Artificial Intelligence and digital transformation

technofutur TIC
centre de compétence

Table of Contents

Chapter 1 - Introduction

Chapter 2 - Algorithms & Models

Chapter 3 - Implementation

Examples - Resources

Brainstorming

What do you think to know about Intelligence Artificial (IA)?

Chapter 1 – Introduction

Definition

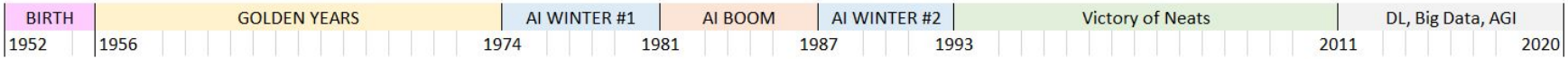
- A system's ability to correctly interpret external data, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation

Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence, Kaplan A, Haenlein M.

- The designing and building of intelligent agents that receive percepts from the environment and take actions that affect that environment.

Artificial Intelligence: A Modern Approach, Stuart Russell and Peter Norvig

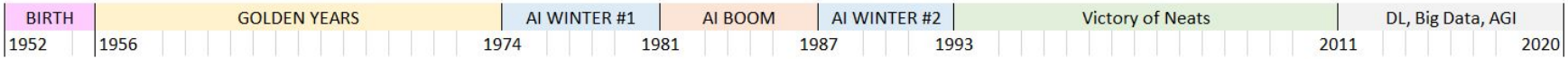
History: some milestones (1)



- **1956:** the formalization of artificial intelligence as a true scientific field at a conference in the United States held at Dartmouth College.
- **in the 60's:** research around AI and a lot of expectations.
- **1974:** AI projects do not succeed ⇨ reduction of funding.
- **in the 80's:** success of expert systems ⇨ relaunch research projects on artificial intelligence.

"An expert system makes it possible to model an expert's reasoning and draw conclusions from a knowledge base"

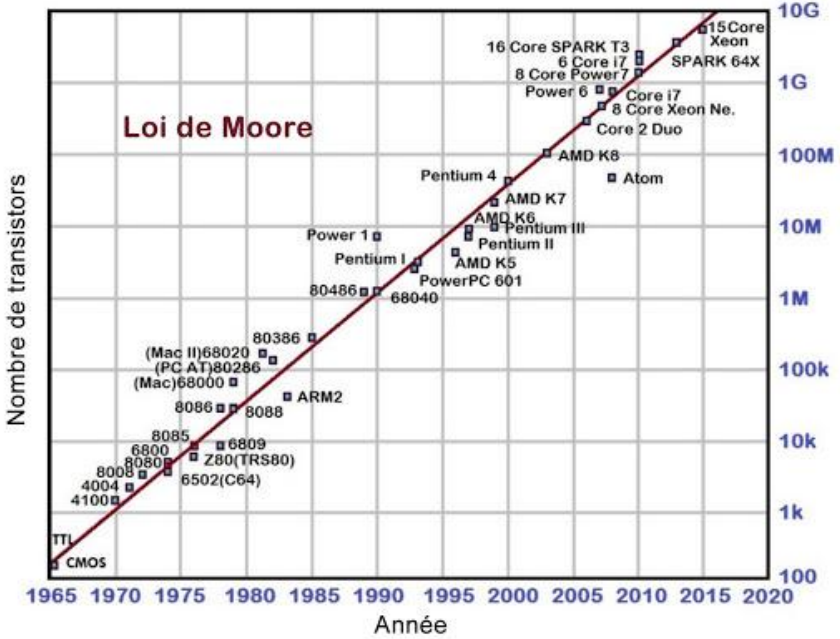
History: some milestones (2)



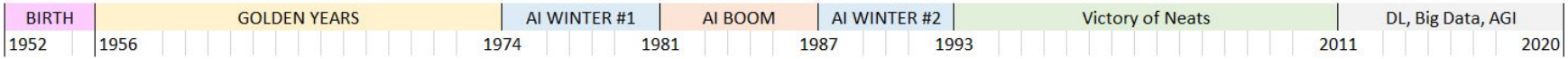
- **in the 90's:** exponential development of IT performance (Moore's Law) ⇨ better use of AI

- **1997:** Garry Kasparov, world chess champion is beaten by Deep Blue

- **in the 2000's:** computer boom ⇨ continuous improvement of IT performance



History: some milestones (3)



- **2011:**
  - Low-cost graphics processors capable of performing large quantities of calculations
  - Introduction of much more sophisticated algorithms ⇨ Deep Learning
  - Availability of very large databases and correctly annotated ⇨ more detailed learning
  - Big companies are investing heavily in AI projects such as Google, Apple, Facebook, … (GAFAM)
  - Change of problematic: having data to process

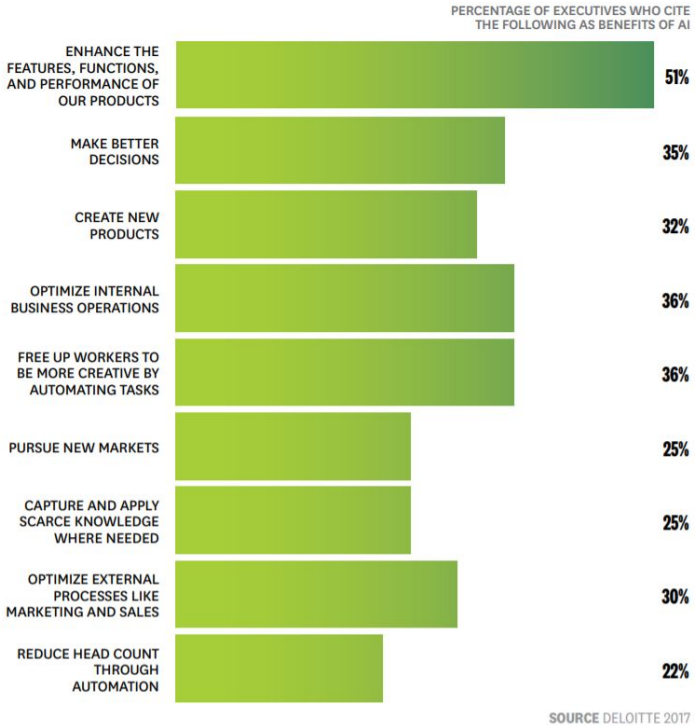
    ⇨ Many “hidden” mechanisms are in place by companies to collect data

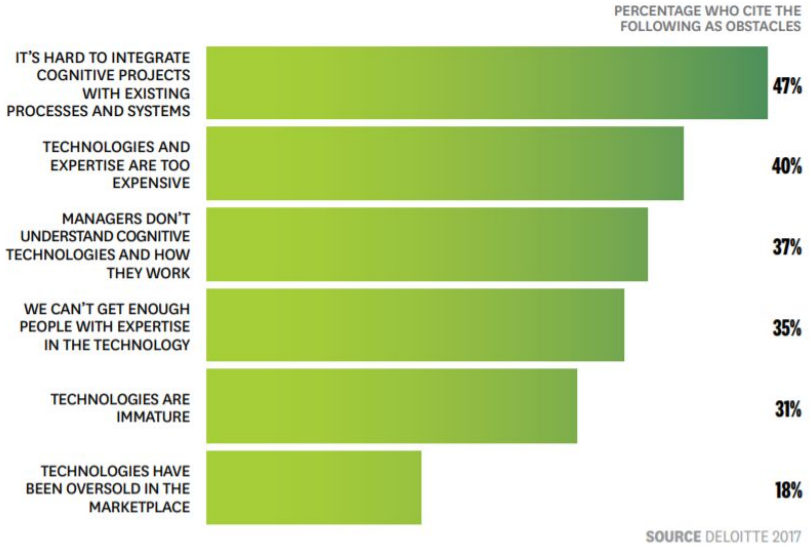
- **Future…** Artificial General Intelligence??

Use Cases

- Sales
  - Sales Rep Next Action Suggestions
  - Meeting Setup Automation (Digital Assistant)
  - Customer Sales Contact Analytics
  - Sales Compensation
- Customer Services
  - Call Classification
  - Customer Service Response Suggestions
  - Chatbot
- Finance
  - Fraud Detection
  - Risk profiling in insurance
  - Credit Lending & Scoring
  - Billing and reminders
- Health Care
  - Personalized Medications and Care
  - Assisted or Automated Diagnosis
  - Real-Time Prioritization and Triage
- Human Resources
  - Hiring
  - HR Retention Management
- Autonomous Things
  - Self-Driving Cars

Expected benefits





Machine Learning Types

Unsupervised Machine Learning

- *Goal :* **Organize** data or **describe** a pattern in historical data
- *Learning process :*
  Learn
- *Historical data :* Data points have no labels associated with them
- *Types :*
  - Clustering algorithms
  - Association rule learning algorithms

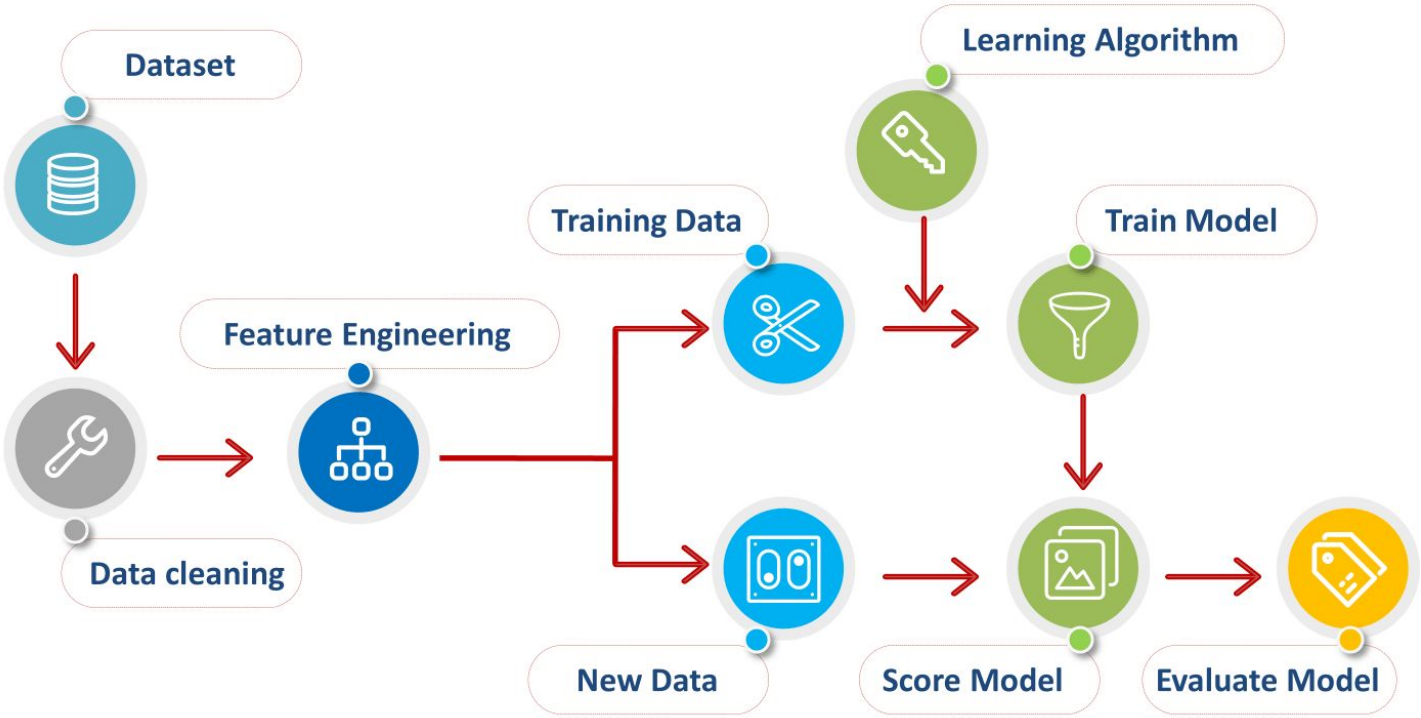
Supervised Machine Learning

- *Goal :* Make **predictions** on new observations
- *Learning process :*
  Learn - Predict
- *Historical data :* Each data point is labeled or associated with a category or value of interest
- *Types :*
  - Classification algorithms
  - Regression algorithms

Reinforcement learning

- *Goal :* **Optimize** a reward function
- *Learning process :*
  Learn - Predict - Get reward - Learn ...
- *Historical data :* Data points, streamingly coming, have labels associated with the action undertaken and a reward value
- *Types :*
  - Criterion of optimality
  - Brute force
  - Value function
  - Direct policy search

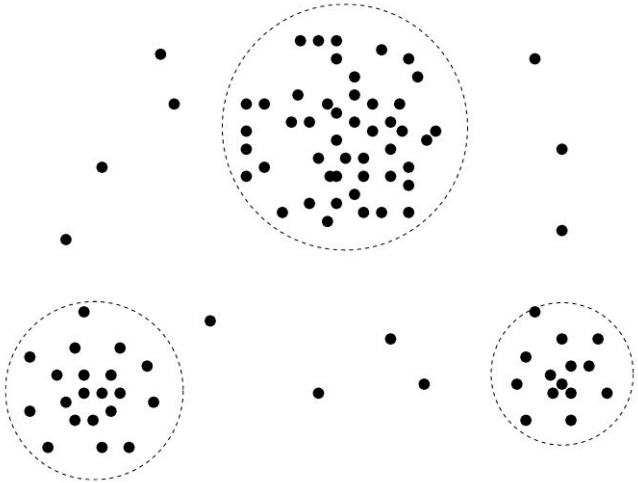
Supervised Machine Learning Process



Afroz Chakure - https://towardsdatascience.com/data-preprocessing-3cd01eefd438

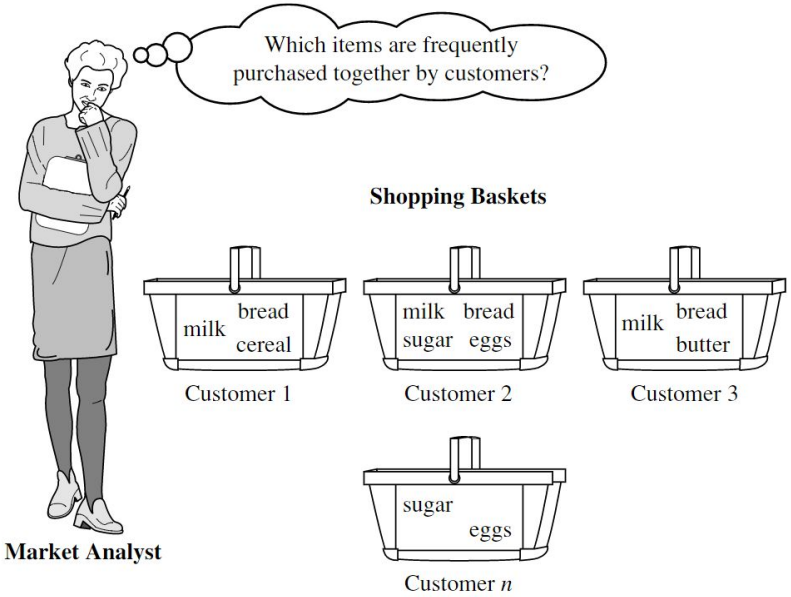
Unsupervised Machine Learning

Clustering algorithms



Unsupervised Machine Learning

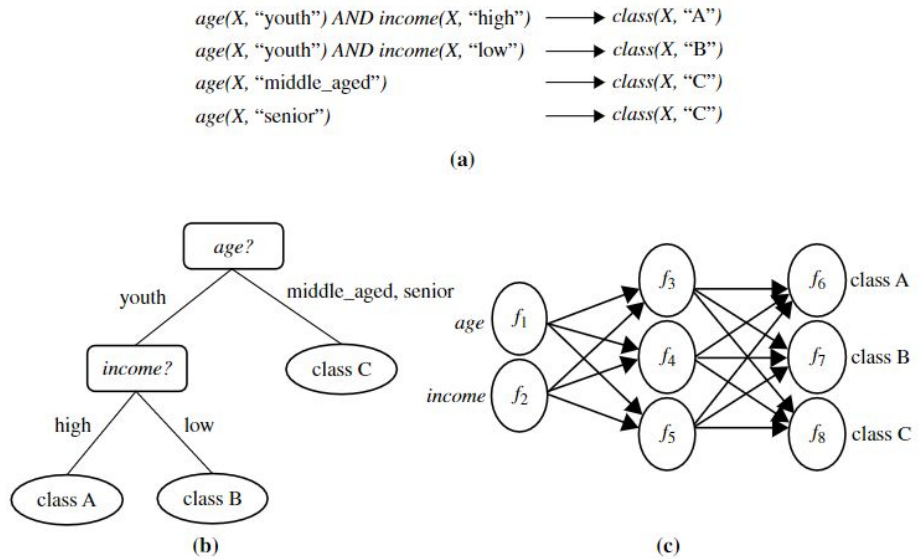
Association rule learning algorithms



milk ⇒ bread [support = 75%; confidence = 100%]

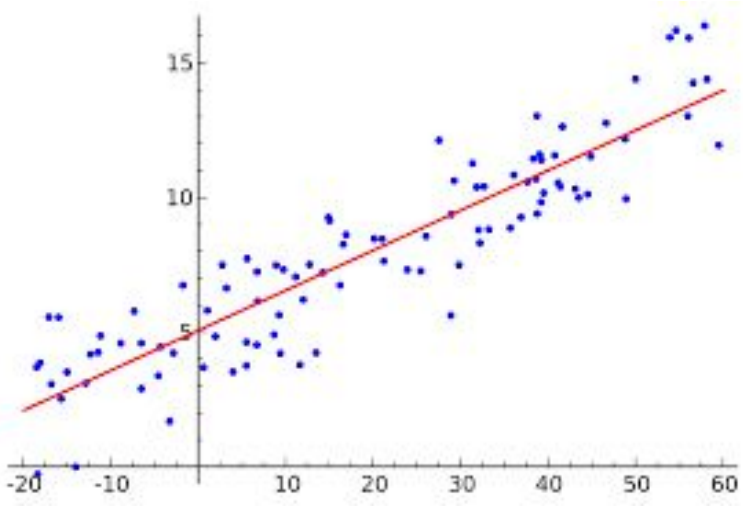
Supervised Machine Learning

Classification algorithms



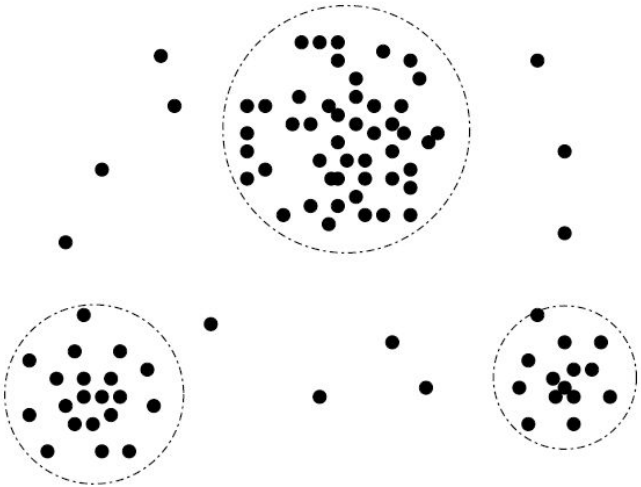
Supervised Machine Learning

Regression algorithms



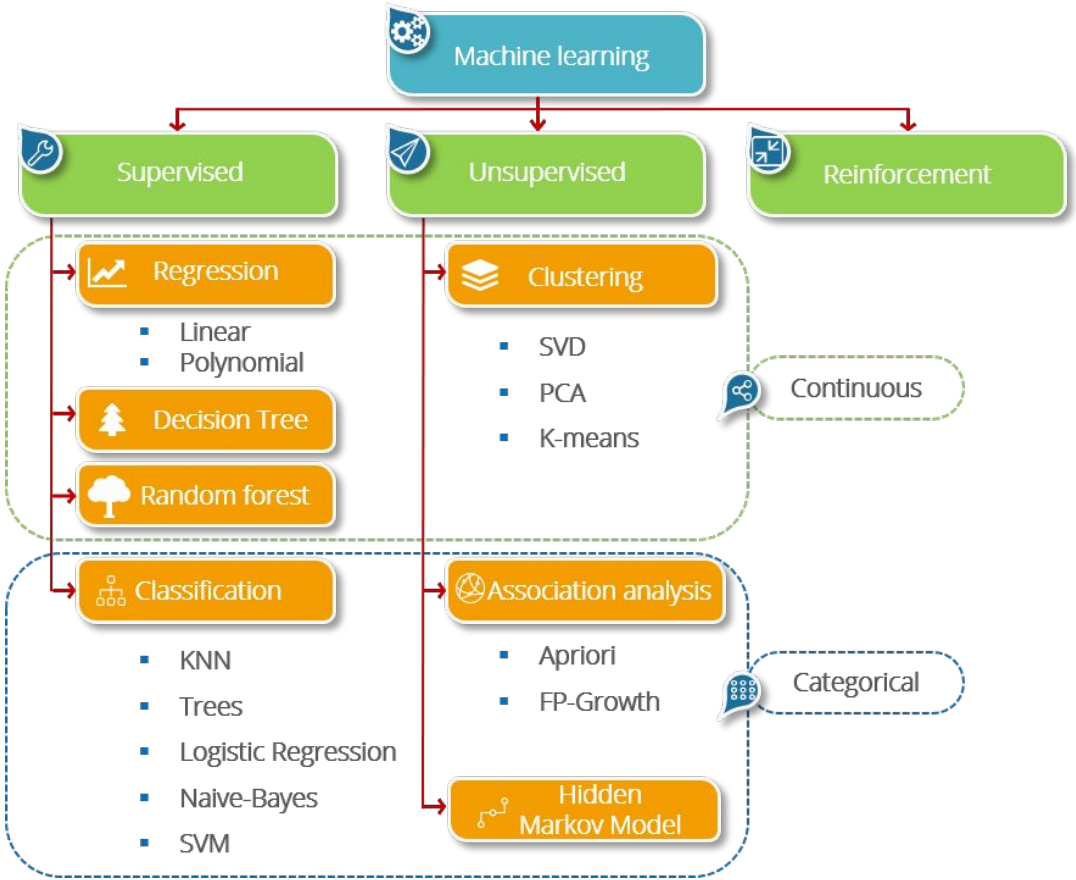
(Un)supervised Machine Learning

Outlier detections



A 2-D customer data plot with respect to customer locations in a city, showing three data clusters. Outliers may be detected as values that fall outside of the cluster sets.

Chapter 2 – Algorithms & Models


Machine learning
Supervised
Unsupervised
Reinforcement
Regression
Linear
Polynomial
Decision Tree
Random forest
Clustering
SVD
PCA
K-means
Continuous
Classification
KNN
Trees
Logistic Regression
Naive-Bayes
SVM
Association analysis
Apriori
FP-Growth
Categorical
Hidden Markov Model

Classification

Basic Concepts

Classification is a form of data analysis that extracts **models** describing important data **classes**.

Such models, called classifiers, **predict** categorical (**discrete, unordered**) class labels.

Examples:

- A bank loans officer needs analysis of her data to learn which loan applicants are “safe” or “risky”.
- A marketing manager needs to guess whether a customer with a given profile will buy a computer.
- A medical researcher wants to analyze breast cancer data to predict which one of three specific treatments a patient should receive.

In each of these examples, the data analysis task is **classification**, where a model or **classifier** is constructed to predict class (categorical, unordered) labels, such as “safe” or “risky” for the loan application data; “yes” or “no” for the marketing data;or “treatment A,”“treatment B,” or “treatment C” for the medical data.

Basic Concepts

General Approach to Classification

Data classification is a three-step process, consisting of :

- a **learning step** (or training phase) where a classification model is constructed
- a **validation step** where the model is used to predict class labels for known data
- a **classification step** where the model is used to predict class labels for new data

Basic Concepts

$$y = f(\boldsymbol{X})$$

A tuple, ***X***, is represented by an *n*-dimensional attribute vector, ***X*** = (x_1,x_2,...,x_n), depicting *n* measurements made on the tuple from *n* database attributes, respectively, A_1,A_2,...,A_n.

Each tuple, ***X***, is assumed to belong to a predefined class as determined by another database attribute, *y*, called the **class label** attribute. The class label attribute is discrete-valued and unordered (i.e. **nominal**).

f can be ***classification rules***, ***decision trees*** or ***mathematical formulae***

In the context of classification, data tuples can be referred to as:
training tuples, samples, examples, instances, data points, or objects.

Basic Concepts

Estimating the predictive accuracy of the classifier.

If the **training set** is used to measure the classifier's accuracy

⇒ the estimate would be **optimistic**, because the classifier tends to overfit the data

Therefore, a test set is used, made up of test tuples and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier.

The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier.

Classification methods

- Bayes Classification Methods
- Decision Tree Induction
- Neural Networks
- Support Vector Machines
- ...

Bayes Classification Methods

Bayesian classifiers are statistical classifiers that can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.

They assume **conditional independence** between attributes

It results in a set of **classification rules**:

age(X, "youth") AND income(X, "high") ⟶ *class(X, "A")*
age(X, "youth") AND income(X, "low") ⟶ *class(X, "B")*
age(X, "middle_aged") ⟶ *class(X, "C")*
age(X, "senior") ⟶ *class(X, "C")*

Bayes Classification Methods

Bayesian classifiers are statistical classifiers that can predict class membership probabilities such as the probability that a given tuple belongs to a particular class.

They assume **conditional independence** between attributes

X is a data tuple described by *n* attributes

H is the hypothesis i that X belongs to a particular class

P(H) = probability to belong to H

P(X) = probability to have those *n* attributes

P(H|X) = probability to belong to H if *X*

P(X|H) = probability to be X if H

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}.$$

Bayes Classification Methods

Naïve Bayesian Classification

Maximize P(H|X)

As P(X) is constant for all classes, only P(X|H) P(H) needs to be maximized

For data with many attributes, we assume the class conditional independence

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$
$$= P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i).$$

Bayes Classification Methods

Naïve Bayesian Classification

To which class belongs?

$X = (age = youth, income = medium, student = yes, credit\_rating = fair)$

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

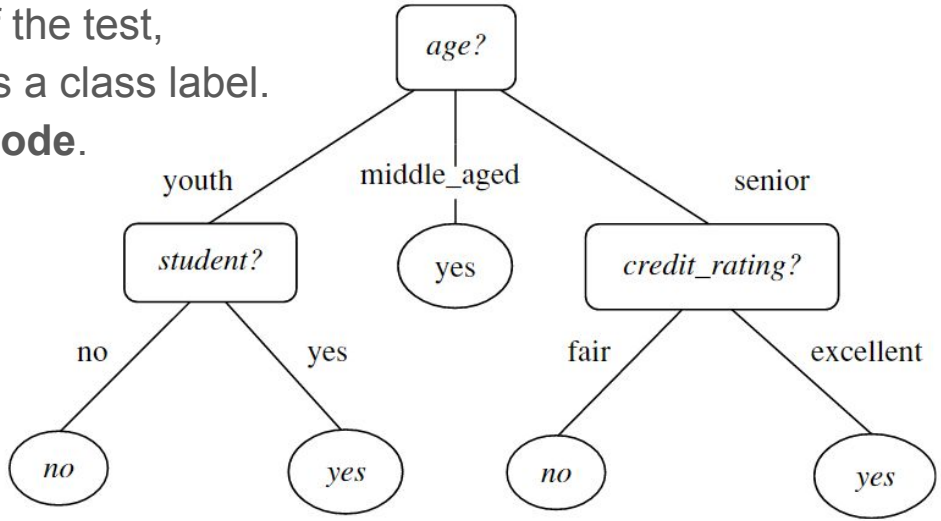
Decision Tree Induction

A **decision tree** is a flowchart-like tree structure, where :

- each **internal node** (non-leaf node) denotes a test on an attribute,
- each **branch** represents an outcome of the test,
- each **leaf node** (or terminal node) holds a class label.
- the topmost node in a tree is the **root node**.

This example predicts wheter a customer would

buy a computer



Decision Tree Induction

History

During the late 1970s and early 1980s, J. Ross Quinlan, a researcher in machine learning, developed a decision tree algorithm known as **ID3** (Iterative Dichotomiser). This work expanded on earlier work on concept learning systems, described by E. B. Hunt, J. Marin, and P. T. Stone.

Quinlan later presented **C4.5** (a successor of ID3), which became a benchmark to which newer supervised learning algorithms are often compared.

In 1984, a group of statisticians (L. Breiman, J. Friedman, R. Olshen, and C. Stone) published the book Classification and Regression Trees (**CART**), which described the generation of binary decision trees.

ID3 and CART were invented independently of one another at around the same time, yet follow a similar approach for learning decision trees from training tuples. These two cornerstone algorithms spawned a flurry of work on decision tree induction.

Decision Tree Induction

ID3, C4.5, and CART are greedy (i.e., nonbacktracking) approaches

Decision trees are constructed in a top-down recursive divide-and-conquer manner.

Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels.

The training set is recursively partitioned into smaller subsets as the tree is being built.

Decision Tree Induction

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, *D*.

Input:

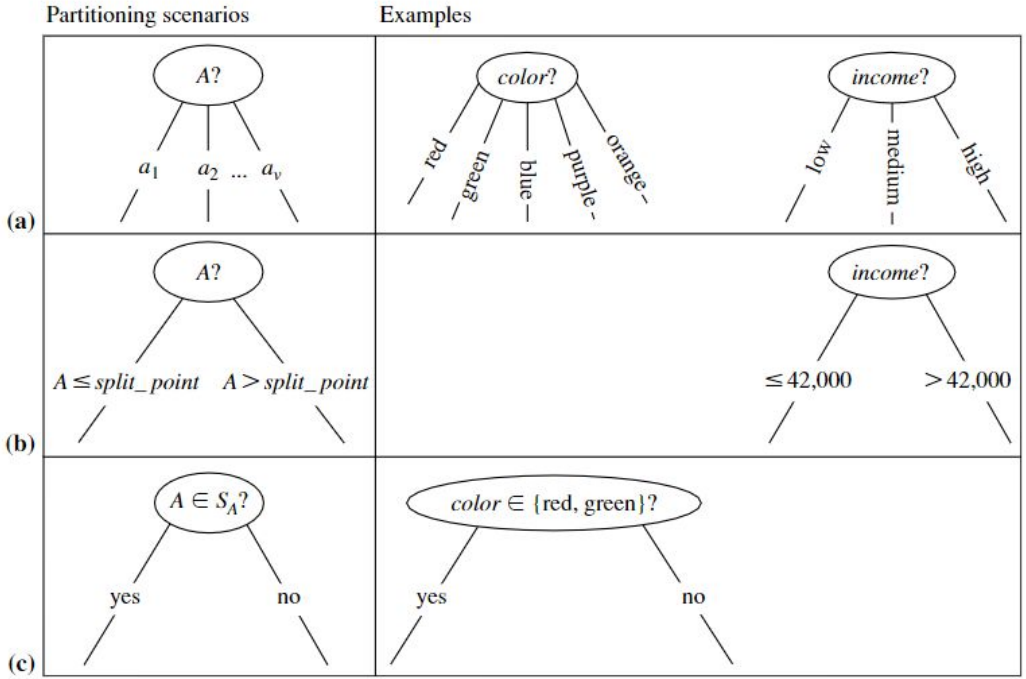
- Data partition, *D*, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

```
(1)  create a node N;
(2)  if tuples in D are all of the same class, C, then
(3)      return N as a leaf node labeled with the class C;
(4)  if attribute_list is empty then
(5)      return N as a leaf node labeled with the majority class in D; // majority voting
(6)  apply Attribute_selection_method(D, attribute_list) to find the "best" splitting_criterion;
(7)  label node N with splitting_criterion;
(8)  if splitting_attribute is discrete-valued and
         multiway splits allowed then // not restricted to binary trees
(9)      attribute_list ← attribute_list − splitting_attribute; // remove splitting_attribute
(10) for each outcome j of splitting_criterion
     // partition the tuples and grow subtrees for each partition
(11)     let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj is empty then
(13)         attach a leaf labeled with the majority class in D to node N;
(14)     else attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

Decision Tree Induction



Decision Tree Induction

Attribute selection measure

The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the splitting attribute for the given tuples.

Examples: Information Gain (ID3), Gain ratio, Gini Index

Decision Tree Induction

Information Gain

It minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or “impurity” in these partitions.

1. Compute the expected information needed to classify *D* into the *m* classes: $Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$
   a. Info(D) is the entropy of D
   b. p_i is the probability that a tuple from D belongs to class C_i : $|C_{i,D}|/|D|$.

2. For each attribute A observed in D:
   a. Compute the information needed after partitioning D with A: $Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$
      i. v is the number of distinct values in A
   b. Compute the gain in information: $Gain(A) = Info(D) - Info_A(D).$

3. Select the attribute A with highest information gain

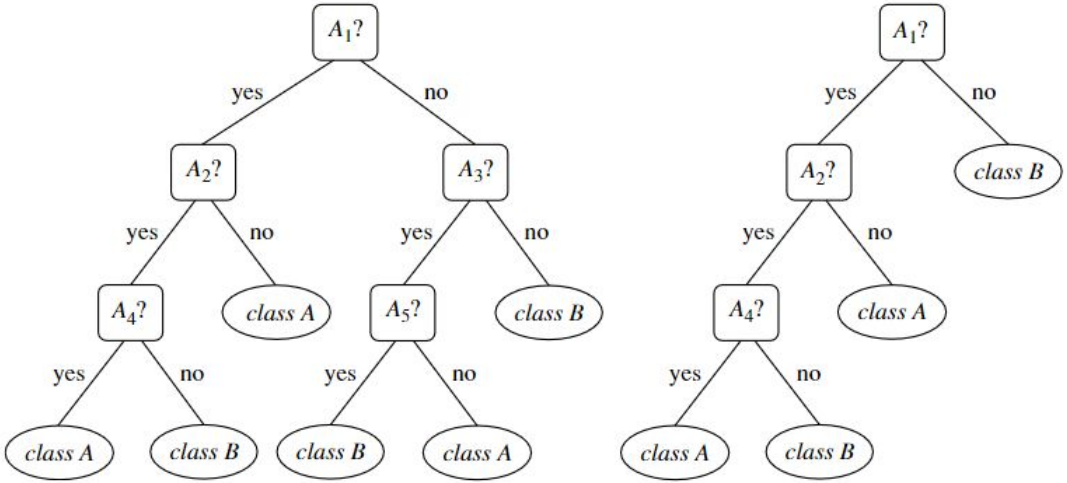
Decision Tree Induction

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

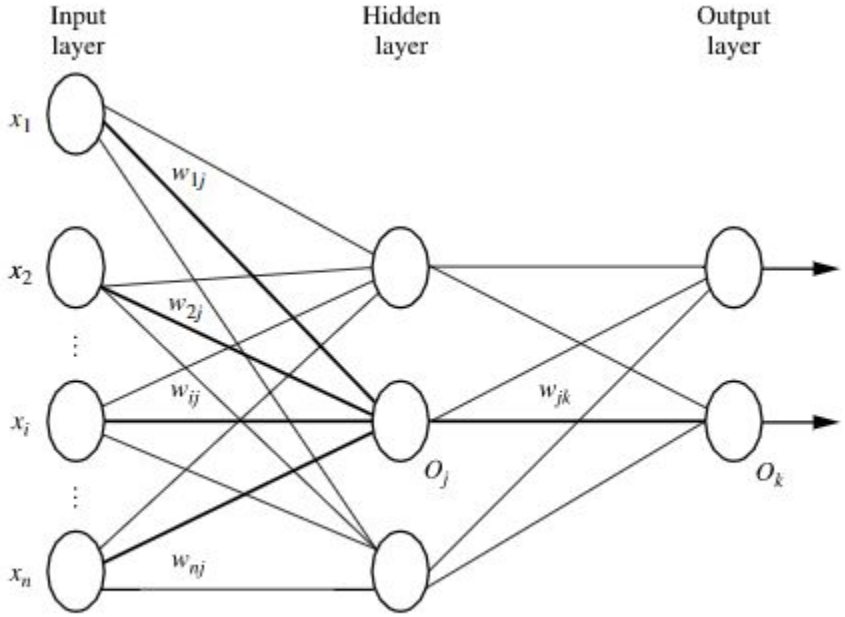
Decision Tree Induction

Tree Pruning



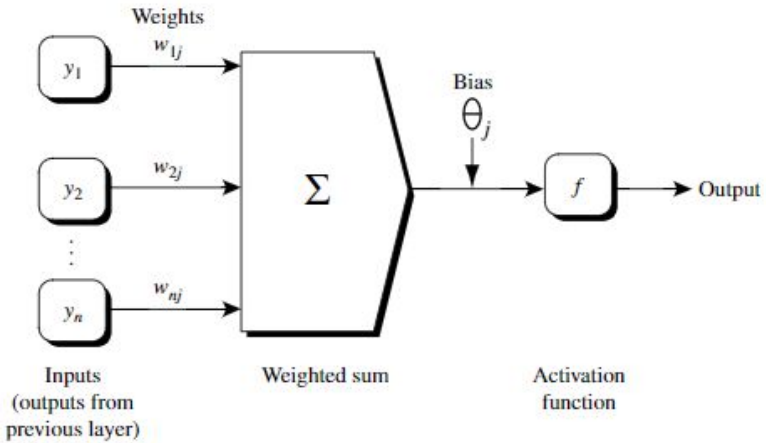
An unpruned decision tree and a pruned version of it.

Backpropagation (Neural Networks)



Multilayer feed-forward neural network.

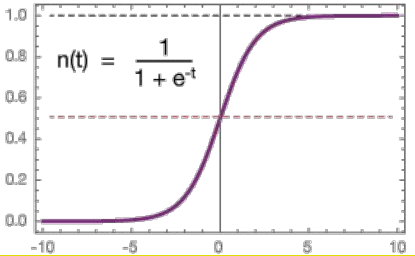
Backpropagation (Neural Networks)



$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

Hidden or output layer unit j: The inputs to unit j are outputs from the previous layer. These are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit j. A nonlinear activation function is applied to the net input. (For ease of explanation, the inputs to unit j are labeled $y_1, y_2, \ldots, y_n$. If unit j were in the first hidden layer, then these inputs would correspond to the input tuple $(x_1, x_2, \ldots, x_n)$.)



Backpropagation (Neural Networks)

Algorithm: Backpropagation. Neural network learning for classification or numeric prediction, using the backpropagation algorithm.

Input:

- D, a data set consisting of the training tuples and their associated target values;
- l, the learning rate;
- *network*, a multilayer feed-forward network.

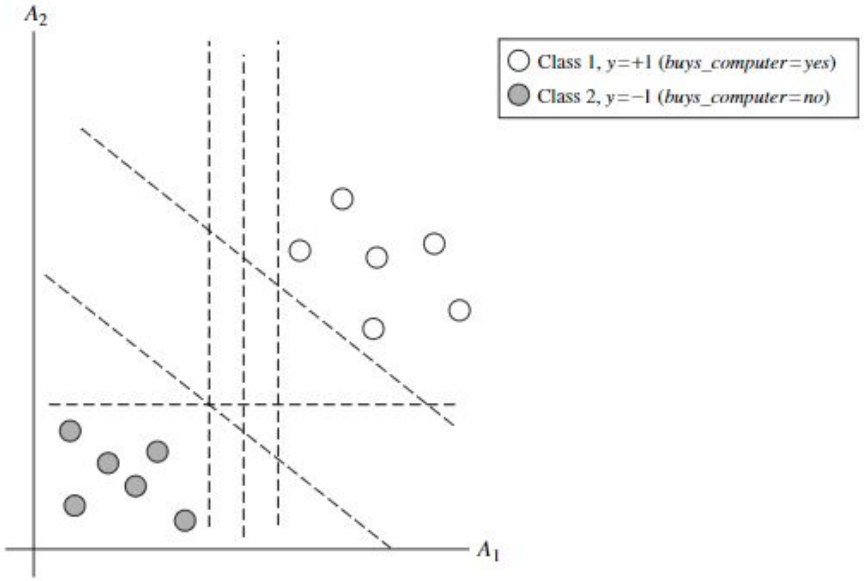
Output: A trained neural network.

Method:

(1) Initialize all weights and biases in *network*;
(2) **while** terminating condition is not satisfied {
(3) **for** each training tuple $\boldsymbol{X}$ in D {
(4) // Propagate the inputs forward:
(5) **for** each input layer unit j {
(6) $O_j = I_j$; // output of an input unit is its actual input value
(7) **for** each hidden or output layer unit j {
(8) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
(9) $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
(10) // Backpropagate the errors:
(11) **for** each unit j in the output layer
(12) $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
(13) **for** each unit j in the hidden layers, from the last to the first hidden layer
(14) $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
(15) **for** each weight w_{ij} in *network* {
(16) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
(17) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
(18) **for** each bias θ_j in *network* {
(19) $\Delta \theta_j = (l) Err_j$; // bias increment
(20) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
(21) } }

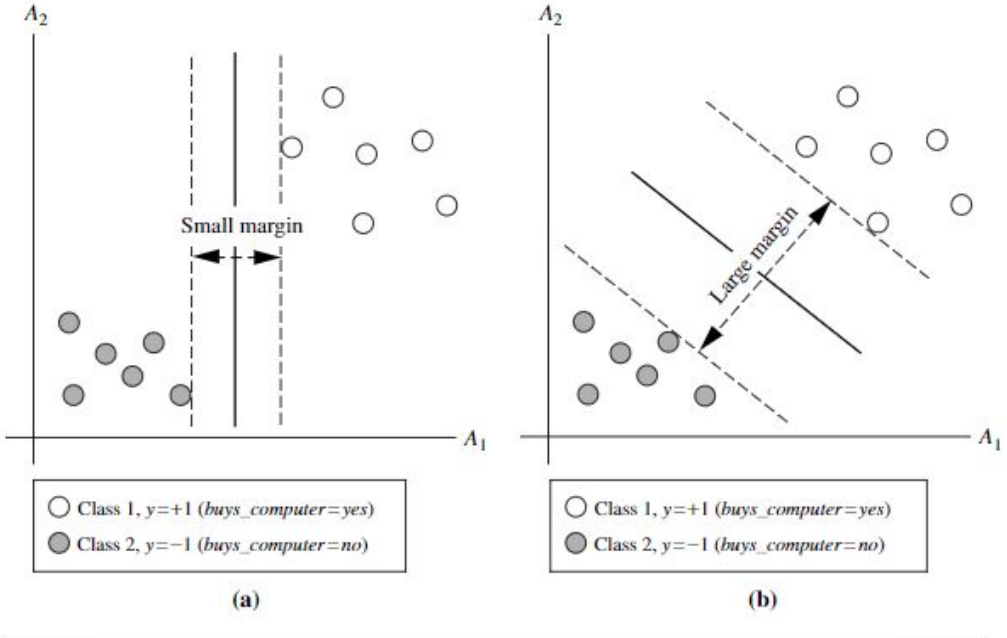
Backpropagation algorithm.

Support Vector Machines



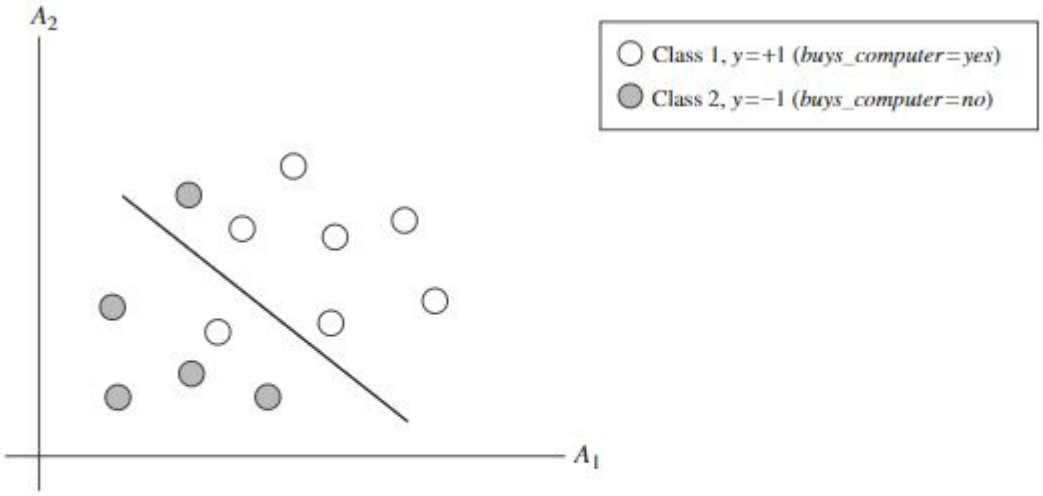
The 2-D training data are linearly separable. There are an infinite number of possible separating hyperplanes or "decision boundaries," some of which are shown here as dashed lines. Which one is best?

Support Vector Machines



Here we see just two possible separating hyperplanes and their associated margins. Which one is better? The one with the larger margin (b) should have greater generalization accuracy.

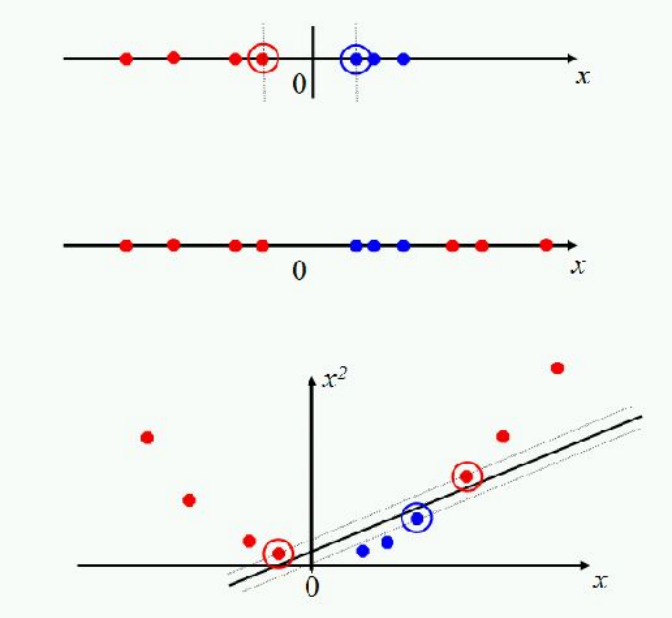
Support Vector Machines



A simple 2-D case showing linearly inseparable data. Unlike the linear separable data of Figure 9.7, here it is not possible to draw a straight line to separate the classes. Instead, the decision boundary is nonlinear.

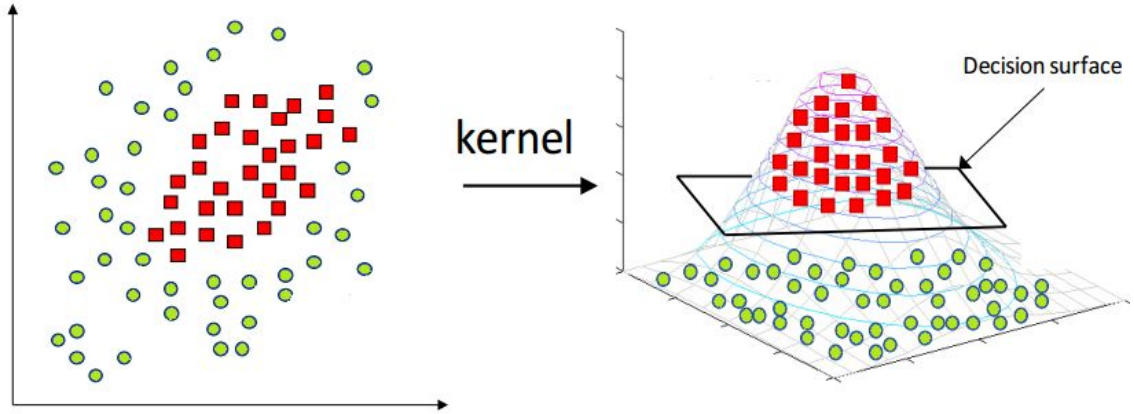
Support Vector Machines

The Case When the Data Are **Linearly Inseparable**



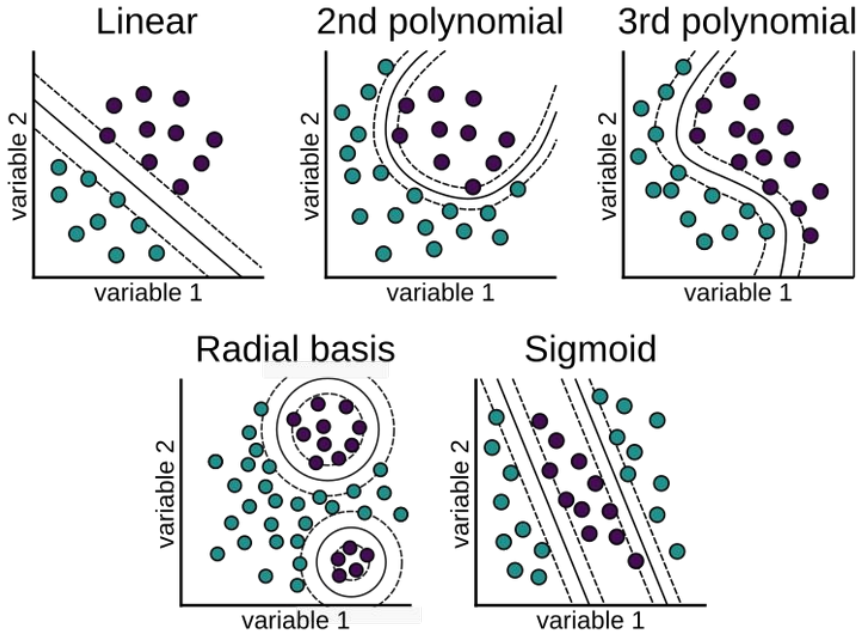
Support Vector Machines

Radial kernel

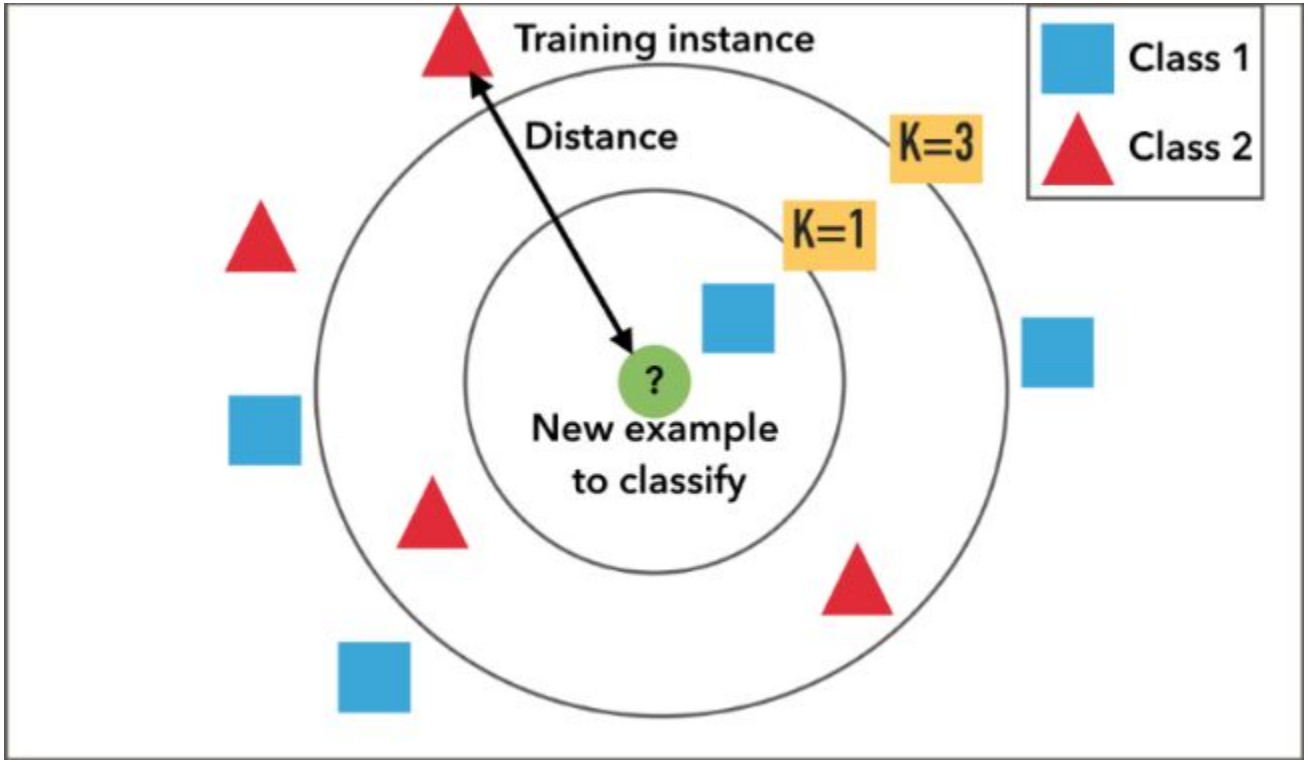


Support Vector Machines

Kernels



k-Nearest-Neighbor



Model Evaluation - Some measures

- **True positives** (*TP*): These refer to the positive tuples that were correctly labeled by the classifier. Let *TP* be the number of true positives.
- **True negatives** (*TN*): These are the negative tuples that were correctly labeled by the classifier. Let *TN* be the number of true negatives.
- **False positives** (*FP*): These are the negative tuples that were incorrectly labeled as positive (e.g., tuples of class *buys_computer* = *no* for which the classifier predicted *buys_computer* = *yes*). Let *FP* be the number of false positives.
- **False negatives** (*FN*): These are the positive tuples that were mislabeled as negative (e.g., tuples of class *buys_computer* = *yes* for which the classifier predicted *buys_computer* = *no*). Let *FN* be the number of false negatives.

Measure	*Formula*
accuracy, recognition rate	$\frac{TP+TN}{P+N}$
error rate, misclassification rate	$\frac{FP+FN}{P+N}$
sensitivity, true positive rate, recall	$\frac{TP}{P}$
specificity, true negative rate	$\frac{TN}{N}$
precision	$\frac{TP}{TP+FP}$

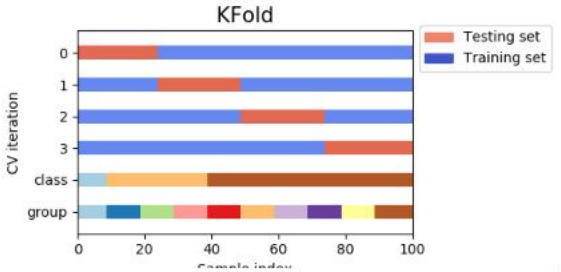
Precision = what percentage of tuples labeled as positive are actually positive

Model Evaluation - Additional Factors

- Speed
- Robustness
- Scalability
- Interpretability

Model Evaluation - Methods

- Holdout :
  - Divide the database into 2 random subsets (train set / test set) with typically a ⅔ rate
  - Pessimist estimation of accuracy (because training on a single subset)
- Random subsampling :
  - Repeat holdout *k* times
- Cross-validation
  - K-fold (typically k = 10)
  - Leave-one-out (K-fold where k = n)
  - Stratified k-fold (preserved proportion of class)
- Bootstrap method
  - Sampling with replacement
  - Best for small data sets

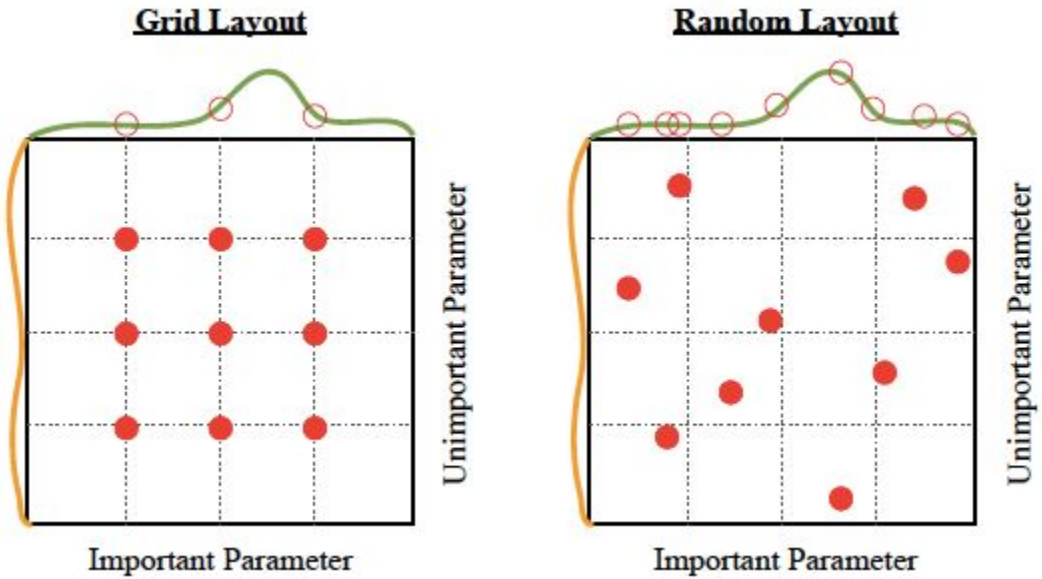


Hyperparameter optimization

Hyperparameter optimization or **tuning** is the problem of choosing a set of optimal hyperparameters for a learning algorithm.

- Grid-search
- Random-search
- Bayesian optimization
- Gradient-based optimization
- Evolutionary optimization

Hyperparameter optimization



Clustering

Requirements for Cluster Analysis

- Scalability
- Ability to deal with different types of attributes
- Discovery of clusters with arbitrary shape
- Requirements for domain knowledge to determine input parameters
- Ability to deal with noisy data
- Incremental clustering and insensitivity to input order
- Capability of clustering high-dimensionality data
- Constraint-based clustering
- Interpretability and usability

Method	General Characteristics
Partitioning methods	– Find mutually exclusive clusters of spherical shape – Distance-based – May use mean or medoid (etc.) to represent cluster center – Effective for small- to medium-size data sets
Hierarchical methods	– Clustering is a hierarchical decomposition (i.e., multiple levels) – Cannot correct erroneous merges or splits – May incorporate other techniques like microclustering or consider object "linkages"
Density-based methods	– Can find arbitrarily shaped clusters – Clusters are dense regions of objects in space that are separated by low-density regions – Cluster density: Each point must have a minimum number of points within its "neighborhood" – May filter out outliers
Grid-based methods	– Use a multiresolution grid data structure – Fast processing time (typically independent of the number of data objects, yet dependent on grid size)

Partitioning Methods

The simplest and most fundamental version of cluster analysis is partitioning, which organizes the objects of a set into several **exclusive** groups or clusters.

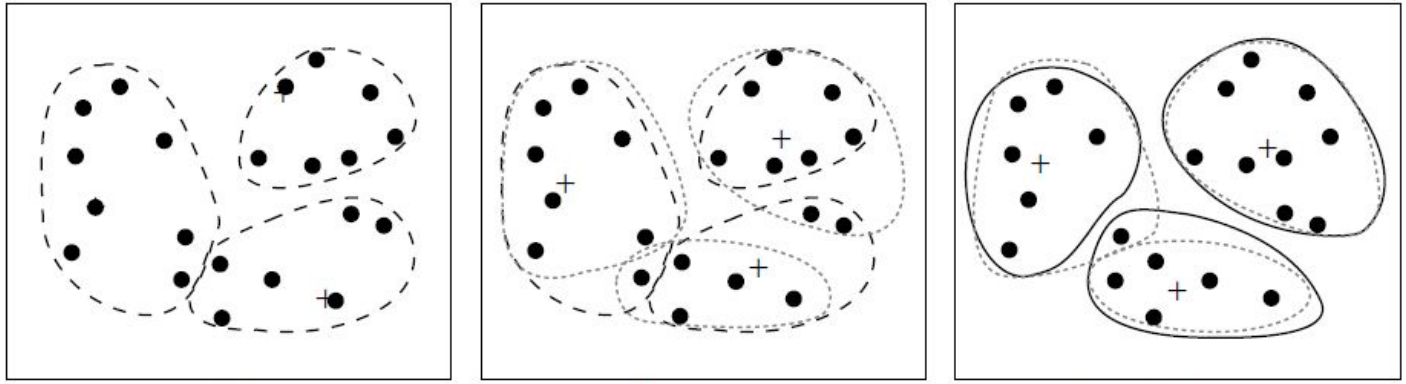
The clusters are formed to optimize an objective partitioning criterion, such as a dissimilarity function based on distance.

The number of cluster should be known.

Examples:

- k-means
- k-modes
- k-medoids

K-Means



(a) Initial clustering (b) Iterate (c) Final clustering

Clustering of a set of objects using the *k*-means method; for (b) update cluster centers and reassign objects accordingly (the mean of each cluster is marked by a +).

K-Means

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
(5) **until** no change;

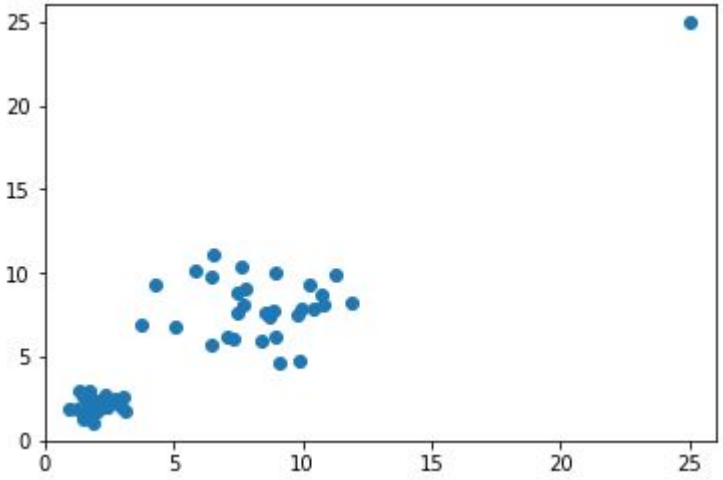
K-Means

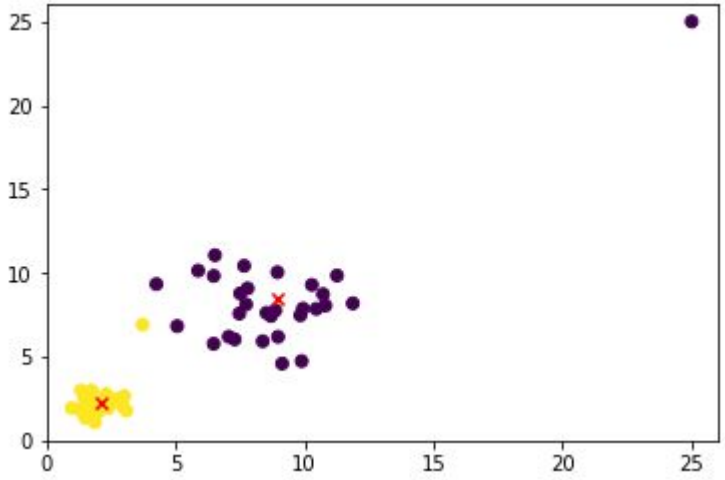
Limitations:

- The k-means method is not suitable for discovering clusters with non-convex shapes or clusters of very different size.
- Moreover, it is sensitive to noise and outlier data points because a small number of such data can substantially influence the mean value.

K-Means

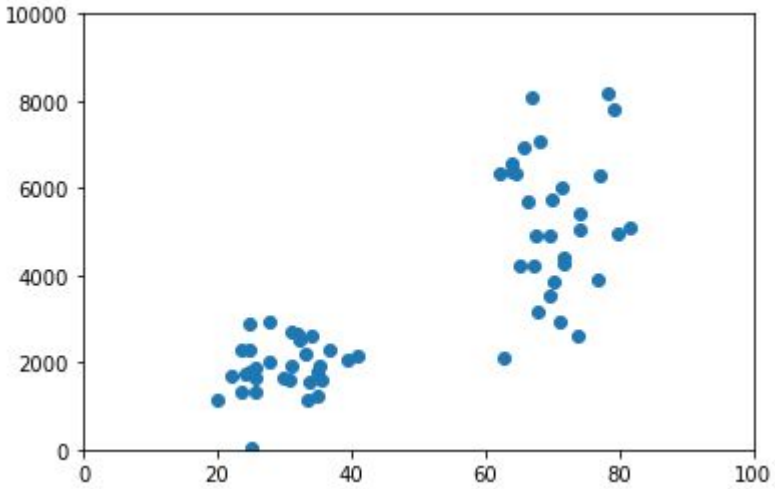
Limitations : K-Means is noise/outlier sensitive

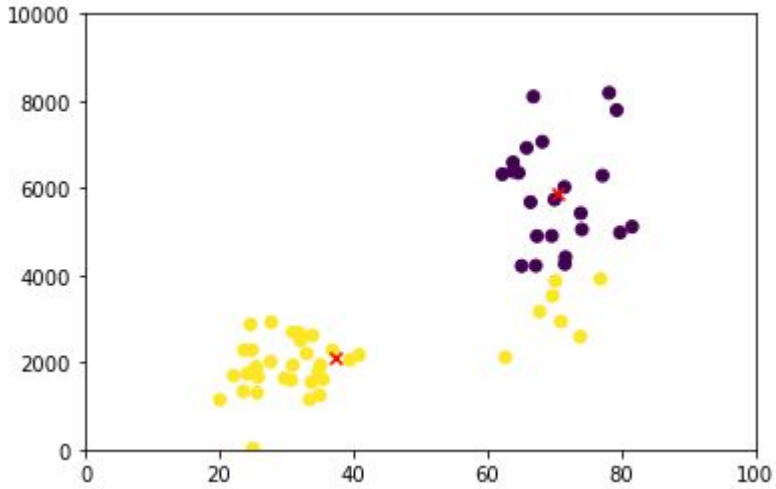




K-Means

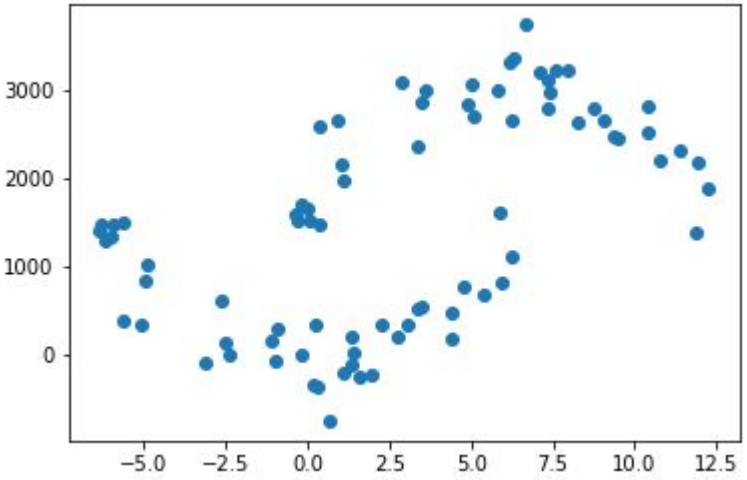
Limitations : All clustering analysis requires homogeneous scales

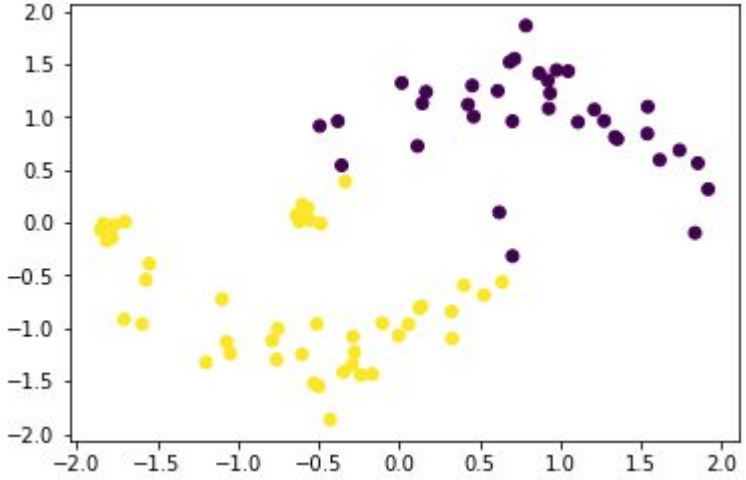




K-Means

Limitations : Non-convex clusters





K-Modes

The k-modes method is a variant of k-means, which extends the k-means paradigm to cluster **nominal data** by replacing the means of clusters with modes.

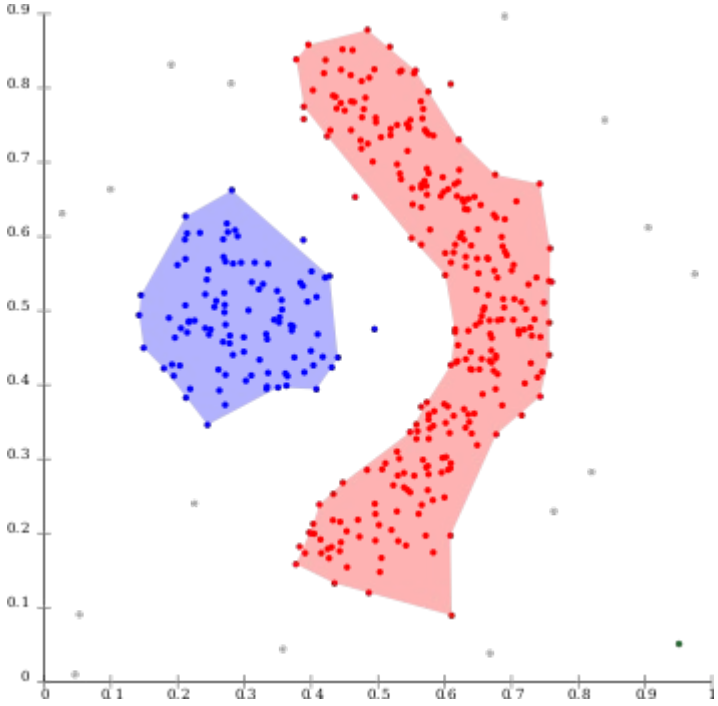
It uses new dissimilarity measures to deal with nominal objects and a frequency-based method to update modes of clusters. The k-means and the k-modes methods can be integrated to cluster data with mixed numeric and nominal values.

K-Medoids

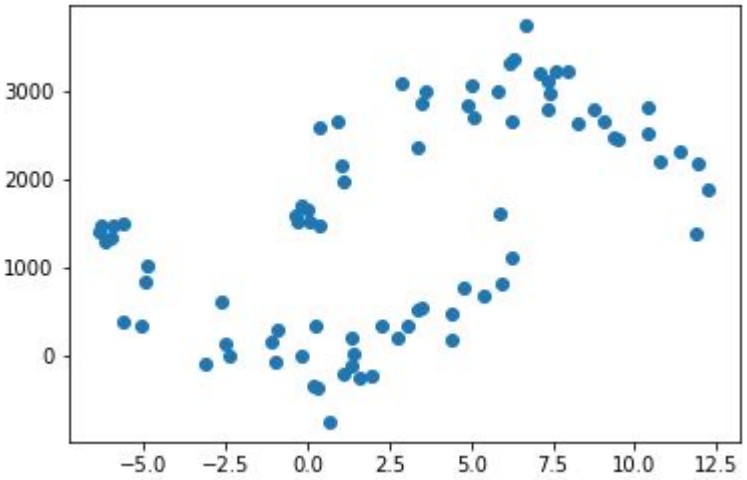
“How can we modify the k-means algorithm to diminish such sensitivity to outliers?”

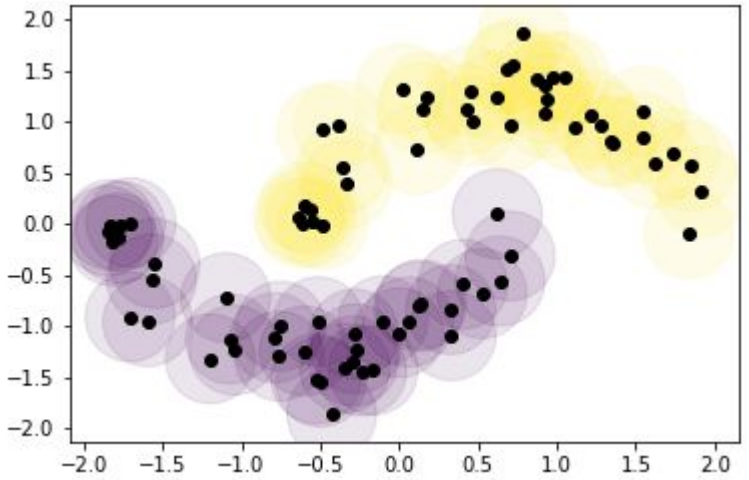
Instead of taking the mean value of the objects in a cluster as a reference point, we can pick **actual objects to represent the clusters**, using one representative object per cluster.

DB-Scan



DB-SCAN


3000
2000
1000
0
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5


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

Association Analysis

Basic Concepts

Frequent patterns are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set.

A **frequent itemset** a set of items, such as milk and bread, that appear frequently together in a transaction data set.

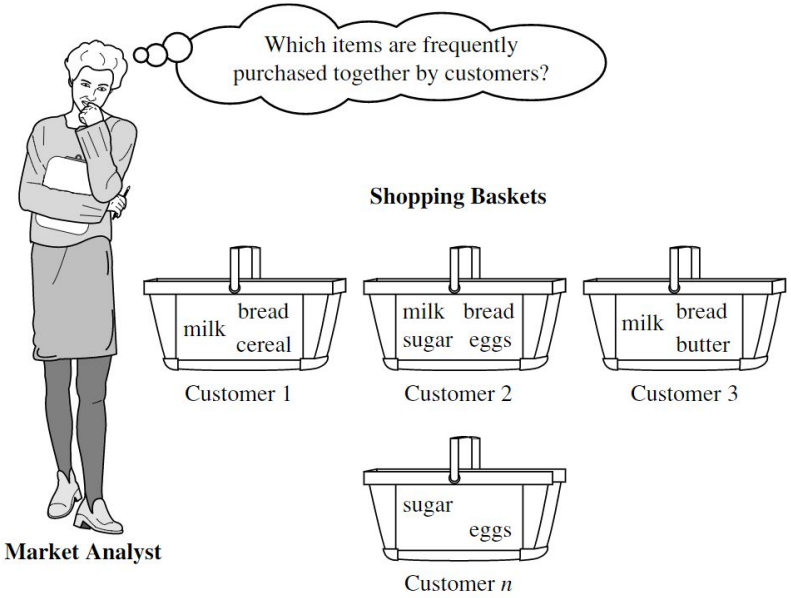
A (frequent) sequential pattern is A subsequence that occurs frequently in a shopping history database such as buying first a PC, then a digital camera, and then a memory card.

A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences.

If a substructure occurs frequently, it is called a (frequent) structured pattern.

Basic Concepts

Market Basket Analysis: A Motivating Example



$$computer \Rightarrow antivirus\_software\ [support = 2\%, confidence = 60\%].$$

Basic Concepts

Frequent Itemsets, Closed Itemsets, and Association Rules

The rule A=>B holds in the transaction set *D* with supports s, where s is the percentage of transactions in D that contain A U B.

The rule A=>B has confidence c in the transaction set D, where c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability ,P(B|A).

$$support(A \Rightarrow B) = P(A \cup B)$$

$$confidence(A \Rightarrow B) = P(B|A).$$

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

Basic Concepts

Frequent Itemsets, Closed Itemsets, and Association Rules

An **itemset** that contains k items is a k-itemset

A **frequent itemset** is an itemset that satisfies a prespecified minimum support (***min_sup***) threshold.

The set of frequent k-itemsets is commonly denoted by L_k.

Basic Concepts

Frequent Itemsets, Closed Itemsets, and Association Rules

Rules that satisfy both a minimum support threshold (**minsup**) and a minimum confidence threshold (**minconf**) are called **strong**. By convention, we write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0.

Basic Concepts

Frequent Itemsets, Closed Itemsets, and Association Rules

Equation (6.4) shows that the confidence of rule $A \Rightarrow B$ can be easily derived from the support counts of A and $A \cup B$. That is, once the support counts of A, B, and $A \cup B$ are found, it is straightforward to derive the corresponding association rules $A \Rightarrow B$ and $B \Rightarrow A$ and check whether they are strong. Thus, the problem of mining association rules can be reduced to that of mining frequent itemsets.

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

Basic Concepts

Frequent Itemsets, Closed Itemsets, and Association Rules

1. **Find all frequent itemsets:** By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, *min_sup*.
2. **Generate strong association rules from the frequent itemsets:** By definition, these rules must satisfy minimum support and minimum confidence.

(2) is much less costly than (1) ⇒ the overall performance is determining by (1)

Why ? If an itemset is frequent, then each of is subsets is frequent as well

Example: if {a,b,c, d} is frequent then {a,b},{a,c},{a,d},{b,c},{b,d},{c,d},{a,b,c},{a,c,d}... are also frequent

A frequent itemset of length 100 contains $\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}$.

Frequent Itemset Mining Methods

Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules.

The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties

Frequent Itemset Mining Methods

Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

Apriori employs an iterative approach known as a level-wise search, where **k-itemsets are used to explore (k+1)-itemsets**.

First, the set L_1 of frequent 1-itemsets is found by scanning the database to accumulate the count for each item and collecting those items that satisfy minimum support.

Next, L_1 is used to find L_2, the set of frequent 2-itemsets, which is used to find L_3, and so on, until no more frequent k-itemsets can be found.

The finding of each L_k requires one full scan of the database.

Frequent Itemset Mining Methods

Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property is used to reduce the search space.

Apriori property: All nonempty subsets of a frequent itemset must also be frequent.

Frequent Itemset Mining Methods

Apriori Algorithm: Finding Frequent Itemsets by Confined Candidate Generation

"How is the Apriori property used in the algorithm?"

Consider how L_{k-1} is used to find L_k for k>=2

1. The join step:
   a. Possible candidates (C_k) for L_k is generated by joining L_{k-1} with itself (why?)
   b. Elements in L_{k-1} needs to be ordered
   c. Join is done if last elements are different (in order to avoid duplicates)
2. The prune step:
   a. Keep an element of C_k only if all k-1 itemsets are in L_{k-1}
   b. Count support only for L_k elements

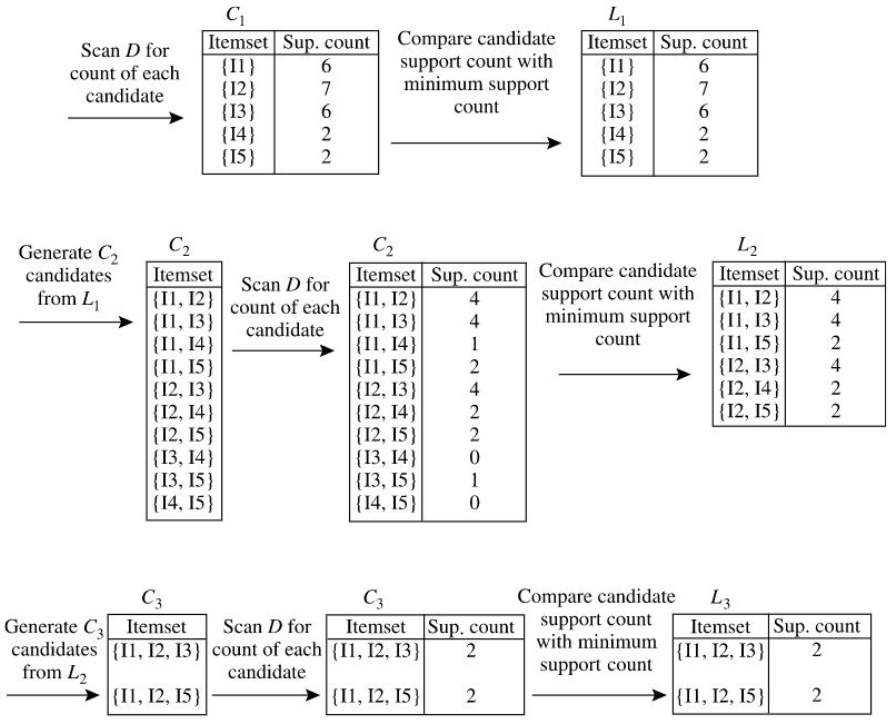
Generating Association Rules from Frequent Itemsets

- For each frequent itemset l, generate all nonempty subsets of l.
- For every nonempty subset s of l, output the rule "$s \Rightarrow (l - s)$" if $\frac{support\_count(l)}{support\_count(s)} \geq$ *min_conf*, where *min_conf* is the minimum confidence threshold.

Frequent Itemset Mining Methods

min_sup= 2

List of item_IDs
I1, I2, I5
I2, I4
I2, I3
I1, I2, I4
I1, I3
I2, I3
I1, I3
I1, I2, I3, I5
I1, I2, I3



Chapter 3 – Implementation

Methodology for the implementation of an AI solution

The different actors:

- Business experts (sector specialists)
- Developers and software architects
- AI specialists: AI expert, data scientists, data analysts, ...
- Governance actors: GDPR officer, Corporate Social Responsibility (in some cases)
- …

Methodology for the implementation of an AI solution

Some important elements (among others):

- Definition of business case: What? For whom? For what? What budget?
- Assessment of the impact: ethical, social, personal safety
- Data governance: what data, accessible by whom, how, what rights of use?
- Solution design: what architecture? What level of data security?
- Industrialization of the solution: monitoring of results, user management, communication, change management...

Methodology for the implementation of an AI solution

Goal: collect as much data as possible

- directly or indirectly related to issue

⇨ Guarantee of project success

- At this step, we don't care if we are going to use all of this
- Use of many techniques to collect data (IoT, API, DB,...)

Methodology for the implementation of an AI solution

Goal: verify the data quality

- Missing data?
- Outliers?
- …

⇨ Need to deal with that!

- Solutions:
  - Replace missing data by statistics or use proxy variables
  - Remove it
  - Keep it and use techniques to deal with outliers

- Data preprocessing is a very important step! So let's focusing a bit more on this...

Data Preprocessing: An Overview

Real-world databases are highly susceptible to **noisy, missing, and inconsistent data.**

This is typically due to their **huge size** and their likely origin from **multiple, heterogenous sources**.

Low-quality data will lead to low-quality mining results and poor ML models.

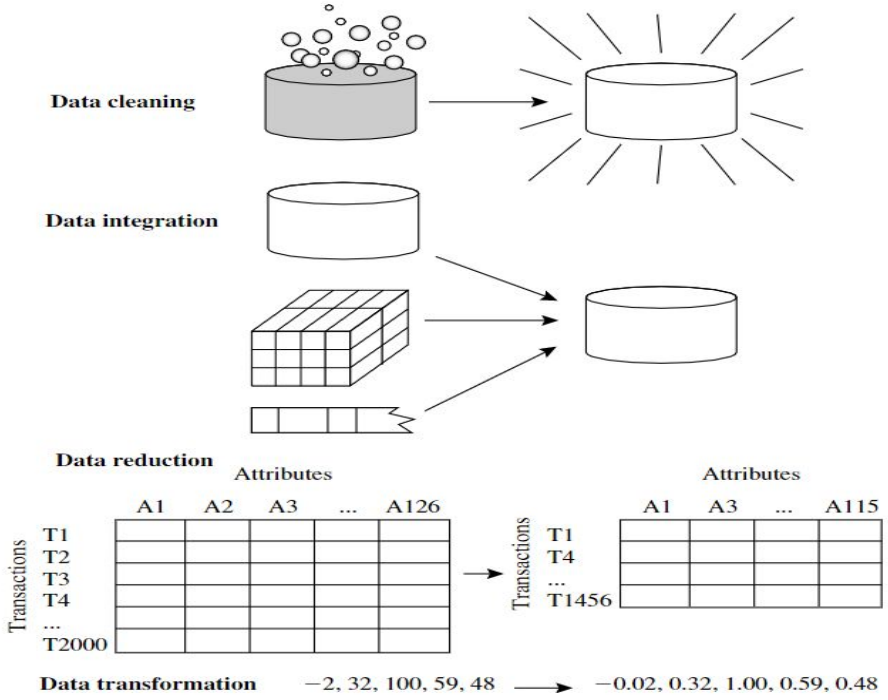
- **Data cleaning** can be applied to remove noise and correct inconsistencies in data.
- **Data integration** merges data from multiple sources into a coherent data store (i.e. data warehouse).
- **Data reduction** can reduce data size by aggregating, eliminating redundant features, clustering...
- **Data transformations** (e.g., normalization) may be applied. This can improve the accuracy and efficiency of mining algorithms involving distance measurements.

Data Preprocessing: An Overview

Data quality factors:

- Accuracy
- Completeness
- Consistency
- Timeliness
- Believability
- Interpretability

Data Preprocessing: An Overview



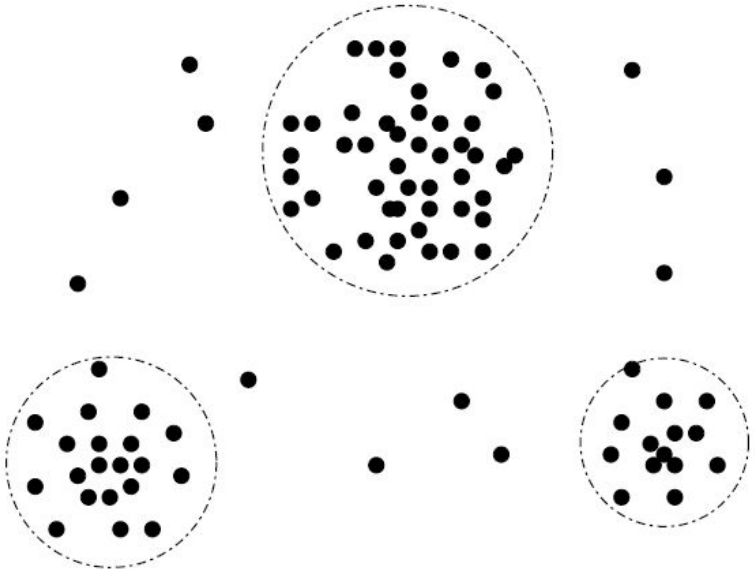
Data Preprocessing: An Overview

Missing values

- Ignore the tuple
  - Not very effective unless the tuple contains several missing attributes or the class label is missing
- Fill in the missing value manually
  - Time consuming
- Use a global constant to fill in the missing value
  - Such as “unknown”, but can lead to misinterpretations
- Use a measure of central tendency for the attribute (e.g., the mean or median) to fill in the missing value
  - The mean for normal (symmetric) data distributions, the median for skewed data distributions
- Use the attribute mean or median for all samples belonging to the same class as the given tuple
- Use the most probable value to fill in the missing value
  - By using the other customer attributes in your data set, you may construct a decision tree to predict the missing values for income.

Data Preprocessing: An Overview

Outliers



A 2-D customer data plot with respect to customer locations in a city, showing three data clusters. Outliers may be detected as values that fall outside of the cluster sets.

Data Preprocessing: An Overview

Data Reduction

Complex data analysis and mining on huge amounts of data can take a long time, making such analysis **impractical or infeasible**.

Data reduction techniques can be applied to **obtain a reduced representation** of the data set that is much smaller in volume, yet closely maintains the integrity of the original data.

- **Dimensionality reduction :** is the process of reducing the number of random variables or attributes under consideration (wavelet transforms, **principal component analysis**, attribute subset selection)

Methodology for the implementation of an AI solution

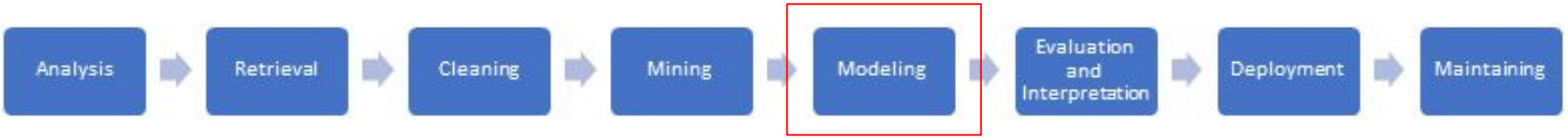
Goal: better understanding data

How?

- working with business experts and data analysts
- some visualisations and data manipulations
- using current BI systems

⇨ Verify our intuitions and our assumptions
⇨ Guide us in the following steps

Methodology for the implementation of an AI solution



Goal: create a Machine Learning (ML) model which handles the issue

How?

- training set: train the model with a part of data collected earlier

  ⇨ the algorithm will improve its predictions as it is learned
  ⇨ a relationship into the data is identified

- a validation step is implemented to figure out the best model

Methodology for the implementation of an AI solution

Goal: verify the reliability of model's predictions

How?

- Test the model on new testing set (which have not been used for the train)

  ⇨ the results obtained will determine the model's power of generalization
  ⇨ a "good" model has to minimize the number of errors

Methodology for the implementation of an AI solution

Deployment step:

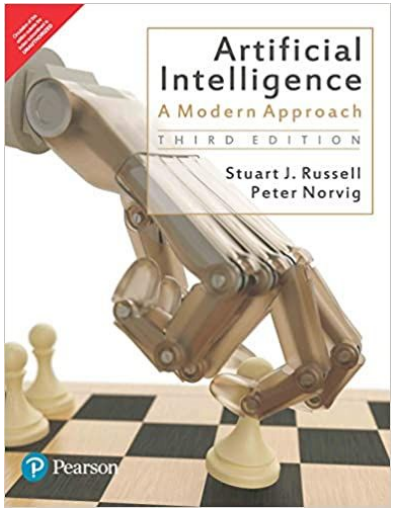
- Deploy the system and ensure that the objectives of the AI solution are met

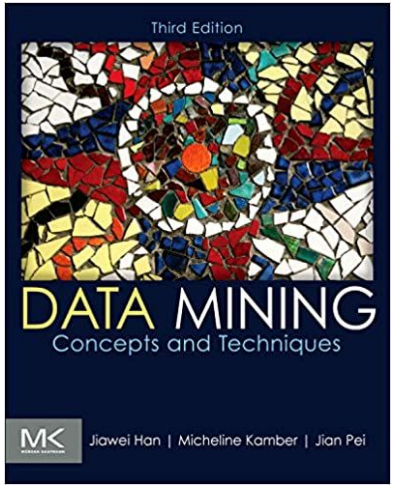
Maintaining step:

- Required to ensure the medium - to long-term relevance of our model
  - retrieval new data
  - cleaning
  - readapt parameters of the model if required
  - ...

Examples with Python

Resources





scikits learn
machine learning in Python

kaggle