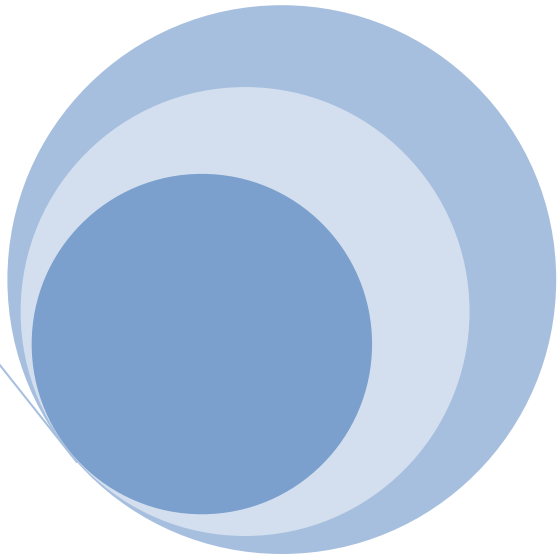
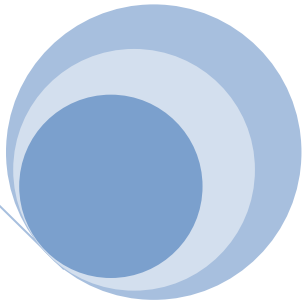
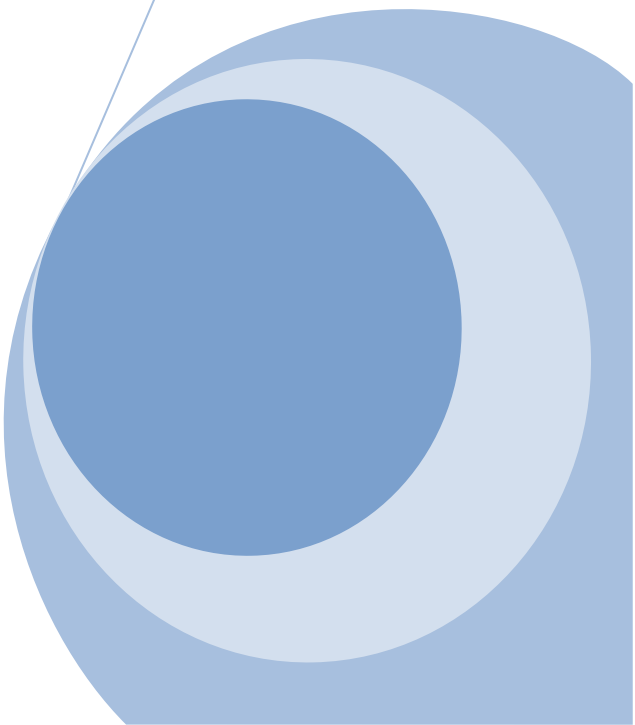

Database Management Systems

Oral exam topics

Theoretical topics for preparation at oral exams of DBMS

Marco
12/03/2013

Reliability Manager

- It's responsible of the atomicity and durability ACID properties.
- It implements the following transactional commands:
    - Begin transaction,
    - Commit work,
    - Abort/Rollback work.
- It provides the following recovery primitives:
    - Warm restart (main memory failures)
    - Cold restart.
- It exploits the LOG file, that is a persistent archive file, recording DBMS activities; it's stored on "stable memory" (idealization of "perfect" memory, with no errors, it's approximated by means of redundancy).
- Log file is written in a chronological order and log records can be transaction records or system records.
- Transaction records are:
    - Begin (B(T)),
    - Commit (C(T)),
    - Abort/Rollback (A(T)),
    - Insert (I(T,O,AS)),
    - Update (U(T,O,BS,AS)),
    - Delete (D(T,O,BS)).
        - Where "T" is the transaction ID, "O" is the involved object, "AS" is the After State of the involved object and "BS" is the Before State of the involved object.
- System records are:
    - Dumb (Backup copy of the database),
    - Checkpoint (CK(L), where "L" is the list of the transactions that are active during the checkpoint time).
- Undoing action: undo a list of transactions.
- Redoing action: redo a list of transactions.
- Idempotency property says that "Undo and Redo can be repeated an arbitrary number of times without changing the final outcome: UNDO(UNDO(Action) = UNDO(Action)."
- The Checkpoint is an operation periodically requested by the Reliability Manager to the Buffer Manager. It's not strictly necessary, but it allows a faster recovery process. During the checkpoint the DBMS writes data on disk for all completed transactions in a synchronous way (with FORCE primitive) and it records all the active transactions. At the end the checkpoint is saved into the log file and the effect of all committed transactions are permanently.
- The Dump creates a complete copy of the database and stores it into an external stable memory (if possible, different from the log memory). At the end the Dump is recorded into the log file with a timestamp and the location of the backup.
- Rules for writing the log:
    - Write Ahead Log (WAL): the Before State of data in a log record is written in stable memory before database data is written on disk; during recovery, it allows the execution of undo operations on data already written on disk.

- Commit procedure: The After State of data in a log record is written in stable memory before commit. During recovery, it allows the execution of redo operations for transactions that already committed, but where not written on disk.
- The standard situation is that the Before States and the After States are written together: the log is written synchronously (FORCE) for data modification written on disk and on commit; the log is written asynchronously (No FORCE) for Abort/Rollback.

- The commit record on the log is a border line:
  - If it's not written in the logs, the transaction should be undone upon failures;
  - If it's written in the log, the transaction should be redone upon failure.
- Writing the log is a very important operation and the usage of robust protocols to guarantee reliability is costly, for this reason log writing is optimized by using compact format, parallelism and commiting of groups of transactions (during the same time period).
- Recovery Management: when a failure occurs in a DBMS, the system is stopped and the recovery method depends on the failure type:
  - System failure: it's caused by software problems or power supply interruptions, it causes only loosing of the main memory content. System failure need the warm restart method.
  - Media failure: it's caused by a failure of devices managing secondary memory, it causes loosing of the database content on disk (or just a portion of that).
  - In both cases, the loosing of the log file is not contemplated because we suppose that it's located into a stable memory. Errors in log file (failures in stable memory) are considered as catastrophic.
- Warm restart method:
  1) Read backward the log until the last checkpoint record,
  2) Detect transactions which should undone/redone
     a. At the last checkpoint
        i. UNDO = {active transactions at checkpoint},
        ii. REDO = {empty}.
     b. Read forward the log
        i. UNDO = add all transactions for which the begin record is found,
        ii. REDO = move transactions from undo list to redo list when the commit record is found,
        iii. Transactions ending with Abort/Rollback remain into undo list.
     c. At the end of this step:
        i. UNDO = {list of transactions to be undone}
        ii. REDO = {list of transactions to be redone}
  3) Data recovery
     a. The log is read backward from the time of failure until the beginning of the oldest transaction in the undo list:
        i. Actions performed by all transactions in the undo list are undone,
        ii. For each transaction the begin record should be reached (even if it's earlier than the checkpoint).
     b. The log is read forward from the beginning of the oldest transaction in the redo list:
        i. Actions of transactions in the redo list are applied to the database,
        ii. For each transaction, the starting point is the begin record for that transaction.

- Cold restart method: it manages failures damaging the database disk (or a portion of that)
    1) Access the last dump to restore the damaged portion of the damage on disk,
    2) Starting from the last dump record, read the log forward and redo all actions on the database and transaction commit/rollback,
    3) Perform a Warm restart.

Concurrency Control Management

- The elementary operations are: read of a single object x (r(x)), write of a single data object x (w(x)); they may require reading from disk or writing to disk an entire page.
- The Scheduler is a block of the concurrency control manager and it's in charge of deciding if and when read/write requests can be satisfied.
- The transaction is a sequence of read and write operations characterized by the same transaction id (TID).
- The schedule is a sequence of read/write operations presented by concurrent transactions; operations in the schedule appear in the arrival order of requests.
- The concurrency control accepts or rejects schedules to avoid anomalies (lost update, dirty read, inconsistent read, ghost update). The scheduler has to accept or reject operation execution without knowing the outcome of the transactions (abort or commit).
- The Serial Schedule is a schedule in which the actions of each transaction appear in sequence, without interleaved actions belonging to different transactions. An arbitrary schedule S_i is correct when it yields the same result as an arbitrary serial schedule S_j of the same transaction and in this case it's possible to say that S_i is Serializable (it's equivalent to an arbitrary serial schedule of the same transactions).
- It's possible to define different equivalence classes between two schedules and they are: View equivalence (VSR), Conflict equivalence (CSR), 2 Phase Locking (2PL) and Timestamp equivalence. Each equivalence class detects a set of acceptable schedules and they are characterized by a different complecity in detecting equivalence.
- View Equivalence (VSR)
    - Definitions:
        - Read-from: $r_i(x)$ reads-from $w_j(x)$ when w_j precedes r_i and $i \neq j$ and when there's no other w_k between them
        - Final write: $w_i(x)$ is a final write if it's the last write of x appearing in the schedule.
    - Two schedules are View Equivalent if they have the same read-from set and the same final write set.
    - A schedule is view serializable if it's view equivalent to an arbitrary serial schedule of the same transactions.
    - Detecting view equivalence to a given schedule has linear complexity. Detecting view equivalence to an arbitrary serial schedule is NP-complete problem (checking for ALL possible permutations of an arbitrary schedule).
- Conflict Equivalence (CSR)
    - Action A_i is in conflict with action A_j ($i \neq j$) if both actions operate on the same object and at least one of them is a write (Read-Write conflict (RW) or Write-Read conflict (WR), Write-Write conflict (WW))

- Two schedules are Conflict Equivalent if they have the same conflict set and each conflict pair is in the same order in both schedules.
- A schedule id conflict serializable if it's equivalent to an arbitrary serial schedule of the same transactions.
- Conflict Graph: it's used to detect conflict serializability. It's represented in this way: a node for each transaction, an edge $T_i \rightarrow T_j$ if there exists at least a conflict between an action A_i in T_i and A_j in T_j and A_i precedes A_j. If this Conflict Graph is acyclic then the schedule is Conflict Serializable (CSR). Checking Conflict Graph is a linear operation.

- 2 Phase Locking
  - A Lock is a block on a resource which may prevent access to others. Lock operations can be: Lock (read lock or write lock) or Unlock. Each read (write) operation is preceded by a request of read (write) lock and it's followed by a request of unlock.
  - The difference between R-Lock and W-Lock is that in the first case the lock is shared among different transactions, while the W-Lock is exclusive and only one transaction per time can obtain the W-Lock on a specific resource. There's another situation that is the Lock escalation: it's a particular situation in which a transaction requests an R-Lock followed by W-Lock on the same specific resource.
  - With this method, the Scheduler becomes a Lock Manager. It receives transaction requests and grants locks based on locks already granted to other transactions:
    - when the lock request is granted then the corresponding resource is acquired by the requesting transaction and when the transaction performs the unlock, the resource becomes again available;
    - when the lock is not granted the requesting transaction is put in a waiting state and this state terminates when the resource is unlocked and become available.
  - The lock manager exploits the information in the lock table to decide if a given lock can be granted to a transaction and the conflict table to manage lock conflicts (conflict table summarizes the rules to grant a lock).
  - Conflict Table

Request	Resource State		
	Free	R-Locked	W-Locked
R-Lock	OK/R-Locked	OK/R-Locked	Nope/W-Locked
W-Lock	OK/W-Locked	Nope/R-Locked or OK/W-Locked (for Lock Escalation	Nope/W-Locked
Unlock	ERROR	OK/R-Locked or Free	OK/Free

  - Read locks are shared and other transactions may lock the same resource; for this reason a counter is used to count the number of transactions currently holding the R-Lock (the resource is free when the counter is zero).
  - The two phases are:

        - Growing phase: this is the initial phase of each transaction, in which all needed locks are acquired;
        - Shrinking phase: this is the last phase of each transaction, in which all locks are released.
        - It's important to notice that if a transactions starts with the shrinking phase, it cannot acquire other resources; in other words: a transaction cannot acquire a now lock after having released any lock.
    - A variant of 2 Phase Locking is the Strict 2 Phase Locking in which a transaction locks may be released only at the end of the transaction (after commit/rollback).
    - The process is articulated in this way: a transaction requests a resource x, if the request can be satisfied, the lock manager modify the state of resource x in its internal tables and it returns control to the requesting transaction; if the request cannot be satisfied immediately, the requesting transaction in inserted in a waiting queue and suspended (wait) and only when the resource become available, the first transaction in the waiting queue is resumed and is granted the lock on the resource. When a timeout of a generic transaction in the waiting queue expires, the lock manager extracts the waiting transaction from the queue, resumes it and returns a not ok error code, the requesting transaction can perform a rollback and request again the same lock after some time.
    - The timeout for each transaction is useful also to avoid Deadlocks.

Distributed Databases

- In a distributed database architecture, data and computation are distributed over different machines. The typical advantages on this are an improvement of performance, increasing of availability and a stronger reliability.
- There are two major different distributed architectures:
    - Client/Server architecture: it's the simplest and more widespread in which the client manages the user interface and it sends all the requests to the server and the server manages this requests and the database;
    - Distributed Database System architecture: different DBMS servers are placed on different network nodes, each of them is autonomous and it must be able to cooperate with other servers. In this type of architecture, guaranteeing the ACID properties requires more complex techniques.
- Distributed Architectures exploits data replications: a replica is a copy of the data stored on a different network node (DBMS server).
- The most important properties of distributed databases are: portability (capability of moving a program from a system to a different system) and interoperability (capability of different DBMS servers to cooperate on a given task).
- The SLQ execution can be done in two ways:
    - Compile & Go: the query is sent to the server and then it's prepared (generation of the execution plan), executed and shipped;
    - Compile & Store: the query is still sent to the server where it's prepared and the execution plan is saved for future usage. This method is particularly efficient for repeated query executions.
- Particularity of distributed database systems are:

    - Local autonomy: each DBMS manages its local data in an autonomous way;
    - Functional advantages such as appropriate localization of data and applications;
    - Technological advantages such as increasing of data availability (total block probability is reduced but local blocks may be more frequent) and enhancing of scalability (providing by the modularity and flexibility of the architecture).
- Data fragmentation: given a relation R, a data fragment is a subset of R in terms of tuples, or schema, or both. There are different criteria to perform fragmentation:
    - Horizontal fragmentation: the horizontal fragmentation of a relation R selects a subset of tuples in R with the same initial schema of R, this tuples are obtained by means a selection using a partitioning predicate. Fragments are not overlapped and there're no intersections between different fragments.
    - Vertical fragmentation: the vertical fragmentation of a relation R selects a subset of schema of R, obtained by means of projection of a subset of the initial schema of R. The primary key should be included in the projection to allow rebuilding the initial relation R; with this fragmentation method, all fragments contain the whole tuples. Fragments are overlapping on the primary key.
    - It's possible to use both previous solutions.
- The fragmentation must have this properties:
    - Completeness: each information in relation R is contained in at least one fragment of R,
    - Correctness: the information in R can be rebuilt from all its fragments.
- Distributed databases are based on fragmentation and data is distributed over different servers: each fragment of a starting relation is usually stored in a different files and, possibly, on different locations. Actually the initial relation doesn't exist, but we can recreate it rebuilding fragments.
- The allocation schema of a distributed database describes how fragments are stored on different server nodes and we can find several situations:
    - Non redundant mapping if each fragment is stored on one single node;
    - Redundant mapping if some fragments are replicated on different servers, this increases data availability but also complex maintenance (because copy synchronization is needed).
- Transparency levels describe the knowledge of how data is distributed among different servers. Transparency levels are:
    - Fragmentation transparency: applications know the existence of tables but not of their fragments because data distribution is not visible. This is the major level in which the programmer only accesses tables of interest, without looking for specific fragments of it.
    - Allocation transparency: applications know the existence of fragments, but not their allocation; for this reason the programmer must select in what fragment he want to make a query, but in the case that the fragment is stored in different allocations, the programmer doesn't care about it. This is the medium level.
    - Language transparency: the programmer should select both the fragment and its allocation; this is the format in which higher level queries are transformed by a distributed DBMS because no SQL dialects are used. This is the lowest level and the complexity of queries increase sensibly (programmer must specify all location details in FROM clause).

- In a distributed DBMS, all nodes participating to a distributed transaction must implement the same decision (commit or rollback); for doing this we need the coordination of some specific protocol, called 2 phase commit.
- The objective of the 2 Phase Commit Protocol is a coordination of the conclusion of a distributed transaction (only the conclusion). For doing this we need two specific roles: the first is a coordinator, called Transaction Manager (TM), the second is represented by all other DBMS servers which take part to the transaction and they are called Resource Managers (RM). It's important to say that any participant may take the role of the transaction manager and also the client requesting the transaction execution can do it.
- 2 Phase Commit Protocol introduces new rules for the Log files. First of all it's important to say that both TM and RM have separate and private logs. For TM 2PC introduces new type of log records:
    - Prepare: it contains the identity of all RMs participating to the transaction;
    - Global commit/rollback: final decision on the transaction outcome, all sub-transactions executed in different nodes are committed or aborted;
    - Complete: written at the end of the protocol.
- New kind of record are introduced also for RMs:
    - Ready: the RM is willing to perform a commit on the transaction ("Ok, I can commit…") and the decision (commit/rollback) cannot be changed afterwards. It's important to say that after this point the RM loses its autonomy for the current transaction.
- Steps for the 2 Phase Commit Protocol:
    - Phase 1:
        1. The TM:
            - Writes the prepare record in the log;
            - Sends the prepare message to all RMs;
            - Sets a timeout, defining maximum waiting time for RM answer (this is absolutely useful for avoiding useless waste of time waiting some broken RM that will never answer).
        2. The RMs:
            - Wait for the prepare message;
            - When they receive it:
                - If they are in a reliable state (affidabile):
                    - Write the ready record in the log,
                    - Send the ready message to the TM;
                - If they are not in a reliable state:
                    - Send a not ready message to the TM,
                    - Terminate the protocol,
                    - Perform local rollback;
                - If the RM crashed:
                    - No answer is sent.
        3. The TM:
            - Collects all incoming messages from the RMs;
            - If it receives ready from ALL RMs:
                - The log commit global decision record is written in the log;

        - If it receives at least one not ready or the timeout expires:
            - The abort global decision record is written in the log.
    - Phase 2:
        1. The TM:
            - Sends the global decision to the RMs;
            - Sets a timeout for the RM answers.
        2. The RMs:
            - Wait for the global decision;
            - When they receive it:
                - The commit/abort record is written in the log,
                - The database is updated,
                - An ACK message is sent to the TM
        3. The TM:
            - Collects the ACK messages from the RMs;
            - If ALL ACK messages are received:
                - The complete record is written in the log;
            - If the timeout expires and some ACK messages are missing:
                - A new timeout is set and the global decision is resent to the RMs which did not answer, until all answers are received.
- Each RM is affected by an uncertainty window that starts after ready message is sent and ends upon receipt of global decision. During this period, that it should be very small, local resources in the RM are locked and RM can do nothing.
- With this protocol is possible that some failures occurs:
    - Failure of a participant (RM): the warm restart procedure is modified with a new case: if the last record in the log for transaction T is "ready", then T does not know the global decision of its TM.
        - Recovery:
            - READY list: it's a new lest collecting the IDs of all transactions in ready state;
            - For all transactions in the ready list, the global decision is asked to the TM at the restart (remote recovery request).
    - Failure of the coordinator (TM): messages that can be lost:
        - Prepare (outgoing),
        - Ready (incoming),
        - Global decision (outgoing).
        - Recovery:
            - If the last record in the TM log is prepare then the global abort decision is written in the log and sent to all participants;
            - If the last record in the TM log is the global decision than there's the repetition of Phase II.
    - It's important to say that any kind of network problem in Phase I causes global abort (the prepare or the ready message are not received). Any network problem in Phase II causes the repetition of Phase II.

- Parallel DBMS: in this DBMS it's used parallelism for computation for increasing efficiency, in fact queries can be effectively parallelized. For doing that, different technological solutions are available, such as multiprocessor systems or computer clusters.
- Inter-query parallelism it's used to schedule and execute different queries (or just subparts of one single query) on different processors, of course queries must be non-conflicting requests, in terms of memory locations. Inter-query parallelism is used mostly in OLTP systems.
- Benchmarks describe the conditions in which performance is measured for a system and in the case of DBMS, they are standardized by the Transaction Processing Council (TPC). Each benchmark is characterized by:
    - Transaction load,
        - Distribution of arrival time of transactions;
    - Database size and content,
        - Randomize data generation;
    - Transaction code,
    - Techniques to measure and certify performance.

Data Preparation

- Before starting with data mining activities we need to make some kind of data preprocessing to objects that we're going to analyze for having good results without noises or bad attributes.
- The most important techniques for data preparation are:
    - Aggregation
        - Aggregation means combining two or more attributes (or objects) into a single attribute (or object). The main purposes are: data reduction (reducing number of attributes or objects), change of scale and more "stable" data (aggregated data tends to have less variability).
    - Data reduction is a generalization of aggregation, it generates a reduced representation of the dataset. This representation is smaller in volume, but it can provide similar analytical results. The methods for doing data reduction are:
        - Sampling
            - This is the main technique employed for data selection and it's often used for both the preliminary investigation of the data and the final data analysis. Sampling is used in data mining because processing the entire set of data of interest is too expensive or time consuming, especially in case of very big data sets. The key principle for effective sampling is the following:
                - Using a sample will work almost as well as using the entire data sets, if the sample is representative;
                - A sample is representative if it has approximately the same property (of interest) as the original set of data.
            - There are different types of sampling:
                - Simple random sampling, in which there's an equal probability of selecting any particular item from the dataset;

        - Sampling without replacement, where as each item is selected, it's removed from the population of the data set;
        - Sampling with replacement, objects are not removed from the population as they are selected for the sample and in this way it's possible that the same object can be selected more than once;
        - Stratified sampling, where data is split into several partitions (similar with clustering); then draw random samples from each partition.
- Dimensionality Reduction
    - When the dimensionality of a data set increases, data becomes increasingly sparse in the space that it occupies; for this reason different techniques are used for: avoiding curse of dimensionality, reducing amount of time and memory required by data mining algorithms, allowing data to be more easily visualized and helping to eliminate irrelevant features or reducing noises. One of this technique is the Principle Component Analysis that is used to find a projection that captures the largest amount of variation in data, this projection replaces some specific attributes in data set, reducing the dimensionality.
- Feature subset selection
    - Another way to reduce dimensionality of data is to eliminate some attributes or features that are not useful to the analysis, such as eliminating of:
        - Redundant features: duplicate much or all of the information contained in one or more other;
        - Irrelevant features: contain no information that is useful for the data mining task at hand.
    - There are different techniques for applying the feature subset selection and they are:
        - Brute-force approach: try all possible feature subsets as input to data mining algorithm and select the subset which gives better results (actually this method is never used because of its higher computational cost);
        - Embedded approaches: feature selection occurs naturally as part of the data mining algorithm;
        - Filter approaches: features are selected before data mining algorithm is run (filtering first);
        - Wrapper approaches: use the data mining algorithm as a black box to find the best subset of attributes, by using some heuristic filtering and exploring just a subset of attributes.
- Feature creation
    - It consists on creating new attributes that can capture the important information in a data set much more efficiently than the original attributes.
- Mapping data to a new space
    - Eg: Fourier transform or Wavelet transform.
- Discretization and Binarization

- Discretization splits the domain of a continuous attribute in a set of intervals, reducing in this way the cardinality of the attribute domain.
- There are several techniques for making Discretization and they are:
    - N intervals with the same width ($W = (v_{max} - v_{\min})/N$)
        - Easy to implement but it can be badly affected by outliers and sparse data; it has an incremental approach.
    - N intervals with (approximately) the same cardinality
        - It's better than the previous one because it better fits sparse data and outliers, but it hasn't incremental approach.
    - Clustering
        - It's the best technique and it well fits sparse data and outliers.

- Attribute transformation
    - It consists in the usage of a function that maps the entire set of values of a given attribute to a new set of replacement values such that each old value can be identified with one of the new values. This method can be performed by using simple functions such as exponentials, logarithms, modulo, etc. or by using standardization and normalization.
        - Normalization is a type of data transformation and it's used when working with different ranges values; the values of an attribute are scaled so as to fall within a small specified range, typically (-1,+1) or (0,+1). There are three techniques:
            - Min-max normalization: it's a sort of proportion of values and typically it's the most used; it follows this formula:
            $$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$
            - Z-score normalization: it's used with no uniform distribution of values and in presence of noises; it follows this formula: $v' = \frac{v - mean_A}{stand\_dev_A}$
            - Decimal scaling: it makes just some scaling by ten of a value $v' = \frac{v}{10^j}$ where j is the smallest integer such that max(|v'|)<1.

- In some cases is very useful making some comparison between two (or more) different objects, this can be done by exploiting some comparing parameters of calculating the "distance" between objects. This techniques are: similarity, dissimilarity and distance.
    - Similarity is a numerical measure of how alike two data objects are. If often falls in the range [0,1] and is higher when objects are more alike (similar or equal).
    - Dissimilarity is a numerical measure of how different are two data; it's lower when object are more alike (similar or equal) and the minimum dissimilarity is often 0. Upper limit varies and can be an high value.
    - There are several formulas for making this numerical measures based on what type of attribute we're analyzing for:
    p and q are the attribute values for two data objects

Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 \; if \; p = q \\ 1 \; if \; p \neq q \end{cases}$	$s = \begin{cases} 1 \; if \; p = q \\ 0 \; if \; p \neq q \end{cases}$
Ordinal	$d = \frac{\lvert p - q \rvert}{n - 1}$ (values mapped to integers 0 to n-1, where n is the number of values)	$s = 1 - \frac{\lvert p - q \rvert}{n - 1}$
Interval or Ratio	$d = \lvert p - q \rvert$	$s = -d, s = \frac{1}{1+d}$ or $s = 1 - \frac{d - \min\_d}{\max\_d - \min\_d}$

- In some cases it's possible to calculate the distance between two or more object, according to its attributes. There are two methods:
  - Euclidean Distance: it's the standard method for having the distance between objects. It's based on this formula: $dist = \sqrt{\sum_{k=1}^{n}(p_k - q_k)^2}$ where *n* is the number of dimensions (attributes) and $p_k$and q_k are, respectively, the k^{th} attributes (components) or data object *p* and *q*. Standardization (normalization) is necessary when the objects have different scales of representation.
  - Minkowski Distance: it's a generalization of Euclidean Distance and consist of a modification of the radix and exponential values; it's based on this formula: $dist = (\sum_{k=1}^{n}|p_k - q_k|^r)^{\frac{1}{r}}$, where *r* is a parameter, *n* is the number of dimensions (attributes) and $p_k$and q_k are, respectively, the k^{th} attributes (components) or data object *p* and *q*. The values of parameter *r* depends on what kind of analysis we're going to do:
    - $r = 1$ → City block distance (a common example of this is the Hamming distance, which is just the number of bits that are different between two binary vectors);
    - $r = 2$ → Euclidean distance;
    - $r \to \infty$ → "Supremum" (L_{max}norm, L_∞norm) distance (this is the maximum difference between any component of the vectors).
  - There're some common properties related to the measurement of distance:
    - Positive definiteness: d(p,q) ≥ 0 for all *p* and *q* and d(p,q) = 0 only if *p*=*q*;
    - Symmetry: d(p,q) = d(q,p) for all *p* and *q*;
    - Triangle Inequality: d(p,r) ≤ d(p,q)+d(q,r) for all points *p*,*q* and *r*. A distance that satisfies these properties is a metric.
  - Similarities also have some well known properties:
    - Maximum similarity : S(p,q) = 1 only if p = q;
    - Symmetry: S(p,q) = S(q,p) for all p and q.
  - In the case when two objects are composed by binary attributes only, we can exploit two different methods to evaluate the distance between this objects and they are Simple Matching and Jaccard Coefficients. First of all it's important to define some values:
    - M_{01} = the number of attributes where p was 0 and q was 1;
    - M_{10} = the number of attributes where p was 1 and q was 0;
    - M_{00} = the number of attributes where p was 0 and q was 0;

- M_{11} = the number of attributes where p was 1 and q was 1.

- Simple Matching Coefficient (SMC) is defined as number of matches / number of attributes and it follows this formula: $SMC = (M_{11} + M_{00})/(M_{01} + M_{10} + M_{11} + M_{00})$.
- Jaccard Coefficients is defined as the number of 11 matches / number of not-both-zero attributes values; this method is more precise with sparse dataset (eg: 0010100000 vs. 010010000). It follows this formula: $J = (M_{11})/(M_{01} + M_{10} + M_{11})$.
- Cosine Similarity is another method to compute the distance between objects and it's implemented by this formula: $\cos(d_1, d_2) = (d_1 \cdot d_2)/\|d_1\|\|d_2\|$, where "·" indicates vector dot product and $\|d\|$ is the length of vector *d*.
- Final, it's possible to do not treat all attributes the same, and it can be done by using some weights which are between 0 and 1 and sum to 1.

Association rules

- The objective of the association rules is the extraction of frequent correlations or pattern from a transactional database (transactional database means a database which contains transactions as records, each transaction is a collection of unsorted item).
- The association rule is definite in this way: $A, B \Rightarrow C$, where *A* and *B* are items in the *rule body* and *C* is the item in the *rule body*; ⇒ means *co-occurrence* and identify the correlation.
- Here some definition:
  - Itemset: it's a set including one or more items;
  - K-Itemset: it's an itemset that contains k items;
  - Support count(#): it's the frequency of occurrence of an itemset;
  - Support: it's the fraction of transactions that contain an itemset;
  - Frequent itemset: it's an itemset whose support is greater than or equal to a minsup threshold.
- It's possible to define some rule quality metrics: given the association rule $A \Rightarrow C$ we have:
  - Support: it's the fraction of transaction containing both A and B: $\frac{\#\{A,B\}}{|T|}$, where |T| is the cardinality of the transactional database; support defines a priori probability of itemset AB and rule frequency in the database.
  - Confidence: it's the frequency of B in transaction containing A: $\frac{\sup(A,B)}{\sup(A)}$; confidence defines the conditional probability of finding B having found A and the strength of the implication.
- Given a set of transaction T, association rule mining is the extraction of the rules satisfying the constrains: support ≥ minsup threshold and confidence ≥ minconf threshold. The result of the extraction can be compared to a normal SQL statement because it's complete (all rules satisfying both constraints) and correct (only the rules satisfying both constraints); it' possible to add other more complex constraints.
- The association rule extraction can be done by using a brute-force approach that consists on enumerate all possible permutations of items (association rule), compute support and confidence for each rule and prune the rules that do not satisfy the minsup and minconf constraints. Actually the huge amount of possibly permutations makes this approach computationally unfeasible; for this reason, given an itemset, the extraction process may be split in this way: first generate frequent itemsets, next generate rules from each frequent itemset. The extraction of frequent itemsets can be done by exploiting different techniques level-

wise approaches (Apriori, …) or approaches without candidate generation (such as FP-growth); generally this is the most computationally expensive step (it's possible to limit the extraction by means of support threshold). The extraction of association rules is the generation of all possible binary partitioning of each frequent itemset (possibly enforcing a confidence threshold).

- In the brute-force approach, each itemset in the lattice is a candidate for frequent itemset and for each candidate a scan of the database is performed to count the support. The complexity of this algorithm is more or less $O(|T|2^d w)$ where |T| is the number of transactions, d is the number of items and w is the transaction length.
- It's possible to improve the brute-force approach by making some modification, improving efficiency. The possibilities are:
    - Reduce the number of candidates (prune the search space);
    - Reduce the number of transactions (prune the transactions as the size of itemsets increases);
    - Reduce the number of comparisons (using of efficient data structures to store the candidates or transactions).
- The Apriori Principle is a method to reduce the number of candidates and it's based on the fact that "if an itemset is frequent, then all of its subsets must also be frequent"; the support of an itemset can never exceed the support of any of its subsets. It holds due to the antimonotone property of the support measure: given two arbitrary itemsets A and B, if $A \subseteq B$ then $sup(A) \geq sup(B)$.
- The Apriori Algorithm (Agr94) is a level-based approach, means that at each iteration it extracts itemsets of a given length k. Two main steps are performed for each level:
    - Candidate generation:
        - Join step: generate candidates of length k+1 by joining frequent itemsets of length k;
        - Prune step by applying Apriori Principle: prune length k+1 candidate itemsets that contain at least one k-itemset that is not frequent.
    - Frequent itemset generation:
        - Scan DB to count support for k+1 candidates;
        - Prune candidates below minsup.
    - The generation of candidates are performed in this way:
        - Sort L_k candidates in lexicographical order;
        - For each candidate of length k:
            - Self-join with each candidate sharing among L_{k-1} prefix;
            - Prune candidates by applying Apriori Principle.
    - During the counting of support for candidates, scan transaction database is needed to count support of each itemset, this can be considerate as a sort of bottleneck because total number of candidates may be large and/or one transaction may contain many candidates. The approach of Agr94 is based on using hash-tree structures for storing candidate itemsets (leaf node of hash-tree contains a list of itemsets and count, interior node contains a hash table); a subset function finds all candidates contained in a transaction.
    - Another bottleneck is the generation of the candidate: candidate sets may be huge and 2-itemset candidate generation is the most critical step; additionally n+1 database scans are needed when longest frequent pattern length is n.
    - Generally the major factors affecting performance of Arg94 algorithm are:

        - Minimum support threshold: lower support threshold increases number of frequent itemsets (large number of candidates, larger length of frequent itemsets);
        - Dimensionality (number of items) of the dataset: more space is needed to store support count of each item and if number of frequent items also increases, both computation and I/O costs may also increases;
        - Size of database: since Apriori Principle makes multiple passes, run time of algorithm may increase with number of transactions;
        - Average transaction width: transaction width increases in dense data set and may increase max length of frequent itemsets and traversal of hash tree (number of subsets in a transaction increases with its width).
    - There are several methods that are designed to improve Arg94 and one of this is the Hash-based itemset counting (Yu95): a k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent.
- Frequent Pattern - growth Algorithm (Han00) is another method that is used to extract rules; it exploits a main memory compressed representation of the database that is the FP-Tree; this high compression if good for dense data distribution, less so for sparse one. There's a complete representation for frequent pattern mining, enforces support constraint. Frequent pattern mining is make by means of FP-growth by using recursive visit of FP-Tree (that is located in main memory) and applying divide-and-conquer approach (decomposing mining task into smaller subtasks). Because of main memory exploiting, only two database scans are needed: the first for counting item supports then the second for building FP-Tree, after this last operation all data of the FP-Tree is located in main memory and no other database scans are needed.
- Han00 algorithm follows this steps:
    - Constructions of the FP-Tree:
        - Count item support and prune items below minsup threshold;
        - Build Header Table by sorting items in decreasing support order;
        - Create FP-Tree: for each transaction *t* in database:
            - Order transaction *t* items in decreasing support order (same order as in Header Table);
            - Insert transaction *t* in FP-Tree by using existing path for common prefix and creating a new branch when path becomes different.
    - Scan Header Table from lowest support item up;
        - For each item *i* in Header Table extract frequent itemsets including item *i* and items preceding it in Header Table:
            - Build Conditional Pattern Base for item *i* (i-CPB) (select prefix-paths of item *i* from FP-Tree);
            - Recursive invocation of FP-Growth on i-CPB.
        - Repeat until all elements in the Header Table are computed.
    - At the end Han00 returns all possible combination, each of them with its support.
- According to its features, an itemset can be Maximal Frequent or Closed:
    - An itemset is Frequent Maximal if none of its immediate supersets is frequent;
    - An itemset is closed if none of its immediate supersets has the same support as the itemset.

    - Based on this we can say that Maximal Frequent Itemsets are Closed Itemsets but Closed Itemsets may be not Maximal Frequent Itemsets.
- Selection of the appropriate minsup threshold is not obvious:
    - If minsup is too high, itemsets including rare but interesting items may be lost (example: pieces of jewelry or other expensive products);
    - If minsup is too low, computation may becomes very expensive and the number of frequent itemsets becomes very large.

Classification

- Given a collection of records (called training set), in which each record contains a set of attributes and one of them is the class of the record, it's possible to find a model for class attribute as a function of the values of other attributes. The main goal is to assign a class as accurately as possible at the previously unseen records or new records; this records are called test set and it's used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.
- The steps for the classification tasks are:
    - Induction: learn the model by applying the training set and using one of the learning algorithm;
    - Deduction: after the creation of the model, the classification (model) is applied to the test set, for checking the model evaluation.
- There are several classification techniques and each of them is applicable on different situations. The most important methods are:
    - Decision Tree based method;
    - Rule-based method;
    - Memory based reasoning;
    - Neural networks;
    - Naïve Bayes and Bayesian Belief Networks;
    - Support vector machines.
- The evaluation of classification techniques are:
    - Accuracy: quality of the prediction;
    - Efficiency: model building time and classification time;
    - Scalability: training set size and attribute number (dimensionality of the objects);
    - Robustness: avoiding of noises and sensibility to the missing data;
    - Interpretability: model interpretability (by humans) and model compactness (properties of classes).
- A Decision Tree is a decision support tool that uses a tree-like graph or model of decision and their possible consequences. Starting by a root, each node is represented by a splitting attribute and each leaf represents the class label; there could be more than one tree that fits the same data. The building time is represented by the Induction phase of classification; after that the Deduction is done by checking the values of the attribute in the test set and treading the tree.
    - There are several algorithms that lead the learning of the model (Induction) and one of the earliest is the Hunt's Algorithm. The general structure of Hunt's algorithm is this: let D_t be the set of training records that reach a node t, the general procedure is:

- If D_t contains records that belong to the same class y_t, then t is a leaf node labeled as y_t;
- If D_t is an empty se, then t is a leaf node labeled by the default class y_d (the default class is a class with the higher probability);
- If D_t contains records that belong to more than one class, use an attribute test to slpit the data into smaller subsets. Recursively apply the procedure to each subset.

- The Induction phase follows a Greedy strategy that is split the records based on attribute test that optimizes certain criterion.
- The main issues are: determine how to split the records (attribute test condition and best split) and determine the stop condition.
  - The specification of test condition depends on attribute type that can be nominal, ordinal and continuous and it depends also on number of ways to split: 2-way split for binary tree, where each node have at most 2 children, multi-way split when each node can have more than two children.
    - For nominal attributes:
      - Multi-way split: use as many partitions as distinct values;
      - Binary split: divides values into two subsets (need to find the optimal partitioning).
    - For ordinal attributes:
      - Multi-way split: use as many partitions as distinct values;
      - Binary split: divides values into two subsets (need to find the optimal partitioning).
    - For continuous attributes there are different ways of handling:
      - Discretization (multi-way split) to form an ordinal categorical attribute
        - Static (discretize once at the beginning),
        - Dynamic (ranges can be found by equal interval bucketing, equal frequency bucketing or clustering);
      - Binary decision (A<v) or (A≥v)
        - Consider all possible splits and finds the best cut,
        - Can be more compute intensive.
  - To determine the best split condition for each node a Greedy approach is followed: nodes with homogeneous class distribution are preferred (pure partition is the best situation). For doing that we need a measure of node impurity; the main measures of node impurity are: Gini index, entropy and misclassification error.
    - Gini index for a given node t is:

$$GINI(t) = 1 - \sum_j [p(j|t)]^2$$

      where $p(j|t)$ is the relative frequency of class j at node t.
      The maximum value is $(1 - 1/n_c)$ when records are equally distributed among all classes, implying least interesting information; the minimum value is 0, when all records belong to one single class, implying most interesting information (pure partition).
      When a node p is split into k partitions (children), the quality of split is computed

as:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where n_i is the number of records at child i and n is the number of records at node p. Also in this case we have different type of splitting, based on type of attributes:

- Binary attributes: splits into two partitions; effect of weighing partitions (larger and purer partitions are sought for);
- Categorical attributes: for each distinct value, gather counts for each class in the dataset; use the count matrix to make decision;
- Continuous attributes: this is the most critical situation because the attribute can also have an huge domain. Use of binary decision based on one value and several choices for the splitting value and each of them has a count matrix associated with it. For choosing the best splitting attribute v there is a simple method, but it's also computationally inefficient for a lot of repetition of work (for each v, scan the database to gather count matrix and compute its Gini index); there is also a variant that works in this way:
  - for each attribute, sort the attribute on values,
  - linearly scan these values, each time updating the count matrix and computing Gini index;
  - choose the split position that has the least Gini index.

- Entropy at given node t is:

$$Entropy(t) = -\sum_{j} p(j|t) \log(p|jt)$$

where p(j|t) is the relative frequency of class j at node t. Entropy measures homogeneity of a node and the maximum value is $\log(n_c)$ when records are equally distributed among all classes implying least information; minimum values is 0 when all records belong to one class, implying most information. Entropy based computations are similar to the GINI index computations. The Information Gain measure is based on Entropy and follow this formula:

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

where parent node p is split into k partitions and n_i is number of records in partition i. Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes the GAIN). The disadvantage is that is tends to prefer splits that result in large number of partitions, each being small but pure. The Gain Ratio is definite in this way: $GainRATIO_{split} = \frac{GAIN_{split}}{SplitINFO}$ and $SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$ where parent node p is split into k partitions and n_i is the number of records in partition i. Gain Ratio adjust Information Gain by the entropy of the partition (SplitINFO); higher entropy partitioning (large number of

small partitions) is penalized. This formula is designed to overcome the disadvantage of Information Gain.

- Classification error at a node t is: $Error(t) = 1 - \max P(i|t)$. It measures misclassification error made by a node. The maximum value is (1-1/n_c) when records are equality distributed among all classes, implying least interesting information; the minimum value is 0, when all records belong to one class, implying most interesting information.

- There are several possible Stopping Criteria for the Tree Induction and some of this are:
  - Stop expanding a node when all the records belong to the same class;
  - Stop expanding a node when all the records have similar attribute values;
  - Early termination.

- The advantages by using a decision tree based classification are:
  - Inexpensive to construct;
  - Extremely fast at classifying unknown records;
  - Easy to interpret for small-sized trees;
  - Accuracy.
- The main disadvantage is that accuracy may be affected by missing data.
- Practical issues of Classification are:
  - Underfitting: we have underfitting situation when model is too simple, both training and test errors are large;
  - Overfitting: we have overfitting situation when model is extremely specialized for the model that it can falls in mistakes. Decision boundary is distorted by noise point, introduced into the initial training set; insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task. It's possible to address the overfitting by following two different methods:
    - Pre-pruning (early stopping rule): stop the algorithm before it becomes a fully-grown tree (forcing some impurity) or stop if all instances belong to the same class or stop if all the attribute values are the same;
    - Post-pruning: grow decision tree to its entirety then trim the nodes of the decision tree in a bottom-up fashion; if generalization error improves after trimming, replace sub-tree by a leaf node, class label of leaf node is determined from majority class of instances in the sub-tree.
  - Missing values: missing values affect decision tree construction in three different ways:
    - Affects how impurity measures are computed;
    - Affects how to distribute instance with missing value to child nodes;
    - Affects how a test instance with missing value is classified.
  - Costs of classification;
  - Data fragmentation: number of instances at the leaf node could be too small to take any statistically significant decision.

        - Search strategy: finding an optimal decision tree is NP-hard problem; the algorithm presented so far uses a greedy, top-down, recursive partitioning strategy to induce a reasonable solution.
        - Expressiveness;
        - Tree replication: it's possible that in a decision tree, some sub-trees appear in multiple branches.
- Alternative techniques for classification are:
    - Rule-Bases classifier: it's based on classify records by using a collection of "if…then…" rules, formally written in this way: $(Condition) \rightarrow y$ where *Condition* is a conjunction of attributes and *y* is the class label.
        - A rule *r* covers an instance *i* if the attributes of the instance satisfy the condition of the rule. The characteristics of Rule-Based classifier are:
            - Mutually exclusive rules: classifier contains mutually exclusive rules if the rules are independent of each other and every record is covered by at most one rule;
            - Exhaustive rules: classifier has exhaustive coverage if it accounts for every possible combination of attribute values and each record is covered by at least one rule.
        - It's possible to extract rules from a decision tree, where rules are mutually exclusive and exhaustive. Rule set contains as much information as the tree and for this reason rules can be simplified, but in this way they are no longer mutually exclusive because a record may trigger more than one rule (a solution is based on ordered rule set and assign the first rule to a record), also, rules are no longer exhaustive and a record may not trigger any rules (a solution is based on usage of a default class.
        - Building Classification rules can be done by using a direct method (extract rules directly from data) or by indirect method (extract rules from other classification models, eg: decision tree).
        - The advantages of Rule-Based classifier are:
            - As highly expressive as decision trees,
            - Easy to interpret,
            - Easy to generate,
            - Can classify new instances rapidly,
            - Performance comparable to decision tree.
    - Associative classification: this classification model is defined by means of association rules $(Condition) \rightarrow y$ where rule body is an itemset. The model generation is based on rule selection and sorting (based on support, confidence and correlation thresholds) and rule pruning (databse coverage). The strong points of associative classification are: interpretable model, higher accuracy than decision tree, efficient classification, unaffected by missing data and good scalability in the training set size; the weak points are: rule generation may be slow (it depends on support threshold) and reduced scalability in the number of attributes (rule generation may become unfeasible).
    - Bayesian classification: it's based on Bayes theorem; let the class attribute and all data attributes be random variables: C = any class label, X = $<x_1,\ldots,x_n>$ record to be classified.

- Bayesian classification is based on compute P(C|x) for all classes (probably that record x belong to C) and assign x to the class with maximal P(C|x). Applying Bayes theorem P(C|X) = P(X|C) * P(C)/P(X), P(X) constant for all C, disgregarded for maximum computation, P(C) is a priori probability of C (P(C) = N_c/N).
  How to estimate P(X|C):
  - Naïve hypothesis at the base of this classification (statistical independence of attributes $x_1,\ldots,x_n$) that actually it's not always true;
  - Computing P(x_k|C):
    - For discrete attribute by using P(x_k|C) = |x_k|/N_c;
    - For continuous attributes by using probability distribution.
- Strong points for Bayesian classification are: efficient classification, medium model interpretability, robust to isolated noise points and irrelevant attributes, incremental model update; weak points are: model generation without Naïve hypothesis is unfeasible and Naïve hypothesis may significantly reduce accuracy.

- Neural Networks: they are inspired to the structure of the human brain (neurons as elaboration units, synapses as connection network).
  - The neural network is built in this way:
    - for each node, definition of set of weights and offset values, providing the highest accuracy on the training data;
    - iterative approach on training data instances.
  - The base algorithm follows this steps:
    - Initially assign random values to weights and offset;
    - Process instances in the training set one at a time:
      - For each neuron, compute the result when applying weights, offset and activation function for the instance;
      - Forward propagation until the output is computed;
      - Compare the computed output with the expected output and evaluate errors;
      - Backpropagation of the error, by updating weights and offset for each neuron.
    - The process ends when:
      - % of accuracy above a given threshold;
      - % of parameter variation (error) below a given threshold;
      - The maximum number of epochs (iterations) is reached.
  - Strong points for neural networks are:
    - Extremely high accuracy,
    - Robust to noise and outliers (they works with percentages so noise and outliers don't came out),
    - Supports both discrete and continuous output,
    - Efficient during classification.
  - Weak points are:

        - Very long training time (weakly scalable in training data size) and absolutely not interpretable model (by human eyes).
    - Support Vector Machines: this method is based on finding a linear hyperplane (decision boundary) that will separate the data; it's possible to identify a lot of hyperplane but it's selected the one which maximizes the margin. We can find also the case in which the decision boundary is not linear and it's possible to solve this situation by transforming data into higher dimensional space.
    - Instance-Based classifier: this classifier doesn't need a model, it exploits directly the itemsets. Instance-Based classifier stores the training records in memory and it uses training records to predict the class table of unseen cases. There are two main examples of this type of classification:
        - Rote-learner: memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly;
        - Nearest Neighbor classifier: uses k "closest" points (nearest neighbor) for performing classification.
            - It requires three things: the set of stored records, distance metric to compute distance between records and the value of k, the number of nearest neighbors to retrieve.
            - To classify an unknown record:
                - Compute distance to other training records;
                - Identify k nearest neighbors;
                - Use class labels of nearest neighbors to determine the class label of unknown record.
            - To compute the distance between two points the Euclidean distance is exploited.
            - Choosing the value of k is a very important point, because if k is too small, the method is sensitive to noise points; if k is too large, neighborhood may include points from other classes.
- The model evaluation is based on different points:
    - Metrics for performance evaluation;
        - They focus on the predictive capability of a model and it's based on the Confusion Matrix, where are reported all cases for a correct/wrong prediction: true positive, true negative, false positive and false negative. Exploiting this values it's possible to compute the accuracy, that is the most widely-used metric and is based on this formula: $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$; anyway this formula is not appropriate for unbalanced class label distribution and different class relevance and for avoiding this is possible to exploit other parameters evaluated for each class and they are: $Recall = \frac{Number\ of\ objects\ correctly\ assigned\ to\ C}{Number\ of\ objects\ belonging\ to\ C}$ and $Precision = \frac{Number\ of\ objects\ correctly\ assigned\ to\ C}{Number\ of\ objects\ assigned\ to\ C}$.
    - Methods for performance evaluation: there are two methods of estimation an both are based of partitioning labeled data in training set for modeling building and test set for model evaluation, they are:

        - Holdout: it's the simplest method and consists on reserve 2/3 of data for training and 1/3 for testing, it's appropriate for large dataset and it may be repeated several times (repeated holdout);
        - Cross validation: it's used with medium size datasets and consists of:
            - Partition data into k disjoined subsets (called folds);
            - K-fold train ok k-1 partitions, test on the remaining one
                - Repeat for all folds;
            - Reliable accuracy estimation, not appropriate for very large dataset.
    - Methods for model comparison: they're based on the representation of the Receiver Operating Characteristic curve (ROC) in which performance of each classifier is represented as a point on the ROC curve (changing the threshold of algorithm changes the location of the point).

Cluster Analysis

- Cluster analysis are represented by finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects on other groups.
- A clustering is a set of clusters and each cluster represent a group of objects with similar features. There are two main types of clustering:
    - Partitional Clustering: a division data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset;
    - Hierarchical Clustering: a set of nested clusters organized as a hierarchical tree (exploiting of dendogram), this type of clustering can be traditional or non-traditional.
- There are other distinctions between sets of clusters and they are:
    - Exclusive vs. Non-exclusive: in non-exclusive clustering, points may belong to multiple clusters;
    - Fuzzy vs. Non-Fuzzy: in Fuzzy clustering, a point belongs to every cluster with some weight between 0 and 1, weights must sum to 1 and probabilistic clustering has similar characteristics;
    - Partial vs. Complete: in some cases, we only want to cluster some of the data (portion of);
    - Heterogeneous vs. Homogeneous: cluster of widely different sizes, shapes and densities.
- There are several different types of clusters and they are:
    - Well-separated clusters: a cluster is a set of points such that any point in a cluster is closer (or more similar) to every point in the cluster than to any point not in the cluster;
    - Center-based clusters: a cluster is a set of points such that any point in a cluster is closer (or more similar) to every point in the cluster than to any point not in the cluster. This type is different from the precedent because the center of a cluster is identified by a centroid, that is the average of all the points in the cluster (probably not a physical point), or a medoid, that is the most "representative" point of a cluster (represented by a real point in the cluster);
    - Contiguous clusters (Nearest neighbor or Transitive): a cluster is a set of points such that a point in a cluster is a closer (or more similar) to one or more other points in the cluster than to any point not in the cluster.
    - Density-based clusters: a cluster is a dense region of points, which is separated by low-density regions, from other regions of high density. This type of cluster is used when the cluster are irregular or intertwined, and when noise and outliers are present;

- Property or Conceptual clusters (Shared Property): finds clusters that share some common property or represent a particular concept;
- Described by an Objective Function.

- The main Clustering Algorithms are:
  - K-Means and its variants:
    - K-means clustering is based on Partitional clustering approach, where each cluster is associated with a centroid; each point is assigned to the cluster with the closest centroid. This algorithm needs in input, the number of clusters K and this are the basic steps:
      - Select K points as the initial centroids,
      - Form K clusters by assigning all points to the closest centroid,
      - Recomputed the centroid of each cluster,
      - Repeat steps (2),(3) until the centroids don't change.
    - Here some details about this algorithm:
      - Initially centroids are often chosen randomly (clusters produced vary from one run to another) and the centroid is typically the mean of the points in the cluster.
      - "Closeness" is measured by Euclidean distance, cosine similarity, correlation or some other measures that can be applied.
      - Most of the convergence happens in the first few iteration and often the stopping condition is changed to "until relatively few points change clusters".
      - The complexity is O(n*K*i*d) where: n is the number of points, K is the number of clusters, i is the number of iterations and d is the number of attributes.
    - Because of the input value K of the algorithm (number of clusters), this method probably will be executed more times, for finding the best clustering criteria, especially when it's not possible to represent the whole dataset. To measure the quality of each solution, the most common measure is Sum of Squared Error (SSE):
      - For each point, the error is the distance to the nearest cluster and to get SSE, we square these errors and sum them: $SSE = \sum_{i=1}^{k} \sum_{x \in C_i} dist^2(m_i, x)$ where x is a data point in cluster C_i and m_i is the representative point for cluster C_i.
      - One easy way to reduce SSE is to increase K, the number of clusters; a good clustering with smallest K can have a lower SSE than a poor clustering with higher K.
    - As previously said, K-means needs some multiple runs to determine the best clustering; this is due to the individuation of the initial centroids; a possible solution is to select more than K initial centroids and then select among these initial centroids (select the most widely separated). After that we need some postprocessing for eliminating artifacts (errors).
    - Basic K-means algorithm can yield empty clusters and for avoiding this there are several strategies:
      - Choose the point that contributes most to SSE;
      - Choose a point from the cluster with the highest SSE;
      - If there are several empty clusters, the above can be repeated several times.
    - Other strategies are based on pre-processing and post-processing:

- Pre-processing:
    - normalize the data,
    - eliminate outliers (hard to do);
- Post-processing:
    - Eliminate small clusters that may represent outliers,
    - Split “loose” clusters (clusters with relatively high SSE),
    - Merge clusters that are “close” and that have relatively low SSE,
    - Can use these steps during the clustering process.

- Bisecting K-means algorithm is a variant of K-means that can produce a Partitional or a hierarchical clustering, it’s a locally optimization of error. The main steps are:
    - Initialize the list of clusters to contain the cluster containing all points,
    - Select a cluster from the list of clusters,
    - For each iteration, bisect the selected cluster using K-means,
    - At the end of all iterations, add the two clusters from the bisection with the lowest SSE to the list of clusters,
    - Repeat steps (2),(3),(4) until the list of clusters contains K clusters.
- K-means has problems when clusters are of differing:
    - Sizes,
    - Densities,
    - Non-globular shapes;
- K-means also has problems when the data contains outliers.

- Hierarchical clustering: it produces a set of nested clusters organized as a hierarchical tree, it can be visualized as a dendogram, that is a tree like diagram that records the sequences of merges or splits.
    - The Strengths of Hierarchical clustering are that do not have to assume any particular number of initial clusters (any desired number of clusters can be obtained by “cutting” the dendogram at the proper level).
    - There are two main types of hierarchical clustering:
        - Agglomerative: it starts with the points as individual clusters and at each step, it merges the closest pair of clusters until only one cluster (or k clusters) left;
        - Divisive: it starts with one, all-inclusive cluster and at each step, it splits a cluster until each cluster contains a point (or there are k clusters).
    - Traditional hierarchical algorithms use a similarity or distance matrix to merge or split one cluster at a time.
    - The most popular hierarchical clustering technique is the agglomerative clustering algorithm and its basic algorithm is straightforward:
        - The main steps are:
            - Compute the proximity matrix,
            - Let each data point be a cluster
            - Merge the two closest clusters,
            - Update the proximity matrix,
            - Repeat steps (3),(4) until only a single cluster remains.

- The key operation is the computation of the proximity of two clusters and there are different approaches to defining the distance between clusters distinguish the different algorithms, this approaches are:
    - MIN or Single Link: similarity of two clusters is based on the two most similar (closest) points in the different clusters;
        - Strength: it can handle non-elliptical shapes,
        - Limitation: sensitive to noise and outliers.
    - MAX or Complete Linkage: similarity of two clusters is based on the two least similar (most distant) points in the different clusters;
        - Strength: less susceptible to noise and outliers,
        - Limitation: it tends to break large clusters and biased towards globular clusters.
    - Group Average: proximity of two clusters is the average of pairwise proximity between points in the two clusters; it's a compromise between Single and Complete link;
        - Strength: less susceptible to noise and outliers,
        - Limitation: biased towards globular clusters.
    - Ward's Method: similarity of two clusters is based on the increase in squared error when two clusters are merged (similar to group average if distance between points is distance squared); it's less susceptible to noise and outliers, based towards globular clusters and hierarchical analogue of K-means and in fact it can be used to initialize K-means.
- Computational time and space requirements of hierarchical clustering tend to be high, in fact the space since it uses the proximity matrix is $O(N^2)$ (where N is the number of points) and computational time is $O(N^3)$ in many cases (there are N steps and at each step the size N^2, proximity matrix must be updated and searched.
- The main problems for hierarchical clustering are:
    - Once a decision is made to combine two clusters, it cannot be undone;
    - No objective function is directly minimized;
    - Different schemes have problems with one or more of the following: sensitivity to noise and outliers, difficulty handling different sized clusters and convex shapes, breaking large clusters.

- Density-based clustering (DBSCAN): DBSCAN is a density-based algorithm; density represents a number of points within a specified radius (Eps);
    - Here some definitions:
        - A point is a core point if it has more than a specified number of points (MinPts) within Eps (these are points that are at the interior of a cluster);
        - A border point has fewer than MinPts within Eps, but it's in the neighborhood of a core point;
        - A noise point is any point that is not a core point or a border point.
    - DBSCAN eliminates noise points and perform clustering on the remaining points; the pseudo-code for the algorithm is this:

- Current_cluster_label <- 1
- For all core points do
    - If the core point has no cluster label then
        - Current_cluster_label=Current_cluster_label+1
        - Label the current core point with cluster label Current_cluster_label
    - End if
    - For all points in the Eps-neighborhood, except i^{th} the point itself do
        - If the point does not have a cluster label then
            - Label the point with cluster label Current_cluster_label
        - End if
    - End for
- End for

- DBSCAN doesn't work well with varying density clusters and high-dimensional data.

- The validation of clustering structures is the most difficult task; to evaluate "goodness" of the resulting clusters, some numerical measures can be exploited and they are classified into two main classes:
    - External index: used to measure the extent to which cluster labels match externally supplied class labels (such as entropy, purity, …);
    - Internal index: used to measure the goodness of clustering structure without respect to external information (such as Sum of Squared Error (SSE), cluster cohesion, cluster separation, Rand-Index, adjusted rand-index, …).
        - Cluster Cohesion: measures how closely related are objects in a cluster;
        - Cluster Separation: measure how distinct or well-separated a cluster is from other clusters.
- The validation of clustering structures is the most difficult and frustrating part of cluster analysis; without a strong effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage.