Designing Algorithms for Machine Learning and Data Mining

Antoine Cornuéjols and Christel Vrain

Abstract Designing Machine Learning algorithms implies to answer three main questions: First, what is the space $\mathscr{H}$ of hypotheses or models of the data that the algorithm considers? Second, what is the inductive criterion used to assess the merit of a hypothesis given the data? Third, given the space $\mathscr{H}$ and the inductive criterion, how is the exploration of $\mathscr{H}$ carried on in order to find a as good as possible hypothesis? Any learning algorithm can be analyzed along these three questions. This chapter focusses primarily on unsupervised learning, on one hand, and supervised learning, on the other hand. For each, the foremost problems are described as well as the main existing approaches. In particular, the interplay between the structure that can be endowed over the hypothesis space and the optimisation techniques that can in consequence be used is underlined. We cover especially the major existing methods for clustering: prototype-based, generative-based, density-based, spectral based, hierarchical, and conceptual and visit the validation techniques available. For supervised learning, the generative and discriminative approaches are contrasted and a wide variety of linear methods in which we include the Support Vector Machines and Boosting are presented. Multi-Layer neural networks and deep learning methods are discussed. Some additional methods are illustrated, and we describe other learning problems including semi-supervised learning, active learning, online learning, transfer learning, learning to rank, learning recommendations, and identifying causal relationships. We conclude this survey by suggesting new directions for research.

1 Introduction

Machine Learning is the science of, on one hand, discovering the fundamental laws that govern the act of learning and, on the other hand, designing machines that learn from experiences, in the same way as physics is both the science of uncovering the

A. Cornuéjols (✉)
UMR MIA-Paris, AgroParisTech, INRA, Université Paris-Saclay, 75005 Paris, France
e-mail: antoine.cornuejols@agroparistech.fr

C. Vrain
LIFO, EA 4022, University of Orléans, 45067 Orleans, France
e-mail: christel.vrain@univ-orleans.fr

© Springer Nature Switzerland AG 2020

P. Marquis et al. (eds.), *A Guided Tour of Artificial Intelligence Research*,
https://doi.org/10.1007/978-3-030-06167-8_12

laws of the universe and of providing knowledge to make, in a very broad sense, machines. Of course, "understanding" and "making" are tightly intertwined, in that a progress in one aspect generally benefits to the other aspect. But a Machine Learning scientist can feel more comfortable and more interested in one end of the spectrum that goes from '*theorize* Machine Learning' to '*making* Machine Learning'.

Machine Learning is tied to data science, because it is fundamentally the science of induction, that tries to uncover general laws and relationships from some set of data. However, it is interested as much in understanding how it is possible to use very few examples, like when you learnt how to avoid a "fork" in chess from one experience only, as how to make sense of large amount of data. Thus, "big data" is not synonymous with Machine Learning. In this chapter, we choose not to dwell upon the problems and techniques associated with gathering data and realize all the necessary preprocessing phases. We will mostly assume that this has been done in such a way that looking for patterns in the data will not be too compromised by imperfections of the data at hand. Of course, any practitioner of Machine Learning will know that this is a huge assumption and that the corresponding work is of paramount importance.

Before looking at what can be a science of designing learning algorithms, it is interesting to consider basic classical Machine Learning scenarios.

2 Classical Scenarios for Machine Learning

A learning scenario is defined by the exchanges between the learner and its environment. Usually, this goes hand in hand with the target task given to the system.

In **supervised learning**, the learner receives a set of examples $\mathscr{S} = \{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq m}$ from the environment, each composed of a set of *explanatory* or *input variables* $\mathbf{x}_i$ and of *output variable(s)* y_i, of which the value must be predicted when the explanatory variables are observed. The goal for the learner is to be able to make predictions about the output values given input values. For example, the learner may receive data about patients registered in an hospital, in the form of pairs (*measures made on the patient*, *diagnostic*), and aims at being able to give a correct diagnostic for new arriving patients on which measurements are available.

By contrast, the objective of **unsupervised learning** is not to make predictions from input values to output values, but to reveal possible hidden structures in the data set, $\mathscr{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, or to detect correlations between the variables. If these putative structures or regularities may sometimes be extrapolated to other data collections, this is not the primary goal of unsupervised learning.

A third type of learning, of growing importance, is **reinforcement learning** (see chapter "Reinforcement Learning" of Volume 1). There, the learner acts in the environment and therefore must be able to decide on the action to take in each successive state encountered in its peregrinations. The trick is that the learner receives reinforcement signals, positive or negative, from time to time, sometimes long after the action that triggered it has been performed. It is therefore not easy to determine which actions are the best in each possible state. The goal of the learner is to maximize the cumulated reinforcement over time even though the credit assignment problem is hard to solve. Reinforcement learning is at the core of the famous AlphaGo system that beat one of the strongest Go player in the world in March 2016, and is now believed to far outclass any human player (see chapter "Artificial Intelligence for Games" of this volume).

One important distinction is between descriptive learning and predictive learning. *Descriptive learning* aims at finding regularities in the data in the hope that they may help to better understand the phenomenon under study. For instance, descriptive learning may uncover different groups in a population of customers, which may in turn help to understand their behavior and suggest different marketing strategies. Descriptive learning is strongly linked to unsupervised learning. *Predictive learning* is concerned with finding rules that allow one to make prediction when given a new instance. Predictive learning is therefore inherently extrapolative. Its justification is in making prediction for new instances, while descriptive learning is turned towards the examples at hand, and not, at least directly, towards telling something about new instances. Predictive learning is tightly associated with supervised learning.

Sometimes, a third type of learning, called *prescriptive learning*, is mentioned. The goal there is to extract information about what can be levers or control actions that would allow one to alter the course of some phenomenon (e.g. climatic change, the diet of the general population). Generally, gaining control means that causal factors have been identified. And this is not the same as being able to predict some events based on the occurrence of some other ones, which might be done by discovering correlations between events. Therefore, special techniques and some specific form of knowledge have to be called up to confront this challenge.

2.1 The Outputs of Learning

It is useful to clarify what is the output of learning. It can indeed change in function of the application. A learning algorithm $\mathscr{A}$ can be seen as a machine that takes as input a data set $\mathscr{S}$ and produces as output either a model (loosely speaking) of the world $\mathscr{M}$ or a decision procedure h, formally $\mathscr{A} : \mathscr{S} \mapsto \mathscr{M}$ or h.

Let us consider this last case, there the decision procedure h is able to associate an output $y \in \mathscr{Y}$ to any input $\mathbf{x} \in \mathscr{X}$. Then we have $h : \mathbf{x} \in \mathscr{X} \mapsto y \in \mathscr{Y}$.

So, we see that we naturally speak of different outputs–either a model $\mathscr{M}$ or a function h–without using different words. And the decision function h outputs a prediction y given any input $\mathbf{x}$. The right interpretation of the word *output* is provided by the context, and the reader should always be careful about the intended meaning. In the following of this section, *output* will mean the output of the learning algorithm $\mathscr{A}$.

One important distinction is between the *generative* and the *discriminative* models or decision functions.

In the *generative approach*, one tries to learn a (parametric) probability distribution $\mathbf{p}_{\mathscr{X}}$ over the input space $\mathscr{X}$. If learning a precise enough probability distribution is successful, it becomes possible in principle to generate further examples $\mathbf{x} \in \mathscr{X}$

of which the distribution is indistinguishable from the true underlying distribution. Using the learnt distribution $\mathbf{p}_{\mathscr{X}}$, it is possible to use it as a model of the data in the unsupervised regime, or as a basis for a decision function using maximum a posteriori criterion (see Sect. 3.3 below). Some say that this makes the generative approach "explicative". This is only true as far as a distribution function provides an explanation. Not every one would agree on this.

The *discriminative* approach does not try to learn a model that allows the generation of more examples. It contents itself with providing either means of deciding when in the supervised mode, or means to express some regularities in the data set in the unsupervised mode. The regularities or these decision functions can be expressed as logical rules, graphs, neural networks, etc. While they do not allow to generate new examples, they nonetheless can be much more interpretable than probability distributions. Furthermore, as Vapnik, a giant in this field, famously said "*If you are limited to a restricted amount of information, do not solve the particular problem you need by solving a more general problem*" (Vapnik 1995), p. 169. This can be translated by, if you need to make prediction or to summarize a data set, it might not be convenient to look first for a generative model, something that generally implies to have large quantities of data.

Now, a decision function might provide a yes or no answer when someone would like to have an associated confidence score. The generative techniques are generally able to provide this uncertainty level rather naturally, while the discriminative techniques must be adapted, often through some heuristic means.

2.2 The Inputs of Learning

According to the discussion about the possible outputs of learning, the inputs can be either a whole data set $\mathscr{S}$, or a particular instance $\mathbf{x}$ for which one wants a prediction y. Here, we list some possible descriptions that the elements $\mathbf{x} \in \mathscr{X}$ can take depending on the application domain.

1. *Vectorial*. This is generally the case when the data is taken from a relational database. In this case, the descriptors are considered as dimensions of an input space, which is often considered as a vectorial space. In addition, when a distance is defined, we get a normed vectorial space: this is very convenient for lots of mathematical techniques have been devised for such spaces.
2. *Non vectorial*. This is the case for instance when the number of internal elements of an example is not fixed. For instance, genomes or documents have a undefined number of elements (e.g. nucleotides or words). Then, it is more difficult to define proper distances, but in most cases, adapted distances have been defined.
3. *Structured data*. In this case, one can exploit the internal structure of the data points, thus adding new structural features. It is possible to further distinguish:

- *Sequential data*. In sequential data, the description of a data point is composed of ordered elements: the individual measurements cannot be exchanged without changing its information content. A time series for instance can be characterized by some trend, or some periodic variations.
- *Spatial data*. Spatial data exhibit a spatial structure, expressing some dependencies between adjacent or distant elements of the description.
- *Graphical data*. Some data, like social networks, are best described as graphs with directed or undirected, weighted or not, edges.
- *Relational data*. More generally, complex structures, like molecules or textual documents, may need to be described, relying on formalisms close to first order logic.

Some data, like videos, share several descriptions, for instance being both sequentially and spatially organized.

It must be emphasized that finding the appropriate description of the data is very often a tricky task, which requires skilled experts. This must be done before some extra techniques that seek to massage the data further be put to work, like, for instance, identifying the principal components, or selecting the most informative descriptors.

3 Designing Learning Algorithms

3.1 Three Questions that Shape the Design of Learning Algorithms

Although predictive and descriptive algorithms have different goals and different success criteria, the overall approach to their design is similar, and it can be cast as the answer to three main questions.

1- *What type of regularities is of interest for the expert in the data*? In unsupervised learning, this question is paramount since the goal of descriptive learning is to uncover structures in the data. This question seems less central in supervised learning where the first concern is to make prediction, and, it could be said, whatever the means. However, even in the supervised setting, the type of decision function that one is ready to consider to make predictions determines the type of algorithm that is adapted.

2- *What is the performance criterion* that one wants to optimize? In the early days of machine learning, algorithms were mostly designed in an algorithmic way. The algorithm had to fill a function, and if it did, if possible with smarty procedures, all was good. The approach, nowadays, is different. One starts by specifying a performance criterion. It evaluates the quality of the model or of the hypothesis learned. In supervised learning, the criterion takes into account the fit to the training data plus a component that expresses how much the hypothesis satisfies some prior bias. In unsupervised learning, the criterion conveys how much the structure discovered in the data matches the kind of regularities one is expecting.

3- *How to organize the search in the space of possible structures*? Learning is viewed as a search procedure in a space of possible structures, given a performance criterion. Once a space of possibilities and a performance measure have been decided upon, it is time to devise an algorithm that is able to search efficiently the space of possibilities, called the *search space* in order to find one that has the best, or at least a good, performance measure. This where computer science comes to the fore.

In the following of this section, we stay at a general level of description. Details about the choice of responses for these questions will be given in later sections.

3.2 Unsupervised Learning

3.2.1 The Problems of Unsupervised Learning

Unsupervised learning works on a dataset described in an input space $\mathscr{X}$ and aims at understanding the underlying structure of data and of the representation space w.r.t. this data. Two kinds of problems are considered: the first one, clustering, tends to find an organization of data into classes, whereas the second one aims at finding dependencies, such as correlations, between variables.

- Given a dataset, unsupervised classification, also called *clustering* aims at finding a set of compact and well separated clusters, which means that objects in a same cluster are similar (their pairwise distance is low) and objects in different clusters are dissimilar (their pairwise distance is large). This set of clusters can form a partition of the data (*partitioning problem*) or it can be organized into a hierarchy (*hierarchical clustering*). The problem is usually modeled by an optimization criterion specifying properties the output must satisfy. For instance, in partitioning problems, the most used optimization criterion popularized by *k-means* algorithm is the minimization of the sum of the squared distance of the points to the center of the cluster they belong to. Given m objects $\mathbf{x}_1, \ldots, \mathbf{x}_m$ usually in $\mathbb{R}^d$, let $\{C_1, \ldots, C_k\}$ denote the k clusters and let μ_j denote the centroid of the cluster C_j, it is written

$$\sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} ||\mathbf{x}_i - \mu_j||_2^2.$$

  This domain has received a lot of attention, and the framework has been extended in several directions. For instance, requiring a partition of the data can be too restrictive. As an example, in document classification, a text can be labelled with several topics and thus could be classified into several clusters. This leads to soft clustering (in opposition to hard clustering), including fuzzy clustering where a point belongs to a cluster with a degree of membership, or overlapping clustering where an object can belong to several clusters. Fuzzy-c-means (Dunn 1973; Bezdek 1981) for instance is an extension of k-means for fuzzy clustering. Data can also be described by several representations (for instance, texts and images for a webpage), thus leading to *multi-view clustering* that aims at finding a consensus clustering between the different views of the data.

Classic clustering methods are usually heuristic and search for a local optimum and different local optima may exist. Depending on the representation of data, on the choice of the dissimilarity measure, on the chosen methods and on the parameter setting, many different results can be produced. Which one is the *correct* partition, the one the expert expects? To overcome this problem, two directions have been taken. The first one initiated in (Wagstaff and Cardie 2000) integrates user knowledge, written as constraints, in the clustering process. Constraints can be put on pairs of points: a *must-link* constraint between two points require these two points to be in the same cluster whereas a *cannot-link* constraint between two points require these two points to be in different clusters. This leads to a new field, called *Constrained Clustering*, that is presented in chapter "Constrained Clustering: Current and New Trends" of this volume. The second solution is to generate many partitions and then to combine them in a hopefully more robust partition, this research direction is called *cluster ensemble* (Vega-Pons and Ruiz-Shulcloper 2011).

Another domain related to clustering is *bi-clustering*, also called *co-clustering*: given a matrix M, bi-clustering aims at finding simultaneously a clustering of rows and of columns. Its goal is therefore to identify blocks or biclusters, composed of rows and columns, satisfying a given property: for instance the elements in a block are constant or the elements in each row of the block are constant. See for instance (Busygin et al. 2008; Madeira and Oliveira 2004) for an introduction to co-clustering.

- Mining *interesting patterns* has been introduced in the 90s by (Agrawal and Srikant 1994) and has known a growing interest since then. The initial problem was to find association rules, modeling a dependency $(A_1 \wedge \cdots \wedge A_p) \rightarrow (B_1 \wedge \cdots \wedge B_q)$ between two sets of variables. The interest of the dependency was measured by two criteria: the *support*, defined as the proportion of the population satisfying both sets of variables and the *confidence*, an estimation of $\mathbf{P}(B_1 \wedge \cdots \wedge B_q | A_1 \wedge \cdots \wedge A_p)$ measured by the proportion of the population satisfying the conclusion of the rule among those satisfying the conditions.

Unsupervised learning has to face two important problems: controlling the complexity of the algorithms and ensuring the validation of the results. Indeed, the number of partitions of a set of m elements is given by the Bell number B_m and when the number k of clusters is fixed, it is given by the Stirling number of the second kind $S(m, k)$.[1] When mining frequent patterns, the complexity is linked to the size of the search space (in case of a boolean dataset, the size of the search space is 2^d, where d is the number of boolean attributes) and to the size of the dataset (computing the frequency of a pattern require to consider all points of the dataset). This explains why many works in pattern mining have tried to reduce the complexity by pruning

[1] $S(m, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} (k_j) j^m$.

the search space and/or changing the representation of the database. From the perspective of finding an actual solution, clustering is often defined as an optimization problem, e.g., finding the best partition given an optimization criterion whereas pattern mining is an enumeration problem, e.g. finding all the patterns satisfying some given constraints.

Another important difficulty for unsupervised learning is the problem of validation. It is well-known that the notion of confidence for association rules can be misleading: it only measures the probability of the conclusion of the rule given the condition ($\mathbf{P}(B|A)$ for a rule $A \rightarrow B$) but it does not measure the correlation between A and B, and it is possible to have a rule $A \rightarrow B$ with a high confidence despite a negative correlation between A and B, therefore, different measures have been introduced for assessing the interest of a rule (see for instance Han et al. 2011). Moreover this domain has to confront the large amount of patterns that can be generated, thus making an expert evaluation difficult. Regarding clustering, the problem is aggravated: how can we validate an organization of data into classes, while, in addition, it can depend on points of views. Classically two kinds of evaluation are performed: either relying on an internal criterion or by evaluating the classification in regards of a ground truth.

3.2.2 Approaches to Unsupervised Learning

Pattern mining and clustering are two distinct tasks, with their own family of methods. The first one works at the attribute level whereas the latter one works at the data level. Nevertheless, they interact in the domain of conceptual clustering that addresses the problem of finding an organization of data in concepts, where a concept is defined by two components: the *extent of the concept*, a subset of observations belonging to this concept, and the *intent of the concept*, a subset of attributes satisfied by elements of the concept.

Distance-based/similarity-based clustering

In distance-based clustering, the notion of dissimilarity between pairs of points is fundamental. When points are described by real features, the Euclidean distance ($||.||_2$) is generally considered, but when dealing with more complex data (texts, images) the identity ($d(x, y) = 0$ if and only if $x = y$) and the triangle inequality properties can be difficult to enforce, and therefore a dissimilarity measure must be defined.

Many clustering tasks are often defined as an *optimization problem*. But because of the complexity of most optimization criteria, there exist only few exact methods, either exploring the search space by Branch and Bound strategies or based on declarative frameworks, such as Integer Linear Programming or Constraint Programming (see Sect. 4.1). Methods are mainly heuristic and search for a local optimum. The heuristic nature of the methods depends of several factors

- Initialization of the method: some methods, such as k-means or k-medoids, start from a random initialization and search for a local optimum that depends on the initial choice.

- Search strategy: usually, the optimization of the criterion relies on a gradient descent procedure which is prone to converge to a local optimum.

Another family of clustering methods is no longer based on an optimization criterion but on the notion of density (see Sect. 4.4). This is illustrated by DBSCAN (Ester et al. 1996) and it relies on the notion of *core points*: a core point is a point the neighborhood of which is dense. The important notions are then the neighborhood defined as a ball of a given radius around the point and the density specified by a minimum number of points in the neighborhood. Relying on core points, dense regions can be built. One interest of such approaches is that they allow finding clusters of various shapes.

Spectral clustering (see Sect. 4.5) is the most recent family of methods. It is based on a similarity graph, where each point is represented by a vertex in the graph and edges between vertices are weighted by the similarity between the points. The graph is not fully connected: for instance edges between nodes can be removed when their similarity is less than a given threshold. The unnormalized graph Laplacian L of the graph has an interesting property: the multiplicity of the eigenvalue 0 of L is equal to the number of connected components in the similarity graph. This property leads to different algorithms. Classically, the k first eigenvalues of the Laplacian are computed, inducing a change in the data representation and then a k-means procedure is applied on the new representation. It has been shown that there are some tight links between spectral clustering, Non Negative Matrix factorization, kernel k-means and some variant of min-cut problems (see Sect. 4.5).

Finally, generative models that aim at modeling the underlying distribution $\mathbf{p}(\mathbf{x})$ of data can be applied. For clustering, data are modeled by a mixture of Gaussian and the parameters are learned, usually by maximizing the log likelihood of data, under the assumption that the examples are i.i.d. The EM algorithm is the most widely used algorithm in this context.

Conceptual clustering. Conceptual clustering was first introduced in (Michalski 1980). The aim is to learn concepts where a concept is defined by a set of objects (extent of the concept) and a description (intent of the concept). It is well illustrated by the system Cobweb (Fisher 1987): it incrementally builds a hierarchy of concepts where a concept C is described by the quantities $\mathbf{P}(X_i = v|C)$ for each feature X_i and each possible value v this feature can take.

The interest for conceptual clustering has been revived for a decade now with the emergence of pattern mining. Truly, an important notion in itemset mining is the notion of closed itemsets, where a closed itemset is a set of items that forms a concept, as defined in Formal Concept Analysis (see Sect. 7.1.4). Therefore once interesting closed itemsets, and therefore concepts, have been found, it becomes natural to study the organization of (some of) these concepts in a structure, leading to a partition of data or to a hierarchical organization of data.

Pattern mining. The problem introduced in (Agrawal and Srikant 1994) was mining association rules in the context of transactional databases: given a predefined set of items $\mathcal{I}$, a transaction is defined by a subset of items, called an *itemset*. A transactional database can be also represented by means of $|\mathscr{I}|$ Boolean features, each feature X_i representing the presence ($X_i = 1$) or absence ($X_i = 0$) of the item in the transaction. In this context a pattern is a conjunction of items, represented by an itemset. Mining association rules is divided into two steps, first mining the frequent patterns, i.e., those with a support greater than a predefined threshold and then from these frequent patterns, building the association rules. The first step is the most time-consuming, with a complexity depending on the number of features and on the number of observations: in the Boolean case, the complexity of the search space is 2^d, where d is the number of Boolean features and the evaluation of the patterns, (e.g. computing their supports) require to go through the entire database. Many algorithms, as for instance Eclat (Zaki 2000), FP-Growth (Han et al. 2000) or LCM (Uno et al. 2004), have been developed to improve the efficiency of pattern mining, relying on different representations of the database or/and on different search strategies. The closedness of an itemset is an important property, since the set of frequent closed itemsets forms a condensed representation of the set of frequent itemsets, requiring less memory to store it. Some methods find all the frequent patterns, some searches only for maximal frequent itemsets or only for closed frequent itemsets (Bastide et al. 2000).

Pattern mining has also been developed for handling more complex databases containing structured data such as sequences or graphs.

3.3 Supervised Learning

Supervised learning aims at finding prediction rules from an input space $\mathscr{X}$: the description of examples, or situations of the world, to an output space $\mathscr{Y}$: the decisions to be made. The goal is to make predictions about the output values given input values, and this is done through the learning of a decision procedure $h : \mathscr{X} \longrightarrow \mathscr{Y}$ using a training sample $\mathscr{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$.

Both the input space and the output space can take various forms according to the task at hand. Classically, the input space often resulted from extractions from a relational data base, therefore taking the form of vectorial descriptions (e.g. age, gender, revenue, number of dependents, profession, ...). Recently, non vectorial input spaces have become fashionable, like texts, images, videos, genomes, and so on. Similarly, the output space may vary from binary labels (e.g. likes, dislikes), to a finite set of categories (e.g. the set of possible diseases), or to the set of real numbers. When the output space is a finite set of elements, the learning task is known as *classification*, while, when it is infinite, it is called *regression*. In the case of *multivariate* supervised learning, the output space is the cartesian product of several spaces (e.g. one may want to predict both the profession and the age of a customer). Some learning tasks involve *structured outputs*. For instance, one may want to infer the structure of a molecule given a set of physical and chemical measurements, or to infer the grammatical structure of an input sentence.

3.3.1 The Problems of Supervised Learning

In the following, we focus on the classification task and do not deal directly with regression.

A supervised learning algorithm $\mathscr{A}$ takes as input the learning set $\mathscr{S}$ and a space of decision functions or hypotheses $\mathscr{H}$, and it must output a decision function $h : \mathscr{X} \longrightarrow \mathscr{Y}$. The search for a good hypothesis h can be done directly by an exploration of the space $\mathscr{H}$. This is called the *discriminative* approach. Or it can be done indirectly by first inferring a joint probability distribution $\mathbf{p}_{\mathscr{X}\mathscr{Y}}$ over $\mathscr{X} \times \mathscr{Y}$ by estimating both $\mathbf{p}_{\mathscr{Y}}$ and $\mathbf{p}_{\mathscr{X}|\mathscr{Y}}$ and then computing the likelihood $\mathbf{p}(y|\mathbf{x})$ for all possible values of $y \in \mathscr{Y}$, and choosing the most probable one. This approach is known as *generative* since, in principle, it is possible to generate artificial data points $\mathbf{x}_i$ for all classes $y \in \mathscr{Y}$ using the estimated distribution $\mathbf{p}_{\mathscr{X}|\mathscr{Y}}$. This is not possible if one has learned a decision function like, for instance, a logical rule or an hyperplane in the input space $\mathscr{X}$ separating two classes.

The generative viewpoint lies at the core of Bayesian learning, while the discriminative one is central to the machine learning approach. We adopt the later perspective in the following.

The Inductive Criterion

The learning algorithm $\mathscr{A}$ must solve the following problem: given a training sample $\mathscr{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ and an hypothesis space $\mathscr{H}$, what is the optimal hypothesis $h \in \mathscr{H}$?

Under the assumption of a stationary environment, the best hypothesis is the one that will minimize the expected loss over future examples. This expected loss, also called the *true risk*, writes as:

$$R(h) \;=\; \int_{\mathscr{X} \times \mathscr{Y}} \ell(h(\mathbf{x}), y)\, \mathbf{p}_{\mathscr{X}\mathscr{Y}}\, d\mathbf{x}dy$$

where $\ell(h(\mathbf{x}), y)$ measures the cost of predicting $h(\mathbf{x})$ instead of the true label y, while $\mathbf{p}_{\mathscr{X}\mathscr{Y}}$ is the joint probability governing the world and the labeling process. A best hypothesis is thus: $h^* = \mathrm{argmin}_{h \in \mathscr{H}} R(h)$. However, the underlying distribution $\mathbf{p}_{\mathscr{X}\mathscr{Y}}$ is unknown, and it is therefore not possible to estimate $R(h)$ and thus to determine h^*.

Short of being able to compute the true value of any $h \in \mathscr{H}$, it is necessary to resort to a proxy for the true risk $R(h)$. This is called the inductive criterion.

One of the best known inductive criterion is the *empirical risk*. It consists in replacing the expected loss by an empirical measure: the mean loss computed on the training set.

$$\widehat{R}(h) \;=\; \frac{1}{m} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i)$$

Searching the best hypothesis using $\hat{h} = \mathrm{argmin}_{h \in \mathscr{H}} \widehat{R}(h)$ is called the *Empirical Risk Minimization* (ERM) principle.

The ground for using an inductive criterion, such as the ERM, and the guarantees it offers, in order to find an hypothesis with a true risk not too far from the true risk of the best hypothesis h^* is the object of *the statistical learning theory* (see chapter "Statistical Computational Learning" of Volume 1). Under the assumption that the data points are identically and independently distributed, the theory is able to show that the ERM must be altered with the incorporation of a bias on the hypotheses to be considered by the learner. Indeed, if no such bias is imposed, then it is always possible to find hypotheses that have low empirical risk, they fit the training data very well, while their true risk is high, meaning they behave badly over instances that do not belong to the training set. This is called *overfitting*.

The hypothesis space $\mathscr{H}$ must consequently be limited in its capacity to accommodate any target concept, or, alternatively, the empirical risk must be "regularized" with the addition of a term that imposes a cost over hypotheses not well behaved according to some prior preference. For instance, one may suppose that the target concepts obeys some smoothness over the input space, which can be translated in penalizing functions h with high values of their second derivative. Another way of limiting the space of hypotheses considered by the learner is to prevent it to search the whole space $\mathscr{H}$ by stopping the search process early on. This is for instance the role of the "early stopping rule" known in artificial neural networks.

The challenge in supervised induction is therefore to identify an hypothesis space rich enough so that a good hypothesis may be found (no underfitting) but constrained enough so that overfitting can be controlled. Sophisticated learning algorithms are able to automatically adjust the "capacity" of the hypothesis space in order to balance optimally the two competing factors.

Once the inductive criterion is set, it remains to explore the hypothesis space in order to find an hypothesis that optimizes as best as possible the inductive criterion.

Controlling the Search in the Hypothesis Space and the Optimization Strategy

Finding the hypothese(s) that optimize(s) the inductive criterion can be done analytically only in very specific cases. Typically, learning algorithms implement methods that update estimates of the solution via an iterative process. These processes include optimization techniques, solving systems of linear equations, and searching lattice-like state spaces. Usually, the learning algorithms require large amounts of numerical and other operations, and it is therefore of foremost concern to control the computational and space complexities of these processes as well as ensuring that the approximations for real numbers stay correct.

When the hypothesis space is the space of vectors of real numbers $\mathbb{R}^n$ ($n \in \mathbb{N}$), as is the case for artificial neural networks for instance, gradient-based methods are the method of choice for optimizing a numerical criterion. When the hypothesis space is discrete, for instance when it is a space of logical expressions, it is important to find operators between expressions that render the exploration of the hypothesis space amenable to classical artificial intelligence search techniques, and in particular to efficient pruning of the search space. This is central to the version space learning algorithm (see Mitchell 1982, 1997) and to the search for typical patterns in databases (see Agrawal and Srikant 1994; Aggarwal 2015).

3.3.2 Approaches to Supervised Learning

There exist a wide variety of supervised learning methods, with new methods, or at least variations of methods, invented almost daily. It is nonetheless possible to group these methods into broad categories.

Parametric Methods

It often happens that the expert knows in advance precisely the type of regularities he/she is looking for in the data. For instance, one may want to fit a set of data points with linear regression. In this case, the learning problem is generally to estimate the coefficients, or parameters, of the model.

E.g., in a linear regression in $\mathbb{R}^n$, $h(\mathbf{x}) = \sum_{i=0}^{n} w_i \, x^{(i)}$, where the $x^{(i)}$ are the coordinates of the input $\mathbf{x}$, the $n+1$ parameters w_i $(0 \leq i \leq n)$ must be estimated.

Parametric methods include most of the generative models (which hypothesize some probability distributions over the data), linear and generalized linear models and simple neural networks models.

Non Parametric Methods

The difference between parametric methods and non parametric methods is not as clear cut as the terms would suggest. The distinction is between families of models that are constrained by having a limited number of parameters and those which are so flexible that they can approximate almost any posterior probabilities or decision functions.

This is typically the case of learning systems that learn a non a priori fixed number of prototypes and use nearest neighbors technique to decide the class of a new input. These systems are ready to consider any number of prototypes as long as it allow them to fit well the data. The *Support Vector Machines* (SVM) fall in this category since they adjust the number of support vectors (learning examples) in order to fit the data. *Deep neural networks* are in between parametric methods and non parametric methods. They have often indeed a fixed number of parameters, but this number is usually so large that the system has the ability to learn any type of dependency between the input and the output.

Other non parametric methods include *decision tree learners*, and more generally all learning systems that partition the input space into a set of "boxes" of which the structure depends on the learning data. Systems that *learn logical descriptions* of the data, often in the form of a collection of rules, are equally non parametric, and may be seen as variants of technique that partition the input space into a set of categories.

Finally, *ensemble learning methods*, such as bagging, boosting and random forests, are also non parametric since the number of base hypotheses that are combined to form the final hypothesis is determined during learning.

It is worth noting that methods that use *regularized inductive criteria* or that adjust the hypothesis space to the learning task at hand (like the Structural Risk Minimization (SRM) principle of Vapnik) may be seen as belonging to an intersection between parametric and non parametric methods. They are parametric because they impose a constrained form to the learning hypotheses, but they adjust the number of non null parameters in function of the learning problem.

Changes of Representations in Supervised Learning

Many learning methods rely at their core on some change of representation of the input space, so that the structure of the data or the decision function becomes easy to discover. These changes of representation may be obtained through a preprocessing step, when e.g. a *Principal Component Analysis* (PCA) or *Singular Value Decomposition* (SVD) or *Non Negative Matrix Factorization* (NMF) are performed. They can also result from the learning process itself, like when using MultiLayer Perceptrons and deep neural networks, where the first layers adapt their weights (parameters) so that the top layer of neurons can implement a linear decision function.

Likewise, regularized techniques, like LASSO, that select the descriptive features that play a role in the final hypothesis, can be considered as methods for changing the representation of the input space.

3.4 The Evaluation of Induction Results

Most methods in inductive learning entail building a model of the relationships between the attributes and the outcome (supervised learning) or between attributes themselves (unsupervised learning), with a penalty term that penalizes the complexity of the model. In order to select the best model, and therefore, the optimal complexity parameter, and the best meta parameters of the methods (e.g. the architecture of the neural networks), which ensure the best predictive performance without overfitting, one has to be able to evaluate the model's performance.

The biggest hurdle to evaluate the value of a learning model is that one does not know the future events that the system will have to process. One has therefore to rely on the known (training) instances and on some a priori assumptions about the relationship between the training data and the future environment in which the system will have to perform. One such assumption is that the world is stationary.

Learning entails the *optimization of two different sets of parameters*: the parameters that define one specific hypothesis in the model which is also the class of possible hypotheses (e.g. the weights of the neural network of which the architecture is given), and the parameters, aka meta-parameters, that control the model (e.g. the architecture of the neural network, the learning step and other such choices that govern the optimization process). In order for these two optimization problems to be properly carried out, it is essential that the training data used for these two optimization tasks be different to obviate the risk of obtaining optimistic and biased results. Thus, ideally, the available data is split into three different subsets: the *learning set* used to set the parameters of the hypothesis, the *validation set* that is used both to evaluate the generalization performance of the hypothesis and to learn the meta-parameters in order to optimize the model, and the *test set* that is used just once, at the end of the whole learning procedure in order to estimate the true value of the final hypothesis.

When data is abundant, it is possible to reserve a significant part of the sample for each of the three subsets. Often, however, data is scarce and methods such as cross-validation must be used which, in essence, uses repeated learnings and tests on different subsets of the training sample in order to compensate for the lack of data.

Evaluating the performance of a learned model differs in supervised learning and in unsupervised learning. We first look at evaluation in supervised learning.

In *supervised learning*, the estimation of the rate of error in generalization is sometimes insufficient as a way to evaluate the value of the learned hypothesis. Thus, for example, it may be useful to estimate more precisely the rates of false positives and false negatives, or of precision and recall. Confusion matrices are then an useful tool.

It is even possible to obtain curves of the evolution of the learning performance when some meta-parameters are varied. The ROC curve (English Receiver Operating Characteristics, motivated by the development of radars during the Second World War) is the best known example. Typically it is sought to optimize the area under the ROC curve which characterizes the discriminating power of the classification method employed. The book (Japkowicz 2011) is a good source of information about evaluation methods for supervised classification.

In *unsupervised learning*, the goal is not to be able to predict the value of the output for some new input, but to uncover structures of interest in the data. Therefore, the error in generalization is replaced by other criteria that appreciate to which extent the structures discovered in the data fit the expectations set by the user. For instance, if one wishes to uncover subgroups or clusters in the data, performance criteria will be based on measures of the compactness of each cluster together with a measure of the dissimilarity between clusters. One can then choose for instance the number of clusters that maximizes the chosen criterion. However, in contrast with supervised learning, where the ground truth is known on the training data, in unsupervised learning, the estimation criteria are very dependent on a priori assumptions about the type of patterns present in the world, and this can easily give very misleading results if these assumptions are not verified.

3.5 Types of Algorithms

The presentation of learning algorithms can be organized along several dimensions:

1. The *optimization method* used by the learner. For instance, gradient-based, divide-and-conquer, and so on. As an illustration, many recent learning algorithms have been motivated by the allure of convex optimization.
2. The *types of hypotheses* that are considered for describing the world or to make predictions. For instance, linear models, non linear models, by partitioning the input spaces, etc.
3. The type of *structure of the hypothesis space*. For instance, vectorial spaces, Galois lattice, and so on.

Obviously, the optimization methods and the structure of the hypothesis space are closely interrelated, while the type of hypotheses considered commands, in a large part, the structure of the hypothesis space.

A specialist of Machine Learning tries to solve inductive problems, that is discovering general patterns or models from data by processes that can be automatized. The first questions asked are: what are the available data? What is the task? How one expects to measure the quality or performance of the learned regularities? What kind of regularities are of interest? Only after these questions have answers does the problem of actually searching the space of possible regularities arise. Of course, Machine Learning is also concerned with feasibility issues, and this all the more that the data is increasingly massive and that the regularities considered become more complex. Therefore the space and time requirements of the computations involved in learning are also a factor in choosing or devising a learning method. This is where Machine Learning blend with Computer Science. However, if a specialist of Machine Learning has to be aware of the computational demands of various types of algorithms, this has not to be his/her foremost forte, in the same way that a specialist in Artificial Intelligence is centrally interested in knowledge representation and automatic reasoning, and less centrally, even though this is important, in computational issues.

In accordance with these considerations, in the following, the chapter is structured around families of regularities that current learning algorithms are able to uncover. For each of these types, however, algorithmic and computational issues will be addressed.

4 Clustering

Clustering aims at finding the underlying organization of a dataset $\mathscr{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. An observation is usually represented by the values it takes on a set of descriptors $\{X_1, \ldots, X_d\}$, thus defining the input space $\mathscr{X}$. Most methods require the definition of either a dissimilarity measure, denoted by dis, between pairs of objects, or a similarity (also called affinity) measure, denoted by sim. Clustering has attracted significant attention since the beginning of exploratory data analysis, and many methods have been developed. In this chapter we focus on partitioning methods that aim at building a partition of $\mathscr{S}$ (see Sects. 4.1–4.5) and hierarchical methods that aim at organizing data in a hierarchical structure (see Sects. 4.6 and 4.7). The methods differ according to the way clustering is modeled. Prototype-based methods (Sect. 4.2) seek for representative points of the clusters, density-based methods (Sect. 4.4) assume that a cluster is built from connected dense regions whereas generative methods (Sect. 4.3) assume that data has been generated by a mixture of gaussians. Spectral clustering (Sect. 4.5) relies on a similarity measure and the construction of a graph reflecting the links in terms of similarity between objects.

We present in this section a few well-known methods that are representative of the various existing algorithms. More complete overviews of clustering and analysis of clustering methods can be found in (Bishop 2006), (Hastie et al. 2009). Let us also mention that outlier detection is a domain close to clustering that we do not address in this chapter. An outlier is defined in (Hawkins 1980) as *an observation which deviates so much from the other observations as to arouse suspicions that it*

Table 1 Criteria on a cluster C with centroid μ: homogeneity (left)/separation (right)

Homogeneity of C to be minimized	Separation of C to be maximized
diameter (diam): $\max_{o_i,o_j \in C} dis(o_i, o_j)$	*split* [a]: $\min_{o_i \in C, o_j \notin C} dis(o_i, o_j)$
radius (r): $\min_{o_i \in C} max_{o_j \in C} dis(o_i, o_j)$	*cut*: $\sum_{o_i \in C} \sum_{o_j \notin C} dis(o_i, o_j)$
star (st): $\min_{o_i \in C} \sum_{o_j \in C} dis(o_i, o_j)$	*ratio_cut*: $\frac{cut(C)}{\lvert C \rvert}$
normalized star: $st(C)/\lvert C \rvert$	*normalized_cut(C)*[c]: $\frac{cut(C)}{\lvert C \rvert \times (n - \lvert C \rvert)}$
clique (cl): $\sum_{o_i,o_j \in C} dis(o_i, o_j)$	
normalized cl: $st(C)/(\lvert C \rvert \times (\lvert C \rvert - 1))$	
sum of squares (ss): $\sum_{o_i \in C} \lvert\lvert o_i - \mu \rvert\rvert_2^2$	
variance(var)[b]: $\frac{\sum_{o_i \in C} \lvert\lvert o_i - \mu \rvert\rvert_2^2}{\lvert C \rvert}$	

[a] Or margin

[b] Also called the error sum of squares, for instance in Ward (1963)

[c] Given a weighted similarity graph, the normalized cut is defined differently by $cut(C)/vol(C)$, with $vol(C) = \sum_{o_i \in C} d_i$

was generated by a different mechanism. Some clustering methods, as for instance density-based methods, allow the detection of outliers during the clustering process whereas other methods, as for instance k-means, need the outliers to be detected before the clustering process. An overview of outlier detection methods can be found in (Aggarwal 2015).

Defining a metric between objects is fundamental in clustering. We discuss in Sect. 5 the kernel trick allowing to embed data in a higher dimension, without expliciting the new representation. Many clustering methods, relying on metrics, have been "kernelized".

4.1 Optimization Criteria and Exact Methods

Clustering looks for a partition of data made of compact and well separated clusters. Several criteria have been defined for assessing the quality of a single cluster. A large list is given in (Hansen and Jaumard 1997), and some examples are given in Table 1, assuming a dissimilarity measure dis between pairs of objects, or the Euclidean distance $||.||_2$.

Once criteria have been defined for assessing the property of a cluster, we can define different objective functions, specifying the properties of the expected output partition $\mathscr{C} = (C_1, \ldots, C_k)$. For instance the minimization of the maximal diameter (defined by $max_{j=1}^{k} diam(C_j)$), or the minimization of the within-clusters sum of squares (WCSS), defined in Eq. 1, rely on the notion of compactness. On the other hand the maximization of the minimal margin (defined by $min_{j=1}^{k} split(C_j)$) emphasizes the notion of separation. The first problem is polynomial only for $k = 2$, the second one is NP-hard, the third one is polynomial. A criterion can be to maximize the cut ($\sum_{j=1}^{k} cut(C_j)$), which can be solved efficiently for $k = 2$, but which leads to unbalanced clusters. This is why usually the normalized cut or the ratio cut are preferred. In case where we have a similarity matrix between points (see Sect. 4.5), the criterion becomes to minimize the cut or the ratio cut.

Because of the complexity of the problem, there are few exact methods. (Hansen and Delattre 1978) presents a method based on graph coloring for the minimization of the maximum diameter. Branch and bound algorithms (see for instance Brusco and Stahl 2005) have been proposed for different criteria. Clustering has also been modeled in Integer Linear Programming (Rao 1969; Klein and Aronson 1991; du Merle et al. 1999) and there exists a new stream of research on using declarative frameworks (Integer Linear Programming, Constraint Programming, SAT) for clustering as for instance (Aloise et al. 2012; Dao et al. 2017) and for constrained clustering (see chapter "Constrained Clustering: Current and New Trends" of this volume).

4.2 K-Means and Prototype-Based Approaches

The most well-known algorithm for clustering is k-means (Lloyd 1982; Forgy 1965). The main ideas are that a cluster can be characterized by its centroid (the most central point in the cluster) and an observation $\mathbf{x}$ is assigned to the closest cluster, measured by the distance between $\mathbf{x}$ and the centroid of the cluster. The number k of classes must be provided a priori. First, k observations, $\mu_1^0, \ldots, \mu_k^0$, are chosen as initial seeds and then the algorithm alternates two phases until convergence (i.e. when the partition is no longer changed): it first assigns each observation to the cluster whose centroid is the closest and then computes the new centroid of the clusters with the observations assigned to this cluster. Let $\mu_1^t, \ldots, \mu_k^t$ denote the centroids at iteration t, the main scheme of the algorithm is given in Algorithm 1.

Algorithm 1: k-means algorithm

```
Initialization: Choice of $k$ observations $\mu_1^0, \ldots, \mu_k^0$ as initial seeds;
$t \leftarrow 0$
repeat
    For each point $\mathbf{x_i}$, $i = 1 \ldots m$, assign $\mathbf{x}_i$ to the cluster $C_l$ with
        $l = \text{argmin}_{j \in 1 \ldots k} ||\mathbf{x}_i - \mu_j^t||_2$;
    For each $j$, $j = 1, \ldots k$, compute the new centroid:
        $\mu_j^{t+1} = \frac{1}{|C_j|} \sum_{\mathbf{x} \in C_j} \mathbf{x}$
        $t \leftarrow t + 1$
until Convergence;
```

As already stated, clustering is often viewed as an optimization problem, stating the quality of the desired partition. In this case, the optimization criterion is called the within-clusters sum of squares and it is defined by

$$WCSS = \Sigma_{j=1}^{k} \Sigma_{\mathbf{x} \in C_j} ||\mathbf{x} - \mu_j||_2^2 \tag{1}$$

when $|| \, ||_2$ denotes the Euclidian distance. It is equivalent to

$$WCSS = \Sigma_{j=1}^{k} \Sigma_{\mathbf{u},\mathbf{v} \in C_j} \frac{||\mathbf{u} - \mathbf{v}||_2^2}{|C_j|}$$

The complexity of the algorithm is $O(mkt)$, when m is the number of observations, k the number of clusters and t the number of iterations. Let us notice two important points. First, to be applicable, the number of classes must be given and computing the means of a subset of observations must be feasible. It means that all the features must be numeric. K-means can be adapted to handle non numeric attributes, for instance by replacing the centroid by the most central observation (the observation minimizing the star, i.e. defined by $\mathrm{argmin}_{o_i \in C} \sum_{o_j \in C} dis(o_i, o_j)$). But in case of non numeric attributes, the most used method consists in looking for k observations, called *medoids*, that are the most representative of the clusters, as for instance in the system PAM (Kaufman and Rousseeuw 1990): the search space is the space of all k-subsets of the observations; the search consists in considering pairs of objects composed of a medoid and a non medoid and analyzing whether a swap between them would improve WCSS. It has been further extended to improve its efficiency (see for instance Ng and Han 1994; Park and Jun 2009).

The second observation is that the optimization criterion is not convex and therefore the initialization step is fundamental: the results depend heavily on it. Several methods have been proposed for the initialization, as for instance K-means++ (Arthur and Vassilvitskii 2007).

K-means is an iterative alternate optimization method. When looking at Eq. 1, it can be seen that it depends on two parameter sets: the assignment of points to clusters and the centroids of the clusters. Let us define a boolean variable $X_{ij}, i = 1, \ldots, m, j = 1 \ldots k$ which is true when the observation $\mathbf{x}_i$ belongs to cluster C_j. We have for all i, $\sum_{j=1}^{k} X_{ij} = 1$, that is each point is assigned to one and only one cluster. Then Eq. 1 can be written:

$$WCSS = \Sigma_{i=1}^{m} \Sigma_{j=1}^{k} X_{ij}.||\mathbf{x}_i - \mu_j||_2^2 \qquad (2)$$

This form highlights the two sets of parameters: X_{ij} $(i = 1, \ldots, m, j = 1 \ldots k)$ (assignment of points to clusters) and μ_j $(j = 1, \ldots k)$ (determination of the centroid of cluster) and in fact, the two alternate steps correspond to (1) fix $\mu_j, j = 1, \ldots k$ in Eq. 2 and optimize $X_{ij}, i = 1, \ldots, m, j = 1 \ldots k$ and (2) fix $X_{i,j}, i = 1, \ldots, m, j = 1 \ldots k$ and optimize $\mu_j, j = 1, \ldots k$. For solving (1) we can notice that the terms involving X_{ij} are independent from those involving $X'_{ij}, i' \neq i$ and can be optimized independently, thus giving $X_{il} = 1$ iff $l = \mathrm{argmin}_{j \in 1 \ldots k} ||\mathbf{x}_i - \mu_j||_2$. The variables X_{ij} being fixed, finding the best $\mu_j, j = 1 \ldots k$, is a quadratic optimization problem that can be solved by setting the derivatives with respect to μ_j equal to 0, leading to $\mu_j = \frac{\sum_{i=1}^{m} X_{ij}\mathbf{x}_i}{\sum_{i=1}^{m} X_{ij}}$. For more details, see (Bishop 2006).

K-means relies on the Euclidian distance and makes the underlying assumption that the different clusters are linearly separable. As many clustering algorithms, it has been extended to kernel k-means, thus mapping points in a higher dimensionality space using a non linear function Φ. Kernel k-means relies on the fact that $||\mathbf{u} - \mathbf{v}||_2^2 =< \mathbf{u}, \mathbf{u} > -2 < \mathbf{u}, \mathbf{v} > + < \mathbf{v}, \mathbf{v} >$ and that the dot product can be replaced by a kernel function. Compared to k-means, the first part of the iteration, i.e. the assignment of a point $\mathbf{x}$ to a cluster, has to be changed: $\text{argmin}_{j \in 1...k} ||\Phi(\mathbf{x}) - \mu_j||$, with $\mu_j = \frac{1}{m} \sum_{\mathbf{u} \in C_j} \Phi(\mathbf{u})$. Replacing μ_j by its value and developing the norm using the dot product allows one to express $||\Phi(\mathbf{x}) - \mu_j||^2$ in terms of the kernel, thus avoiding the computation of $\Phi(\mathbf{x})$.

Self-organizing maps (SOM) are also based on prototypes. The simplest approach called *vector quantization* (VQ) follows the same philosophy as k-means, in the sense that k prototypes are initially chosen at random in the input space, each point is assigned to the cluster of its closest prototype and prototypes are updated. Nevertheless, at each iteration of VQ a single point $\mathbf{x}$ is chosen at random and the closest prototype is updated, so as to move closer to $\mathbf{x}$. Let us call m_j^t the prototype of cluster j at iteration t. If $\mathbf{x}$ is assigned to C_j, then its prototype is updated by the formula (ε is a fixed parameter):

$$m_j^{t+1} \leftarrow m_j^t + \varepsilon(\mathbf{x} - m_j^t)$$

thus $m_j^{t+1} - \mathbf{x} = (1 - \varepsilon)(m_j^t - \mathbf{x})$. Self-organizing map (Kohonen 1997) adds a structure (often a grid) on the prototypes and at each iteration, the closest prototypes and its neighbors are updated. The prototypes and their structure can be seen as a neural network, but learning can be qualified as competitive since at each iteration only the winning prototype and its neighbors are updated.

4.3 Generative Learning for Clustering

In Sect. 2.1, we have mentioned that there are two approaches in Machine Learning: *generative* versus *discriminative*. Generative methods aims at learning a (parametric) probability distribution $\mathsf{p}_{\mathscr{X}}$ over the input space $\mathscr{X}$. On the other hand, clustering as an exploratory process aims at discovering the underlying structure of data, and this structure can naturally be modeled by a probability distribution $\mathsf{p}_{\mathscr{X}}$. This leads to the use of generative methods for clustering. The model that is chosen is usually a linear combination of k Gaussian densities (intuitively a Gaussian density for each cluster):

$$\mathsf{p}(\mathbf{x}) = \sum_{j=1}^{k} \pi_j \mathscr{N}(\mathbf{x}|\mu_j, \Sigma_j) \tag{3}$$

$$with\ for\ all\ j\ \pi_j \geq 0\ and \sum_{j=1}^{k} \pi_j = 1 \tag{4}$$

The coefficients π_j are called the *mixing* coefficients and can be interpreted as the prior probability for a point to be generated from the jth cluster, $\mathscr{N}(\mathbf{x}|\mu_j, \Sigma_j)$ is the probability of the point given that it is drawn from the jth component.

This can also be formulated by introducing a k dimensional binary random variable, $\mathbf{z} = (z_1, \ldots, z_k)$, intuitively representing the assignment of a point to a cluster ($z_j = 1$ if $\mathbf{x}$ is assigned to cluster j). Therefore, only one z_j is equal to 1 and the other ones are equal to 0 ($\sum_{j=1}^{k} z_j = 1$). The marginal distribution of $\mathbf{z}$ is given by $\mathsf{p}(z_j = 1) = \pi_j$ and the conditional probability of $\mathbf{x}$ given z is given by $\mathsf{p}(\mathbf{x}|z_j = 1) = \mathscr{N}(\mathbf{x}|\mu_j, \Sigma_j)$ and since $\mathsf{p}(\mathbf{x}) = \sum_j \mathsf{p}(z_j = 1)\mathsf{p}(\mathbf{x}|z_j = 1)$, we fall back on Eq. 3. The k dimensional variable $\mathbf{z}$ is called a latent variable. It can be seen as a hidden set of variables: observations are described in the input space $(\mathscr{X}, \mathscr{Z})$ of dimension $d + k$, where hidden variables in $\mathscr{Z}$ give the assignment of points to clusters.

Given the observations $\mathscr{S}$, learning a generative model for $\mathscr{S}$ is classically achieved by maximizing the log likelihood of the data, defined as $ln(\mathsf{p}(\mathscr{S}|\pi, \mu, \Sigma))$, where π, μ, Σ denote respectively $\pi = (\pi_1, \ldots, \pi_k)$, $\mu = (\mu_1, \ldots, \mu_k)$ and $\Sigma = (\Sigma_1, \ldots, \Sigma_k)$. Supposing that examples have been drawn independently from the distribution we get

$$ln(\mathsf{p}(\mathscr{S}|\pi, \mu, \Sigma)) = \sum_{i=1}^{m} ln(\sum_{j=1}^{k} \pi_j \mathscr{N}(\mathbf{x_i}|\mu_j, \Sigma_j)) \qquad (5)$$

Maximizing the log likelihood, or, in other words, learning the coefficients $(\pi_j,\mu_j,\Sigma_j), j = 1 \ldots k$ is usually achieved by the *Expectation/Minimization algorithm* (EM) algorithm. Roughly, after having initialized the parameters to learn, EM iterates two phases: an expectation phase, computing the probability of the assignment of a point to a cluster given that point ($\mathsf{p}(z_j = 1|\mathbf{x}_n)$) and a maximization step that given the assignments of points to cluster, optimize the parameters (π_j, μ_j, Σ_j) for maximizing the likelihood. The algorithm for learning mixture of Gaussians is given in Algorithm 2 (Bishop 2006).

Algorithm 2: EM algorithm for learning Gaussian mixtures (Bishop 2006)

```
Initialize, for all $j$ in $1, \ldots k$, the means $\mu_j$, the covariance $\Sigma_j$ and the mixing coefficients $\pi_j$ ;
Evaluate the log likelihood using Eq. 5;
repeat
    E step Evaluate the responsibilities $\gamma(z_{ij}) = \mathsf{p}(z_j = 1|\mathbf{x}_i)$
        $\gamma(z_{ij}) = \frac{\pi_j \mathscr{N}(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^{k} \pi_l \mathscr{N}(x_i|\mu_l, \Sigma_l)}$;
    M step Reestimate the parameters
        $\mu_j^{new} = \frac{1}{m_j} \sum_{i=1}^{m} \gamma(z_{ij})\mathbf{x}_i$, with $m_j = \sum_{i=1}^{m} \gamma(z_{ij})$
        $\Sigma_j^{new} = \frac{1}{m_j} \sum_{i=1}^{m} \gamma(z_{ij})(\mathbf{x}_i - \mu_j^{new})(\mathbf{x}_i - \mu_j^{new})^t$
        $\pi_j^{new} = \frac{m_j}{m}$;
    Evaluate the log likelihood using Eq. 5 ;
until Convergence;
```

The structure of this algorithm looks similar to the structure of k-means algorithm (Algorithm 1). Let us notice that in k-means a point is assigned to a single cluster (*hard assignment*) whereas with this approach a point is assigned to the cluster with a probability (*soft assignment*).

A more detailed presentation of EM can be found in (Bishop 2006).

4.4 Density-Based Clustering

Density-based methods are based on the assumption that clusters correspond to high-density subspaces. The first method was proposed in the system DBSCAN (Ester et al. 1996): it relies on the search for core points, that are points with a dense neighborhood. The notion of density is based on two parameters: the radius ε defining the neighborhood of a point and a density threshold $MinPts$, defining the minimum number of points that must be present in the neighborhood of a point for the density around this point to be high. More formally the *ε-neighborhood* of a point $\mathbf{u}$ is defined by:

$$N_\varepsilon(\mathbf{u}) = \{\mathbf{v} \in \mathcal{S} | dis(\mathbf{u}, \mathbf{v}) \leq \varepsilon\}$$

and a *core point* is a point $\mathbf{u}$ satisfying $|N_\varepsilon(\mathbf{u})| \geq MinPts$. A point $\mathbf{v}$ is linked to a core point $\mathbf{u}$ when there exists a sequence of core points $\mathbf{u}_1, \ldots, \mathbf{u}_n$ such that $\mathbf{u}_1 = \mathbf{u}$, and for all $i, i = 2, \ldots, n, \mathbf{u}_i \in N_\varepsilon(\mathbf{u}_{i-1})$ and $\mathbf{v} \in N_\varepsilon(\mathbf{u}_n)$ A *dense cluster* is a maximal set of connected objects, that is objects that are linked to a same core point. It is thus composed of core points (the internal points of the cluster) and non core points (the border points of the cluster). The method is described in Algorithms 3 and 4: roughly for each point that has not yet been visited (marked by $Uncl$) and that is a core point, a new cluster is built and this cluster is iteratively expanded considering the core points in its neighborhood (Ester et al. 1996).

Algorithm 3: Outline of DBSCAN algorithm

```
Data: SetofPoints, $\varepsilon$, $MinPts$
$ClustId \longleftarrow first(ClusterId)$ ;                       // First cluster label
for each point $\mathbf{x}$ do
    if Mark($\mathbf{x}$) = $Uncl$ ;                                  // x unclassified
    then
        if Expand(SetofPoints, $\mathbf{x}$, $ClustId$, $\varepsilon$, $MinPts$) then
            $ClustId \longleftarrow$ next($ClusterId$) ;              // New cluster label
        end
    end
end
```

Algorithm 4: Expand($SetofPoints$, $\mathbf{x}$, $ClustId$, ε, $MinPts$)

```
$N \longleftarrow N_\varepsilon(\mathbf{x})$;
if $|N| < MinPts$ ;                                   // Not a core point
then
    Mark($\mathbf{x}$) $\leftarrow Noise$ and Return $False$ ;     // Mark may change later
else
    for all points $\mathbf{z}$ in $N$ Mark($\mathbf{z}$) $\leftarrow ClustId$ ;  // Init the cluster with $N_\varepsilon(\mathbf{x})$
    Delete $\mathbf{x}$ from $N$;
    while $N \neq \emptyset$ do
        Let $\mathbf{s}$ be the first element of $N$;
        if $N_\varepsilon(\mathbf{s}) \geq MinPts$ ;                  // Expand the cluster
        then
            for all points $\mathbf{z}$ in $N_\varepsilon(\mathbf{s})$ do
                if Mark($\mathbf{z}$) = $Uncl$ then add $\mathbf{z}$ to $N$;
                if Mark($\mathbf{z}$) = $Uncl$ or $Noise$ then Mark($\mathbf{z}$) $\leftarrow ClustId$;
            end
        end
        Delete $\mathbf{s}$ from $N$;
    end
    Return $True$;
end
```

The average complexity of this algorithm is $O(n \times log(n))$ with adapted data structures, such as $R * -trees$. Density-based clustering allows the detection of outliers, that are defined in DBSCAN as points that are connected to no core points. The second important property is that by merging neighborhoods, density-based clustering allows one to find clusters of arbitrary shape. On the other hand the drawback is that the output is highly dependent on the parameters ε and $Minpts$, although heuristics have been proposed to set these parameters. Moreover the algorithm cannot handle cases where the density varies from one cluster to another. Some extensions, as for instance OPTICS proposed in (Ankerst et al. 1999), have been defined to take into account learning in presence of clusters with different density.

4.5 Spectral Clustering, Non Negative Matrix Factorization

A clear and self-contained introduction to spectral clustering can be found in von Luxburg (2007). Spectral clustering takes as input a similarity graph $\mathcal{G} = (S, E)$: the nodes S of the graph are the observations and E is a set of weighted edges, where the weight w_{ij}, $w_{ij} \geq 0$, between two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ represent the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$ ($w_{ij} = 0$ when there are no edges between $\mathbf{x}_i$ and $\mathbf{x}_j$ or when the similarity is null).

When the input is a pairwise dissimilarity or a distance dis between pairs of points, a function transforming the distance into a similarity must first be applied;

for instance it can be a Gaussian similarity function: $sim(\mathbf{x}_i, \mathbf{x}_j) = exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{2\sigma^2})$, where σ is a parameter. Several methods have been proposed to build the graph. All nodes can be connected and the edges weighted by the similarity. It is also possible to use an unweighted graph, connecting points whose distances are less than a given parameter ε.

In the following $Sim = (sim_{ij})_{i,j=1...m}$ denotes a similarity matrix with sim_{ij} the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, $W = (w_{ij})_{i,j=1...m}$ denotes the weight matrix and D the degree matrix. D is a diagonal matrix whose diagonal terms are the degree d_i of points, defined by $d_i = \sum_{j=1}^{m} w_{ij}$, $i = 1, \dots m$.

Given this graph, a Laplacian matrix is built. The Laplacian matrix is particularly interesting since the multiplicity of the eigenvalue 0 of the Laplacian matrix is equal to the number of connected components of the graph. If the multiplicity is equal to k then the partition of the observations in k clusters is naturally given by the k connected components of the graph. More precisely, several Laplacian graphs can be built: the unnormalized graph Laplacian L, or the normalized graph Laplacians, L_{sym} or L_{rw}, defined by:

$$L = D - W, \quad L_{sym} = D^{-\frac{1}{2}} L D^{\frac{1}{2}} \quad and \quad L_{rw} = D^{-1} L.$$

Laplacian have important properties. For instance considering L, we have:

$$\forall \mathbf{x} = (x_1, \dots, x_m)^t \in \mathbb{R}^m, \mathbf{x}^t L \mathbf{x} = \frac{1}{2} \sum_{i,j=1}^{m} w_{ij} (x_i - x_j)^2$$

The definition of L and this property allow to show that L is a symmetric, positive semi-definite matrix, it has m non negative, real-valued eigenvalues, $0 \leq \lambda_1 \leq \dots \leq \lambda_m$ and 0 is the smallest eigenvalue with as eigenvector the unit vector $\mathbf{1}$ whose components are all equal to 1.

This leads to two kinds of approaches:

- *Spectral clustering*, that given the graph Laplacian L computes the first k eigenvectors u_1, ..., u_k of L. Considering $U = [u_1, \dots, u_k]$, each element u_{ij}, $i = 1 \dots m$, $j = 1 \dots k$ can be seen as the belonging of $\mathbf{x}_i$ to C_j. K-means is then applied on the rows to obtain a partition. See Algorithm 5.
- *Structured graph learning* (Nie et al. 2014) that aims at modifying the similarity matrix Sim so that the multiplicity of 0 in its graph Laplacian is equal to k. This leads to the following minimization problem, where a regularization term is introduced to avoid a trivial solution ($||.||_F$ denotes the Frobenius norm)

$$\min_{Sim} \sum_{i,j=1}^{m} ||\mathbf{x}_i - \mathbf{x}_j||_2^2 sim_{ij} + \mu ||Sim||_F$$
$$s.t. \;\; Sim.\mathbf{1} = \mathbf{1}, sim_{ij} \geq 0, rank(L_{sim}) = m - k.$$

Algorithm 5: Unnormalized spectral clustering (von Luxburg 2007)

```
Input  : a similarity matrix $Sim = (s_{ij})_{i,j=1...m}, \in \mathbb{R}^{m\times m}$
         a number $k$ of classes
Output: $k$ clusters, $C_1, \ldots, C_k$
Compute the similarity graph $G = (S, E)$;
Compute the unnormalized Laplacian $L$, $L \in \mathbb{R}^{m\times m}$;
Compute the first $k$ eigenvectors $u_1, \ldots, u_k$ of $L$;
Let $U = [u_1, \ldots, u_k]$ , the matrix whose columns are $u_i$ $(U \in \mathbb{R}^{m\times k})$;
Let $z_i, i = 1, \ldots, m$, the vectors corresponding to the rows of $U$;
Cluster the points $(z_i)_{i=1,\ldots,m}$ into $k$ clusters $A_1 \ldots A_k$;
Return for all $j, j = 1 \ldots l, C_j = \{\mathbf{x}_i | z_i \in A_j\}$
```

Spectral clustering can be seen as a relaxation of the problem of finding a partition $\mathscr{C} = (C_1, \ldots, C_k)$ minimizing the *ratioCut* of $\mathscr{C}$, defined by $ratioCut(\mathscr{C}) = \sum_{j=1}^{k} \frac{cut(C_j)}{|C_j|}$. In fact it can be shown that $ratioCut(\mathscr{C})$ can be rewritten by introducing a $m \times k$ matrix H defined by $h_{ij} = \frac{1}{\sqrt{|C_j|}}$ if $\mathbf{x}_i \in C_j$ and $h_{ij} = 0$ if $\mathbf{x}_i \notin C_j$.

$$ratioCut(\mathscr{C}) = Trace(H^t LH)$$

H satisfies $H^t H = I$. Each row of H contains a single non zero value, denoting the cluster the point belongs to whereas each column of H is an indicator vector representing the composition of the corresponding cluster. The relaxation is performed by allowing H to take arbitrary real values. The minimization problem:

$$minimize_{H\in\mathbf{R}^{m\times k}} Trace(H^t LH)$$
$$under\ the\ condition\ H^t H = I$$

has as solution the matrix $U = [u_1, \ldots, u_n]$ composed by the first k eigenvectors of L (theorem of Rayleigh–Ritz). U has to be transformed to model a partition and this is achieved by applying k-means. Other relations between clustering frameworks have been addressed: Dhillon et al. (2004) also shows a relation on a weighted version of kernel k-means, spectral clustering as defined in Ng et al. (2001) and normalized cut; Ding and He (2005) has also shown the relation between Spectral Clustering, kernel k-means and Nonnegative Matrix Factorization of W. Let us recall that non Negative Factorization of W consists in finding a matrix H, $H \in \mathbb{R}_+^{n\times k}$ minimizing $||W - HH^t||_F$, where $||.||_F$ denotes the Frobenius norm.

4.6 Hierarchical Clustering

Searching for an organization of data into a single partition requires to choose the level of granularity, defined by the number of clusters in the partition. Hierarchical clustering solves this problem by looking for a sequence of nested partitions.

Hierarchical clustering (Ward 1963; Johnson 1967) aims at building a sequence of nested partitions: the finest one is the partition $\mathscr{P}$, $\mathscr{P} = \{\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_m\}\}$ where each point is put in a cluster reduced to this point and the coarsest one is $\mathscr{Q}$, $\mathscr{Q}= \{\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}\}$, composed of a single class containing all the points. There exists two family of algorithms. Divisive algorithms start with the partition composed of a single class and iteratively divide a cluster into 2 or more smaller clusters, thus getting finer partitions. On the other hand, agglomerative algorithms start with the finest partition $\mathscr{P}$, composed of m clusters, each cluster being reduced to a single point. The two closest clusters are merged and the process is iterated until getting the partition $\mathscr{Q}$ composed of a single cluster. A sketch of these algorithms is given in Algorithm 6

Algorithm 6: Agglomerative hierarchical clustering

```
Compute $dis(\mathbf{x}_i, \mathbf{x}_j)$ for all pairs of points $\mathbf{x}_i$ and $\mathbf{x}_j$ in $\mathscr{S}$;
Let $\Pi = \{\{\mathbf{x}\} | \mathbf{x} \in S\}$ ;
Let $\mathscr{D}$ a dendrogram with $m$ nodes at height 0, one node for each element in $\Pi$;
while $|\Pi| > 1$ do
    Choose two clusters $C_u$ and $C_v$ in $\Pi$ so that $dis(C_u, C_v)$ is minimal ;
    Remove $C_u$ and $C_v$ from $\Pi$ and add $C_u \cup C_v$;
    Add a new node to the dendrogram labeled by $C_u \cup C_v$ at height $dis(C_u, C_v)$;
    Compute $dis(C_u \cup C_v, C_w)$ for all $C_w \in \Pi$;
end
```

This algorithm is parameterized by the choice of the dissimilarity between two clusters. For instance (Ward 1963) proposes to optimize the variance, defined in Table 1. The most known strategies for defining a dissimilarity between two clusters are:

- single linkage (nearest-neighbor strategy):
  $dis(C_u, C_v) = min\{dis(\mathbf{u}, \mathbf{v}) | \mathbf{u} \in C_u, \mathbf{v} \in C_v\}$ (split between C_u and C_v)
- average linkage:
  $dis(C_u, C_v) = mean\{dis(\mathbf{u}, \mathbf{v}) | \mathbf{u} \in C_u, \mathbf{v} \in C_v\}$
- complete linkage (furthest-neighbor strategy):
  $dis(C_u, C_v) = max\{dis(\mathbf{u}, \mathbf{v}) | \mathbf{u} \in C_u, \mathbf{v} \in C_v\}$

Single linkage suffers from the chain effect (Johnson 1967): it merges clusters with a minimal split but it can iteratively lead to clusters with large diameters. On the other hand, complete linkage aims at each step at minimizing the diameter of the resulting clusters and thus finding homogeneous clusters, but quite similar objects may be classified in different clusters, in order to keep the diameters small. Average linkage tends to balance both effects.

(Lance and Williams 1967) reviews different criteria used in hierarchical clustering and proposes a general formula, allowing to update $dis(C_u \cup C_v, C_w)$ from $dis(C_u, C_v)$, $dis(C_u, C_w)$ and $dis(C_v, C_w)$.

4.7 Conceptual Clustering

Conceptual clustering was introduced in (Michalski 1980), (Michalski and Stepp 1983). The main idea of conceptual clustering is that, in order to be interesting, a cluster must be a concept, where a concept can be defined by characteristic properties (properties satisfied by all the elements of the concept) and discriminant properties (properties satisfied only by elements of the concept).[2] The system Cobweb, proposed in (Fisher 1987) is original in the sense that it learns probabilistic concepts organized into a hierarchy and it is incremental, that is examples are sequentially processed and the hierarchy is updated given a new example. More precisely, each concept C is described by the probability of the concept $\mathbf{P}(C)$ (estimated by the rate of observations in this concept w.r.t. the total number of observations) and for each attribute X and each value v, $\mathbf{P}(X = v|C)$ (estimated by the proportion of observations in the concept satisfying $X = v$).

Given a hierarchy of concepts and a new observation $\mathbf{x}$, the aim is to update the hierarchy taking into account this new observation. The new observation is first inserted at the root of the hierarchy and then iteratively integrated at the different levels of the tree. The algorithm relies on four main operators, the two last operators aims at repairing the bad choices that could have been done, because of the sequential processing of the observations:

- *Creating* a new concept: when, at a given level, $\mathbf{x}$ seems too different from the existing concepts, a new concept is created, reduced to this observation and the process stops.
- *Integrating* $\mathbf{x}$ in an existing concept C. When this concept is composed of a single observation, $\mathbf{x}$ is added to C (thus getting a concept with 2 observations) and two leaves are created one for each observation. Otherwise, probabilities of C are updated and the process goes on, considering the sons of C as the new level.
- *Merging* two concepts: when $\mathbf{x}$ seems close to two concepts, the two concepts are merged, $\mathbf{x}$ is integrated in the new resulting concept and the process goes on.
- *Splitting* two concepts: the concept is removed and its descendants are put at the current level.

The choice of the best operator relies on a criterion, called *category utility*, for evaluating the quality of a partition. It aims at maximizing both the probability that two objects in the same category have values in common and the probability that objects in different categories have different property values. The sum is taken across all categories C_k, all features X_i and all feature values v_{il}

$$\sum_j \sum_i \sum_l \mathbf{P}(X_i = v_{il})\mathbf{P}(X_i = v_{il}|C_j)\mathbf{P}(C_j|X_i = v_{il})$$

[2] If P is a property and C is a concept, a property is characteristic if $C \rightarrow P$ and discriminant if $P \rightarrow C$.

- $\mathbf{P}(X_i = v_{il}|C_j)$ is called predictability. It is the probability that an object has the value v_{il} for feature X_i given that the object belongs to category C_j
- $\mathbf{P}(C_j|X_i = v_{il})$ is called predictiveness. It is the probability with which an object belongs to the category C_j given it has a value v_{il} for a feature X_i.
- $\mathbf{P}(X_i = v_{il})$ serves as a weight. Frequent features have a stronger influence.

4.8 Clustering Validation

Validation of a clustering process is a difficult task since clustering is per nature exploratory, aiming at understanding the underlying structure of data, which is unknown. Thus, contrary to supervised learning, we have no *ground truth* for assessing the quality of the result. Several approaches have been developed:

- deviation to a null hypothesis reflecting the absence of structure (Jain and Dubes 1988): for instance samples can be randomly generated and the result is compared to the output computed on real data.
- comparison of the result with some prior information on the expected results. They can be formulated in terms of the expected structure of the clustering, as for instance getting compact and/or well separated clusters, or in terms of an expected partition. This method is mostly used for assessing the quality of a new clustering method, by running it on supervised benchmarks in which the true partition is given by the labels. It can also be used when only partial information is given.
- stability measures that study the change in the results, either when the parameters of the clustering algorithm (number of clusters, initial seeds, ...) vary or when data is slightly modified.

In all methods, performance indexes must be defined, either measuring the quality of the clustering by itself (called *internal indexes*) or comparing the result with other clusterings (either obtained by a ground truth, or on randomly generated data, or with different settings of the parameters) (called *external indexes*). There are also some *relative indexes* that compare the results of several clusterings.

4.8.1 Internal Indexes

Such indexes allow measuring the intrinsic quality of the partition. There are many indexes (a list can be found in Halkidi et al. 2002). They tend to integrate in a single measure the compactness of the clusters and their separation: the first one must be minimized whereas the second one must be maximized, and usually they are aggregated using a ratio. Some criteria for assessing the compactness of a cluster or its separation from other clusters are given in Table 1, this list is far from being exhaustive. For example, Davies–Bouldin index is defined by

$$\frac{1}{k}\Sigma_{i=1}^{k}\max_{j,j\neq i}\frac{\delta_i+\delta_j}{\Delta_{ij}}$$

In this expression, δ_i is the average distance of the objects of cluster i to the centroid μ_i, it measures the dispersion of the cluster i (to be minimized for compactness). Δ_{ij} is the distance between the centroid of cluster i and the centroid of cluster j ($dis(\mu_i, \mu_j)$), it measures the dissimilarity between the two clusters (to be maximized for cluster separation). The term $\frac{\delta_i+\delta_j}{\Delta_{ij}}$ represents a kind of similarity between clusters C_i and C_j. Clustering aims at finding dissimilar clusters and therefore the similarity of a cluster with the other ones must be small. Davies–Bouldin index averages out the similarity of each cluster with its most similar one, the quality is higher when this index is small.

4.8.2 External Indexes

We suppose that the result of the clustering is a partition $\mathscr{C} = (C_1, \ldots, C_k)$ and we already know a partition $\mathscr{P} = (P_1, \ldots, P_k)$, which is called the *reference partition*. The external indexes compare the situation of pairs of points $(\mathbf{x}, \mathbf{y})$ in each cluster: do they belong to the same cluster in both partitions? Do they belong to different clusters in both partitions? More precisely comparing the two partitions can involve the following numbers:

- a: number of pairs of points belonging to a same cluster in both partitions
- b: number of pairs of points belonging to a same cluster in $\mathscr{C}$ but not in $\mathscr{P}$
- c: number of pairs of points belonging to a same cluster in $\mathscr{P}$ but not in $\mathscr{C}$
- d: number of pairs of points belonging in different clusters in both partitions.

This leads to the definition of the *Rand Index* defined by

$$RI = \frac{a+d}{\frac{n(n-1)}{2}}$$

where $a + d$ represents the number of agreements between $\mathscr{C}$ and $\mathscr{P}$: when the partitions are identical, the Rand index is equal to 1. The adjusted rand index (ARI) is usually preferred. It corrects the Rand Index by comparing it to the expected one: when $ARI = 0$, the learned partition is not better than a random partition, whereas if $ARI = 1$, the two partitions are identical.

5 Linear Models and Their Generalizations

We now turn to various hypothesis spaces, which can be used either in the unsupervised context or in the supervised one. Most of the following presentation, however, is put in the supervised setting. We start with the simplest model: the linear ones.

When the input space $\mathscr{X}$ is viewed as a vectorial space, for instance $\mathbb{R}^d$, where d is the dimension of this space, it becomes natural to think of regularities in the data as geometric structures in this input space. The simplest such structures are linear, like lines or (hyper)planes. Their appeal for inductive learning comes from their simplicity which helps in understanding or interpreting what they represent, and from the property of the associated inductive criteria that are often convex and therefore lead to efficient ways of approximating their global optimum. Additionally, because these models have limited expressive power, they are not prone to overfitting and are stable to small variations of the training data.

Typically, the expressions considered for representing regularities are of the form:

$$\boxed{h(\mathbf{x}) \;=\; f\Big(\sum_{i=1}^{n} w_i \, g_i(\mathbf{x}) + w_0\Big)}$$

1. When the $g_i(\cdot)$ are the projection on the ith coordinate of $\mathscr{X}$, and f is the identity function, we have the classical *linear regression model.*
2. When f is the *sign* function, we have a *binary linear classification model*, where $\sum_{i=1}^{n} w_i \, g_i(\mathbf{x}) + w_0 \;=\; 0$ is the equation of the separating hyperplane.
3. *Mixtures models* in the generative approach are also often expressed as linear combination of simple density distributions, like mixtures of Gaussians: $\mathbf{p}(\mathbf{x}) = \sum_{i=1}^{n} \pi_i \, \mathbf{p}_i(\mathbf{x}|\theta_i)$.
4. When the functions $g_i(\cdot)$ are themselves non linear transformations of the input space, we get the so-called *generalized linear models.*
5. Even though the *Support Vector Machine* is a non parametric method for classification _ meaning that its number of parameters depends on the training data _, it can also be cast as a linear model of the form:

$$h(\mathbf{x}) \;=\; \sum_{i=1}^{n} \alpha_i \, \kappa(\mathbf{x}, \mathbf{x}_i) \, y_i$$

   where the kernel function κ measures a "similarity" between instances in the input space, and can be seen as special cases of function $g_i(\cdot)$, with each $g_i(\mathbf{x}) = \kappa(\mathbf{x}_i, \mathbf{x})$.
6. Some ensemble learning methods, such as boosting for binary classification, are equally members of the linear models family. In these methods, the hypotheses generally take the form:

$$H(\mathbf{x}) \;=\; sign\Big(\sum_{i=1}^{n} \alpha_i \, h_i(\mathbf{x})\Big)$$

   where the number n of base (or "weak") hypotheses controls the expressive power of the hypothesis, and hence the risk of overfitting the data.

The first problem to solve in learning these models is to choose the "dictionary" of functions $g_i(\cdot)$. The second one is to estimate the parameters w_i $(1 \leq i \leq n)$.

The **choice of the dictionary** is either trivially done using classical base functions, like the splines for linear regression, or is the result on the expert's insights. When there exists a large amount of training data, methods for "dictionary learning" can be used. The design of these methods, however, remains largely a research problem because the search for a good hypothesis within an hypothesis space is now compounded by the problem of constructing the hypothesis space itself. Dictionary learning methods have been mostly used in the context of vision systems and scene analysis (Qiu et al. 2012; Rubinstein et al. 2010; Tosic and Frossard 2011).

The **estimation of the parameters** demands first that a performance criterion be defined, so that it can be optimized by controlling their values.

In *regression*, the problem is to learn a function $h : \mathscr{X} \rightarrow \mathbb{R}$ from a set of examples $(\mathbf{x}_i, y_i)_{1 \leq i \leq m}$. The differences between the actual and estimated function values on the training examples are called *residuals* $\varepsilon_i = y_i - h(\mathbf{x}_i)$. The least-squares method adopts as the empirical risk the square of the residuals $\sum_{i=1}^{m} \varepsilon_i^2$, and, according to the ERM principle, the best hypothesis $\hat{h}$ is the one minimizing this criterion. One justification for this criterion is to consider that the true target function is indeed linear, but that the observed y_i values are contaminated with a Gaussian noise.

The problem of optimizing the empirical risk can be solved by computing a closed-form solution. The matrix inversion of $\mathbf{X}^\top \mathbf{X}$ is needed, where $\mathbf{X}$ denotes the m-by-d data matrix containing m instances in rows described by d features in columns. Unfortunately, this can be prohibitive in high-dimensional feature spaces and can be sensitive to small variations of the training data. This is why iterative gradient descent methods are usually preferred.

It is also in order to control this instability, and the overfitting behavior it can denote, that *regularization* is called for. The idea is to add penalties on the parameter values.

In *shrinkage*, the penalty is on the square norm of the weight vector $\mathbf{w}$:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \, ||\mathbf{w}||^2 \right\}$$

This favors weights that are on average small in magnitude.

In *Lasso* (least absolute shrinkage and selection operator), the penalty is on the absolute values of the weights $\mathbf{w}$:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \, |\mathbf{w}| \right\}$$

Lasso uses what is called the L_1 norm and favors *sparse solutions*, in that it favors solutions with zero values for as many weights as possible while still trying to fit the data.

We now look at two simple, yet still widely used, **discriminative methods**: logistic regression and the perceptron.

Logistic regression assumes that the decision function is linear and that the distance in the input space of an example from the decision boundary is indicative of the probability of the example to belong to the class associated with this side of the boundary. In fact, the method assumes that the histogram of these distances follows a normal distribution. If $d(\mathbf{x})$ is the distance of $\mathbf{x}$ to the decision function, we have:

$$\hat{p}\big(class(\mathbf{x}) = \text{`}+\text{'} | d(\mathbf{x})\big) \; = \; \frac{\exp(\mathbf{w} \cdot \mathbf{x} - w_0)}{\exp(\mathbf{w} \cdot \mathbf{x} - w_0) + 1} \; = \; \frac{1}{1 + \exp\big(-(\mathbf{w} \cdot \mathbf{x} - w_0)\big)}$$

Because the model is based on generative assumptions (i.e. the model is able to generate data sets, in contrast to, say, decision functions), one can associate a likelihood function to the training data:

$$L(\mathbf{w}, w_0) \; = \; \prod_i P(y_i | \mathbf{x}_i) \; = \; \prod_i \hat{p}(\mathbf{x}_i)^{y_i} \, (1 - \hat{p}(\mathbf{x}_i))^{(1-y_i)}$$

We want then to maximize the log-likelihood with respect to the parameters, which means that all partial derivatives must be zero:

$$\nabla_{\mathbf{w}} L(\mathbf{w}, w_0) = \mathbf{0}$$
$$\frac{\partial}{\partial w_0} L(\mathbf{w}, w_0) = 0$$

The corresponding weight parameters $\mathbf{w}$ and w_0 can be obtained through a gradient descent procedure applied to the negative log-likelihood.

The logistic regression algorithm is based on the assumption that the distances from the decision function in the input space $\mathscr{X}$ follows a normal distribution. If this assumption is not valid, it is possible that a linear separation exists between the two classes, but that the logistic regression outputs a decision function that does not separate them properly.

By contrast, the *perceptron algorithm* guarantees that if a linear separation exists between the classes, it will output a linear decision function making no errors on the training set. The perceptron considers the training examples one at a time, and updates its weight vector every time the current hypothesis $\mathbf{h}_t$ misclassifies the current example $\mathbf{x}_i$, according to the following equation:

$$\mathbf{w}_{t+1} \; = \; \mathbf{w}_t \; + \; \eta \, y_i \, \mathbf{x}_i \tag{6}$$

The algorithm may cycle several times through the training set. Learning stops when there is no more training example misclassified. The perceptron algorithm is simple to implement and is guaranteed to converge in a finite number of steps if the classes are linearly separable.

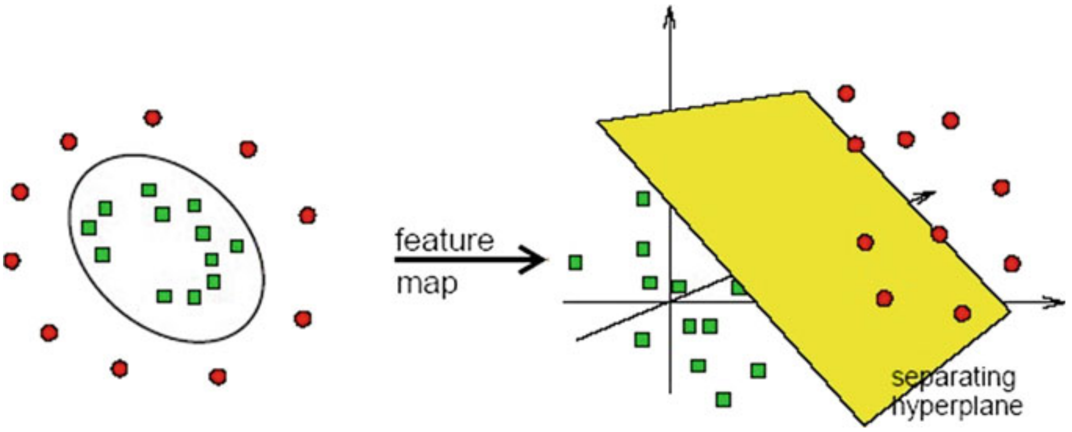


Fig. 1 A complex decision function in the input space can be made linear in a feature space with an appropriate mapping

All methods described above are limited to finding linear models within the input space. One way to circumvent this serious limitation is by changing the input space. This is what the so-called *generalized linear models* do by using sets of non linear basis functions g_i defined over $\mathscr{X}$. In the new description space spanned by the functions g_i, a linear separation can thus be associated with a non linear separation in the input space $\mathscr{X}$ (Fig. 1).

Aside its limited expressive power, the perceptron has another unsatisfactory property: it outputs a linear decision function as soon as it has found one between the training data. However, intuitively, some linear separations can be better than others and it would be preferable to output the best one(s) rather than the first one discovered. This realization is the basis of methods that attempt to maximize the "margin" between examples of different classes (Fig. 2).

Suppose we call the margin of an example $\mathbf{x}$ with respect to a linear decision function defined by its weight vector $\mathbf{w}$ the distance of this example from the decision function: $\mathbf{w} \cdot \mathbf{x}$, then we want to have all the training examples on the good side of the decision function that is learned (i.e. all the positive instances on one side, and the negative ones on the other side), and the smallest margin for the positive training examples and for the negative ones to be as large as possible (see Fig. 2). In this way, the decision boundary is the most robust to small changes of the training points, and accordingly, it can be expected that it is the best separator for new unseen data that follow the same distribution as the training set. This leads to a quadratic constrained optimization problem:

$$\mathbf{w}^* \;=\; \underset{\mathbf{w}}{\text{argmin}}\, \frac{1}{2}\,||\mathbf{w}||^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1,\, 1 \leq i \leq m$$

This optimization problem is usually solved using the method of Lagrange multipliers. Ultimately, it can be shown that the solution only depends on the set $\mathscr{S}_S$ of the so-called support vectors which are the training examples nearest, i.e. with the smallest margin to the decision boundary. Each support vector $\mathbf{x}_i$ is associated with

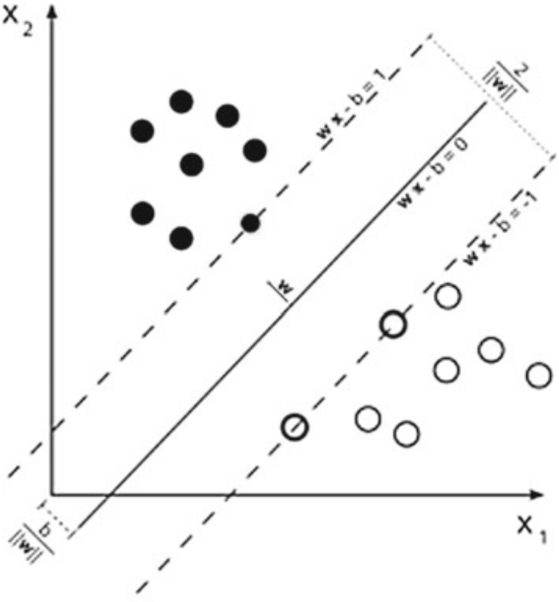


Fig. 2 A linear decision function between two classes of data points can be defined by the support vectors that are on the margin. Here only three points suffice to determine the decision function

a weight α_i. The decision function then becomes:

$$h(\mathbf{x}) \ = \ \text{sign}\left\{ \sum_{\mathbf{x}_i \in \mathscr{S}_S} \alpha_i \ y_i \ \langle \mathbf{x}_i . \mathbf{x} \rangle \right\}$$

where $\langle \mathbf{x}_i . \mathbf{x} \rangle$ is the dot product of $\mathbf{x}_i$ and $\mathbf{x}$ in the input space $\mathscr{X}$. The dot product can be considered as a measure of ressemblance between $\mathbf{x}_i$ and $\mathbf{x}$. Accordingly, the SVM can be seen as a kind of nearest neighbors classifier, where a new example $\mathbf{x}$ is labeled by comparison to selected neighbors, the support vectors $\mathbf{x}_i$, weighted by the coefficients α_i.

However, even though SVM outputs decision functions that are likely to be better than the ones produced by perceptrons, they would still be limited to produce linear boundaries in the input space. A fundamental realization by Vapnik and his co-workers (Cortes and Vapnik 1995) has changed the expressive power of the SVM, and indeed of many linear methods, such as linear regression, Principal Component Analysis, Kalman Filters, and so on.

Indeed, an essential problem of the methods based on examples is the choice of an appropriate measure of similarity. Thus, any chess player is aware that two exactly identical game situations, but for the position of a pawn, may actually lead to completely different outcomes of the game. The same can happen when comparing various objects such as molecules, texts, images, etc. In every case, the choice of the right similarity measure is crucial for methods that decide on the basis of similarities to known examples.

If mathematics provides us with many measures suited to vector spaces, for example in the form of distances, the problem is much more open when the data involve symbolic descriptors and/or are defined in non-vector spaces, such as texts. This is why a significant part of current machine learning contributions is the definition and testing of new similarity measures appropriate for particular data types: sequences, XML files, structured data, and so on. The invention and popularization of SVM by Vapnik and its co-workers have been very influential in reviving and renewing the problem of the design of appropriate similarity functions, and this can be seen especially with the rise of the so-called *kernel methods* (Schölkhopf and Smola 2002).

If one wants to adapt linear methods to learn non-linear decision boundaries, a very simple idea comes to mind: transform the data non-linearly to a new feature space in which linear classification can be applied. The problem, of course, is to find a suitable change of representation, from the input space to an appropriate feature space. We will see that one approach is to learn such a change of representation in a multilayer neural network (see Sect. 6). But one remarkable thing is that, in many cases, the feature space does not have to be explicitly defined. This is the core of the so-called "kernel trick".

Suppose that you must compare two points in $\mathbb{R}^2$: $\mathbf{u} = (u_1, u_2)$ and $\mathbf{v} = (v_1, v_2)$. One way to measure their distance is through their dot-product $\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + u_2 v_2$. Suppose now that you decide to consider the mapping $(x, y) \to (x^2, y^2, \sqrt{2}\, x\, y)$, which translates points in $\mathbb{R}^2$ into points in $\mathbb{R}^3$. Then, the points $\mathbf{u}$ and $\mathbf{v}$ are respectively mapped to $\mathbf{u}' = (u_1^2, u_2^2, \sqrt{2}\, u_1 u_2)$ and $\mathbf{v}' = (v_1^2, v_2^2, \sqrt{2}\, v_1 v_2)$. The dot product of these two vectors is:

$$\mathbf{u}' \cdot \mathbf{v}' \;=\; u_1^2 v_1^2 + u_2^2 v_2^2 + 2\, u_1 u_2 v_1 v_2 \;=\; (u_1 v_1 + u_2 v_2)^2 \;=\; (\mathbf{u} \cdot \mathbf{v})^2$$

In other words, by squaring the dot product in the input space, we obtain the dot product in the new 3-dimensional space without having to actually compute it explicitly. And one can see on this example that the mapping is non linear.

A function that computes the dot product in a feature space directly from the vectors in the input space is called a *kernel*. In the above example, the kernel is: $\kappa(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2$. Many kernel functions are known, and because kernel functions form a group under some simple operators, it is easy to define new ones as wished.

In order to be able to use the kernel trick to transform a linear method into a non linear one, the original linear method must be entirely expressible in terms of dot products in the input space. Many methods can be expressed that way, using a "dual form". This is for instance the case of the perceptron, which thus can be transformed into a kernel perceptron able to separate non linearly separable data in the input space. A kernel perceptron that maximizes the margin between the decision function and the nearest examples of each class is called a Support Vector Machine.

The general form of the equation of Support Vector Machines is:

$$h(\mathbf{x}) \;=\; \text{sign}\left\{\sum_{i=1}^{m} \alpha_i\, \kappa(\mathbf{x}, \mathbf{x}_i)\, y_i\right\}$$

where m is the number of training data. However, it suffices to use the support vectors (the nearest data points to the decision function) in the sum, which most often drastically reduces the number of comparisons to be made through the kernel function.

It is worth stressing that kernel functions have been devised for discrete structures, such as trees, graphs and logical formulae, thus extending the range of geometrical models to non-numerical data. A good introductory book is (Shawe-Taylor and Cristianini 2004).

The invention of the *boosting algorithm*, another class of generalized linear classifiers, is due to Shapire and Freund in the early 1990s (Shapire and Freund 2012). It has stemmed from a theoretical question about the possibility of learning "strong hypotheses", ones that perform as well as possible on test data, using only "weak hypotheses" that may perform barely above random guessing on training data.

Algorithm 7: The boosting algorithm

```
Input: Training data set $\mathscr{S}$ ; number of combined hypotheses $T$ ;
       weak learning algorithm $\mathscr{A}$

Initialization of the distribution on the training set: $w_{i,1} = 1/|\mathscr{S}|$ for all $\mathbf{x}_i \in \mathscr{S}$ ;

for $t = 1, \ldots, T$ do
    Run $\mathscr{A}$ on $\mathscr{S}$ with weights $w_{i,t}$ to produce an hypothesis $h_t$;
    Calculate the weighted error $\varepsilon_t$ of $h_t$ on the weighted data set;
    if $\varepsilon_t \geq 1/2$ then
        set $T \leftarrow t - 1$ and break
    end
    $\alpha_t \leftarrow \frac{1}{2} \log_2\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$      (confidence for the hypothesis $h_t$) ;
    $w_{i,t+1} \leftarrow \frac{w_{i,t}}{2 * \varepsilon_t}$      for instances $\mathbf{x}_i$ misclassified at time $t$ ;
    $w_{j,t+1} \leftarrow \frac{w_{j,t}}{2(1-\varepsilon_t)}$      for instances $\mathbf{x}_j$ correctly classified at time $t$ ;
end

Output: the final combined hypothesis $H : \mathscr{X} \rightarrow \mathscr{Y}$:

    $H(\mathbf{x}) = \text{sign}\left\{\sum_{t=1}^{T} \alpha_t \, h_t(\mathbf{x})\right\}$      (7)
```

The idea is to find a set of basis decision functions $h_i : \mathscr{X} \rightarrow \{-1, +1\}$ such that used collectively in a weighted linear combination, they can provide a high-performing decision function $H(\mathbf{x})$. How to find such a set of basis functions or weak hypotheses? The general principle behind boosting and other *ensemble learning methods* is to promote diversity in the basis functions so as to eliminate accidental fluctuations and combine their strengths (Zhou 2012). The way boosting does that is by modifying the training set between each stage of the iterative algorithm. At each stage, a weak hypothesis h_t is learnt, and the training set is subtly modified so that h_t does not perform better than random guessing on the new data set. This way, at the next stage, one is guaranteed that if a better than random hypothesis can be learnt, it has used information not used by h_t and therefore brings new leverage on the rule to classify the data points.

In boosting, the training set is modified at each stage by changing the weights of the examples. Specifically, at start the weights of all training examples are uniformly set to $1/|\mathcal{S}|$. At the next step, half of the total weight is assigned to the misclassified examples, and the other half to the correctly classified ones. Since the sum of the current weight of the misclassified examples is the error rate ε_t, the weights of the misclassified examples must be multiplied by $1/2\varepsilon_t$. Likewise the weights of the correctly classified examples must be multiplied by $1/2(1-\varepsilon_t)$. That way, at stage $t+1$, the total weight of the misclassified examples is equal to the total weight of the other examples, and equal to 1/2. In each successive round, the same operation is carried out. Finally, a confidence is computed for each learned weak hypothesis h_t. An analysis of the minimization of the empirical risk (with a exponential loss function) leads to associate $\alpha_t = \frac{1}{2}\log_2\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$ to each h_t. The basic algorithm is given in Algorithm 7.

From the Eq. 7, it can be seen that boosting realizes a linear classification in a feature space made of the values of the learned weak hypotheses (see Fig. 3). It can therefore be considered as generalized linear method which learns its feature map.

While not strictly linear in a geometrical sense, another method, the *random forest*, belongs to the family of ensemble learning, like boosting (Breiman 2001). There, the idea is to foster diversity in the basis hypotheses by changing both the training data and the hypothesis space at each round of the algorithm. The training set is changed at each time step using a bootstrap strategy, by sampling with replacement m training examples (not necessarily different) from the initial training set $\mathcal{S}$ of size m. The hypothesis space is made of decision trees, and in order to accentuate diversity, a subset of the attributes is randomly drawn at each step, forcing the individual "weak decision tree" to be different from each other. Random forests have been the winning method in numerous competitions on data analysis, and while they are no longer well publicized since the advent of the deep neural networks, they remain a method of choice for their simplicity and their propensity to reach good level of performance.

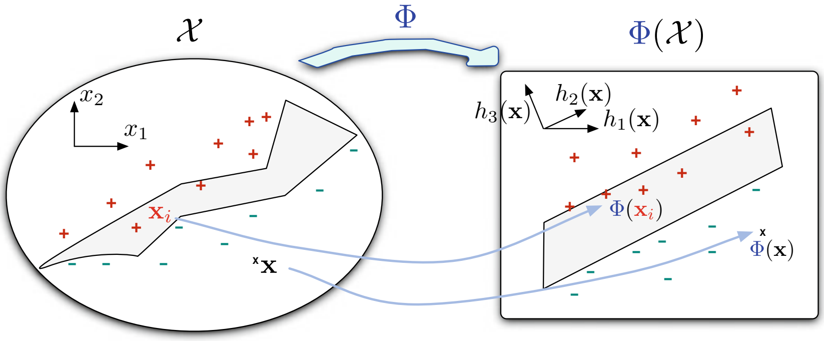


Fig. 3 Boosting in effect maps the data points into a feature space with T dimensions corresponding to the values of the weak hypotheses $h_t(\cdot)$. In this new feature space the final hypothesis $H(\mathbf{x}) = \text{sign}\left\{\sum_{t=1}^{T} \alpha_t\, h_t(\mathbf{x})\right\}$ corresponds to a linear classifier

Linear methods are said to be shallow since they combine only one "layer" of descriptors, and, supposedly, this would limit their expressive power. Methods that change the representation of the data, and specially deep neural methods are motivated by overcoming this limitation.

6 Multi-layer Neural Networks and Deep Learning

The preceding section has introduced us to linear models and ways to learn them. Among these models, the *perceptron* was introduced by Frank Rosenblatt in the late 1950s with the idea that it was a good model of learning perceptual tasks such as speech and character recognition. The learning algorithm (see Eq. 6) was general, simple and yet efficient. However, it was limited to learning linear decision functions with respect to the input space. This limitation was forcibly put forward in the book *Perceptrons* by Marvin Minsky and Seymour Papert in 1969 (see Minsky and Papert 1988) and this quickly translated into a "winter period" for connectionism. Actually, Minsky and Papert acknowledged in their book that layers of interconnected neurons between the input and the last linear decision function could perform some sort of representation change that would allow the system to solve non linear decision tasks, but they did not see how it would be possible to disentangle the effects of modifications in the various weights attached to the connections, and thus to learn these weights. They believed that these first layers would have to be hand-coded, which would be a daunting task but for the simplest perceptual problems.

For more than ten years, no significant activity occurred in the study of artificial neural networks, not to lessen the works of some courageous scientists during this period. In 1982, a change happened coming from a field foreign to Artificial Intelligence. John Hopfield, a solid state physicist, noticed a resemblance between the problem of finding the state of lowest energy in spin glass and the problem of pattern recognition in neural networks (Hopfield 1982). In each case, there is a "basin of attraction", and changing the weights in the connections between atoms or neurons can alter the basins of attractions which translates into modifying the ideal patterns corresponding to imperfect or noisy inputs. For the first time, a non linear function from the input space to the output one was shown to be learnable with a neural network. This spurred the development of other neural models, noticeably the "Boltzman machine", unfortunately slow and impractical, and the Multi-layer perceptron.

6.1 *The Multi-layer Perceptron*

In multi-layer perceptrons, the signal is fed to an input layer made of as many neurons (see Fig. 4) as there are descriptors or dimensions in the input space, and is then propagated through a number of "hidden layers" up until a final output layer which computes the output corresponding to the pattern presented to the input layer.

Fig. 4 Model of a formal neuron

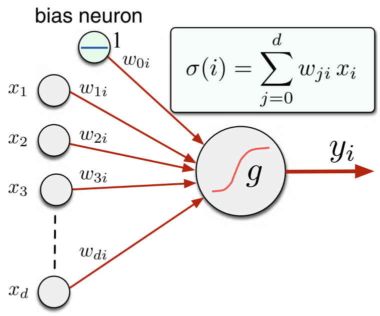
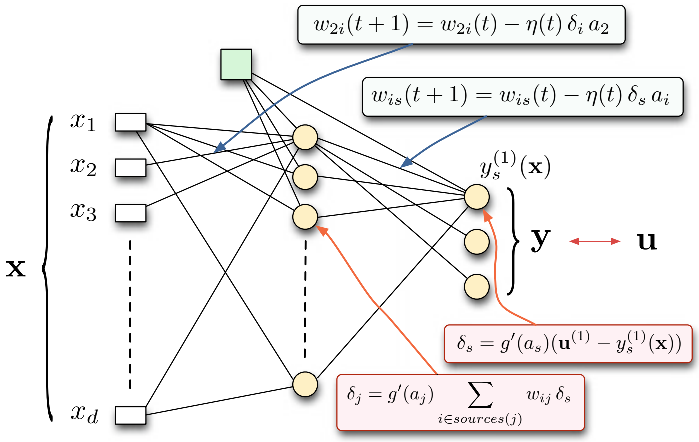


Fig. 5 The back-prop algorithm illustrated. Here the desired output **u** is compared to the output produced by the network **y**, and the error is back-propagated in the network using local equations


$w_{2i}(t+1) = w_{2i}(t) - \eta(t)\, \delta_i\, a_2$
$w_{is}(t+1) = w_{is}(t) - \eta(t)\, \delta_s\, a_i$
x_1
x_2
x_3
x_d
$\mathbf{x}$
$y_s^{(1)}(\mathbf{x})$
$\mathbf{y}$
$\mathbf{u}$
$\delta_s = g'(a_s)(\mathbf{u}^{(1)} - y_s^{(1)}(\mathbf{x}))$
$\delta_j = g'(a_j) \sum_{i \in sources(j)} w_{ij}\, \delta_s$

The hidden layers are in charge of translating the input signal in such a way that the last, output, layer can learn a linear decision function that solve the learning problem. The hidden layers thus act as a non linear mapping from the input space to the output one. The question is how to learn this mapping. It happens that the solution to this problem was found the very same year that the book “Perceptrons” was published, by Arthur Bryson and Yu-Chi Ho, and then again in 1974 by Paul Werbos in his PhD. thesis. However, this is only in 1986 that the now famous “back-propagation algorithm” was widely recognized as the solution to the learning of hidden layers of neurons and credited to David Rumelhart, Geoff Hinton and Ronald Williams (Rumelhart et al. 1987), while Yann Le Cun, independently, in France, homed on the same algorithm too (Le Cun 1986) (see Fig. 5).

To discover the learning algorithm, it was needed that the neurons were no longer seen as logical gates, with a $\{0, 1\}$ or $\{\text{False}, \text{True}\}$ output, but as continuous functions of their inputs: the famous ‘S’ shaped logistic function or the hyperbolic tangent one. This allowed the computation of the gradient of the signal error _the difference between the desired output and the computed one_ with respect to each connection in the network, and thus the computation of the direction of change for each weight in order to reduce this prediction error.

The invention of the back-propagation algorithm arrived at a timely moment in the history of Artificial Intelligence when, on one hand, it was felt that expert systems were decidedly difficult to code because of the knowledge bottleneck and, on the other hand, symbolic learning systems started to show brittleness when fed with noisy data, something that was not bothering for neural networks.

Terry Sejnowski and Charles Rosenberg showed to stunned audiences how the system NETtalk could learn to correctly pronounce phonemes according to the context, reproducing the same stages in learning as exhibited by children, while other scientists applied with success the new technique to problems of speech recognition, prediction of stock market prices or hand-written character recognition. In fact, almost a decade before the DARPA Grand Challenge on autonomous vehicles, a team of Carnegie-Mellon successfully trained a multilayer perceptron to drive a car, or, more modestly but yet, to appropriately turn the steering wheel by detecting where the road was heading in video images recorded in real time. This car was able to self-direct itself almost 97% of the time when driving from the East coast of the United States to the West coast in 1997 (see https://www.theverge.com/2016/11/27/13752344/alvinn-self-driving-car-1989-cmu-navlab).

However, rather rapidly, in the mid-90s, multilayer perceptrons appeared limited in their capacity to learn complex supervised learning tasks. When hidden layers were added in order to learn a better mapping from the input space to the output one, the back propagated error signal would rapidly become too spread out to induce significant and well-informed changes in the weights, and learning would not succeed. Again, connectionism subsided and yielded to new learning techniques such as Support Vector Machines and Boosting.

6.2 *Deep Learning: The Dream of the Old AI Realized?*

Very early, researchers in artificial intelligence realized that one crucial key to successful automatic reasoning and problem solving was how knowledge was represented. Various types of logics were invented in the 60s and 70s, Saul Amarel, in a famous paper (Amarel 1968), showed that a problem became trivial if a good representation of the problem was designed, and Levesque and Brachman in the 80s strikingly exhibited the tradeoff between representation expressiveness and reasoning efficiency (Levesque and Brachman 1987) demonstrating the importance of the former.

Likewise, it was found that the performance of many machine learning methods is heavily dependent on the choice of representation they use for describing their inputs. In fact, most of the work done in machine learning involves the careful design of feature extractors and generally of preprocessing steps in order to get a suitable redescription of the inputs so that effective machine learning can take place. The task can be complex when vastly different inputs must be associated with the same output while subtle differences might entail a difference in the output. This is often the case in image recognition where variations in position, orientation or illumination of an object should be irrelevant, while minute differences in the shape, or texture can be significant. The same goes for speech recognition, where the learning system should be insensitive to pitch or accent variations while being attentive to small differences in prosody for instance. Until 2006, and the first demonstration of successful "deep neural networks", it was believed that these clever changes of representation necessary in some application areas could only be hand-engineered. In other words, the dream of AI: automatically finding good representations of inputs and/or knowledge could only be performed for "simple" problems, as was done, in part, by multilayer perceptrons.

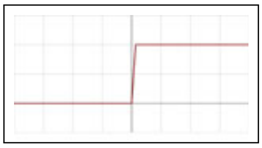
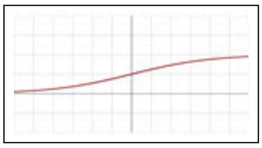
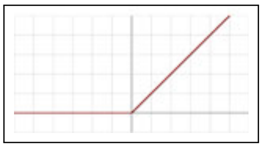
If it was known that representations using hierarchical levels of descriptors could be much more expressive than shallow representations given the same number of features, it was also believed that learning such multiple levels of raw features and more abstract descriptors was beyond the reach of algorithmic learning from data. All that changed in 2006 when a few research groups showed that iteratively stacking unsupervised representation learning algorithms could yield multiple levels of representation, with higher-level features being progressively more abstract descriptions of the data. These first successes made enough of an impression to rekindle interest in artificial neural nets. This interest spectacularly manifested itself in the unexpected large audience of the "Deep Learning Workshop: Foundations and Future Directions" organized aside of the NIPS-2007 conference. Since then, there has been an increasing number of scientific gatherings on this subject: noticeably one workshop every year at the NIPS and ICML conferences, the two major conferences in machine learning, and a new specialized conference created in 2013: ICLR, the International Conference on Learning Representations.

But then, what was the novelty with respect to the previous multilayer perceptrons? It was not a revolution, but still there were three significant new ingredients that changed the realm of possibilities.

The *first* one was related to the fact that it would help learning tremendously if the weights of the connections were not initialized randomly but in a cleverer way. And the new idea was to train each layer one by one with unsupervised learning, and then finishing with a round of supervised learning with the standard back-propagation technique. In the early stages of the deep learning, this was done by variants of the Boltzman machines invented in the 1980s, later it was carried out with auto encoders that learn to associate each input with itself but with a limited bandwidth represented by a constrained hidden layer of neurons. That way, enormous volumes of unsupervised training data could be put to use in order to learn the initial weights of the network, and more importantly to learn meaningful hierarchically organized descriptors of the data.

The *second* ingredient was the use of vastly more computing power than a decade earlier. To learn the millions of weights typical of the new deep networks, the classical CPU, even with parallelism were not enough. The computing power of Graphics Processing Units (GPU) had to be harnessed. This was quickly realized and lead to new record breaking achievements in machine learning challenges. Thus the possibility of using vastly larger sets of training data and correspondingly much faster computation lead to new horizons. Indeed, the use of very large data sets is instrumental in avoiding overfitting.

Table 2 Some usual activation functions in artificial neurons. The Heaviside activation function was used in the perceptron

Name	Graph	Equation	Derivative
Heaviside		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ ? & \text{if } x = 0 \end{cases}$
Logistic or sigmoid function		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)\left(1 - f(x)\right)$
ReLU		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$

However, a third ingredient was also of considerable importance, and it was related to the analysis of the inefficiencies of the back-propagation algorithm when the number of hidden layers exceeded a small value, like 2 or 3. This analysis lead to two major findings. First, that the non-linear activation function in neurons has a big impact on performances, and that the classical 'S' shaped ones, like the sigmoid function, results easily in vanishing gradients and thus in very slow or inexistent learning. It was found that rectified linear units (ReLU), which is simply the half-wave rectifier $g(z) = max(z, 0)$, albeit not strictly differentiable, lead to far more efficient back-propagation of the error signal (see Table 2). The second finding was that to better channel the error signal, it was possible to randomly "paralyse" a proportion of neurons during each back-propagation learning step. This trick, called "dropout", not only improved the back-propagation of the error signal in the deep networks, but it also proved to be related to the bagging technique that enlists an ensemble of classifiers in order to reduce the variance of the learned models, and therefore to increased learning performance.

A good book about deep learning is (Goodfellow et al. 2016).

6.3 Convolutional Neural Networks

While the early revival of research in neural networks, in 2006 and later, was largely spurred by the realization that iterative construction of deep neural networks was possible by using iterative unsupervised learning of each individual layer of neurons, another competing neural architecture soon regained attention. This architecture was the Convolutional Neural Network on which Yann Le Cun had been working as early as 1990 (Le Cun et al. 1990, 1995).

Deep neural networks in general exploit the idea that many natural signals are hierarchical and compositional, in which higher-level features are composed of lower level ones. For instance, in images, low-level detectors may attend to edge and local contrasts, those being combined in motifs in higher levels, motifs that are then combined into parts and parts finally into objects. Similar hierarchies exist in speech or in texts and documents.

Whereas the deep architectures obtained from iteratively using unsupervised learning trust the successive layers to encode such hierarchical descriptors, the convolutional neural networks are designed to realize some invariance operations, through convolutions, that conduct to such hierarchies. Four key ideas are drawn upon in ConvNets: local connections, shared weights, pooling and the use of many layers.

The first few stages of a typical ConvNet are composed of two types of layers: convolutional layers and pooling layers (see Fig. 6). The input to a ConvNet must be thought of as arrays (one dimensional or multi-dimensional). The units in convolutional layers see only a local patch in the input array, either the raw input in the first layer, or an array from the previous layer in later layers. Each unit is connected to its input patch through a set of weights called a filter bank. The result of this local weighted sum is then passed through the function activation, usually a ReLU. Units in convolutional layers are organized in several feature maps, and all units in the same feature map share the same filter bank (aka weight sharing). This way, each feature map specializes into the detection of some local motif in the input, and these motifs can be detected anywhere in the input. Mathematically, this amounts to a discrete convolution of the input. Then, the role of the pooling layer is to merge

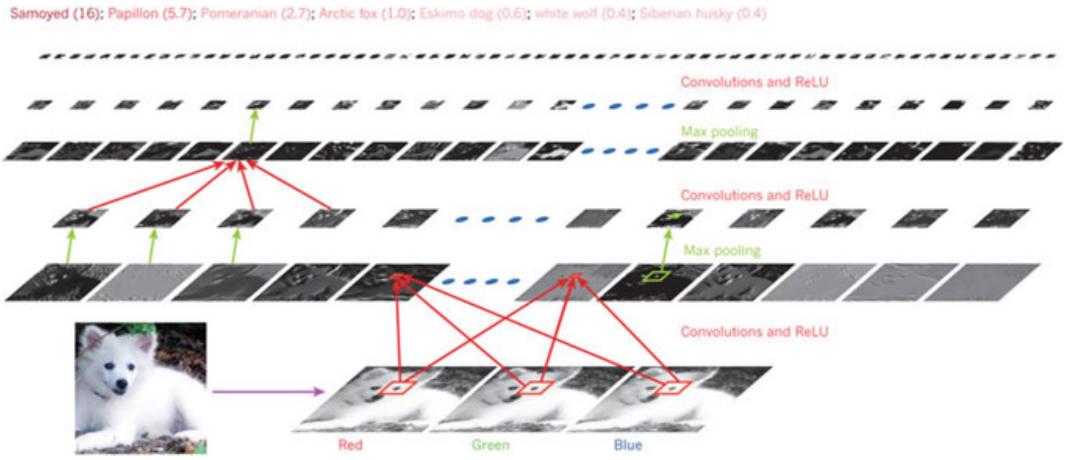

Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

Fig. 6 A convolutional network taking as input an image of a dog, transformed into RGB input arrays. Each rectangular image is a feature map corresponding to the output for one of the learned feature detected at each of the image positions. Information flows bottom up with ever more combined features as the signal goes up toward the last layer where a probability is assigned to each class of object. (image taken from Le Cun et al. 2015)

local motifs into one. One typical pooling unit may compute the maximum of the output of the local detectors realized by the convolutional unit, thus claiming that some motif has been found anywhere in the array if this maximum is high enough. This encodes a detector invariant by translation. Pooling units may also be used to reduce the dimension of the input, or to create invariance to small shifts, rotations or distortions. Typically, several stages of convolutions, non-linearity and pooling are stacked, followed by more convolutional and fully-connected layers. The back propagation algorithm, possibly using dropout and other optimization tricks is used allowing all the weights in all the filter banks to be trained. Recent Convolutional neural networks have more than 10 layers of ReLU, and hundreds of millions, sometimes billions, of weights.

A ConvNet takes an input expressed as an array of numbers and returns the probability that the input, or some part of it, belongs to a particular class of objects or patterns. ConvNets have been applied to speech recognition, with time-delay neural networks, and hand-written character recognition in the early 1990s. More recently, ConvNets have yielded impressive levels of performance, sometimes surpassing human performance, in the detection, segmentation and recognition of objects and regions in images, like traffic signal recognition, the segmentation of biological images, the detection of faces in photos, and so on. These networks have also been instrumental in the success of the AlphaGo program that now beats routinely all human champions at the game of Go (Silver et al. 2016, 2017). The other area where ConvNets are gaining importance are speech recognition and natural language understanding.

6.4 The Generative Adversarial Networks

The preceding sections have dealt mainly with supervised learning: learning to associate inputs to outputs from a training sample of pairs (*input, output*). In 2014, Ian Goodfellow and co-workers (Goodfellow et al. 2014) presented an intriguing and seducing idea by which deep neural networks could be trained to generate synthetic examples that would be indistinguishable from a training example. For instance, given a set of images of interiors in apartments, a network can be trained to generate other interiors that are quite reasonable for apartments. Or a network could be trained to generate synthetic paintings in the style Van Gogh.

The idea is the following. In GANs, two neural networks are trained in an adversarial way (see Fig. 7). One neural network, G, is the generator network. It produces synthetic examples using a combination of latent, or noise, variables as input, and its goal is to produce examples that are impossible to distinguish from the examples in the training set. For this, it learns how to combine the latent input variables through its layers of neurons. The other neural network, D, is the discriminator. Its tasks is to try to recognize when an input is coming from the training set or from the generator G. Each neural network evolves in order, for G to fool D, while for D the task is to learn to be able to discriminate the training inputs from the generated ones. Learning is successful when D is no longer able to do so.

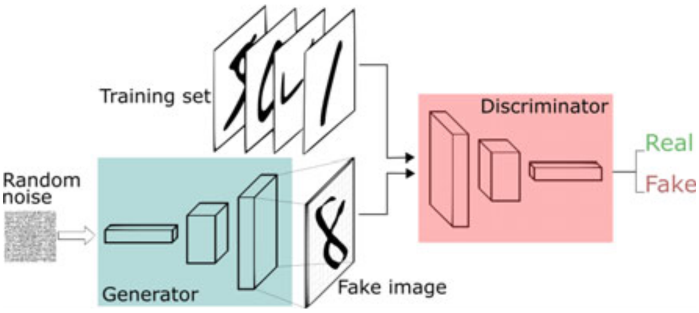


Fig. 7 A typical GAN network

6.5 Deep Neural Networks and New Interrogations on Generalization

A recent paper (Zhang et al. 2016) has drawn attention to a glaring lack of understanding of the reason of the success of the deep neural networks. Recall that supervised learning is typically done by searching for the hypothesis function that optimizes as well as possible the empirical risk, which is the cost of using one hypothesis on the training set. In order for this empirical risk minimization principle to be sound, one has to constraint the hypothesis space $\mathscr{H}$ considered by the learner. The tighter the constraint, the tighter the link between the empirical risk and the real risk, and the better the guarantee that the hypothesis optimizing the empirical risk is also a good hypothesis with regard to the real risk. If no such constraint on the hypothesis space exists, then there is no evidence whatsoever that a good performance on the training set entails a good performance in generalization.

One way to measure the constraint, or capacity, of the hypotheses space is to measure to which extent one can find an hypothesis in $\mathscr{H}$ that is able to fit any training set. If, for any training set, with arbitrary input and arbitrary output, one can find a hypothesis with low, or null, empirical risk, then there is no guaranteed link between the measured empirical risk and the real one.

But this is exactly what happens with deep neural networks. It has been shown in several studies that typical deep neural networks, used successfully for image recognition, can indeed be trained to have a quasi null empirical risk on any training set. Specifically, these neural nets could learn perfectly images where the pixels and the output were randomly set. For all purposes, there was no limit to the way they could fit any training set. According to the common wisdom acquired from the statistical learning theory, they should overfit severely any training set. Why, then, are they so successful on "natural" images?

A number of papers have been hastily published since this finding. They offer at the time of this writing only what seem partial explanations. One observation is that learning is not done in the same way when learning, by heart, a set of "random" data and learning a set of "reasonable" data. Further studies are needed, and it is expected that they bring new advances in the understanding of what makes successful induction.

More information about deep neural networks can be found in Le Cun et al. (2015), Goodfellow et al. (2016).

7 Concept Learning: Structuring the Search Space

Concept learning is deeply rooted in Artificial Intelligence, at the time when knowledge appeared to be the key for solving many AI problems. The idea was then to learn concepts defined by their intent, and thus meaningful models easily understandable by experts of the domain. Logic - propositional or first order logic - was therefore the most used representation formalism. Nevertheless, modeling the world require dealing with uncertainty and thus to study formalisms beyond classical logic: fuzzy logic, probabilistic models, ...Learning a representation of the world becomes much more difficult, since two learning problems must then be faced: learning the structure (a set of rules, dependencies between variables, ...) and learning the parameters (probabilities, fuzzy functions, ...). We focus in this section on learning the structure in classic logic, and probabilistic learning will be addressed in Sect. 8. The structure depends on the input data and on the learning task. Data can be given in a table, expressing the values taken by the objects on attributes or features, or it can be more complex with a set of objects described by attributes and linked by relations, such as a relational database or a social network. Data can also be sequential or structured in a graph. Depending on the data types, the outputs vary from a set of patterns (conjunction of properties, rules, ...), often written in logics or graphical models (automata, Hidden Markov model, Bayesian networks, conditional random fields, ...).

Before going further, let us specify some basic notions about the different representation languages used in this section. We focus mainly on rule learning.

A rule is composed of two parts, the left-hand side of the rule expresses the conditions that must be satisfied for the rule to be applied and the right-hand side or conclusion specifies what become true when the conditions are realized. Several languages can be used to express rules: the propositional logic, composed only of propositional symbols, as for example in the rule $\texttt{vertebrate} \wedge \texttt{tetrapod} \wedge \texttt{winged} \rightarrow \texttt{bird}$, the attribute-value representation as for instance $\texttt{temperature} \geq \texttt{37.5} \rightarrow \texttt{fever}$. First order logic allows one to express more complex relations between objects, such as `father (X, Y), father (Y, Z)` → `grandFather (X, Z)`. Sometimes, when the concept to be learned is implicit (for example, learning the concept `mammal`), the conclusion of the rule is omitted and only the conjunction of properties defining the concept is given. More complex expressions can be written, like general clauses allowing to specify alternatives in the conclusion or to introduce negation. Learning knowledge expressed in first-order logic is the field of study of Inductive Logic Programming (Raedt 2008; Dzeroski and Lavrac 2001). Another representation formalism is the family of *graphical models* (Koller and Friedman 2009): they are based on graphs representing dependencies between variables. We distinguish mainly the Bayesian networks, oriented graphs associating to each node the conditional probability of this node, given its parents and the Markov models, non-oriented graphs for which a variable is independent of the others, given its neighbors. A research stream, called *Statistical Relational Learning* (Raedt et al. 2008; Getoor and Taskar 2007) tends to couple Inductive Logic Programming with probabilistic approaches.

Finally, *grammatical inference* (de la Higuera 2010 or Miclet 1990) aims at learning grammars or languages from data: the family of models considered is often that of automata, possibly probabilistic ones.

Let us notice that often we do not look for an hypothesis but for a set of hypotheses, for instance a disjunction of hypotheses. Consider learning a concept from positive and negative examples. It may be unrealistic to look for a single rule covering positive examples and rejecting negative ones, since this would assume that all positive examples follow the same pattern: we then look for a set of rules. However, extending the model to a set of hypotheses introduces again the necessary compromise between the inductive criterion and the complexity of the hypothesis space. Indeed, the disjunction of the positive examples, $\mathbf{x}_1 \vee \cdots \vee \mathbf{x}_m$, represents a set of hypotheses that covers all positive examples and rejects all negatives, under the condition that the negative examples differ from the positive ones. We then have an arbitrarily complex hypothesis, varying according to the learning examples, and with a null empirical error. This is called *learning by heart*, involving no generalization. In order to solve this problem, constraints on the hypotheses must be introduced, that is, making regularization as mentioned in Sect. 3.3.

7.1 Hypothesis Space Structured by a Generality Relation

7.1.1 Generalization and Coverage

When the hypothesis space $\mathscr{H}$ is no longer parameterized, the question is how to perform an informed exploration of the hypothesis space. The notion of hypothesis space structured by a generality relation has been first introduced for concept learning, where the aim is to learn a definition of a concept in presence of positive and negative examples of this concept. A hypothesis describes a part of the space $\mathscr{X}$, and we search for a hypothesis that covers the positive examples and excludes the negative ones. Two hypotheses can then be compared according to the subspace of $\mathscr{X}$ they describe, or according to the observations they cover. We define the *coverage of a hypothesis* as the set of observations of $\mathscr{X}$ satisfied or covered by this hypothesis. A hypothesis is *more general than* another one, denoted by $h_1 \geq h_2$, if the coverage of the former contains the coverage of the latter ($h_1 \geq h_2$ iff $coverage(h_2) \subseteq coverage(h_1)$).

The inclusion relation defined on $\mathscr{X}$ thus induces a generality relation on $\mathscr{H}$, as illustrated in Fig. 8, which is a partial preorder. It is not antisymmetric since two different hypotheses may cover the same subspace of $\mathscr{X}$, but it can be transformed into an order relation by defining two hypotheses as equivalent when they cover the same subspace of $\mathscr{X}$ and by considering the quotient set of $\mathscr{H}$ with respect to this equivalence relation. Thus only one representative hypothesis among all the equivalent ones has to be considered.

The coverage relationship is fundamental for induction, because it satisfies an important property: when a hypothesis is generalized (resp. specialized), then its coverage becomes larger (resp. smaller). Indeed, when learning a concept, an *incon-*

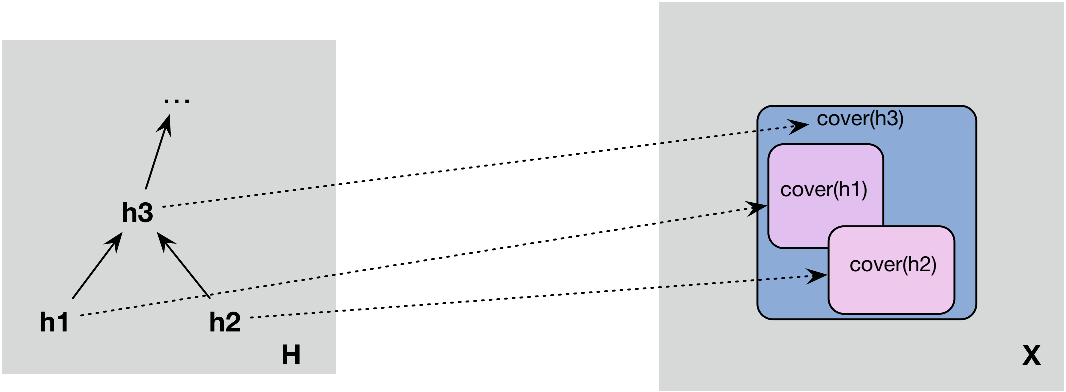


Fig. 8 The inclusion relation in $\mathscr{X}$ induces a generalization relation in $\mathscr{H}$. It is a partial preorder: h_1 and h_2 are not comparable, but they both are more specific than h_3

sistent hypothesis covering negative examples must be specialized to reject these negative examples, whereas an *incomplete hypothesis* covering not all known positive examples has to be generalized, in order to cover them. In other words, an inconsistent hypothesis when generalized remains inconsistent, whereas an incomplete hypothesis when specialized, remains incomplete. The induction process is therefore guided by the notion of coverage, which allows to define quantitative criteria such as the number of covered positive examples, the number of covered negative examples, useful for guiding the search and for pruning the search space.

7.1.2 Generalization in Logic

This notion of generalization is defined in terms of the input space $\mathscr{X}$, used to describe the observations while learning is performed by exploring the hypothesis space $\mathscr{H}$. From the point of view of search, it is more efficient to define operators working directly in the set of hypotheses $\mathscr{H}$ but respecting the generality relation defined in $\mathscr{H}$ in terms of the inclusion relation in $\mathscr{X}$. Let us notice that in some cases the representation language of the examples is a subset of $\mathscr{H}$, thus making the computation of the coverage easier. To illustrate this point, let us consider a dataset describing animals inspired by the `zoo` dataset of the UCI Machine Learning Repository.[3] An observation is described by some boolean properties (`presence of feathers` for instance) that can be true or false. In propositional logic, it can be represented by a conjunction of literals, where a literal is either a property of the negation of a property. Given a set of animals, it may be interesting to find the properties they have in common. This set of properties can still be expressed by a conjunction of literals: $\mathscr{X}$ and $\mathscr{H}$ share the same representation space: a conjunction of literals or equivalently a set of literals. In this context, a hypothesis is more general than another when it contains less properties ($h_1 \geq h_2$ if $h_1 \subseteq h_2$). It is easy to show that if $h_1 \subseteq h_2$, then $coverage(h_2) \subseteq coverage(h_1)$. The first definition is more

[3] http://archive.ics.uci.edu/ml/datasets/zoo.

interesting than this latter since comparison are made in $\mathscr{H}$ and does not involve $\mathscr{X}$. Thus defining a generality relation is quite easy in propositional logic, it becomes more problematic in predicate logic.

Indeed, in first order logic, a natural definition would be the logical implication between two formulas, but this problem is not decidable and this is why (Plotkin 1970) introduced the notion of θ-subsumption between clauses: given two clauses A and B, A is more general than B if there exists a substitution θ such that $A.\theta \subseteq B$.[4] Let us consider 4 people: John (jo), Peter (pe), Ann (an) and Mary (ma) and the two rules:

$$father(X, Y), father(Y, Z) \rightarrow grandFather(X, Z)$$
$$father(jo, pe), father(pe, ma), mother(an, ma) \rightarrow grandFather(jo, ma)$$

They can be transformed into clauses:

$$\neg father(X, Y) \vee \neg father(Y, Z) \vee grandFather(X, Z)$$
$$\neg father(jo, pe) \vee \neg father(pe, ma) \vee \neg mother(an, ma) \vee grandFather(jo, ma)$$

Consider these clauses as sets of literals. If we instantiate X by jo, Y by pe and Z by ma, the first instantiated clause is included in the second, it is therefore considered as more general w.r.t. θ-subsumption. The θ-subsumption test involves potentially costly comparisons between hypotheses. Computing the l.g.g. of 2 clauses under θ-subsumption is $\mathscr{O}(n^2)$, where n is the size of the clauses and computing the l.g.g. of s clauses under θ-subsumption is $\mathscr{O}(n^s)$.

In this example, we have made the assumption that examples were described in the same representation space as the hypotheses: an example is a fully instantiated clause and a hypothesis is a clause. This is called the *single representation trick* and this generally makes the coverage test simpler. We could also have given a set of atoms describing many people, including John, Peter, Mary, Ann and the relations between them and the subsumption test would have been even more costly, since many comparisons would have to be performed with non relevant information.

The definition of θ-subsumption can be extended to take into account knowledge on the domain, for example that a father or a mother are parents, but then reasoning mechanisms must be introduced at the price of complexity. It is important to realize that the complexity of the *coverage test* is often a determining factor for choosing the representation space $\mathscr{X}$ and the hypothesis space $\mathscr{H}$.

The representation language chosen for the hypothesis space is thus essential for determining a generality relation allowing an efficient exploration of the hypothesis space. Among the possible order relations, a special interest has been put on the relations that form a lattice: in this case given two hypotheses h_i and h_j, there exists a least upper bound and a greatest lower bound. They are called *the least general generalization*, $lgg(h_i, h_j)$, and *the least specific specialization*, $lss(h_i, h_j)$. θ-subsumption induces a lattice on the space of clauses, whereas this is not true for

[4] A clause is a disjunction of literals, assimilated in this definition to a set of literals.

logical implication. These notions are easily extended to more than 2 hypotheses, leading to $lgg(h_i, h_j, h_k, \ldots)$ and $lss(h_i, h_j, h_k, \ldots)$. Finally, it is assumed that the lattice is bounded and that there exists a hypothesis, denoted by $\top$, that is more general than all the others and an hypothesis, denoted by $\perp$, that is more specific than the others.

7.1.3 Exploration of the Search Space

The generality relation allows one to structure the hypothesis space and generalization/specialization operators can be defined to explore the search space. Let us consider a quasi-ordering[5] $\geq$ on $\mathscr{H}$ ($h_2 \geq h_1$ if h_2 is more general than h_1). A *downward refinement operator* or specialization operator is a function ρ from $\mathscr{H}$ to $2^{\mathscr{H}}$ such that for all h in $\mathscr{H}$, $\rho(h) \subseteq \{h' \in \mathscr{H} | h \geq h'\}$, whereas a *upward refinement operator* or generalization operator is a function ρ such that for all h in $\mathscr{H}$, $\rho(h) \subseteq \{h' \in \mathscr{H} | h' \geq h\}$.

Let us consider the zoo example and let us assume that the hypothesis space is composed of all the conjunctions of literals in propositional logic. A hypothesis could be: hair $\wedge$ milk $\wedge$ four_legs. It will cover all the animals satisfying these properties, as for instance a bear or a cat. Let us consider now two simple operators that consist in adding/removing a literal to a hypothesis. Removing a literal for instance allows producing the hypothesis hair $\wedge$ milk that covers more instances, and is therefore more general. Removing a literal (resp. adding a literal) allows the production of a more general (resp. a more specific) hypothesis. Refinement operators coupled with a coverage measure can be used to guide efficiently the exploration of the hypothesis space. Indeed, if a hypothesis does not cover all the positive examples, a generalization operator must be used to produce more general hypotheses covering this positive example, whereas, conversely, if negative examples are covered, specialization operators must be considered. This leads to different strategies for exploring the hypothesis space.

The generalization/specialization operators depend on the description language of the hypotheses. If it is simple enough in propositional logic, it becomes more complicated in the presence of domain knowledge or in more complex languages such as first order logic. In this case, different generalization operators can be defined, such as deleting a literal, transforming a constant into a variable or transforming two occurrences of the same variable into different variables.

To be interesting a refinement operator must satisfy some properties: it must be locally finite (i.e. it generates a finite and calculable number of refinements), proper (each element in $\rho(h)$ is different from h) and complete (for a generalization operator, it means that for all h, if $h' > h$ then h' or an hypothesis equivalent to h' can be reached by the application of a finite number of ρ to h; the definition is similar for a specialization operator). Such an operator is called ideal. See van der Laag

[5] A reflexive and transitive relation.

and Nienhuys-Cheng (1998) for a discussion on the existence of ideal refinement operators.

7.1.4 Formal Concept Analysis

As we have seen, there is a duality between the observation space $\mathscr{X}$ and the hypothesis space $\mathscr{H}$. This duality is formalized by the notion of Galois connection, which forms the basis of Formal Concept Analysis (Ganter et al. 1998, 2005). Let us consider a set $\mathscr{O}$ of objects (animals in the `zoo` dataset), a set $\mathscr{A}$ of propositional descriptors and a binary relation r on $\mathscr{O} \times \mathscr{A}$ such that $r(o, p)$ is true when object o satisfies property p. The connection between the objects and the properties is expressed by two operators, denoted here by f and g. The first one corresponds to the notion of coverage: given a set of descriptors A, $A \subseteq \mathscr{A}$, $f(A)$ returns the `extent` of A, i.e., the set of objects satisfying A. The second one corresponds to the notion of generalization: given a set of objects O, $O \subseteq \mathscr{O}$, $g(O)$ returns the `intent` of O, i.e., the set of descriptors that are true for all objects of O.

Let us consider again the concept of `animals`, described by only 4 properties (hair, milk, four_legs, domestic) and consider only 3 objects (a bear, a cat and a dolphin)

$name$	$hair$	$milk$	$four\_legs$	$domestic$
$bear$	T	T	T	F
cat	T	T	T	T
$dolphin$	F	T	F	F

The extent of the descriptor `milk` contains the three animals whereas the extent of `hair` and `milk` contains two animals, namely the `bear` and the `cat`. The intent of the first two animals, i.e., the descriptors they have in common is composed of `hair`, `milk` and `four_legs`.

These two operators form a Galois mapping satisfying the following properties: (i) f and g are anti-monotonous (if $A_1 \subseteq A_2$ then $f(A_2) \subseteq f(A_1)$ and if $O_1 \subseteq O_2$ then $g(O_2) \subseteq g(O_1)$), (ii) any set of descriptors is included in the intent of its extent (for any A of $\mathscr{A}$, $A \subseteq g(f(A))$) and every set of objects is included in the extent of its intent (for any O of $\mathscr{O}$, $O \subseteq f(g(O))$) and (iii) $f(A) = f(g(f(A)))$ and $g(O) = g(f(g(O)))$. The operator $g \circ f$ has received particular attention in the field of frequent pattern mining and is called the `closure` of a set of descriptors.

A *concept* is then defined as a pair (O, A) of objects and descriptors such that O is the extent of A and A is the intent of O. In the previous example, the pair `({bear, cat}, {hair, milk, four_legs})` is a concept. Let us notice that if (O, A) is a concept then A is a closed set ($A = g(f(A))$) and the set of closed descriptors with $\subseteq$ is a lattice. This notion is particularly useful in frequent itemset mining since the set of frequent closed itemsets is a condensed representation of the set of frequent itemsets.

For more details on Formal Concept Analysis, see chapter "Formal Concept Analysis: From Knowledge Discovery to Knowledge Processing" of this volume.

7.1.5 Learning the Definition of a Concept

name	*hair*	*four_legs*	*domestic*	*class*
bear	*T*	*T*	*F*	*mammal*
cat	*T*	*T*	*T*	*mammal*
dolphin	*F*	*F*	*F*	*mammal*
honeybee	*T*	*F*	*F*	*insect*
moth	*T*	*F*	*F*	*insect*

Let us suppose now that we want to learn a definition of the concept mammal given the three positive examples (bear, cat, dolphin) and the negative examples (honeybee, moth), with only three attributes, namely hair, four_legs, domestic. If we consider a generalization of the three positive examples, it is empty: they share no common properties. In fact these 3 examples do not belong to the same concept and must be split into two groups for common properties to emerge. To learn a concept from positive and negative examples, the most common method is to learn a hypothesis covering some positive examples and then to iterate the process on the remaining examples: this is a *divide and conquer* strategy. Alternative methods exist, as for instance first separating positive examples into classes (unsupervised classification) and then learning a rule for each class.

This leads to the problem of constructing a rule covering positive examples. Several strategies can be applied: a greedy approach, which builds the rule iteratively by adding conditions, as in Foil system (Quinlan 1996) or a strategy driven by examples, as for instance in Progol system (Muggleton 1995). Progol starts from a positive example, constructs a rule covering this example and as many positive examples as possible, while remaining consistent and iterates with the positive examples that are not yet covered by a rule. Other approaches propose to search exhaustively all the rules with sufficient support and confidence (one finds then the problem of association rule mining) and to construct a classifier from these rules (see for instance Liu et al. 1998; Li et al. 2001). For example, to classify a new example, each rule applicable on this example vote for its class with a weight depending on its confidence.

7.1.6 Extensions

An area in which learning is based on a generality relation in the hypothesis space is that of the induction of languages or grammars from examples of sequences. In *grammatical inference*, the description of hypotheses generally takes the form of automata. Most of the work has focused on the induction of regular languages that correspond to finite state automata. It turns out that the use of this representation naturally leads to an operator associated with a generality relation between automata. Without entering into formal details, considering a finite state automaton, if we merge in a correct way two states of this automaton, we obtain a new automaton, which accepts at least all the sequences accepted by the first one; it is therefore more general. Using this generalization operator, most methods of grammatical inference thus start from a particular automaton, accepting exactly the positive sequences and generalize it, by a succession of merging operations, stopping either when a negative sequence is covered or when a stop criterion on the current automaton is verified. (de la Higuera 2010) describes thoroughly all the techniques of grammatical inference.

Sometimes data results from inaccurate or approximate measurements, and it is possible that the precision on the values of the attributes is unrealistic. In this case, it may be advantageous to change the representation to take into account a lower accuracy and possibly to obtain more comprehensible concept descriptions. The *rough sets* (Suraj 2004) formalism provides a tool for approximate reasoning applicable in particular to the selection of informative attributes and to the search for classification or decision rules. Without going into details, the rough sets formalism makes it possible to seek a redescription of the examples space taking into account equivalence relations induced by the descriptors on the available examples. Then, given this new granularity of description, the concepts can be described in terms of lower and upper approximation. This leads to new definitions of the coverage notion and of the generality relation between concepts.

7.2 *Four Illustrations*

7.2.1 Version Space and the Candidate Elimination Algorithm

In the late 1970s Tom Mitchell showed the interest of generalization for Concept Learning. Given a set of positive and negative examples, the *version space* (Mitchell 1982, 1997) is defined as the set of all the hypotheses that are consistent with the known data, that is to say that cover all the positive examples (completeness) and cover no negative examples (consistency). The empirical risk is therefore null. Under certain conditions, the set of hypotheses consistent with the learning data is bounded by two sets in the generalization lattice defined on $\mathscr{H}$: one, called the *S-set* is the set of the most specific hypotheses covering positive examples and rejecting negative examples, whereas the *G-set* is the set of maximally general hypotheses consistent with the learning data.

Tom Mitchell proposed then an incremental algorithm, called the *candidate elimination algorithm*: it considers sequentially the learning examples and at each presentation of a new example, it updates the two frontiers accordingly. It can be seen as a bidirectional width-first search, updating incrementally, after each new example, the *S-set* and the *G-set*: an element of the *S-set* that does not cover a new positive example is generalized if possible (i.e. without covering a negative example), whereas an element of the *G-set* that covers a new negative example is specialized if possible.

Inconsistent hypotheses are removed. Assuming that the language of hypotheses is perfectly chosen, and that data is sufficient and not noisy, the algorithm can in principle converge towards a unique hypothesis that is the target concept. An excellent description of this algorithm can be found in Chap. 2 of Tom Mitchell (Mitchell 1997).

While this technique, at the basis of many algorithms for learning logical expressions as well as finite state automata, was experiencing a decline in interest with the advent of more numerical methods in Machine Learning, it knows a renewal of interest in the domain of Data Mining, which tends to explore more discrete structures.

7.2.2 Induction of Decision Trees

The induction of decision trees is certainly the best known case of learning models with variable structure. However, contrary to the approaches described above, the learning algorithm does not rely on the exploitation of a generality relation between models, but it uses a greedy strategy constructing an increasingly complex tree, corresponding to an iterative division of the space $\mathscr{X}$ of the observations.

A decision tree is composed of internal nodes and leaves: a test on a descriptor is associated to each internal node, as for instance `size > 1.70m`, while a class label is associated to each leaf. The number of edges starting from an internal node is the number of possible responses to the test (`yes/no` for instance). The test differs according to the type of attribute. For a binary attribute, it has 2 values and for a qualitative attribute, there can be as many branches as the domain of the attribute or it can be transformed into a binary test by splitting the domain into two subsets. For a quantitative attribute, it takes the form $A < \theta$ and the difficulty is to find the best threshold θ.

Given a new observation to classify, the test at the root of the tree is applied on that observation and according to the response, the corresponding branch towards one of the subtrees is followed, until arriving at a leaf, which gives then the predicted label. It is noteworthy that a decision tree can be expressed as a set of classification rules: each path from the root to a leaf expresses a set of conditions and the conclusion of the rule is given by the label of the leaf. Figure 9 gives the top of a decision tree built on the `zoo` dataset.

A decision tree is thus the symbolic expression of a partition of the input space. We can only obtain partitions *parallel to the axes* insofar as the tests on numerical variables are generally of the form $X \geq \theta$. Each subspace in the partition is then labelled, usually by the majority class of the observations in this subspace. When building a decision tree, each new test performed on a node refines the partition by splitting the corresponding subspace. Learning consists in finding such a partition. The global inductive criterion is replaced by a local criterion, which optimizes the homogeneity of the nodes of the partition, where the homogeneity is measured in terms of the proportion of observations of each class. The most used criteria are certainly the *information gain* used in C5.0 (Quinlan 1993; Kotsiantis 2007) and

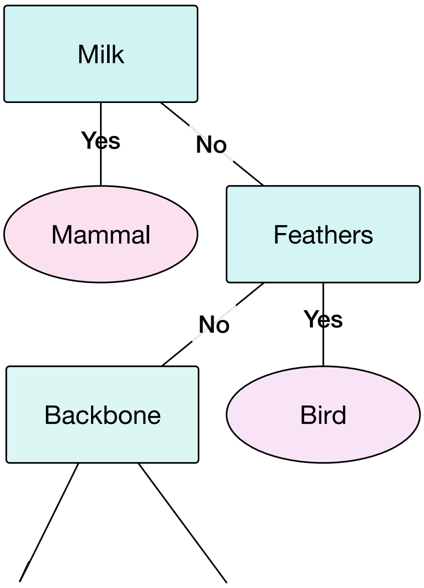


Fig. 9 A decision tree on the zoo dataset

based on the notion of entropy or the *Gini index*, used in Cart (Breiman et al. 1984). Techniques for pruning the built tree are then applied in order to avoid overfitting.

This algorithm is of reduced complexity: $\mathcal{O}(m \cdot d \cdot \ log(m))$ where m is the size of the learning sample and d the number of descriptive attributes. Moreover, an induction tree is generally easy to interpret (although this depends on the size of the decision tree). This is a typical example of a *divide and conquer* algorithm with a greedy exploration.

7.2.3 Inductive Logic Programming

Inductive Logic Programming (ILP) has been the subject of particular attention since the 1980s. Initially studied for the synthesis of logical programs from examples, it has gradually been oriented towards learning knowledge from relational data, thus extending the classical framework of data described in vector spaces and allowing to take into account relations in the data. When developed for the synthesis of logical programs, a key problem was learning recursive or mutually recursive concepts, whereas learning knowledge requires to take into account quantitative and uncertain information.

One interests of ILP is the possibility to integrate knowledge domain, allowing to obtain more interesting concept definitions. For example, if one wishes to learn the concept of `grandfather` from examples of persons linked by a father-mother relationship, introducing the concept of `parent` will allow a more concise definition of the concept. The definition of θ-subsumption defined in Sect. 7.1.2 must then be modified accordingly.

One of the major challenges ILP has to face is the complexity due to the size of the search space and to the coverage test. To overcome this problem, syntactic or semantic biases have been introduced, thus allowing to reduce the search space.

Another idea is to take advantage of the work carried out in propositional learning. In this context, a rather commonly used technique, called propositionalization, consists in transforming the first-order learning problem into a propositional or attribute-value problem (see an example in Zelezný and Lavrac 2006). The difficulty then lies in the construction of relevant characteristics reflecting the relational character of the data and minimizing the loss of information.

The best known ILP systems are Foil (Quinlan 1996) and Progol (Muggleton 1995). As already mentioned, Foil iteratively builds rules covering positive examples and rejecting the negative ones. For building a rule, it adopts a top-down strategy: the most general clause (a rule without conditions) is successively refined to reject all negative examples. It is based on a greedy strategy relying on a heuristic, close to the information gain: it takes into account the number of instantiations that cover positive, respectively negative, examples. The quality of each refinement is measured and the best one is chosen without backtracking. This strategy suffers from a well known problem: it may be necessary to add refinements, corresponding to functional dependencies (for example introducing a new object in relation with the target object), which are necessary to construct a consistent clause, but which have a null information gain (they are true for all positive and all negative examples). Progol differs in the way clauses are built: the construction of a clause is driven by a positive example. More precisely, Prolog chooses a positive example and constructs the most specific clause covering this example–it is called the saturation; search is performed in the generalization space of this saturated clause. The results produced by Prolog depend on the order positive examples are processed. It was implemented in the system Aleph (see http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/aleph).

During the 1990s, it was shown that a large class of constraint satisfaction problems presented a *phase transition* phenomenon, namely a sudden variation in the probability of finding a solution when the parameters of the problem vary. It was also noted that finding a hypothesis covering (in terms of θ-subsumption) positive examples, but no negative ones can be reduced to a constraint satisfaction problem. It was then shown empirically that a phenomenon of phase transition actually may appear in ILP. This discovery has been extended to certain types of problems in grammatical inference. (Saitta et al. 2011) is devoted to the study of Phase Transitions in Machine Learning.

Finally, it should be noted that while supervised classification has long been the main task studied in ILP, there are also work on searching for frequent patterns in relational databases, or on subgroup discovery. Important references include (Lavrac and Dzeroski 1994; Dzeroski and Lavrac 2001; Raedt 2008; Fürnkranz et al. 2012).

Inductive Logic Programming is still an active research field, mainly these recent years in Statistical Relational Learning (Koller and Friedman 2009; Raedt et al. 2008). The advent of deep learning has launched a new research stream, aiming at encoding relational features into neural networks, as shown in the last International Conferences on Inductive Logic Programming (Lachiche and Vrain 2018).

7.2.4 Mining Frequent Patterns and Association Rules

Searching for frequent patterns is a very important Data Mining problem, which has been receiving a lot of attention for twenty years. One of its application is the discovery of interesting association rules, where an association rule is a rule $I \rightarrow J$ with I and J two disjoint patterns. The notion of pattern is fundamental: it depends on the input space $\mathscr{X}$, and more precisely on the representation language of $\mathscr{X}$. It may be a set of items, called an *itemset*, a list of items when considering sequential data, or more complex structures, such as graphs. A pattern is said to be *frequent* if it occurs in the database a number of times greater than a given threshold (called the *minimal support threshold*).

As in the version space, it is quite natural to rely on a generality relation between patterns. A pattern is more general than another when it occurs in more examples in the database. This is yet the notion of coverage defined in Sect. 7.1. When a pattern is a conjunction (or a set) of elementary expressions taken in a fixed vocabulary set $\mathscr{V}$ ($\mathscr{V}$ can be a set of Boolean features, a set of pairs (attribute, value), ...), the generalization (resp. specialization) operator is the removal (resp. addition) of a term in the pattern. The pattern space is then ordered by the inclusion relation, modeling the generality relation ($h_1 \geq h_2$ if $h_1 \subseteq h_2$) and it has a lattice structure. This ordering induces an anti-monotonous property, which is fundamental for pruning the search space: the specialization of a non frequent pattern cannot be frequent, or in other terms, when a pattern is not frequent it is useless to explore its specializations, since they will be non frequent too.

This observation is the basis of Apriori algorithm (Agrawal and Srikant 1994), the first and certainly the most well-known algorithm for mining frequent itemsets and solid association rules. It is decomposed into two steps: mining frequent itemsets and then building from these itemsets association rules, which are frequent by construction. The confidence of the association rules can then be computed from the frequency of the itemsets. The first step is the most difficult one since the search space is 2^d, (with $d = |\mathscr{V}|$) and computing the frequency of itemsets require to go through the entire database.

Apriori performs a breadth-first search in the lattice, first considering 1-itemsets, then 2-itemsets and so on, where a l-itemset is an itemset of size l. At each level, the support of all l-itemsets is computed, through a single pass in the database. To prune the search space, the anti-monotonicity property is used: when an itemset is not frequent, its successors can be discarded. The complexity depends on the size of the database and on the number of times it is necessary to go through the database. It depends also on the threshold: the lowest the minimum support threshold, the less pruning. Complexity is too high to be applicable on large data and with a low minimum support threshold and therefore new algorithms have been developed, based either on new search strategies (for instance partitioning the database, or sampling), or on new representations of the database, as for instance in FP-Growth (Han et al. 2004) or in LCM (Uno et al. 2004).

Another source of complexity is the number of frequent itemsets that are generated. Condensed representations have been studied, they are usually based on closed patterns, where a pattern is closed if the observations that satisfy this pattern share only the elements of this pattern. It corresponds to the definition of concepts, underlying Formal Concept Analysis and described in Sect. 7.1.4. We can notice that the support of an itemset can be defined as the cardinal of its extent (the number of elements in the database that satisfy it) and we have interesting properties stating for instance that two patterns that have the same closure have the same support, or that if a pattern is included in another pattern and has the same support then these two patterns have the same closure. These properties allow one to show that the set of closed patterns with their support are a condensed representation of all the itemsets with their support, and therefore only closed patterns have to be stored in memory (see for instance Zaki 2000; Bastide et al. 2000).

Itemset mining has been extended to deal with structured data, such as relational data, sequential data, graphs. All these structures are discrete. A remaining difficult problem is pattern mining in the context of numeric data, since learning expressions such as $(age > 60) \wedge (HDL\text{-}cholesterol > 1.65\ \text{mmol/L})$ require to learn the threshold (60, 1.65 for instance) in a context where the structure is not fixed and has to be learned simultaneously.

8 Probabilistic Models

So far we have mainly addressed the geometric and symbolic views of Machine Learning. There is another important paradigm in Machine Learning, that is based on a probabilistic modeling of data. It is called *generative* in the sense that it aims at inducing the underlying distribution of data and given that distribution, it is then possible to generate new samples following the same law. We have addressed it in the context of clustering in Sect. 4.3 but it is also widely used in supervised learning with the naive Bayesian classifier. Generative and discriminative approaches differ, since as stated in (Sutton and McCallum 2012), a discriminative model is given by the conditional distributions $\mathsf{p}(y|\mathbf{x})$, whereas a generative model is given by $\mathsf{p}(\mathbf{x}, y) = \mathsf{p}(y|\mathbf{x}) \times \mathsf{p}(\mathbf{x})$.

Let us consider a sample $\mathscr{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, where $\mathbf{x}_i$ is described by d features X_l, $l = 1 \ldots d$, with domain D_l and y_i belongs to a discrete set $\mathscr{Y}$ (the set of labels or classes). The features $X_l, l = 1 \ldots d$ and the feature Y corresponding to the class of the observations can be seen as random variables. Let X denote the set of features X_l. Given a new observation $\mathbf{x} = (x_1, \ldots, x_d)$ the Bayesian classifier assigns to $\mathbf{x}$ the class y in $\mathscr{Y}$ that maximizes $\mathsf{p}(Y = y|X = \mathbf{x})$. Thus we have:

$$h : \mathbf{x} \in \mathscr{X} \mapsto argmax_{y \in \mathscr{Y}} \mathsf{p}(Y = y|X = \mathbf{x}) \tag{8}$$

Using Bayes theorem $\mathsf{p}(Y = y|X = \mathbf{x})$ can be written $\frac{\mathsf{p}(X=\mathbf{x}|Y=y)\mathsf{p}(Y=y)}{\mathsf{p}(X=\mathbf{x})}$. Since the denominator is constant given $\mathbf{x}$, it can be forgotten in the argument of argmax, thus leading to the equivalent formulation

$$h : \mathbf{x} \in \mathscr{X} \mapsto argmax_{y \in \mathscr{Y}} \mathbf{p}(X = \mathbf{x}|Y = y)\mathbf{p}(Y = y) \tag{9}$$

Learning consists in inferring an estimation of the probabilities given the data. When the input space $\mathscr{X}$ is described by d discrete features X_l, $l = 1 \dots d$, with domain D_l, this means learning $\mathbf{p}(X_1 = x_1, \dots, X_d = x_d|Y = y)$ for all possible tuples and $\mathbf{p}(Y = y)$ for all y, leading to $(|D_1| \times \dots \times |D_d| + 1) \times |Y|$ probabilities to estimate. This would require many observations to have reliable estimates. An assumption is needed to make the problem feasible: the features are assumed to be independent conditionally to the class, that means:

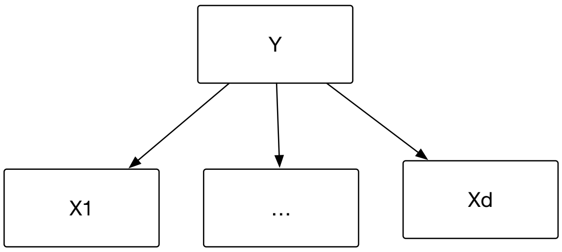
$$\mathbf{p}(X_1 = x_1, \dots, X_d = x_d|Y = y) = \mathbf{p}(X_1 = x_1|Y = y) \times \dots \times \mathbf{p}(X_d = y_d|Y = y)$$

This leads to the definition of the naive Bayesian classifier

$$h : \mathbf{x} = (x_1, \dots, x_d) \mapsto argmax_{y \in \mathscr{Y}} \Pi_{l=1}^{d} \mathbf{p}(X_l = x_l|Y = y) \times \mathbf{p}(Y = y) \tag{10}$$

Nevertheless this assumption is often too simple to accurately model the data. More complex models, grouped under the term *graphical models* (Koller and Friedman 2009) have been introduced to take into account more complex dependencies between variables. Dependencies between variables are represented by a graph whose nodes are the variables and whose edges model the dependencies. There exist two main families of models: *Bayesian networks* that are acyclic oriented graphs associating to each node the conditional probability of this node given its parents and *Conditional Random Fields* (CRF), that are non oriented graphs in which a variable is independent from the other ones, given its neighbors. The naive Bayesian classifier is a special case of a Bayesian network, as illustrated by Fig. 10. A third model, Hidden Markov Model (HMM) is also frequently used for sequential data: a HMM is defined by a directed graph, each node of the graph represents a state and can emit a symbol (taken from a predefined set of observable symbols), two families of probabilities are associated to each state: the probabilities of emitting a symbol s, given this state ($\mathbf{P}(s|n)$) and the probability of moving in state n', given state n ($\mathbf{P}(n'|n)$). A HMM models a Markovian process: the state in which the process is at time t depends only of the state reached at time $t - 1$ ($\mathbf{P}(q_t = n'|q_{t-1} = n)$), it also assumes that the symbol emitted at time t depends only on the state the automaton

Fig. 10 Dependency between variables for the naive Bayesian classifier



has reached at time t. Let us notice that a HMM can be modeled by a CRF: indeed the probability distribution in a CRF is often represented by a product of factors put on subsets of variables, the probability $\mathbf{P}(n'|n)$ and $\mathbf{P}(s|n)$ are easily converted into factors. CRFs, by means of factors, allow the definition of more complex dependency between variables and this explains why it is now preferred to HMMs in natural language processing. An introduction to CRF and a comparison with HMM can be found in (Sutton and McCallum 2012).

Learning graphical models can be decomposed in two subproblems: either the structure is already known and the problem is then to learn the parameters, that is the probability distribution, or the structure is not known and the problem is to learn both the structure and the parameters (see Koller and Friedman 2009, and Pearl 1988). The first problem is usually addressed by methods such as likelihood maximization or such as Bayesian maximization a priori (MAP). Different methods have been developed for learning the structure. For more information, the reader should refer to chapter "Belief Graphical Models for Uncertainty Representation and Reasoning" of this volume, devoted to graphical models.

Statistical Relational Learning (Raedt et al. 2008; Getoor and Taskar 2007) is an active research stream that aims at linking Inductive Logic Programming and probabilistic models.

9 Learning and Change of Representation

The motivation for changing the input space $\mathscr{X}$ is to make the search for regularities or patterns more straightforward. Changes can be obtained through unsupervised learning or through supervised learning, guided by the predictive task to solve.

Unsupervised learning is often used in order to estimate density in the input space, or to cluster the data into groups, to find a manifold where most of the data lies near, or to make denoising in some way. The overall principle is generally to find a representation of the training data that is the simplest while preserving as much information about the examples as possible. Of course, the notion of "simplicity" is multifarious. The most common ways of defining it are: lower dimensional representations, sparse representations, and independent representations.

When looking for *lower dimensional representations*, we are interested in finding smaller representations that keep as much useful information as possible about the data. This is advantageous because it tends to remove redundant information and generally allows for more efficient processing. There are several ways to achieve this. The most straightforward is to perform feature selection. Another one is to change the representation space and to project the data into a lower dimensional space.

An altogether different idea is to use a high dimensional representation space, but to make sure that each piece of data is expressed using as few of these dimensions, or descriptors, as possible. This is called a *sparse representation*. The idea is that each input should be expressible using only a few "words" in a large dictionary. These dictionaries are sometimes called overcomplete representations from earlier studies on the visual system (Olshausen and Field 1996).

Independent representations seek to identify the sources of variations, or latent variables, underlying the data distribution, so that these variables are statistically independent in some sense.

10 Other Learning Problems

10.1 Semi-supervised Learning

Learning to make prediction, that is to associate an input to an output, requires a training set with labelled inputs, of the form $(\mathbf{x}_i, y_i)_{(1 \leq i \leq m)}$. The larger the training set, the better the final prediction function produced by the learning algorithm. Unfortunately, obtaining labels for a large set of inputs is often costly. Think of patients at the hospital. Determining the right pathology from the symptoms exhibited by a patient requires a lot of expertise and often costly medical examinations. However, it is easy to get a large data set comprised of the description of patients and their symptoms, without a diagnostic. Should we ignore this (potentially large) unsupervised data set?

Examining Fig. 11 suggests that this might not be the case. Ignoring the unlabelled examples would lead a linear SVM to learn the decision function on Fig. 11 (left). But this decision function sure does feel inadequate in view of the unlabelled data points in Fig. 11 (right). This is because, it seems reasonable to believe that similarly labelled data points lie in “clumps”, and that a decision function should go through low density region in the input space. If we accept this assumption as a prior bias, then it becomes possible to use unlabelled data in order to improve learning. This is the basis of semi-supervised learning (Chapelle et al. 2009).

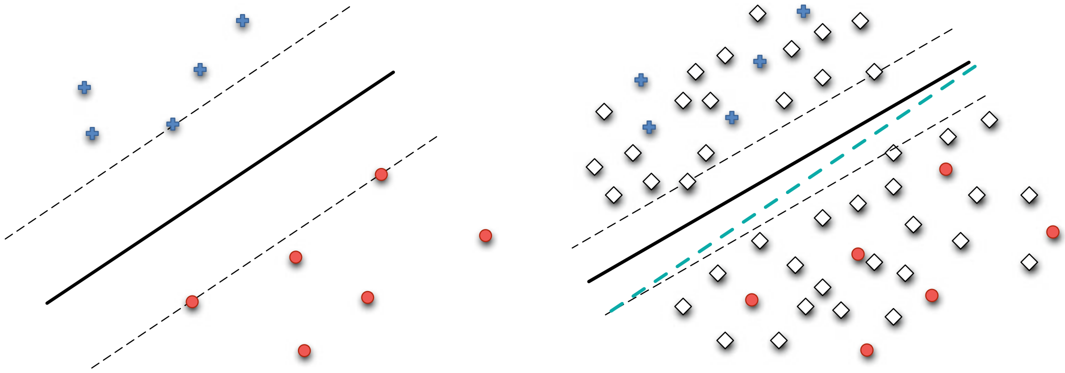


Fig. 11 Given a few labelled data points, a linear SVM would find the decision function on the left. When unlabelled data points are added, it is tempting to change the decision function to better reflect the low density region between the apparent two clouds of points (right)

Semi-supervised learning is based on the assumption that the unlabelled points are drawn from the same distribution as the labelled ones, and that the decision function lies in the low density region. If any of these assumption is erroneous, then semi-supervised learning can deteriorate the learning performance as compared to learning from the labelled data points alone.

10.2 Active Learning

The learning scenarios we have presented so far suppose that the learner is passively receiving the training observations. This hardly corresponds to learning in natural species and in humans in particular who are seekers of information and new sensations, and this seems wasteful when, sometimes, a few well chosen observations could bring as much information as a lot of random ones. Why not, then, design learning algorithms that would actively select the examples that seem the most informative? This is called *active learning*.

Suppose that inputs are one dimensional real valued in the range [0, 100], and that you know that their label is decided with respect to a threshold: all data points of the same class ('$+$' or '$-$') being on one side of the threshold (see Fig. 12). If you can ask the class of any data point in [0, 100], then you start by asking the class of the point '50', and of the point '0'. If they are of the same class, you should now concentrate on the interval (50, 100], and ask for the class of the point '75', otherwise concentrate on the interval (0, 50) and test the point '25'. By systematically halving the interval at each question, you can be ε-close to the threshold with $\mathscr{O}(log_2\, 1/\varepsilon)$ questions, whereas you should ask $\mathscr{O}(1/\varepsilon)$ random questions in order to have the same precision on the threshold.

In this example, active learning can bring an exponential gain over a passive learning scenario. But is this representative of what can be expected with active learning? In fact, four questions arise:

1. Is it possible to *learn* with active learning concept classes *that cannot be learned with passive learning*?
2. What is the *expected gain* if any in terms of number of training examples?
3. *How to select* the best (most informative) training examples?
4. *How to evaluate* learning if the assumption of equal input distribution in learning and in testing is no longer valid, while it is the foundation of the statistical theory of learning?

The answer of question (1) is that the class of concepts learnable in the active learning scenario is the same as with the passive scenario. For question (2), we have

$$h_w(x) = \begin{cases} 1 & \text{if } x \geq w \\ 0 & \text{if } x < w \end{cases}$$

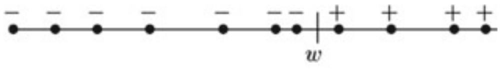


Fig. 12 Active learning on a one-dimensional input space, with the target function defined by h_w

exhibited, in the example, a case with an exponential gain. However, there exist cases with no gain at all. On average, it is expected that active learning provides an advantage in terms of training examples. This, however, should be put in regards to the computation gain. Searching the most informative examples can be costly. This is related to question (3). There exist numerous heuristics to select the best examples. They all rely on some prior assumptions about the target concept. They also differ in their approach to measure the informativeness of the examples. Question (4) itself necessitates that further assumptions be made about the environment. There are thus different results for various theoretical settings (See Chapter "Statistical Computational Learning").

10.3 Online Learning

In the batch learning scenario, the learner is supposed to have access to all the training data at once, and therefore to be able to look at it at will in order to extract decision rules or correlations. This is not so in online learning where the data arrives sequentially and the learner must take decisions at each time step. This is what happens when learning from data streams (Gama 2010). In most cases, the learner cannot store all the past data and must therefore compress it, usually with loss. Consequently, the learner must both be able to adapt its hypothesis or model of the world iteratively, after each new example or piece of information, and be able to decide what to forget about the past data. Often, the learner throws out each new example as soon as it has been used to compute the new hypothesis.

Online learning can be used in stationary environments, when data arrives sequentially or when the data set is so large that the learner must cope with it in piecemeal fashion. It can as well be applied in non stationary environment, which complicates things since, in this case, the examples have no longer the same importance according to their recency.

Because there can no longer be a notion of expectation, since the environment may be changing with time, the empirical risk minimization principle, or the maximization of likelihood principles can no longer be used as the basis of induction.

In the classical "batch scenario", one tries several values of the hyperparameters that govern the hypothesis space being explored, and for each of them record the best hypothesis, the one minimizing:

$$\hat{h} = \operatorname*{argmin}_{h \in \mathscr{H}} R_{\text{Reg}}(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i, y_i)) + \Omega_{\text{hyperparameters}}(\mathscr{H})$$

where $\Omega_{\text{hyperparameters}}(\mathscr{H})$ penalizes the hypothesis space according to some prior bias encoded by the hyper parameters. The best hyper parameters are found using the validation set (see Sect. 3.4).

In on line learning, this is no longer the best hypothesis space that is looked for, but rather the best adaptive algorithm, the one that best modifies the current hypothesis after each new arriving piece of information. Let us call this algorithm $\mathscr{A}_{\text{adapt}}$. Then the best adaptive algorithm is the one having the best performance on the last T inputs, if T is supposedly relevant as a window size:

$$\mathscr{A}^*_{\text{adapt}} = \underset{\mathscr{A}_{\text{adapt}} \in \mathscr{A}}{\text{argmin}} \left\{ \frac{1}{T} \sum_{t=1}^{T} \ell\big(h_t(\mathbf{x}_t), y_t\big) \right\}$$

In fact $\mathscr{A}^*_{\text{adapt}}$ is found using several "typical" sequences of length T, that are supposed to be representative of the sequences the learning system will encounter.

Numerous heuristical online learning systems are based on this general principle.

But how do you get by if no regularity is assumed about the sequence of arriving data? Can you still devise a learning algorithm that can cope with any sequence of data whatsoever, even if it is ruled by an adversary who tries to maximize your error rate? And, if yes, can you still guarantee something about the performance of such an online learning system?

This is the province of the *online learning theory*. Now, the performance criterion is called a *regret*. What we try to achieve is to have the learner to be competitive with the best fixed predictor $h \in \mathscr{H}$. The *regret* measures how "sorry" the learner is, in retrospect, not to have used the predictions of the best hypothesis $h*$, in retrospect, in $\mathscr{H}$.

$$R_T = \sum_{t=1}^{T} \ell(\widehat{y_t} - y_y) - \min_{h \in \mathscr{H}} \sum_{t=1}^{T} \ell(h(\mathbf{x}_t), y_t)$$

where $\widehat{y_t}$ is the guess of the online learner for the input $\mathbf{x}_t$.

Surprisingly, it is found that it is possible to devise learning algorithms with guarantees, meaning bounds, over the regret, whatever can be the sequence feed to the learner. One good reference about these algorithms is (Cesa-Bianchi and Lugosi 2006).

10.4 Transfer Learning

In the classical supervised learning setting, one seeks to learn a good decision function h from the input space $\mathscr{X}$ to the output space $\mathscr{Y}$ using a training set $S = \{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq m}$. The basis of the inductive step is to assume that the training data and future test data are governed by the same distribution $\mathbf{P}_{\mathscr{X}\mathscr{Y}}$. Often, however, the distributions are different. This may happen for instance when learning to recognize spam using data from a specific user and trying to adapt the learned rule to another user. The resulting learning problem is called *Domain Adaptation*. A further step is taken when one wishes to profit from a solved learning task in order to facilitate another different learning task, possibly defined on another input space $\mathscr{X}$. For instance, a system that knows how to recognize poppy fields in satellite images, might learn more efficiently to recognize cancerous cells in biopsy images than a system that must learn this from scratch. This is known as *transfer learning* (Pan and Yang 2010).

Formally, in transfer learning, there is a *Source domain* $\mathscr{D}_{\mathscr{S}}$ defined as the product of a source input space and a source output space: $\mathscr{X}_{\mathscr{S}} \times \mathscr{Y}_{\mathscr{S}}$. The source information can come either through a source training set $S_{\mathscr{S}} = \{(\mathbf{x}_i^{\mathscr{S}}, y_i^{\mathscr{S}})\}_{1 \leq i \leq m}$ or through a decision function $h_{\mathscr{S}} : \mathscr{X}_{\mathscr{S}} \rightarrow \mathscr{Y}_{\mathscr{S}}$, with or without a training set. If only the decision function $h_{\mathscr{S}}$ is available, this is called *hypothesis transfer learning*. Similarly, the *Target domain* $\mathscr{D}_{\mathscr{T}}$ is defined as a product of a target input space and a target output space: $\mathscr{X}_{\mathscr{T}} \times \mathscr{Y}_{\mathscr{T}}$. Often, it is assumed that the available target training data $S_{\mathscr{T}} = \{(\mathbf{x}_i^{\mathscr{T}}, y_i^{\mathscr{T}})\}_{1 \leq i \leq m}$ is too limited to allow a learning algorithm to yield a good decision function $h_{\mathscr{T}} : \mathscr{X}_{\mathscr{T}} \rightarrow \mathscr{Y}_{\mathscr{T}}$. In some scenarios, the target data is assumed to be unlabeled: $S_{\mathscr{T}} = \{\mathbf{x}_i^{\mathscr{T}}\}_{1 \leq i \leq m}$.

Two questions arise then. *First*, can the knowledge of the source hypothesis $h_{\mathscr{S}}$ help in learning a better decision function in $\mathscr{D}_{\mathscr{T}}$ than would be possible with the training set $\mathscr{S}_{\mathscr{T}}$ alone? *Second*, if yes, how can this be achieved?

Transfer learning is becoming a hot topic in machine learning, both because a growing number of applications could benefit from it, and because it demands new theoretical developments adapted to non stationary environments.

10.5 Learning to Rank

Learning to rank (Li 2011) is related to descriptive learning in that the goal is not to make predictions about new unknown instances, but to order the set of available data according to some "relevance". It is however related to supervised learning in that, usually, there are supervised information in the training data, for instance, that such instance is preferred to some other one.

A typical application is the ordering of the results of a search engine according to their relevance to a query, and, possibly, to a user.

One approach is to first define a loss function that measures the difference between the true ranking of a set of instances and the one produced by the system, and then to apply the empirical risk minimization (ERM) principle to find a good ranking function on the training sets (several sets of which the true ranking is known).

For instance, linear predictors for ranking can be used. In this technique, assuming that $\mathscr{X} \subset \mathbb{R}^d$ for any vector $\mathbf{w} \in \mathbb{R}^d$, a ranking function can be defined as:

$$h_{\mathbf{w}}\big((\mathbf{x}_1, \ldots, \mathbf{x}_r)\big) = \big(\langle \mathbf{w}, \mathbf{x}_1 \rangle, \ldots, \langle \mathbf{w}, \mathbf{x}_r \rangle\big)$$

The elements $\mathbf{x}_i (1 \leq i \leq r)$ can then be ordered according to the values $\langle \mathbf{w}, \mathbf{x}_i \rangle$.

There are other learning algorithms, some of which are based on learning binary classifiers that take two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ and that return the output '$+$' if the first argument $\mathbf{x}_i$ is before $\mathbf{x}_j$, and '$-$' otherwise.

10.6 Learning Recommendations

Learning to make recommendations is somewhat related to learning to rank, but aims at extrapolating the relevance to new instances, not already seen by the user. For instance, a system can learn to make recommendations about movies that should interest someone based on his/her history of seeing movies and the appreciations that he/she reported.

One way to do that is to describe the items to be rated (e.g. movies), and thus recommended or not, on the basis of attributes, and then to learn to associate the resulting descriptions with appreciations, or grades. This is called *content-based recommendation*. One problem is to find relevant attributes. Another is that such recommending systems can only use the past history of each customer or user. Information is not shared among users.

Another approach is to suppose that if another user has expressed ratings that are close to mine for a set of items that we both rated, then it is likely that I would rate similarly to this user other items that I have not yet seen. In this way, it is possible to capitalize on the vast data sets of preferences expressed by the whole community of users. The idea is to compute the similarity (or dissimilarity) with other users based on sets of items rated in common, and then to extrapolate the way I would rate new items based on these similarities and the rates that these other users gave to the new items. This is called *collaborative filtering*.

Among many techniques dedicated to solve the problem of collaborative filtering, the prominent one currently operates using matrix completion. The idea is to consider the matrix $\mathbf{R}$ defined over $user \times item$ with each element $R_{(i,j)}$ of the matrix containing the rate given by $user_i$ to $item_j$. When no rate has been given the element contains 0.

This matrix is usually very sparse (few values $R_{(i,j)} \neq 0$) since users have generally tested only a few tens or at most hundreds of items. The goal is then to complete this matrix, filling the missing values. A range of techniques can be used for this. The most classical relies on the Singular Value Decomposition (SVD), which, in a way, expresses the fact that the columns (or the rows) of this matrix are not independent.

If these techniques are rather powerful, they nonetheless give results that are somewhat less than satisfactorily. This is due to several factors, including the fact that the missing values are not randomly distributed as would demand SVD, the performance measures such as the root mean square error (RMSE) give the same importance to all entries $R_{(i,j)}$ while users are interested in high value ratings, and in fact are not interested in accurate ratings, but rather on the respective ratings given to the most interesting items. Furthermore, recommendations are highly context dependent: for instance, the same item can be quite interesting one day, and not the day after because a similar item has been purchased.

For all these reasons, new approaches for recommendation systems are to be expected in the years to come (see for instance Jannach et al. 2016).

10.7 Identifying Causality Relationships

It might be easy to come up with correlations such as people who eat ice-creams wear swimming suits. But should we rely on a rule that says: "to make people eat ice-cream, make them wear swimming suits"? Clearly, correlations are not causal relationships, and believing they are can lead to disastrous conclusions. In order to be able to "act" on some phenomenon, it is crucial to identify the causes of the phenomenon. Unfortunately, almost all of the current predictive learning systems are geared to discover correlations, but not causal links, and going from correlations to causality is not a trivial problem. It is indeed mostly an open problem.

Judea Pearl has shown that the standard concepts, and notations, of statistics are not able to capture causality (Pearl 2009). Some approaches suppose that a graph of potential causal links is provided by an expert before a learning algorithm tries to identify the true causal links together with their directions and intensities (see chapter "A Glance at Causality Theories for Artificial Intelligence" of Volume 1 for an extended discussion of causality within Artificial Intelligence, and the book by (Peters et al. 2017) for a thorough discussion of causality and machine learning).

Recently an interesting idea has been proposed where the data points corresponding to some phenomenon are analyzed by a classifier (in this work, a deep neural network) which, after learning on synthetic data reflecting causality between variables, can recognize if some variable is likely to cause the value of some other one (Lopez-Paz et al. 2016).

11 Conclusion

Machine learning is all the rage in artificial intelligence at the moment. Indeed, because it promises to eliminate the need to explicitly code the machines by hand when it suffices to feed the machines with examples to allow it to program itself, machine learning seems the solution to obtain high performing systems in many demanding tasks such as understanding speech, playing (and winning) games, problem solving, and so on.

And, truly, machine learning has demonstrated impressive achievements in recent years, reaching superhuman performances in many pattern recognition tasks or in game playing. Autonomous vehicles are, so to speak, around the corner, while Watson, from IBM, and other similar automatic assistant systems that can sift through millions of documents and extract information in almost no time are deemed to profoundly change the way even high level professions, such as law making or medicine, will be carried out in the years to come.

Still for all these breakthroughs and the accompanying hype, today's machine learning is above all the great triumph of pattern recognition. Not much progress has been made in the integration of learning and reasoning since the 1980s. Machines are very limited in learning from unguided observations and, unlike less that 2 years aged children, they are very task-focused, lack contextual awareness, and look for statistical correlations when we look for casual relationships. Progress in reinforcement learning and in neural networks have been highly touted, and in part rightly so, but the improvements in performance are due in large parts to gains in computational power, and in the quantity of training examples rather than on new breakthroughs in concepts, even though it is indisputable that new ideas have been generated.

The field of machine learning is therefore all but a finished, polished, or even a mature domain. Revolutionary ideas are yet to come. They have to come. Accordingly, let us conclude this chapter with an intriguing idea. Maybe the key of true intelligence is not learning per se, but teaching. Infants, and, later, adults, share knowledge in an innate compulsion. We share information and we share it with no immediate gain other than to remedy the gap of knowledge we perceive in others. It starts when a one-year old child see an object falling behind a piece of furniture and points to it to an adult who did not see where the object fell. Teaching is an universal instinct among humans. It is certainly intricately linked to our capacity of learning. We have models of what the others know or ignore, and we act, we teach, accordingly. In order to do so, we need to recognize the state of the world and the state of others, and we need to reason. Is the teaching ability, or rather the teaching instinct, the true frontier that machine learning must conquer? Learning/teaching, do we need to look at the two faces of a same coin in order to understand each one? Intriguing idea isn't it, Watson!

References

Aggarwal CC (2015) Data mining: the textbook. Springer Publishing Company Incorporated, Berlin

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. Very large data bases (VLDB-94). Santiage, Chile, pp 487–499

Aloise D, Hansen P, Liberti L (2012) An improved column generation algorithm for minimum sum-of-squares clustering. Math Program 131(1–2):195–220

Amarel S (1968) On representations of problems of reasoning about actions. Mach Intell 3(3):131–171

Ankerst M, Breunig MM, Kriegel H, Sander J (1999) OPTICS: ordering points to identify the clustering structure. In: SIGMOD 1999, proceedings ACM SIGMOD international conference on management of data, June 1–3, 1999, Philadelphia, Pennsylvania, USA, pp 49–60. https://doi.org/10.1145/304182.304187

Arthur D, Vassilvitskii S (2007) k-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7–9, 2007, pp 1027–1035. http://dl.acm.org/citation.cfm?id=1283383.1283494

Bastide Y, Pasquier N, Taouil R, Stumme G, Lakhal L (2000) Mining minimal non-redundant association rules using frequent closed itemsets. In: Computational logic, pp 972–986

Bezdek JC (1981) Pattern recognition with fuzzy objective function algorithms. Kluwer Academic Publishers, Norwell

Bishop CM (2006) Pattern recognition and machine learning. Springer, Secaucus

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Breiman L, Friedman J, Olshen R, Stone CJ (1984) Classification and regression trees. Wadsworth and Brooks/Cole Advanced Books and Software

Brusco M, Stahl S (2005) Branch-and-bound applications in combinatorial data analysis (Statistics and computing), 1st edn. Springer, Berlin

Busygin S, Prokopyev OA, Pardalos PM (2008) Biclustering in data mining. Comput OR 35:2964–2987

Cesa-Bianchi N, Lugosi G (2006) Prediction, learning, and games. Cambridge University Press, Cambridge

Chapelle O, Scholkopf B, Zien A (2009) Semi-supervised learning (chapelle O, et al eds; 2006). IEEE Trans Neural Netw 20(3):542

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297

Dao T, Duong K, Vrain C (2017) Constrained clustering by constraint programming. Artif Intell 244:70–94. https://doi.org/10.1016/j.artint.2015.05.006

de la Higuera C (2010) Grammatical inference: learning automata and grammars. Cambridge University Press, Cambridge

Dhillon IS, Guan Y, Kulis B (2004) Kernel k-means: spectral clustering and normalized cuts. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining, Seattle, Washington, USA, August 22–25, 2004, pp 551–556. https://doi.org/10.1145/1014052.1014118

Ding CHQ, He X (2005) On the equivalence of nonnegative matrix factorization and spectral clustering. In: Proceedings of the 2005 SIAM international conference on data mining, SDM 2005, Newport Beach, CA, USA, April 21–23, 2005, pp 606–610, https://doi.org/10.1137/1.9781611972757.70

du Merle O, Hansen P, Jaumard B, Mladenovic N (1999) An interior point algorithm for minimum sum-of-squares clustering. SIAM J Sci Comput 21(4):1485–1505

Dunn JC (1973) A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. J Cybern 3(3):32–57. https://doi.org/10.1080/01969727308546046

Dzeroski S, Lavrac N (eds) (2001) Relational data mining. Springer, Berlin

Ester M, Kriegel H, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the second international conference on knowledge discovery and data mining (KDD-96), Portland, Oregon, USA, pp 226–231. http://www.aaai.org/Library/KDD/1996/kdd96-037.php

Fisher DH (1987) Knowledge acquisition via incremental conceptual clustering. Mach Learn 2(2):139–172. https://doi.org/10.1007/BF00114265

Forgy E (1965) Cluster analysis of multivariate data: efficiency versus interpretability of classification. Biometrics 21(3):768–769

Fürnkranz J, Gamberger D, Lavrac N (2012) Foundations of rule learning. Springer, Berlin

Gama J (2010) Knowledge discovery from data streams. Chapman & Hall

Ganter B, Wille R, Franke C (1998) Formal concept analysis: mathematical foundations. Springer, Berlin

Ganter B, Stumme G, Wille R (eds) (2005) Formal concept analysis: foundations and applications. Springer, Berlin

Getoor L, Taskar B (eds) (2007) An introduction to statistical relational learning. MIT Press, Cambridge

Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Ghahramani Z, Welling M, Cortes C, Lawrence ND, Weinberger KQ (eds) Advances in neural information processing systems 27, Curran Associates, Inc., pp 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Getoor L, Taskar B (eds) (2007) An introduction to statistical relational learning. MIT Press

Halkidi M, Batistakis Y, Vazirgiannis M (2002) Clustering validity checking methods: part ii. SIGMOD Rec 31(3):19–27. https://doi.org/10.1145/601858.601862

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. SIGMOD Rec 29(2):1–12. https://doi.org/10.1145/335191.335372

Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min Knowl Discov 8(1):53–87

Han J, Kamber M, Pei J (2011) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco

Hansen P, Delattre M (1978) Complete-link cluster analysis by graph coloring. J Am Stat Assoc 73:397–403

Hansen P, Jaumard B (1997) Cluster analysis and mathematical programming. Math Program 79(1–3):191–215

Hastie T, Tibshirani R, Friedman JH (2009) The elements of statistical learning: data mining, inference, and prediction, 2nd edn. Springer series in statistics. Springer, Berlin

Hawkins D (1980) Identification of outliers. Monographs on applied probability and statistics. Chapman and Hall. https://books.google.fr/books?id=fb0OAAAAQAAJ

Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci 79(8):2554–2558

Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall

Jannach D, Resnick P, Tuzhilin A, Zanker M (2016) Recommender systems-: beyond matrix completion. Commun ACM 59(11):94–102

Japkowicz N (2011) Evaluating learning algorithms: a classification perspective. Cambridge University Press

Johnson S (1967) Hierarchical clustering schemes. Psychometrika 32(3):241–254

Kaufman L, Rousseeuw PJ (1990) Finding groups in data: an introduction to cluster analysis. Wiley, New York

Klein G, Aronson JE (1991) Optimal clustering: a model and method. Nav Res Logist 38(3):447–461

Kohonen T (ed) (1997) Self-organizing maps. Springer, New York Inc, Secaucus

Koller D, Friedman N (2009) Probabilistic graphical models. Principles and techniques. MIP Press

Kotsiantis SB (2007) Supervised machine learning: a review of classification techniques. Informatica 31:249–268

Lance GN, Williams WTA (1967) A general theory of classificatory sorting strategies: 1. Hierarchical systems 9

Lachiche N, Vrain C (eds) (2018) Inductive logic programming - 27th international conference, ILP 2017, Orléans, France, September 4–6, 2017, Revised selected papers, Lecture notes in computer science, vol 10759. Springer. https://doi.org/10.1007/978-3-319-78090-0

Lavrac N, Dzeroski S (1994) Inductive logic programming - techniques and applications. Ellis Horwood series in artificial intelligence. Ellis Horwood

Le Cun Y (1986) Learning process in an asymmetric threshold network. Disordered systems and biological organization. Springer, Berlin, pp 233–240

Le Cun Y, Boser BE, Denker JS, Henderson D, Howard RE, Hubbard WE, Jackel LD (1990) Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems, pp 396–404

Le Cun Y, Bengio Y et al (1995) Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks 3361(10):1995

Le Cun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444. https://doi.org/10.1038/nature14539

Levesque HJ, Brachman RJ (1987) Expressiveness and tractability in knowledge representation and reasoning. Comput Intell 3(1):78–93

Li H (2011) A short introduction to learning to rank. IEICE Trans Inf Syst 94(10):1854–1862

Li W, Han J, Pei J (2001) CMAR: accurate and efficient classification based on multiple class-association rules. In: Proceedings of the 2001 IEEE international conference on data mining, 29 November–2 December 2001, San Jose, California, USA, pp 369–376. https://doi.org/10.1109/ICDM.2001.989541

Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. In: Proceedings of the fourth international conference on knowledge discovery and data mining, AAAI Press, KDD'98, pp 80–86. http://dl.acm.org/citation.cfm?id=3000292.3000305

Lloyd SP (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28(2):129–136. https://doi.org/10.1109/TIT.1982.1056489

Lopez-Paz D, Nishihara R, Chintala S, Schölkopf B, Bottou L (2016) Discovering causal signals in images. arXiv:160508179

Madeira SC, Oliveira AL (2004) Biclustering algorithms for biological data analysis: a survey. IEEE/ACM Trans Comput Biol Bioinform 1:24–45. https://doi.org/10.1109/TCBB.2004.2, www.doi.ieeecomputersociety.org/10.1109/TCBB.2004.2

Michalski RS (1980) Knowledge acquisition through conceptual clustering: a theoretical framework and an algorithm for partitioning data into conjunctive concepts. Int J Policy Anal Inf Syst 4:219–244

Michalski RS, Stepp RE (1983) Automated construction of classifications: conceptual clustering versus numerical taxonomy. IEEE Trans Pattern Anal Mach Intell 5(4):396–410. https://doi.org/10.1109/TPAMI.1983.4767409

Miclet L (1990) Grammatical inference. In: Bunke H, Sanfeliu A (eds) Syntactic and structural pattern recognition theory and applications. World Scientific, Singapore

Minsky ML, Papert S (1988) Perceptrons, expanded ed. MIT Press, Cambridge, vol 15, pp 767, 776

Mitchell T (1982) Generalization as search. Artif Intell J 18:203–226

Mitchell T (1997) Machine learning. McGraw-Hill

Muggleton S (1995) Inverse entailment and progol. New Gener Comput 13(3&4):245–286

Ng AY, Jordan MI, Weiss Y (2001) On spectral clustering: analysis and an algorithm. In: Advances in neural information processing systems 14 [neural information processing systems: natural and synthetic, NIPS 2001, December 3–8, 2001, Vancouver, British Columbia, Canada], pp 849–856. http://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm

Ng RT, Han J (1994) Efficient and effective clustering methods for spatial data mining. In: VLDB'94, proceedings of 20th international conference on very large data bases, September 12–15, 1994, Santiago de Chile, Chile, pp 144–155. http://www.vldb.org/conf/1994/P144.PDF

Nie F, Wang X, Huang H (2014) Clustering and projected clustering with adaptive neighbors. In: The 20th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '14, New York, NY, USA - August 24–27, 2014, pp 977–986. https://doi.org/10.1145/2623330.2623726

Olshausen BA, Field DJ (1996) Natural image statistics and efficient coding. Netw Comput Neural Syst 7(2):333–339

Pan SJ, Yang Q (2010) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359

Park HS, Jun CH (2009) A simple and fast algorithm for k-medoids clustering. Expert Syst Appl 36:3336–3341

Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann

Pearl J (2009) Causal inference in statistics: an overview. Statist Surv 3:96–146. https://doi.org/10.1214/09-SS057

Peters J, Janzing D, Schölkopf B (2017) Elements of causal inference: foundations and learning algorithms. MIT Press

Plotkin G (1970) A note on inductive generalization. In: Machine intelligence, vol 5. Edinburgh University Press, pp 153–163

Qiu Q, Patel VM, Turaga P, Chellappa R (2012) Domain adaptive dictionary learning, pp 631–645

Quinlan J (1993) C4.5: programs for machine learning. Morgan Kauffman

Quinlan JR (1996) Learning first-order definitions of functions. CoRR. arXiv:cs.AI/9610102

Raedt LD (2008) Logical and relational learning. Springer, Berlin

Raedt LD, Frasconi P, Kersting K, Muggleton S (eds) (2008) Probabilistic inductive logic programming - theory and applications. Lecture notes in computer science, vol 4911. Springer, Berlin

Rao M (1969) Cluster analysis and mathematical programming 79:30

Rubinstein R, Bruckstein AM, Elad M (2010) Dictionaries for sparse representation modeling. Proc IEEE 98(6):1045–1057

Rumelhart DE, McClelland JL, Group PR et al (1987) Parallel distributed processing, vol 1. MIT Press, Cambridge

Saitta L, Giordana A, Cornuéjols A (2011) Phase transitions in machine learning. Cambridge University Press

Schölkhopf B, Smola A (2002) Learning with kernels. MIT Press

Shapire R, Freund Y (2012) Boosting: foundations and algorithms. MIT Press

Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489

Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A et al (2017) Mastering the game of go without human knowledge. Nature 550(7676):354

Suraj Z (2004) An introduction to rough sets theory and its applications: a tutorial. In: ICENCO'2004, Cairo, Egypt

Sutton C, McCallum A (2012) An introduction to conditional random fields. Found Trends Mach Learn 4(4):267–373. https://doi.org/10.1561/2200000013

Tosic I, Frossard P (2011) Dictionary learning. IEEE Signal Process Mag 28(2):27–38

Uno T, Kiyomi M, Arimura H (2004) LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: FIMI '04, proceedings of the IEEE ICDM workshop on frequent itemset mining implementations, Brighton, UK, November 1, 2004. http://ceur-ws.org/Vol-126/uno.pdf

van der Laag PR, Nienhuys-Cheng SH (1998) Completeness and properness of refinement operators in inductive logic programming. J Log Program 34(3):201–225. https://doi.org/10.1016/S0743-1066(97)00077-0, http://www.sciencedirect.com/science/article/pii/S0743106697000770

Vapnik V (1995) The nature of statistical learning theory. Springer, Berlin

Vega-Pons S, Ruiz-Shulcloper J (2011) A survey of clustering ensemble algorithms. IJPRAI 25(3):337–372. https://doi.org/10.1142/S0218001411008683

von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416. https://doi.org/10.1007/s11222-007-9033-z

Wagstaff K, Cardie C (2000) Clustering with instance-level constraints. In: Proceedings of the 17th international conference on machine learning, pp 1103–1110

Ward JH (1963) Hierarchical grouping to optimize an objective function. J Am Stat Assoc 58(301):236–244. https://doi.org/10.1080/01621459.1963.10500845, http://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845

Zaki MJ (2000) Generating non-redundant association rules. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20–23, KDD, pp 34–43

Zeleznỳ F, Lavrac N (2006) Propositionalization-based relational subgroup discovery with rsd. Mach Learn 62(1–2):33–63

Zhang C, Bengio S, Hardt M, Recht B, Vinyals O (2016) Understanding deep learning requires rethinking generalization. arXiv:161103530

Zhou ZH (2012) Ensemble methods: foundations and algorithms. CRC Press