

Z-score Normalization

$$V_{new} = \frac{V_{old} - \mu_x}{\sigma_x}$$

Min-Max Normalization

$$V_{new} = \frac{V_{old} - min_x}{max_x - min_x}$$

Data Structure

There are two ways to represent distances in a computer

Data matrix

A 2D matrix where for each row that represent a point contains the attributes for that point. This is the most common form that most clustering algorithms require.

Dissimilarity matrix

A n by n triangle matrix where n is the number of points where the i-th and j-th entry is the distance between point i and point j.

General Applications of Clustering

There are multiple:

- Pattern Recognition and Image Processing
- Spatial Data Analysis - Creating thematic maps in GIS by clustering features
- Economic Science, especially market research
- WWW - Documents and web-logs
- Biology - Clustering of gene expression data

Major Clustering Approaches

There are multiple ways to cluster points

- Partitioning methods like find k partitions, minimizing some objective functions
- Hierarchical methods like creating a hierarchical decomposition of the set of objects
- Density-based methods
- Other methods includes: grid-based, Neural networks like SOMs, Graph methods, ...

Partitioning Methods

The goal of these methods is to construct a partition of a database D of n objects into a set of k clusters minimizing an objective functions. Exhaustively enumerating all possible partitions into k sets in order to find the global minimum is too expensive.

To improve performance we use a **Heuristic methods**:

- Partition the data according to some initialization strategy into k groups and compute their cluster representations
- Improve the initial representations iteratively:
  - Assign each object to the cluster it "fits best" in the current clustering
  - Compute new cluster representations based on these assignments
  - Repeat until the changes in the objective function from one iteration to the next drops below a threshold.

Types of cluster representations:

- **k-means**: A cluster is represented by the centre of a cluster
- **k-medoid**: A cluster is represented by the most central object
- **EM**: a cluster is represented by a probability distribution (generally a Gaussian)

K-Means

The goal: For a given k, form k groups so that the sum of the squared distance between the mean of the groups and their elements is minimum. Basically it minimize the variance of the element and the group's mean within each group.

The object is defined as $p = (x_1^p, \ldots, x_d^p)$ are points in a d-dimensional vector space (the mean of a set of points must be defined)

Definition 2.0.1 (Centroid). *μ_C mean of all points in a cluster C*

Measure for the compactness of a cluster C:

$$TD^2(C) = \sum_{p \in C} dist(p, \mu_C)^2$$

Measure for the compactness of a clustering:

$$TD^2 = \sum_{i=1}^{k} TD^2(C_i)$$

K-Mean Clustering Algorithm

Given k, the k-means algorithm is implemented in 4 steps:

1. Partition the objects into k nonempty subsets
2. Compute the centroid of the clusters of the current partition
3. Assign each object to the cluster with the nearest representative
4. Go back to step 2, stop when the representatives do not change

Strength.

- Relatively efficient: $O(tkn)$ where n is the number of objects, k is the number of clusters, and t is the number of iterations. Normally $k, t \ll n$.
- Easy implementations

Weakness.

- Applicable only when mean is defined
- Need to specify k, the number of clusters, in advance.
- Sensitive to noisy data and outliers
- Clusters are forced to have convex shapes
- Results and runtime are dependent on the initial partition; often terminates at a local optimum.

Note

Due to the greedy nature of the standard K-means algorithm if the initial partitions don't cut along clear clusters then you can have a sub-optimum clustering.

Several variants of the k-means methods exist, e.g., extending k-means by methods to eliminate very small clusters, merging and split of clusters; user has to specify additional parameters.

Model-Based Clustering

Clusters are represented by Gaussian distributions

Definition 2.0.2 (Cluster ϕ_i, $1 \leq i \leq k$ represented as a Gaussian).

$$\phi_i(x|\mu_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^d}\sqrt{|\Sigma i|}} \cdot e^{-\frac{1}{2}\cdot[(x-\mu_i)^T \cdot \Sigma_i^{-1} \cdot (x-\mu_i)]}$$

Where:

Notation	*What it means*
x	*represents the d-dimensional data*
Subscript i	*specifies the i-th cluster*
μ_i	*is the d-dimensional mean vector of the i-th cluster*
Σ_i	*is the $d \times d$ covariance matrix of the i-th cluster*

Iterative EM

Iterative Expectation-Maximization (EM) for maximum likelihood clustering

- One Iteration:
  1. **E-Step** - Computes a matrix z such that z_{ic} is an estimate of the conditional probability that x_i belongs to the cluster ϕ_c, given the current parameter estimates for μ_c and $\sum_C$ for each cluster ϕ_C
  2. **M-Step** - Computes the maximum likelihood parameter estimates for each cluster (i.e., μ_C and $\sum_C$), given the current cluster membership probabilities in z.
- Converges in the limit to the maximum likelihood values for the Gaussian mixture model.

Iterative EM Algorithm

Given k, the *EM* algorithm is implemented in 4 steps:

1. **Initialize** estimate z_{ik}, e.g., from a discrete partition of objects into k nonempty subsets ($z_{ik} = 1$ if x_i belongs to group k, otherwise $z_{ik} = 0$)
2. **M-Step**

$$n_c \leftarrow \sum_{i=1}^{n} z_{ic}$$
$$w_c \leftarrow \frac{n_c}{n}$$
$$\mu_c \leftarrow \frac{\sum_{i=1}^{n} z_{ic} x_i}{n_c}$$

   Σ_c depends on the model

3. **E-Step**

$$z_{ic} \leftarrow \frac{w_c \phi_c(x_i|\mu_c, \Sigma_c)}{\sum_{j=1}^{k} w_j \phi_j(x_i|\mu_j, \Sigma_j)}$$

4. Stop when convergence criteria are satisfied, else go back to Step 2

Parameters in MCLUST

In MCLUST, each covariance matrix ix parameterized by eigenvalue decomposition

$$\Sigma_i = \lambda_i D_i A_i D_i^T$$

Where:

Notation	What it means
D_i	is the orthogonal matrix of eigenvectors; determines the **orientation** of the ellipsoid for the i-th cluster.
A_i	is the diagonal matrix whose elements are proportional to the eigenvalues of Σ_i; determines the shape of the density contour
λ_i	is a scalar; determines (with A_i) the volume of the ellipsoid for the i-th cluster - proportional to $\lambda_i^d \ \|A_i\|$ (where d is the data dimensionality).

Cluster characteristics like the orientation, volume, shape are typically estimated from the data and can vary between clusters or can be constrained.

K-Medoid

The basic idea the objective is: For a given k, find k representatives in the dataset so that, when assigning each object to the closest representative, the sum of the distance between the representatives and objects which are assigned to them is minimal.

Requires arbitrary objects and a distance function.

Definition 2.0.3 (Medoid). *m_C representative object in a cluster C*

Measure for the compactness of a cluster C:

$$TD(C) = \sum_{p \in C} dist(p, m_C)$$

Measure for the compactness of a clustering:

$$TD = \sum_{i=1}^{k} TD(C_i)$$

Basically like k-mean but you do not square the distance.

PAM Algorithm

1. Select k objects arbitrarily as medoids (representatives); assign each remaining (non-medoid) object to the cluster with the nearest representative, and compute $TD_{current}$
2. For each pair (medoid M, non-medoid N) compute the value $TD_{N \leftrightarrow M}$ i.e., the value of TD for the partition that results when swapping M with N.
3. Select the non-medoid N for which $TD_{N \leftrightarrow M}$ is minimal
4. If $TD_{N \leftrightarrow M}$ is smaller than $TD_{current}$, then swap N with M, set $TD_{current}$ with $TD_{N \leftrightarrow M}$, and go back to step 2.
5. Stop

This algorithm is at least quadratic because you have to compare every pair.

K-Medoid Clustering: CLARA and CLARANS

CLARA. :

- Additional parameter: *numlocal*
- Draws *numlocal* samples of the dataset
- Applies PAM on each samples
- Returns the best of these sets of medoids as output

CLARANS.

- Two additional parameters: *maxneighbor* and *numlocal*
- At most *maxneighbor* many pair (medoid M, non-medoid N) are evaluated in the algorithm.
- The first pair (M, N) for which $\text{TD}_{N \leftrightarrow M}$ is smaller than $\text{TD}_{current}$ is swapped (instead of the pair with the minimal value of $\text{TD}_{N \leftrightarrow M}$)
- Finding the local minimum with this procedure is repeated *numlocal* times

In terms of run time CLARANS is faster than CLARA which is faster than PAM.

Strengths and Weaknesses

Strenghts.

- Applicable to arbitrary objects and distance functions
- Not so sensitive to noisy data and outliers as k-means

Weaknesses.

- Inefficient
- Like k-means you need to specify the number of clusters k in advance, and clusters are forced to have convex shapes.
- The results and runtime for CLARA and CLARANS may ary largely due to the randomization.

Initialization of Partitioning Methods

There are many ways to do the initial partitioning of clusters. One way is to:

- Draw m different (small) samples of the dataset
- Cluster each sample to get m estimates for k representatives $A = (A_1, \ldots, A_k), B = (B_1, \ldots, B_k), M = (M_1, \ldots, M_k)$
- Then, cluster the set $DS = A \cup B \cup \cdots \cup M$ m times, using the set $A, B, \ldots C$ as respective initial partitioning
- Use the best of these m clustering as initialization for the partitioning clustering of the while dataset.

Choice of the Parameter k

You can try all possible values between 2 and $n-1$, but that would be too costly. We need to choose the "best" clustering parameter for k.

Measuring a Cluster's Quality

But how can we measure the quality of a clustering? This is something that has to be independent of k and this measurement has to be monotonically decreasing with an increasing value of k when measuring the compactness of a clustering using k-mean and k-medoid.

Silhouette Coefficient

The Silhouette-Coefficient measure the quality of a k-means or a k-medoid clustering that is independent of k.

Definition 2.0.4 (Silhouette Coefficient)**.** *Let:*

- $a(o)$*: the average distance between an object* o *and the objects in its cluster* A*.*
- $b(o)$*: the average distance between an object* o *and the objects in its "second closest" cluster* B*.*

The silhouette of o *is then defined as*

$$s(o) = \frac{b(o) - a(o)}{max\{a(o), b(o)\}}$$

This measures how good the assignment of o *to its cluster is*

- $s(o) = -1$*: bad, on average closer to members of* B*.*
- $s(o) = 0$*: in-between* A *and* B*.*
- $s(o) = 1$*: good assignment of* o *to its cluster* A*.*

The Silhouette Coefficient s_C *of a clustering is the average of all objects:*

- $0.7 < s_C \leq 1.0$ *strong structure*
- $0.5 < s_C \leq 0.7$ *medium structure*
- $0.25 < s_C \leq 0.5$ *weak structure*
- $s_C \leq 0.25$ *no structure*

Density-based Methods: DBSCAN

The basic idea of *Density-based Clustering* is that clusters are dense regions in the data space, separated by regions of lower object density.

The intuition for the formalization of the basic idea is that for any point in a cluster, the local point density around that point has to exceed some threshold and the set of points from one cluster is spatially connected.

Let's have a local point density of a point p defined by two parameters:

- ε - radius for the neighbourhood of point p: $N_\varepsilon(p) = \{q \in \text{dataset } D | dist(p,q) \leq \varepsilon\}$. Meaning $N_\varepsilon(p)$ is the set of points that are within ε distance away from p.
- *MinPts* - minimum number of points in the given neighbourhood $N(p)$.

q is called a **core object** or **core point** with regards to ε, *MinPts* if $|N_\varepsilon| \geq MinPts$.

Basic Definition

- p is **directly density-reachable** from q w.r.t. ε, and *MinPts* if
  1. $p \in N_\varepsilon(q)$
  2. q is a core object w.r.t. ε, and *MinPts*
- **density-reachable**: transitive closure of directly density-reachable. Meaning, p is density-reachable of q by going through another core point that is **density-reachable** or **directly density-reachable**.
- p is **density-connected** to a point q w.r.t. ε, *MinPts* if there is a point o such that both, p and q are density-reachable from o w.r.t. ε, and *MinPts*.
- **Density-based Cluster**: non-empty subset S of database D satisfying:
  1. Maximality: if p is in S and q is density-reachable w.r.t. ε, *MinPts* from p then q is in S.
  2. Connectivity: each object in S is density-connected w.r.t ε, *MinPts* to all other objects.
- **Density-based Clustering** of a database $D : \{S_1, S_2, \ldots, S_n; N\}$ where
  - $S_1, \ldots, S_n$: all density-based clusters in the database D.
  - $N = D \setminus \{S_1, \ldots, S_n\}$ is called **noise** or objects that are not in a cluster.

DBSCAN Algorithm

Basically each object in a density-based cluster C is density-reachable from any of its core-objects. Nothing else is density-reachable from core objects.

```
1 for each o in D:
2   if o is not classified(o):
3     if core_object(o):
4       # collect all objects density-reachable
        ↪ from o and assign them to a new
        ↪ cluster
5     else:
6       # assign o to NOISE
```

Listing 1: Basic DBSCAN Algorithm

Density-reachable objects are collected by performing successive ε-neighbourhood queries.

DBSCAN Algorithm Performance

Runtime complexities:

	N_ε-query	DBSCAN
without DS support	$O(n)$	$O(N^2)$
tree-based support	$O(\log(n))$	$O(n \cdot \log(n))$
direct access to neighbours	$O(1)$	$O(n)$

Determine the parameters ε and *MinPts*

We still have the same issues as before of determining the correct parameters for ε and *MinPts*. One could use the point density of the least dense cluster in the dataset as the parameter but how do we determine this value? A heuristic could be looking at the distances to the k-nearest neighbour. Using a function like $k - distnace(p)$ that will generate a distance from p to its k-nearest neighbour and you can generate a plot of k-distances from all objects and sort them in descending order. A cutoff point could be the first kink in the trend line of the plot.

But this doesn't work for all examples

Strengths and Weaknesses

Advantages.

- Clusters can have arbitrary shapes and size
- Number of clusters can be determine automatically
- Can separate clusters from surrounding noise
- Can be supported by spatial index strictures

Weaknesses.

- Input parameters may be difficult to determine
- In some situations very sensitive to input parameter settings

Hierarchical Methods

Global parameters to separate all clusters with a partitioning clustering method may not exist.

You need a hierarchical clustering algorithm in these situations.

Basic Notation

Hierarchical decomposition of the dataset (with respect to a given similarly measure) into a set of nested clusters.

Results represented by a so called *dendrogram*. Nodes in the dendrogram represent possible clusters. They can constructed bottom-up (agglomerative approach) or top down (divisive approach).

Interpretation of the dendrogram:

- The root represents the while dataset
- A leaf represents a single objects in the dataset
- An internal node represent the union of all objects in its sub-tree
- The height of an internal node represents the distance between its two child nodes.

Agglomeratice Hierarchical Clustering

1. Initially, each object forms its own cluster.
2. Compute all pairwise distances between the initial clusters (objects).
3. Merge the closest pair (A,B) in the set of the current clusters into a new cluster $C = A \cup B$
4. Remove A and B from the set of current clusters; insert C into the set of current clusters.
5. **If** the set of current clusters contains only C then stop
6. **Else** determine the distance between the new cluster C and all other clusters in the set of current clusters; go to step 3.

Requires a distance function for clusters (sets of objects)

Single Link Methods and Variants

Given a distance function $dist(p, q)$ for database objects. The following distance functions for clusters (i.e., sets of objects) X and Y are commonly used for hierarchical clustering.

Definition 2.0.5 (Single-Link)**.**

$$dist_{sl}(X, Y) = min_{x \in X,\ y \in Y} dist(x, y)$$

Definition 2.0.6 (Complete-Link)**.**

$$dist_{cl}(X, Y) = max_{x \in X,\ y \in Y} dist(x, y)$$

Definition 2.0.7 (Average-Link)**.**

$$dist_{al}(X, Y) = \frac{1}{\mid X \mid \cdot \mid Y \mid} \cdot \sum_{x \in X,\ y \in Y} dist(x, y)$$

Density-based hierarchical clustering: OPTICS

Observation: Dense clusters are compelled contained by less dense clusters. The idea is Process objects in the "right" order and keep track of point density in their neighbourhood.

Core- and Reachability Distance

Parameters: "generating" distance ε, fixed value *MinPts*.

Definition 2.0.8 (core-distance$_{\varepsilon,\ MinPts}(o)$)**.** *Smallest dance such that o is a core object if that distance is less than ε; unknown otherwise.*

Definition 2.0.9 (reachability-distance$_{\varepsilon,\ MinPts}(p, o)$)**.** *Smallest distance such that p is directly density-reachable from o if that distance is less than ε; unknown otherwise.*

$$= max(coredistance_{\varepsilon,\ MinPts}(o), dist(o, p))$$

OPTICS Algorithm

The basic data structure is: controlList which memorize shortest reachability distance seen so far ("distance of a jump to that point"). Visit each point and always make the shortest jump. The output is:

- order of points
- core-distance of points
- reachability-distance of points

```
for o in Database:
  # initially, o.processed = false for all objects o
  if o.processed() == False:
    ControlList.insert(o, "?")
  while not ControlList.empty():
    elem = ControlList.get(o, r_dist)
    ep_neighbors = get_epsilon_neighbour(o)
    c_dist = core_distance(o)
    o.processed = True
    file.write(o, r_dist, c_dist)
    if is_core_object(o) and dist(o) <= epsilon:
      for p in ep_neighbors and not p.processed:
        determine r_dist_p = reachability_distance(p,o)
        if (p,_) not in ControlList:
          ControlList.insert(p,r_dist_p)
        elif (p,old_r_dist) in ControlList and r_dist_p < old_r_dist:
          ControlList.update(p,r_dist_p_)
```

Listing 2: OPTICS Algorithm

OPTICS: Properties

Flat density-based clusters w.r.t., $\varepsilon^* \leq \varepsilon$and$MinPts$ afterwards:

- Starts with an object o where c-dist(o) $\leq \varepsilon^*$ and r-dist(o) $> \varepsilon^*$
- Continue while r-dist $\leq \varepsilon^*$

Performance: approximate runtime(DBSCAN(ε, $MinPts$))

- $O(n \cdot \text{runtime}(\varepsilon\text{-neighbourhood-query}))$
- without spatial index support (worst-case): $O(n^2)$
- tree-based spatial index support $O(n \cdot \log(n))$

OPTICS: The Reachability Plot

Represents the density-based clustering structure. It's easy to analyze and independent of the dimension of the data.

OPTICS: Parameter Sensitivity

Relative insensitive to parameter settings, good result if parameters are just ‘"large enough"

Hierarchical Clustering Discussion

Advantages:

- Does not require the number of clusters to be known in advance
- No (standard methods) or very robust parameters as seen in OPTICS
- Computes a complete hierarchy or clusters
- Good result visualization integrated into the methods
- A "flat" partition can be derived afterwards (e.g., via a cut through the dendrogram or the readability plot)

Disadvantages:

- May not scale well, the runtime for the standard method is: $O(n^2 \log(n^2))$
- The runtime for OPTICS isn't better without index support $O(n^2 \log(n))$

Chapter 3: Clustering 2

Density-Based Clustering Revisited: HDBSCAN*

As a review of Density-based Clustering. Statistically sound (if we drop the border points) and it does not require the number of clusters as input. Clusters can have arbitrary shapes and sizes. These clusters can be separated from noise. However, many of these algorithms give clusters corresponding to a single density level, and these density levels are often difficult to set. You can also have clusters that have different densities and/or nested within clusters. If these clusters are hierarchical, then cluster are difficult to automatically extract.

Hierarchy Simplification and Cluster Extraction

The original DBSCAN algorithm is considered a "flat" density-based clustering algorithm. *Direct density reachability* is only defined between core objects:

- Density connectivity -> Transitive cluster of direct density reachability
- Clusters -> connected components of dense (core) objects

The density threshold ε is critical and difficult to set. Moreover, a single, global threshold separating all clusters may not exist in the dataset.

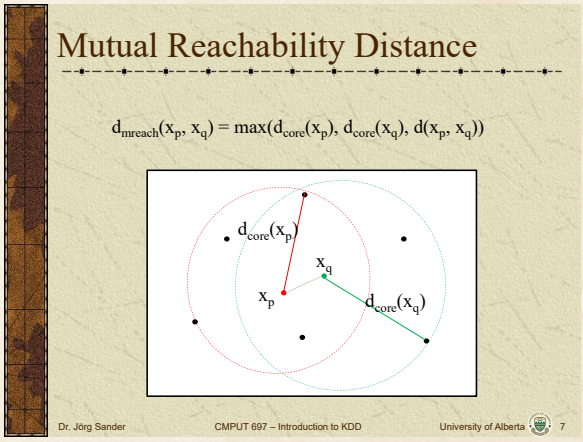
Here comes **HDBSCAN*** to the rescue. **HDBSCAN*** does:

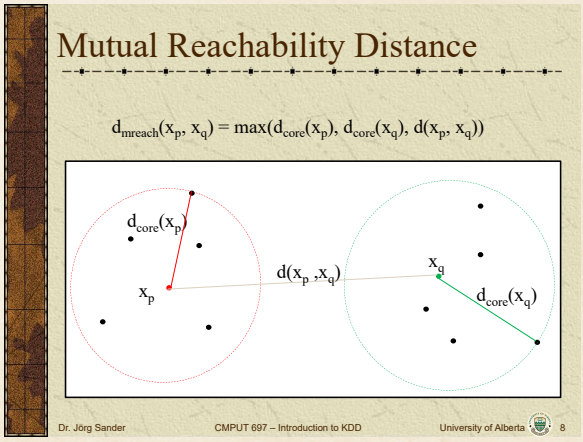
- **Complete density-based clustering hierarchy**
- **Simplified cluster tree** of significant clusters
- **Cluster stability measure** that allows extracting a **flat clustering via local cuts** through the hierarchy
- Incorporation of user-provided constraints for **semi-supervised cluster extraction**
- Basis for **outlier detection** (unifying local and global methods)
- **Visualization** of dendrogram, each ability plot, etc

Key Concepts

There are two key concepts for HDBSCAN*.

- **core-distance** w.r.t m_{Pts} where $d_{core}(p)$ = distance from p to its m_{Pts}-nearest neighbour
- **mutual reachability distance** w.r.t. m_{Pts} where $d_{mReach}(p,q) = \max\{d_{core}(p), d_{core}(q), d(p,q)\}$

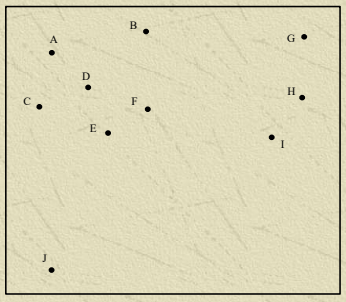




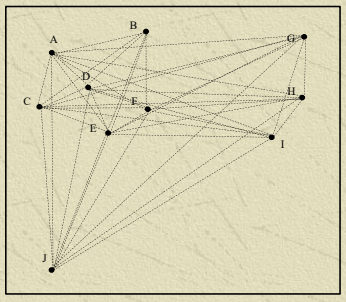
HDBSCAN* Algorithm (Sketch)

1. MST in the transformed space of **symmetric reachability distances** where $edge(p,q)$; edge weight $= d_{mReach}(p,q)$
2. Extend the MST with **self-edges** to model transitions involving isolated objects (dense <-> noise). edge weight; $core_dist(o)$
3. Iteratively remove edges from MST_{ext} in decreasing order while properly relabeling sub-clusters and noise (**hierarchy simplification**) and keeping track of **stabilities** of the clusters.

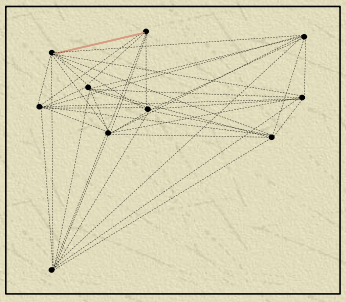
An example dataset



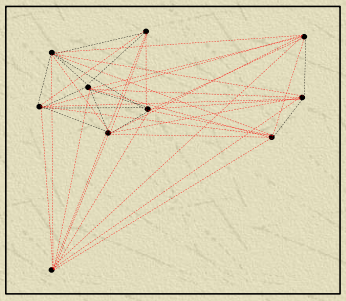
Mutual Reachability Graph



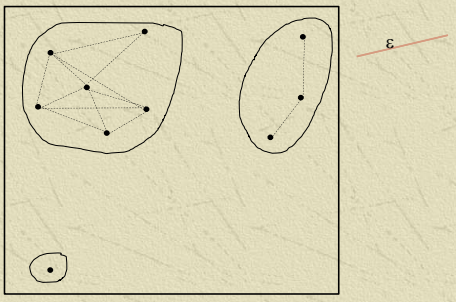
Extracting clusters at a single ε



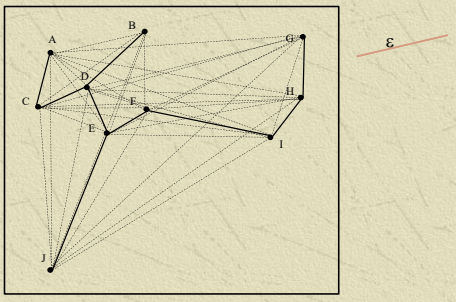
Remove all edges larger than ε



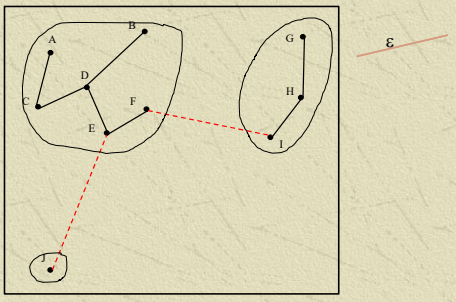
Remove all edges larger than ε



Only need an MST (in the space of mutual reachability distances) to extract clusters at any level



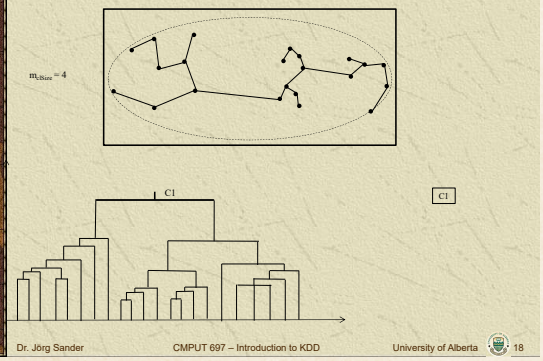
Only need an MST (in the space of mutual reachability distances) to extract clusters at any level



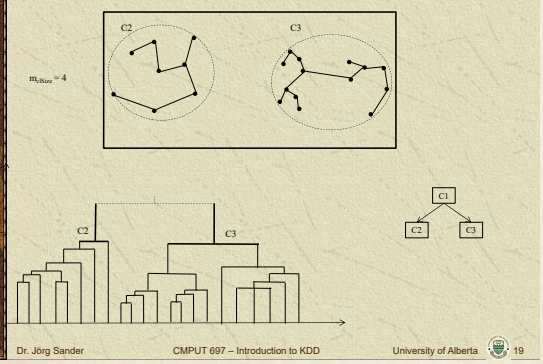
Hierarchy Simplification

Removing edges in decreasing order/increasing density threshold. Consider loosing noise/spurious components not to be a true split, but cluster only shrinks. Simplifies hierarchy by orders of magnitude which leads to a simplified cluster tree

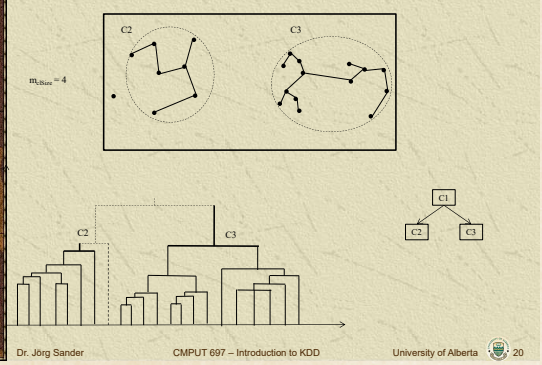
Cluster Tree Example



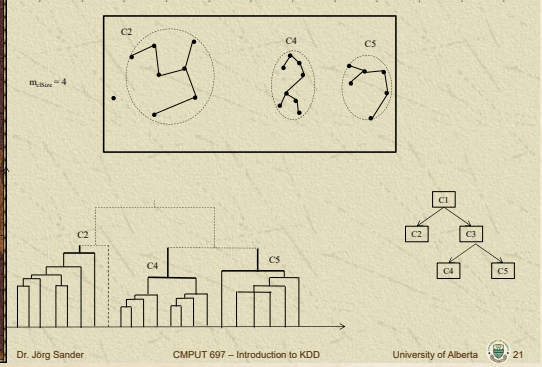
Cluster Tree Example



Cluster Tree Example



Cluster Tree Example

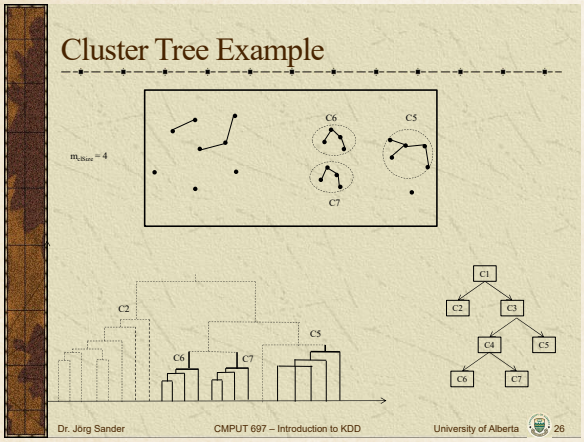


Cluster Tree Example

Cluster Tree Example

Cluster Tree Example

Cluster Tree Example



Extracting Clusters from a Hierarchy

Typically done by a simple horizontal "cut" through a cluster tree (~single density level) A few algorithms to automatically extract clusters and cluster trees, based on "ad-hoc principles" and critical parameters.

Cluster Stability

Definition 3.0.1 (Relative Excess of Mass). *Specially designed for density-based clustering hierarchies. Accounts for the different density profiles of objects in a cluster.*

$$S(C_i) = \sum_{x_j \in C_i} (\lambda_{max}(x_j, C_i) - \lambda_{min}(C_i))$$
$$= \sum_{x_j \in C_i} \left(\frac{1}{\varepsilon_{min}(x_j, C_i)} - \frac{1}{\varepsilon_{max}(C_i)} \right)$$

Notation	What it means
$\lambda_{min}(C_i)$	minimum density level at which C_i exist
$\lambda_{max}(x_j, C_i)$	density level beyond which object x_j no longer belongs to cluster C_i because C_i is split or disappears
$\varepsilon_{max}(C_i)$ and $\varepsilon_{min}(x_j, C_i)$	are the corresponding values for the threshold ε

Optimal Cluster Selection

Flat, non-overlapping clustering solution can be extracted from the set of all K clusters in a simplified hierarchy. Optimization Problem:

- Maximize the overall Stability of the collection of selected clusters
- Such that parent-children clusters are mutually exclusive

$$\max_{\delta_2,\ldots,\delta_k} J = \sum_{i=2}^{k} \delta_i S(C_i)$$

Subject to

- $\delta_i \in \{0, 1\}, i = 2, \ldots, k$
- exactly one $\delta_{(\cdot)} = 1$ in each path from a leaf cluster to the root

Unsupervised

Global Optimal Solution: via DP algorithm (local cuts through the tree)

$$\hat{S}(C_i) = \begin{cases} S(C_i), & C_i \text{ is a leaf node} \\ \max\{S(C_i), \hat{S}(C_{i_l}) + \hat{S}(C_{i_r})\} & C_i \text{ internal node} \end{cases}$$

Where:

Notation	What it means
$\hat{S}(C_i)$	equals total stability
C_{i_l}	left child of C_i
C_{i_r}	right child of C_i

Semi-supervised

Select clusters that overall best satisfy given constraints (should-link or should-not-link), but do not violate the density-based cluster model. Replace cluster stability with constraint satisfaction

$$\Gamma(C_i) = \frac{1}{2n_c} \sum_{x_j \in C_i} \gamma(x_j, C_i)$$

Need to add virtual nodes to account for constraints involving noise

$$\hat{\Gamma}(C_i) =$$
$$\begin{cases} \Gamma(C_i), C_i \text{ is not a non-virtual leaf node} \\ \max\{\Gamma(C_i), \hat{\Gamma}(C_{i_l} + \hat{\Gamma}(C_{i_r} + \Gamma(C_i^{\emptyset})))\}, C_i \text{ internal node} \end{cases}$$

M_c is the total number of constraints. $\gamma(x, c)$ is the number of constraint involving x, satisfied in c

Current and Future Extension

Other or better measures for clustering extraction (cluster validation measures?). What about other density-estimates. Scaling up density-based hierarchical clustering.

- Investigate faster to compute density estimates
- Application of data summarization techniques

Cluster Validation

Motivation and preliminaries

> The validation of clustering structures is the most difficult and frustrating part of cluster analysis.
>
> Without a string effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage - Jain and Dubes, *Algorithms for Clustering Data*, 1988

Cluster validation refers broadly to a collection of procedures to evaluate clustering results in an objective and quantitative way. Such procedures can help the analyst to answer question such as:

- Have we really found groups?
- How unusual or significant are the groups we found?
- What is the quality of the groups we have found?
- What is the most natural / appropriate number of clusters (or grouping based on parameters of a clustering methods)?

External, Internal, and Relative Validation Indices

To validate a cluster or measure its performance we need a sort of **index**, a sort of clustering validation or validity criterion. There are three types of indices:

- **External:** Measure the degree of agreement between two clustering structures (partitions or hierarchies) Typical an obtained clustering against a ground truth
- **Internal:** Measure some sort of quality or suitability of a given clustering structure using only the data themselves.
- **Relative:** Provide an ordering (usually a scoring) for a collection of different clusterings. The term typically referees to internal criteria which are also relative

External Criteria

Clustering is an unsupervised task, but in some cases we have a *ground truth*:

- Domain expert
- Visual inspection of low-dimensional data (2D, 3D)
- Synthetically generated data, benchmark dataset

External criteria can be used to measure the degree of matching between a given clustering and the ground truth, usually for research purposes.

There are different criteria available in the literature:

- Jaccard Coefficient
- (Adjusted) Rand Index
- Γ Statistic
- F-Measure
- Normalized Mutual information

Pair-counting-based External Criteria

Notation	What it mean
a	No. of pair that belong to the same class and to the same cluster
b	No. of pairs that belong to the same class but to different clusters
c	No. of pairs that belong to different classes but to the same cluster
d	No. of pair that belong to different classes and to different clusters

Rand Index

Definition 3.0.2 (Rand Index)**.**

$$RI = \frac{a+d}{a+b+c+d}$$

Jaccard Coefficient

Definition 3.0.3 (Jaccard Coefficient)**.**

$$Jc = \frac{a}{a+b+c}$$

Better than Rand Index because it's not influenced by the number of pairs that are in different clusters and different classes, which is a large number.

Adjusted Rand Index

Both the Rand and Jaccard are not *adjusted for chance*. Their **expected value** is **not null** when two randomly generated partitions are compared. Therefore the adjustment for chance gives us this:

$$\text{Adjusted Criterion} = \frac{\text{Criterion} - E\{\text{Criterion}\}}{\text{Max Criterion} - E\{\text{Criterion}\}}$$

Definition 3.0.4 (Adjusted Rand Index)**.** *ARI can be written as:*

$$ARI = \frac{a - \frac{(a+c)(a+b)}{M}}{\frac{(a+c)+(a+b)}{2} - \frac{(a+c)(a+b)}{M}}$$

Where: $M = a+b+c+d$.

Internal Criteria

Validation for practical scenarios. Usually only the data and the clustering result to be assessed are available.

The **Internal Criteria** is used that make use of the above information **only**. Example the object function of the k-mean algorithm is

$$J = \sum_{c=1}^{k} \sum_{x_j \in C_c} d(x_j, \overline{x}_c)^2$$

Index J has been used to determine the number of clusters. Stopping rule for related hierarchical algorithms (e.g. Ward's)

Detection of the *knee* can be made easier by using some sort of transformation like

$$\Delta J(k) = abs\left(\frac{J(k-1) - J(k)}{J(k) - J(k+1)}\right)$$

But in practice it not always that simple.

Silhouette Width Criterion

Definition 3.0.5 (SWC)**.** *The average of the individual silhouette:*

$$SWC = \frac{1}{N}\sum_{i=1}^{N} s(i)$$

Definition 3.0.6 (Silhouette (i-th object)).

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

where:

- $a(i)$ *average dissimilarity to the cluster the i-th object belongs to*
- $b(i)$ *average dissimilarity to the nearest neighbouring cluster*

Range: $SWC \in [-1, 1]$. *Compute Complexity:* $O(N^2)$

Definition 3.0.7 (Silhouette width). *For* $i \in C_i$

$$a(i) = \frac{1}{\mid C_i \mid -1} \sum_{j \in C_i, j \neq i} d(i, j)$$

$$b(i) = min_{J \neq I} \frac{1}{\mid C_J \mid} \sum_{j \in C_j} d(i, j)$$

Cluster Validation for Density-Based Clustering

Density-based clustering paradigm:

- Allows the concept of the noise to be modelled
- Allow clusters of varied shapes to be modelled.
- Allows to model the presence of nested clusters and varied densities in the data.

All Points Core Distance

Definition 3.0.8 (All Points Core Distance).

$$a_{pts}coredist(o) = \left(\frac{\sum_{i=2}^{n_i} \left(\frac{1}{KNN(o,i)} \right)^{2d}}{n_i - 1} \right)^{-\frac{1}{d}}$$

DBCV: Density-Based Clustering Validation

Definition 3.0.9 (Validation Index of a cluster).

$$V_C(C_i) = \frac{min_{1 \leq j \leq l, j \neq i}(DSPC(C_i, C_j)) - DSC(C_i)}{max(min_{1 \leq j \leq l, j \neq i}(DSPC(C_i, C_j)), DSC(C_i))}$$

Definition 3.0.10 (Validation index of a while Clustering).

$$DBCV(C) = \sum_{i=1}^{i=l} \frac{\mid C_i \mid}{\mid O \mid} V_C(C_i)$$

How to deal with noise for other indices?.

- assign all noise points to a single cluster
- assign each noise to its closest cluster
- assign each noise to a singleton cluster
- remove all noise points (this doesn't make any sense), and
- remove all noise with a proportional penalty

The only reasonable choice is the last one, which is the same as the weighted averaging approach in DBCV

Chapter 4: Outlier Detection

Introduction

> An outline is an observation which deviates so much from the other observation as to arouse suspicious that it was generated An outline r is an observation which deviates so much from the other observation as to arouse suspicious that it was generate by a different mechanism - Hawkins 1980

Statistics-based intuition
Normal data object follow a "generating mechanism", e.g., some given statistical process, Abnormal objects deviate from this generating mechanism

Data is usually multivariate i.e., multi-dimensional the basic model is univariate or 1-dimensional. There is usually more than one generating mechanism or statistical process underlying the data a basic model assumes only one "normal"generating mechanism. Anomalies may represent a different class (generating mechanism) of objects, so there may be a small group of similar objects that are outliers, while a basic model assumes that outliers are rare and are "scattered" observations. A lot of different approaches have been proposed.

Causes for outliers

There are many causes that can generate outliers these include:

- Measurement errors
- Execution faults
- Intrinsic variability
- Outlier generated by a different process

Applications of outlier detection

There are many applications for outlier detection these include:

- Fraud detection - Abnormal buying patterns from the cardholder may indicate that the card info has been stolen.
- Medicine - Unusual symptoms or test results may indicate potential health problems of a patient.
- sport statistics - Moneyball
- Detecting measurement errors - Abnormal values could provide an indication of a measurement error
- etc

Outlier detection settings

- Supervised - Training data with normal and abnormal data objects are available, often, the classification problem is lightly imbalanced
- Semi-supervised - Only training data for the normal class(es) are provided
- Unsupervised - No training data available

We will mostly focus on unsupervised outlier detection methods

Statistical Methods

Normal data objects follow a (given) distribution and occur with high probability with respect to the model. Outliers deviate strongly form this distribution i.e., have low probability with respect to the model. There are many tests available for discordance, largely influenced by:

- Type of distribution
- unimodal vs multi-modal (mixture models)
- univariate vs multivariate (one-dimensional vs multi-dimensional)

Definition 4.0.1 (Multivariate Normal Distribution)**.**

$$N(x \mid \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d}\sqrt{\mid \Sigma \mid}} \cdot^{-\frac{1}{2}\cdot[(x-\mu)^t \cdot \Sigma^{-1} \cdot (x-\mu)]}$$

where:

Notation	*What it means*
x	*represents a d-dimensional data point*
μ	*is the d-dimensional mean vector of all data points*
Σ	*is the $d \times d$ covariance matrix of the data*
$d^2(x,\mu) = (x-\mu)^t \cdot \Sigma^{-1} \cdot (x-\mu)$	*squared Mahalanobis distance*
$d^2(x,\mu)$	*has a χ^2 -distribution with d DoF*

Outliers are points x with $d^2(x,\mu) > \chi^2(0.975)$ or around 3 standard deviation in 1-d
As for ***robustness****: Mean and co-variances are very sensitive to outliers as these values are typical estimated on the while dataset (including the potential outliers), $d^2(x,\mu)$ is used to determine the outliers although its values are influenced by these outliers. You could use the minimum covariance determinant to minimize the influence of outliers on the Mahalanobis distance.*
Outputs a label but can also output a score. Data distribution is fixed and only unimodal.

Distance-Based Approaches

Proposed for multi-variate analysis without knowing data distribution. Judge a point based on the distance(s) to its neighbours. There are several variants proposed

Basic assumptions:

- normal objects have a dense neighbourhood
- Outliers have a less dense neighbourhood, since they are far apart from their neighbour.

DB(ε, π)-Outliers

The basic model is from Knorr and Ng 1997.

Given a radius ε and a percent π: A point p is considered an outlier if at most π percent of all other points have a distance to p less than ε.

Outlier scoring based on kNN distances

There are two general models:

- Take the kNN distance of a point as its outlier score [Ramaswamyet al 2000]
- Aggregate the distance of a point ot its 1NN, 2NN, ..., kNN into an outlier score [Angiulliand Pizzuti 2002]

The result produces "outlier scores"

Density-Based Approaches

With density-based approaches the general idea is that you compare the density around a point with the density around its local neighbours. The relative density of a point compared to its neighbour is computed as an outlier score. Approaches also differ in how to estimate density and in how they compare local densities.

Basic assumptions:

- The density around a "normal" object is similar to the density around its neighbours
- The density around an outlier is considerably different from the density around it neighbours

Local Outlier Factor (LOF)

Distance-based outlier detection has problems with different densities, so how do we compare neighbourhoods from areas with different densities? For example:

- DB(ε, π)-outlier model - The parameters ε and π cannot be chosen so that outliers detected for a super dense cluster will not falsely label normal points as outliers for a less dense cluster.
- Outlier based on kNN-distance - The KNN-distance of objects in a sparse cluster are larger than the kNN-distance of an outlier for a dense cluster.

LOF compares the kNN distance of a point p with the kNN distance of its k distances of its k nearest neighbour.

Definition 4.0.2 (Local Reachability Distance (lrd) of a point p)**.** *Inverse of the average reachability distance of p's k nearest neighbours*

$$lrd_k(p) = \frac{1}{\left(\frac{\Sigma_{o \in kNN(p)} rdist_k(p,o)}{|KNNs(p)i|}\right)}$$

Definition 4.0.3 (Local Outlier Factor (LOF) of a point p)**.** *Average ratio if lrds of p'p neighbours to the lrd of p*

$$LOF_k(p) = \frac{\Sigma_{o \in kNN(P)} \frac{lrd_k(o)}{lrd_k(p)}}{\mid kNNs(p) \mid}$$

Properties of LOF

If the LOF is close to 1 then the point is in a cluster (region with homogeneous density around the point and its neighbours). If the LOF $\gg 1$ then the point is an outlier. The parameter k specifies the size of the reference set.

Global-Local Outlier Score from Hierarchies (GLOSH)

- KNN outliers are "global" outliers, they consider the whole dataset as the reference set
- LOF outliers are "local" outliers, they consider a fixed local neighbourhood as a reference set
- GLOSH outliers can be both "local" or "global" outliers, they consider a dynamic (not predetermined) reference set

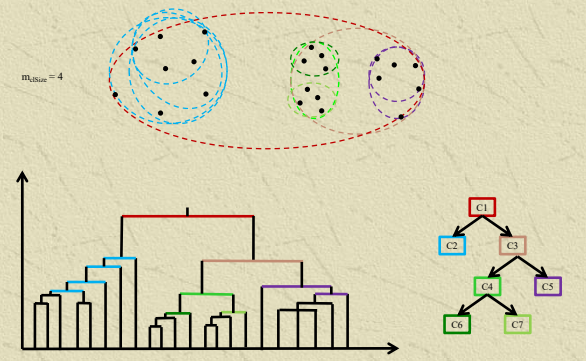
The intuition of GLOSH is that normal objects are generated by some mechanism(s), modelled by the clusters in a dataset. Reference set for an object is based on the closest cluster within the density-based cluster hierarchy.

Definition 4.0.4 (GLOSH)**.** *The **closet cluster for a data object** x is the smallest or densest cluster C that contains x, i.e.,*

1. *C is a node in the hierarchy at density level $\alpha \sim radius\varepsilon$*
2. *$\varepsilon \geq d_{core}(x)$, since x must be a core object*
3. *$x \in C$*
4. *no child node of C satisfies condition 1 -> 3.*

*The **highest density level of a cluster** C corresponds to the lowest radius at which C or any of its sub-clusters still exist, and below which all its objects are labelled as noise.*

Illustration of the Definitions



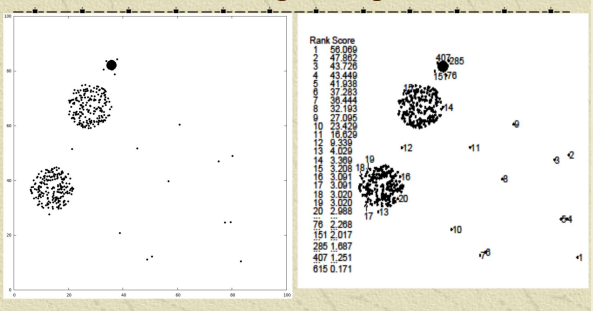
Definition 4.0.5 (GLOSH-score).

$$GLOSH(x) = \frac{\alpha_{max}(C_x) - \alpha(x)}{\alpha_{max}(C_x)}$$

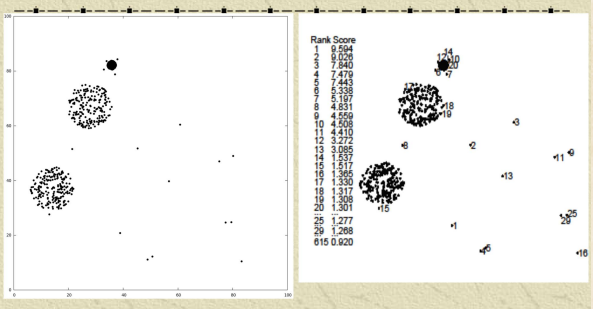
Where:

Notation	What it means
$\alpha(x)$	is the density level at which x is first contained in its closest cluster C_x
$\alpha_{max}(C_x)$	be the highest density level of C_x

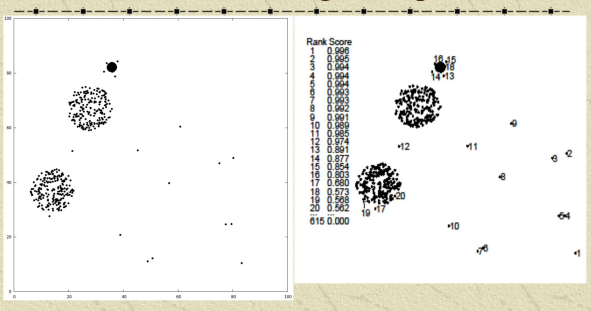
*k*NN Outlier Ranking Example

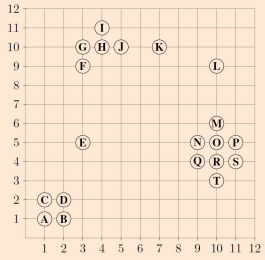


LOF Outlier Ranking Example



GLOSH Outlier Ranking Example





As distance function, use Manhattan distance $L_1(a, b) := |a_1 - b_1| + |a_2 - b_2|$.
Compute the following (without including the query point when determining the kNN):

- LOF using $k = 2$ for the points E, K and O.
- LOF using $k = 4$ for the points E, K and O.
- kNN distance using $k = 2$ for all points.
- kNN distance using $k = 4$ for all points.
- aggregated kNN distances for $k = 2$ and $k = 2$ for all points
  (aggregated kNN distance = sum of the distances to all the kNN!)

30

Chapter 5: Spatial Data and Index Structure

SDBMS vs GIS

Definition 5.0.1 (GIS (Geographic Information System)). *Software to visualize and analyze spatial data (creation of maps; bugger, overlay, flow analysis; connectivity, shortest path, spatial analysis;)*

GIS uses SDBMS (Spatial Database Management System) to store, search, query and share large spatial datasets. SDBMS focus on:

- Efficient storage, querying, sharing of large spatial dataset
- Provides simpler operations that can form the basis for more complex operations
- Uses spatial index structures and query optimization to speed up queries over large spatial datasets

Model and Spatial Data

Properties of Spatial Objects

Spatial objects are described by:

- Spatial Extent
- location and/or boundary with respect to a reference point in a coordinate system, which is at least 2-dimensional
- Basic object types: point, line, polygon, voxel.

There are many ways to model spatial data:

- points or line segments
- polygons
- voxels

Relational Representation of Spatial Data

You can use three tables to represent a polygon:

- One to represent the points
- One to represent the line segments to those points
- One to represents the polygons that is made up with line segments

For spatial queries involving polygons it is necessary to reconstruct the spatial information from the different tables.

Even this simple query requires expensive joins of three tables. Querying the geometry (e.g., P in F2?) is not directly supported.

Extension of the Relational Model to Support Spatial Data

Integration of spatial data types and operations into the core of a DBMS (similar to object-oriented and object-relational databases).

```
SELECT points.PNr, X-coord,Y-Coord
FROM Parcels, Border, Points
WHERE Parcel.FNr = 'F2' AND
  (Parcels.BNr1 = Borders.PNr OR
  Parcels.PNr2 = Points.PNr)
```

Listing 3: DB Query

- Data types like *Point, Line, Region*
- Operations such as *ObjectIntersect, RangeQuery*, etc

Advantages.

- Natural extension of the relational model and query language
- Facilitate design and querying of spatial databases
- Spatial data types and operations can be supported by spatial index structures and efficient algorithms, implemented in the core of a DBMS.

The province decides that a reforestation is necessary in an area described by a polygon S. Find all forest officials affected by this decision.

```
SELECT ForestOfficial
FROM ForestZone
WHERE ObjectIntersects (S,Zone)
```

Listing 4: Better Query

Spatial Queries Processing

DBMS has to support two types of spatial operations:

- Operations to retrieve certain subsets of spatial object from the database, things like Spatial queries and selection, like window query, point query, etc
- Operations that perform basic geometric computations and tests, things like point in polygon test, intersection of two points, etc

Spatial selections like in geographic information systems, are often supported by an interactive graphical user interface.

Basic Spatial Queries

These are basic spatial queries that one could do on a database.

Definition 5.0.2 (Containment Query). *Given a spatial object R, find all objects that completely contain R. If R is a Point then this is called a* ***Point Query***

Definition 5.0.3 (Region Query). *Given a region R (polygon or a circle), find all spatial objects that intersect with R. If R is a rectangle then its called a* ***Window Query****, If R is a circle then it's called a* ***Range Query****.*

Definition 5.0.4 (Enclosure Query). *Given a polygon region R, find all objects that are completely contained in R ("refinement" of a Region Query)*

Definition 5.0.5 (K-Nearest Neighbour Query). *Given a object P, find the k objects that are closets to P (typically for points)*

Spatial Join

Given two sets of spatial objects (typically minimum bounding rectangles) $S_1 = \{R_1, R_2, \ldots, R_m\}$ and $S_2 = \{R'_1, R'_2, \ldots, R'_n\}$ Spatial join: Compute all pairs of object (R, R') such that:

- $R \in S_1, R' \in S_2$
- and R intersects R' ($R \cap R' \neq \emptyset$)
- Spatial predicates other that intersection are also possible, e.g. all pairs of objects that are within a certain distance from each other.

Index Support for Spatial Queries

Conventional index structures such as B-trees are not designed to support spatial queries as they:

- Group objects only along one dimension
- Do not preserve spatial proximity, e.g., nearest neighbor of a point may not be in any single dimension.

Spatial index structures try to preserve spatial proximity by:

- Grouping objects that are close to each other on the same data page
- The issue with this is that the number of bytes to store extended spatial objects (lines, polygons) varies
- The solution of which is to store the **Approximations** of spatial objects in the index structure, typically axis-parallel minimum bounding rectangles (MBR) and have the exact object representation (ER) store separately. Use pointers to the ER in the index i.e., the MBR is the key and the value is a pointer to its ER.

Query Processing Using MBR Approximation

1. Filter Step: Use the index to find all approximations that satisfy the query. Some objects already satisfy the query based on the approximation others have to be checked in the refinement step which will produce the candidate set
2. Refinement Step: Load the exact object representations for candidate left after filter step and test whether they satisfies the query.

Approximations

Conservative Approximation:

- Completely contain the object
- Used to exclude objects that cannot be answers, e.g., for intersection:
  $\neg(a.con_a ppr \cap b.cons_a ppr) \rightarrow \neg(a \cap b)$

Bounding boxes are a conservative approximation as they completely contain the polygon. If the two bounding boxes do not intersect then you know the two objects do not intersect.

Progressive Approximation:

- Completely contained within the object
- Used to determine certain answers. e.g., for intersections $(a.prog_a ppr \cap b.prog_a ppr) \rightarrow (a \cap b)$

Spatial Indexing

Two basic alternatives:

- Embedding into a one-dimensional space and using traditional index structure like Z-order + B+-tree
- Design spacial index structure for spatial data
  - Data organization, like R-tree
  - Space organization like Quad-tree

Embedding of the 2-dimensional space into a 1-dimensional space

Basic idea:

- The data space is partitioned into rectangular cells
- Use a space filling curve to assign cell numbers to the cells (define a linear order on the cell)
  - The curve should preserve spatial proximity
  - Cell numbers should be easy to compute
- For a fixed resolution (e.g., to represent points)
  - Objects are approximated by cells,
  - Store the cell numbers for objects in a conventional index structure

Z-Order preserves spatial proximity relatively well as well easy to compute.

Z-Order - Z-Values

Definition 5.0.6 (Coding of cells).
- *Partition the data space recursively into two halves*
- *Alternate X and Y dimension*
- *Left/Bottom -> 0*
- *Right/Top -> 1*

Definition 5.0.7 (Z-Value: (c, l)). *c is a decimal value of the bit string, and l is the level (number of bits), if all the cells are in the same level then l can be omitted.*

Every two bits represent the left/right and bottom/top position of the square based on the encoding before. The next set of two bits will

represent the position of the cell within the bigger enclosing cell. The bigger the cell the more significant their bits for encoding their position.

Z-Order - Representation of Spatial Objects

For points, use a fixed resolution of the space in both dimensions, i.e., each cell has the same size. Each point is then approximated by one cell. For extended spatial objects, use variable resolution Z-Order defined on pairs: (z-value, level), and minimum enclosing cell, or several cells. Map spatial queries to queries in the one-dimensional space, e.g., for a window query to a range query in the B+-tree. Find all entries (Z-Values) in the range $[l, u]$ where

- l = smallest Z-Value of the window (bottom left corner)
- u = largest Z-Value of the window (top right corner)

Z-Order - Mapping to a B+-Tree

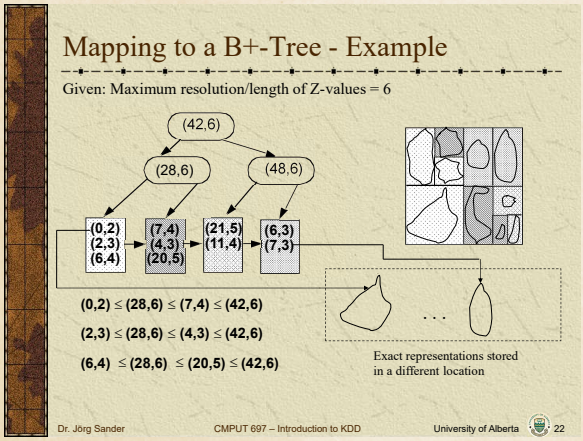
Definition 5.0.8 (**Linear Order for Z-values** to store them in a B+-tree). *Let c_1, l_1 and c_2, l_2 be two Z-Values and let $l = (min)\{l_1, l_2\}$.*
The order relation $\leq_Z$ (that defines a linear order on Z-values) is then defined by

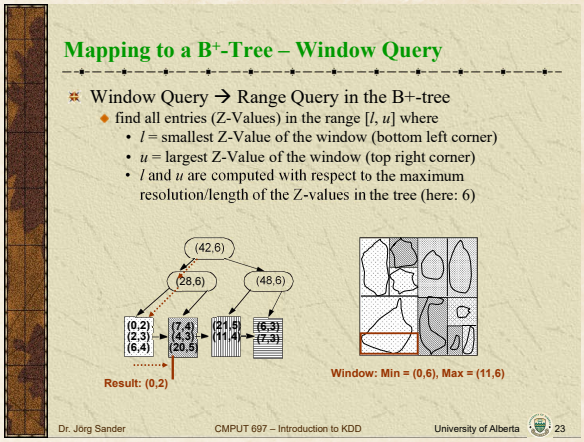
$$(c_1, l_1) \leq_Z (c_2, l_2) \leftrightarrow (c_1 \ div\ 2^{(l_1 - l)}) \leq (c_2 \ div\ 2^{(l_2 - l)})$$

For examples:

- $(1,2) \leq_Z (3,2)$
- $(3,2) \leq_Z (3,2)$
- $(1,2) \leq_Z (10,4)$

The reason why we take the minimum level is to bring the lower Z-value and bring it up to the higher Z-value and compare their location on the same level. We always go up a level because the upper level will always encompass the cells in the lower level.





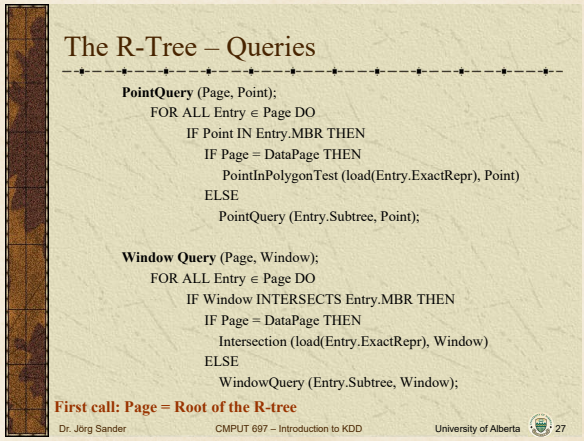
Quad-trees

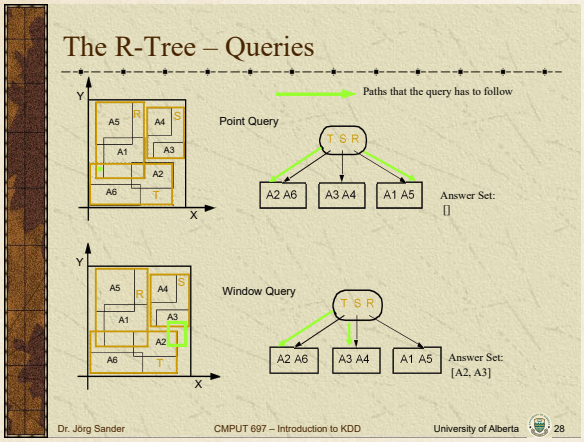
Class of spatial index structures which divides the data space recursively into 4 quadrants (NW, NE, SW, SE). Different algorithms for Quad-Trees are used for processing of points, lines, polygons (i.e., different node types, construction and query algorithms). Frequently used in commercial GIS especially for compressing, storing and manipulating of raster images.

R-trees

Balanced Tree designed to organize rectangles each page contain between m and M entries. Data page entries are of the form (MBR, PointerToExactRepr), the MBR is a minimum bounding rectangle of a spatial object, which PointerTpExactRepr is pointing to. Directory page entries are of the form (MBR, PointerToSubtree), MBR is the minimum bounding rectangles of all entries in the subtree which PointerToSubtree is pointing to. Rectangles can overlap. The height h of an R-Tree for N spatial objects is:

$$h \leq \lceil \log_M N \rceil + 1$$





Variants

Many variants of R-trees exist, for example there are R*-tree, X-tree for higher dimensional point data R-Trees are also efficient index structure for point data since points can be modelled as "degenerated" rectangles. Multidimensional points, where a distance function between the points is defined play an important role for similarity search in so-called "feature" or "multimedia" databases.

Feature Databases

Examples include:

- Measurements of celestial objects like stars and galaxies (e.g., light in different wavelengths) + Euclidean distance
- Color histograms of images + a quadratic form distance function
- Document (word frequency histogram), other image features, shape descriptors, ...

These can all fit into an n d-dimensional feature vector, where each domain of features is their own d-dimensional feature vector.

Similarity Queries

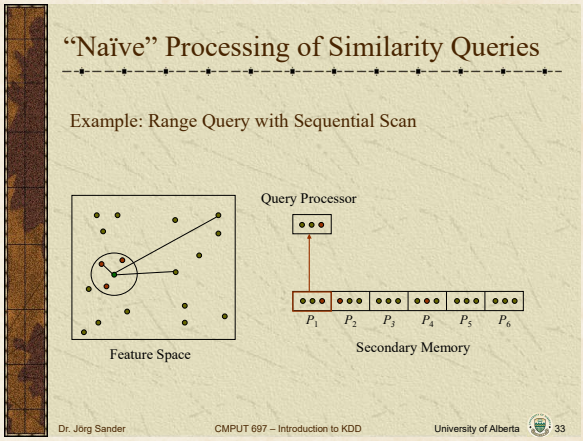
With an object and metric distance function you can measure (dis)similarity between objects. Basic types of similarity queries include:

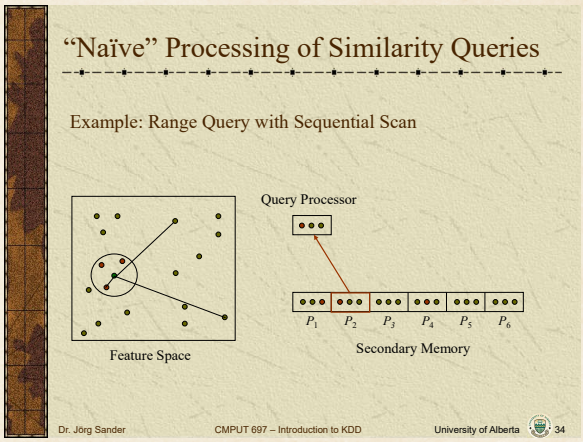
- Range queries with range ε where you retrieves all objects which are similar to the query object up to a certain degree of ε
- k-nearest neighbor queries where you retrieve k most similar objects to the query

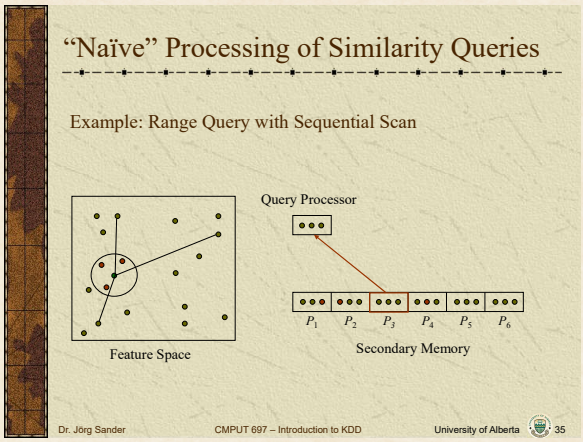
There are many applications for similarity queries:

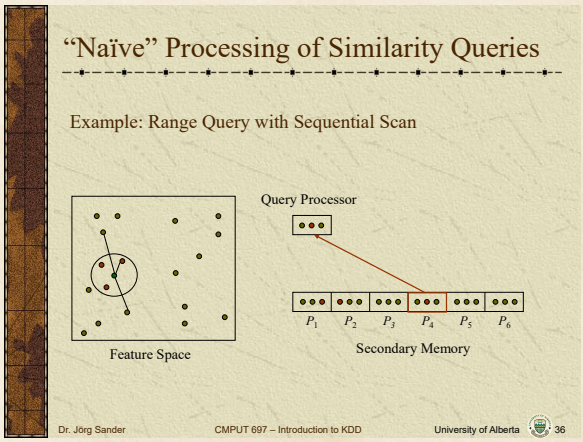
- Manual data exploration
- Density-based clustering
- k-nn classification
- etc

Naive Processing of Similarity Queries



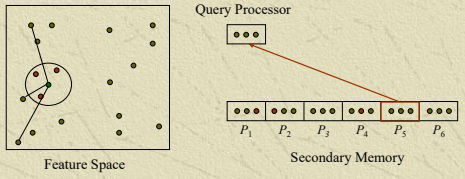






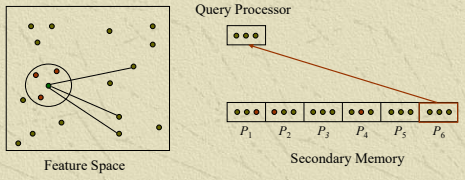
"Naïve" Processing of Similarity Queries

Example: Range Query with Sequential Scan



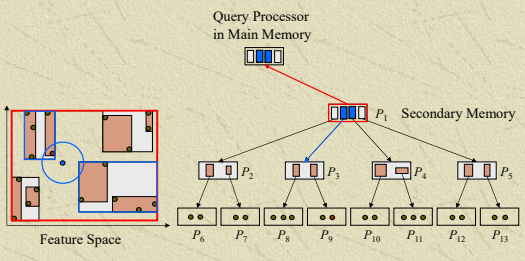
"Naïve" Processing of Similarity Queries

Example: Range Query with Sequential Scan

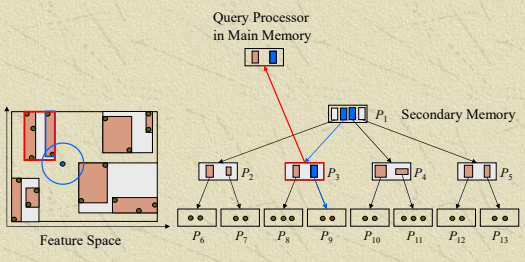


Range Query Using R-tree

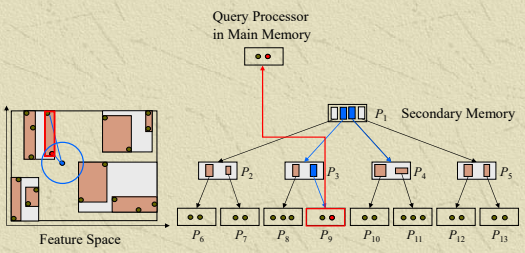
Example Range Query using R-tree



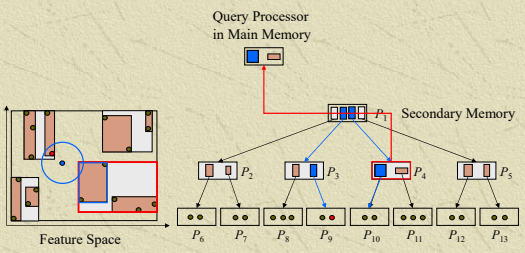
Example Range Query using R-tree



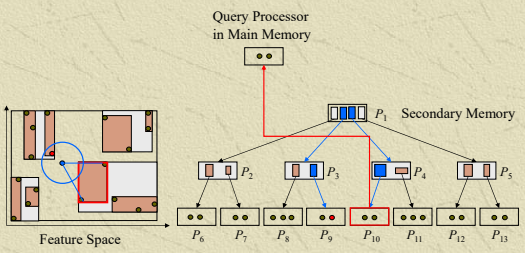
Example Range Query using R-tree



Example Range Query using R-tree



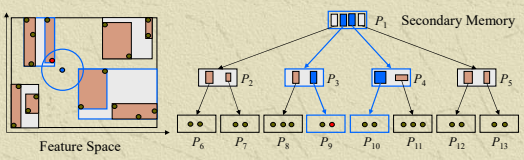
Example Range Query using R-tree



Example Range Query using R-tree

In general:

A query has to follow a limited number of paths in the tree, from the root to the leaves

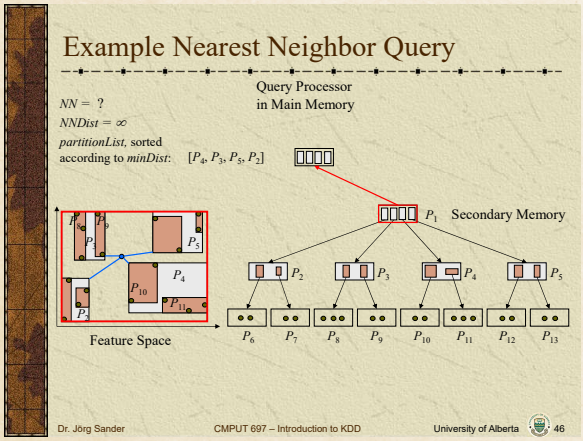


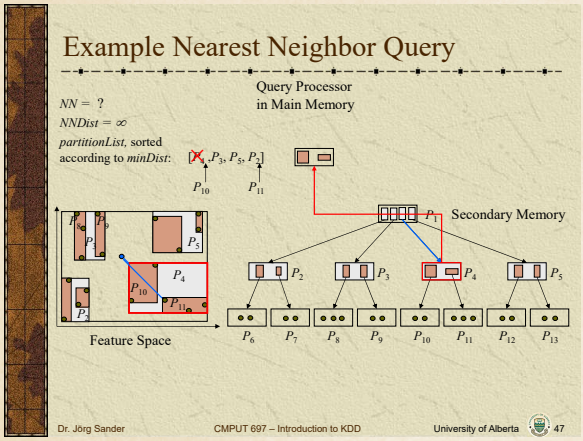
Nearest Neighbour Query using R-tree

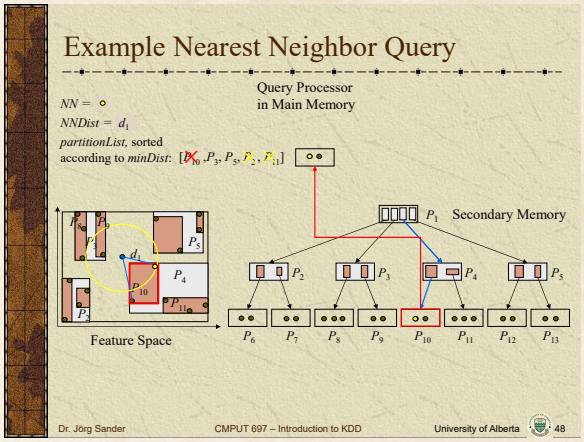
To retrieve the object closest to the query point q:

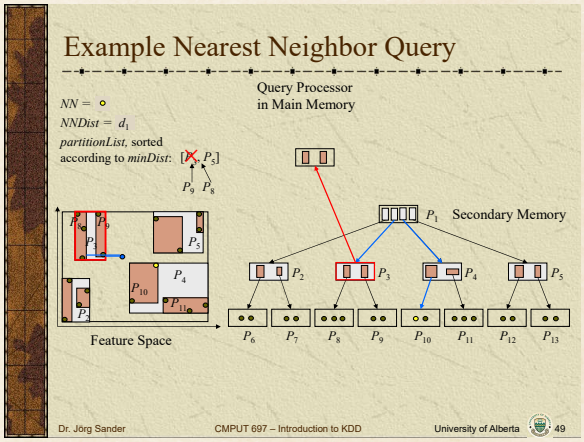
1. Take the *minDist* between a query point and an MBR as an estimate for the distances between the query and any point stored in the corresponding subtree of the MBR
2. Traverse the tree, following always the pointer of the entry which is closest to the query point
3. Once a point p from a data page with a distance $d = dist(p, q)$ is found, all paths from an MBR with $minDist(q, MBR) > d$ can be pruned.
4. Only MBR's with $minstDist(q, MBR) \leq d$ can contain points which might be closer to the query q than the current point p.

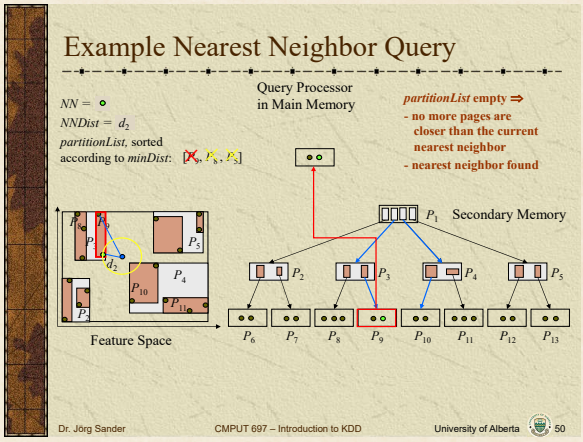
Example



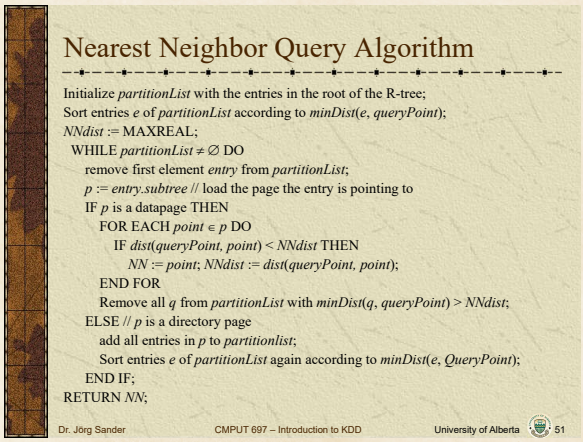








Algorithm



R-Tree Construction - Optimization Goals

Overlap between the MBRs:

- Spatial queries have to follow several paths
- try to minimize overlap

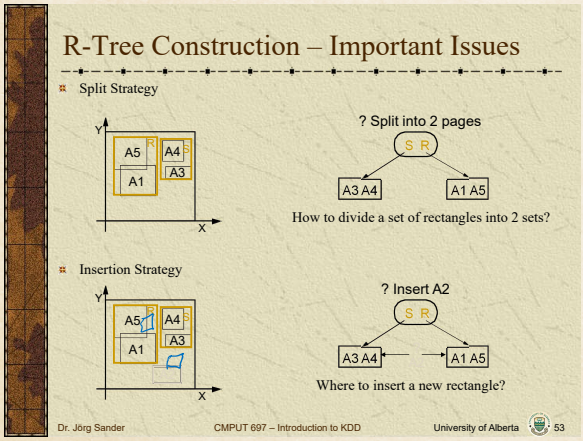
Empty space in MBR

- Spatial queries may have to follow irrelevant paths
- Try to to minimize area and empty space in MBRs

R-Tree Construction - Important Issues

There are two strategies for building a R-tree:

- Split Strategy
- Insertion Strategy



Insertion Strategy

Dynamic construction by insertion of rectangles R

- Searching for the data page into which R will be inserted, traverses the tree from the root to a data page
- When considering entries of a directory page P, 3 cases can occur:
  1. R falls into exactly one $Emtry.MBR$ in which case follow $Entry.Subtree$
  2. R falls into the MBR of more than one entry $e_1, \ldots, e_n$ in which case follow $E_i.Subtree$ for entry e_i with the smallest area of $e_i.MBR$
  3. R does not fall into an $Entry.MBR$ of the current page in which case check the increase in area of the MBR for each entry when enlarging the MBR to enclose R. Choose $Entry$ with the minimum increase in area (if this entry is not unique, choose the one with the smallest area); enlarge $Entry.MBR$ and follow $Entry.Subtree$

Construction by "bulk-loading" the rectangles

- Sort the rectangles e.g., Z-Order
- Create the R-Tree "bottom-up"

Split Strategy

Insertion will eventually lead to an overflow of a data page in that case

- The parent entry for that page is deleted
- The page is split into 2 new pages - according to a split strategy
- 2 new entires pointing to the newly created pages are inserted into the parent page.
- A now possible overflow in the parent page is handled recursively in a similar way; if the root has to be split, a new root is created to contain the entires pointing to the newly created pages.

The overflow of a node K with $\mid K \mid = M + 1$ entries leads to the distribution few the entries into two new nodes K_1 and K_2 such that $\mid K_1 \mid \geq m$ and $\mid K_2 \mid \geq m$. The Exhaustive algorithm works by searching for the "best" split in the set of all possible splits in two expensive ($O(2^M)$ possibilities!). Important issues when splitting a page:

- Overlap of the MBR's of the resulting pages should be avoided/minimized, to minimize the number of paths that a query which falls into this region of the data space, has to investigate.
- Area of the MBR's of the resulting pages should be as small as possible, to minimize the possibilities of queries intersecting the MBR in a region where no data points are located.

Different Split Heuristics:

- Quadratic algorithm
- Linear algorithm
- R*-tree
- ...

Quadratic algorithm.

- Choose the "initial pair" of rectangles R_1 and R_2 that have the largest value $d(R_1, R_2)$ for the empty space in an MBR, which covers both R_1 and R_2.

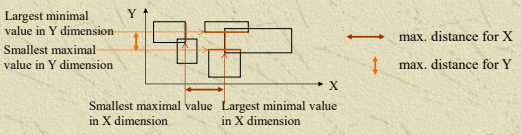
  $$d(R_1, R_2) = \text{Area}(MBR(R_1 \cup R_2)) - (\text{Area}(R_1) + \text{Area}(R_2))$$

- Set $K_1 = \{R_1\}$ and $K_2 = \{R_2\}$
- Repeat until STOP
  - if all R_i are assigned: STOP
  - if all remaining R_i are needed to fill the smallest node to guarantee minimal occupancy m: assign them to the smallest node and STOP
  - else: choose the next R_i and assign it to the node that will have the smallest increase in area of the MBR by the assignment. If not unique: choose the K_i that covers the smallest area (if still not unique the one with fewer entries).

Linear algorithm.

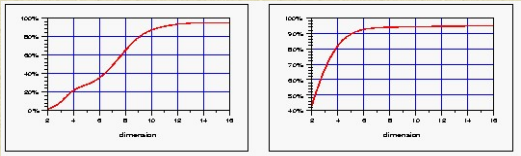
R-Tree Construction – Splitting Strategies

- Linear algorithm:
  - Same as the quadratic algorithm, except for the choice of the initial pair: Choose the pair with the largest normalized distance.
    - For each dimension determine the rectangle with the largest minimal value and the rectangle with the smallest maximal value (the difference is the maximal distance/separation).
    - Normalize the maximal distance of each dimension by dividing by the sum of the extensions of the rectangles in this dimension
    - Choose the pair of rectangles that has the greatest normalized distance. Set K1 := {R1} and K2 := {R2}.



R-trees and High-dimensional data

Overlap of the R-tree Directory Nodes*



a. Overlap (Uniformly Distributed Data) b. Weighted Overlap (Real Data)

Figure from: Berchtold, Keim & Kriegel: The X-tree : An Index Structure for High-Dimensional Data, VLDB 96

⇒ the R*-tree is only efficient for < 10 dimensions

Index Structures for High-Dimensional Data

X-tree [Berchtold, Keim & Kriegel VLDB 96]

Idea

- perform a split only if the resulting overlap is not too large
- otherwise, extend the size of the current node, i.e. create a *supernode*
- combines hierarchical index structure and sequential scan

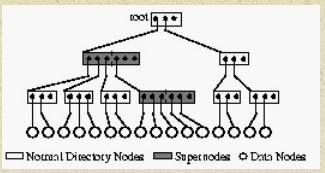


Figure from: Berchtold, Keim & Kriegel: The X-tree : An Index Structure for High-Dimensional Data, VLDB 96

Chapter 6: Association Rules

Definition 6.0.1 (Association rule mining). *Finding frequent patterns or associations in transaction databases, relational databases, and other information repositories*
Applications include:

- *Market basket data analysis*
- *Cross-marketing*
- *Catalog design*
- *Clustering*
- *Classification*

Rule form: "Body -> Head[support, confidence]"
Example: buys(x,"diapers") -> buys(x,"beer")[0.5%, 60%]

Simple Association Rules

Given: (1) database of transactions, (2) each transaction is a list of items (e.g., purchased by a customer in a visit). Find: *all* rules that correlate the presence of one set of items with that of another set of items, e.g., 98% of people who purchase tires and auto accessories also get automotive services done.
Applications:

- For the previous example **Maintenance Agreement** (What the store should do to boost Maintenance Agreement sales)
- **Home Electronics -> * (What other products should the store stocks up?)**

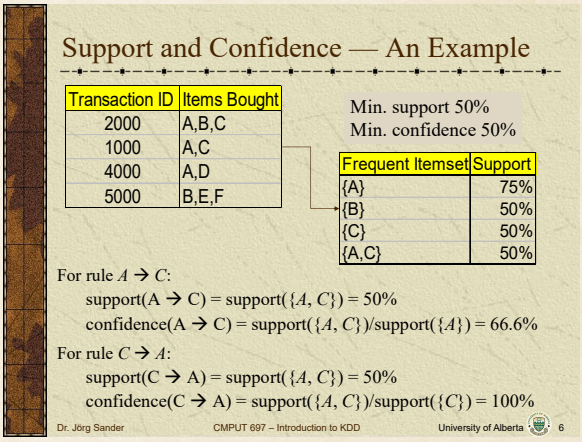
Basic Concepts

- Items $I = \{i_1, \ldots, i_m\}$ a set of literals (denoting items)
- Itemset X: Set of items $X \subseteq I$
- Database D: Set of transactions T, each transaction is a set of items $T \subseteq I$
- T contains X: $X \subseteq T$
- The items in transactions and itemsets are sorted lexicographically: itemset $X = (x_1, x_2, \ldots x_k)$, where $x_1 \leq x_2 \leq \ldots \leq x_k$
- Length of an itemset: number of elements in the itemset
- k-ithemset: itemset of length k
- Support of an itemset X: proportion of the transactions in D containing X
- Association rule: Implication $X \rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$

Riles Measures: Support and Confidence

For a rule $x = X \rightarrow Y$ support and confidence are defined as following:

- **support(s)**, probability that a transaction contains $X \cup Y$
- **confidence(c)**, conditional probability that a transaction containing X also contains Y



Mining Frequent Itemsets

The Key Step: Find the *frequent itemsets*: the sets of items that have minimum support.

- A subset of a frequent itemset must also be a frequent itemset, i.e., if $\{AB\}$ is a frequent itemset, both $\{A\}$ and $\{B\}$ should be a frequent itemset
- Iteratively find frequent itemsets with cardinality from 1 to k (k-itemset)

Use the frequent itemsets to generate association rules.

The *Naive Algorithm* count the frequency of all possible subsets of I in the database D, this would be too expensive since there are 2^m such itemsets.

Definition 6.0.2 (Apriori principle). *Any subset of a "frequent" itemset must be "frequent"*

Method based on the Apriori principle:

- First count the 1-itemsets, then the 2-itemsets, then the 3-itemsets, and so on
- When counting the $k+1$ itemsets, only consider those $k+1$-itemsets where all subsets of length k have been determined as frequent in the previous step

Apriori Algorithm

The Apriori Algorithm

- Join Step: C_k is generated by joining L_{k-1} with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset
- Pseudo-code:

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
    increment the count of all candidates in C_{k+1} that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
return $\cup_k L_k$;

Candidate Generations

Candidate Generation: 1. Join Step

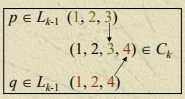
- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

  insert into C_k

  select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$

  from L_{k-1} p, L_{k-1} q

  where $p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$



Candidate Generation: 2. Prune Step

- Step 2: pruning

  forall *itemsets c in* C_k do

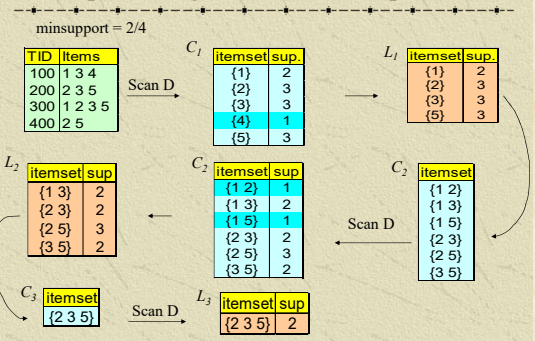
    forall *(k-1)-subsets s of c* do

      if (*s is not in* L_{k-1}) **then delete** *c* **from** C_k

- Example
  - L_3 = {(1 2 3), (1 2 4), (1 3 4), (1 3 5), (2 3 4)}
  - Candidates after the join step: {(1 2 3 4), (1 3 4 5)}
  - In the pruning step: delete (1 3 4 5) because (3 4 5) ∉ L_3
  - C_4 = {(1 2 3 4)}

Example

The Apriori Algorithm — Example



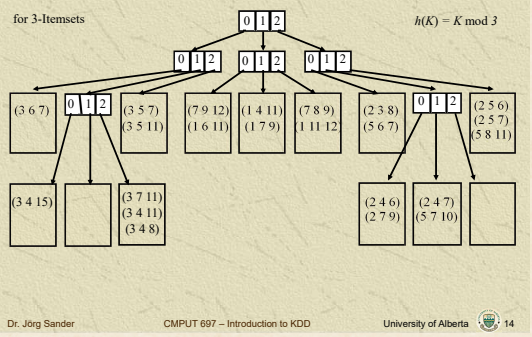
How to Count Supports of Candidates?

Why is counting supports of candidates a problem you might ask? Well

- Total number of candidates can be very large
- One transaction may contain many candidates

A method we can use to resolve this issue is to store the candidate itemsets in a *hash-tree*.

Hash-Tree – Example



Hash-Tree Structure

Hash-Tree structure elements

- **Leaf node** of the hash-tree contains a list of itemsets and counts; each leaf node has a maximum capacity
- **Interior node** contains a hash table; each bucket of the hash table has a pointer to a node on the next level of the tree
- **Subset function**: finds all candidates contained in a transaction

Operations

Searching for an itemset:

- Start at the root
- At level d: Apply the hash function h to the d-th item in the Itemset and follow the pointer in the resulting bucket; until a leaf node is reached
- Scan the leaf node

Insertion of an Itemset:

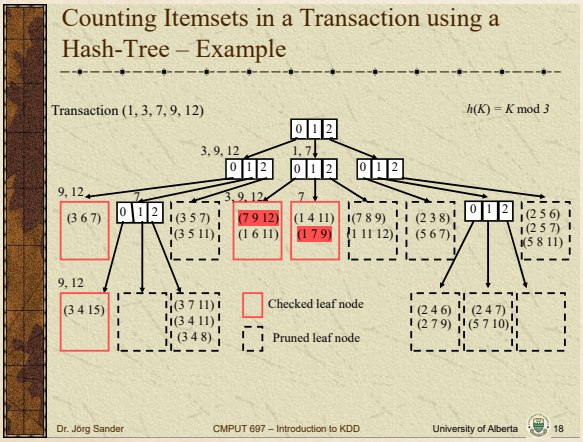
- Search for the corresponding leaf node, and insert the itemset into that leaf
- If an overflow occurs: transform the leaf node into an internal node and distribute the entires to the new leaf nodes according to the hash function

Counting

Search for all candidate itemset contained in a transaction $T = (t_1, t_2, \ldots, t_n)$. At the root:

- Determine the hash value for each item in T
- Continue the search in the resulting child nodes

At the internal node at level d (reached after hashing of item t_i). Determine the hash values and continue the search for each item t_k with $k > i$. At the leaf node check whether the itemsets in the leaf node are contained in transaction T.



Methods to Improve Apriori's Efficiency

- **Hash-based itemset counting**: A k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
- **Transaction reduction**: A transaction that does not contain any frequent k-itemset is useless in subsequent scans
- **Partitioning**: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
- **Sampling**: mining on a subset of given data, lower support threshold + a method to determine the completeness

Generation of Rules from Frequent Itemsets

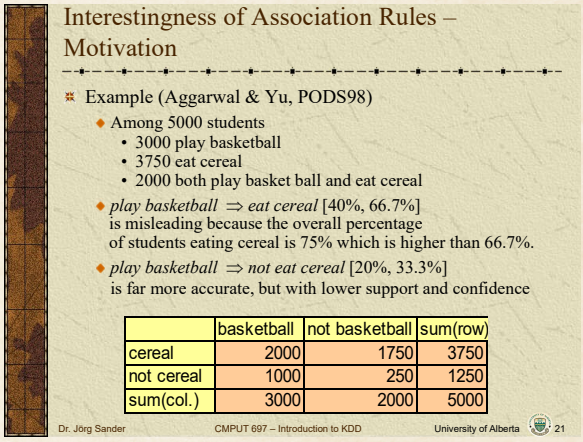
For each frequent itemset X:

- For each subset A of X form a rule $A \rightarrow (X - A)$
- Delete those rules that do not have a minimum confidence

Computation of the confidence of a rule $A \rightarrow (X - A)$

$$confidence(A \rightarrow (X - A)) = \frac{support(X)}{support(A)}$$

Interestingness of Association Rules



Delete "misleading" association rules. The condition for a rule $A \rightarrow B$

$$\frac{P(A \cup B)}{P(A)} > P(B) + d \quad \text{for a suitable threshold d} > 0$$

Measure for the interestingness of a rule

$$\frac{P(A \cup B)}{P(A)} - P(B)$$

The larger the value, the more interesting the relation between A and B, expressed by the rule.

Extensions and Other Methods

- Hierarchical association rules
- Quantitative association rules
- Constraint association rules
- FP-Tree: generating frequent patterns without candidate generation
- ...

Credits

General

- Created by: Jihoon Og, u/DnD_Notes
- Compiled on Wednesday 1st March, 2023 at 20:42
- Typesetting engine: LaTeX
- Dungeon and Dragon (5e) LaTeX Template
- CMPUT-697 lecture slides by Dr. Jörg Sander

Art

- Couatl for the cover art is from D&D Beyond
- Andy the D&D Ampersand is from Dungeon and Dragons
- Book logo is from Adobe Stock
- Cover art formatting and design done in Photoshop CC 2019

Disclaimer