Chapter 2

Supervised automatic learning

In this chapter, the supervised learning problem is formally defined. In the first section, the different notions and notations are introduced. Section 2.2 explains the way learning algorithms proceed. Some classical methods are described in section 2.3 and the way algorithms and models are evaluated are discussed in Section 2.4 and 2.5.

2.1 Definitions and notation

Loosely speaking, automatic learning aims at modeling and/or understanding relationships among a set of variables used to describe objects (or situations) encountered in some particular context, on the sole basis of some available observations of the values of these variables on a sample of already encountered objects. By modeling, we mean being able to predict the values of some of the variables (called output variables) in terms of the values of some other variables (the input variables). By understanding, we mean for example sorting out those input variables which influence (or are correlated with) some of the outputs, determining the type of influences, or constructing logical rules which allow to infer information about outputs given information about inputs.

In this thesis we will consider a restrictive version of this type of problem, using the so-called "object-attribute-value" representation and where objects are viewed as outcomes of a random experiment modeled using probability theory.

2.1.1 Universe, objects, attributes

Under these restrictions, the formal definition of a supervised learning problem starts by the definition of the universe $\mathcal{U}$ of all possible objects o. In a given context, each object o has a, generally unknown, probability $P(o)$ of being observed [1]. In practice, we do not have full information about the objects but only have access to a description of them by their attributes (which are measurable properties of objects, or which can be computed as a function of such properties). An attribute A is a function defined on the universe:

$$A(\cdot) : \mathcal{U} \mapsto \mathcal{A},$$

[1]Although most of the theory that we present can be extended to infinite and non-countable universes, for the sake of mathematical and notational convenience, we assume in this chapter that the universe is a finite (but generally very large) set of objects, and use the notation $P(o)$ instead of $P(\{o\})$ when considering the probability of an atomic event; under these conditions we can further assume that $P(o) > 0, \forall o \in \mathcal{U}$. In the context of our research, we found that the distinction between continuous and discrete probability spaces was not really fundamental. Although we will in some occasions use continuous random variables and densities, mainly in the context of illustrative examples, we will use the discrete notations in most cases.

where $\mathcal{A}$ is the codomain of the function $A(.)$. Equivalently, an attribute A is a random variable whose values are supposed to belong to the set $\mathcal{A}$ [2]. The probability distribution on the universe induces a probability distribution on $\mathcal{A}$ in the following way:

$$P(A = a) = \sum_{o \in \mathcal{U} | A(o) = a} P(o). \tag{2.1}$$

In the literature, attributes are also called features or variables. Usually two types of attributes are considered according to the structure of their codomains:

- **Symbolic attributes:** the codomain is a finite[3] and unstructured set of values, i.e. $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$.
- **Numerical attributes:** the codomain is a subset of the real axis $\mathbb{R}$.

These two general types are sufficient to handle most problems. Symbolic attributes in fact denote all discrete non ordered types of attributes, while the numerical attributes regroup ordered ones (discrete or not, like integer for example). Of course, many other types of attributes are possible. For example, in the third part of this dissertation, we will consider attribute values in the form of time series or temporal signals.

2.1.2 Learning sample

Supervised learning aims at modeling the relationship between some a priori selected input and output attributes. In this dissertation, we reduce this problem to the modeling of a *single* output variable which is denoted by Y. Besides, the vector of input attributes is denoted by $\underline{A} = (A_1, \ldots, A_n)$. This random vector takes its values in

$$\underline{\mathcal{A}} = \{(A_1(o), \ldots, A_n(o)) | o \in \mathcal{U}\} \subseteq \mathcal{A}_1 \times \ldots \times \mathcal{A}_n,$$

where $\mathcal{A}_i$ is the codomain of the attribute A_i. The joint probability distribution of input and output attributes is denoted by $P(\underline{A}, Y)$ and is determined like (2.1). The relationship between input and output attributes is in principle fully characterized by the conditional probability distribution $P(Y|\underline{A}) = P(\underline{A}, Y)/P(\underline{A})$ (if $P(\underline{A}) \neq 0$).

A learning sample[4] of size N, ls^N, is an N-tuple of objects randomly and independently drawn from $\mathcal{U}$, each one described by its input and output attribute values[5]:

$$ls^N = ((\underline{A}(o_1), Y(o_1)), \ldots, (\underline{A}(o_N), Y(o_N))) = ((\underline{a}_1, y_1), \ldots, (\underline{a}_N, y_N)).$$

ls^N is thus the realization of a random variable, denoted LS^N, distributed according to $P^N(\underline{A}, Y)$. In the sequel we will most often drop the superscript ^{N} from our notation, unless it is necessary to explicitly stress the dependencies on the sample size.

---

[2] We use upper case letters to denote random variables and lower case letters to denote values of random variables. Occasionally, we use the notation $A(.)$ to denote the corresponding function defined on $\mathcal{U}$. Finally, we use calligraphic upper case letter to denote subsets of objects or of values of random variables.

[3] Obviously, when $\mathcal{U}$ is finite all attributes have also a finite codomain.

[4] In the literature, the term "learning set" is often improperly used to denote the learning sample. Historically, this terminology originates from the AI community where learning algorithms were first introduced in a deterministic setting and thus duplicates in the learning sample were meaningless. Propagating this bad habit, we will sometimes improperly use the term "set" to denote a sample.

[5] In practice, some of the attribute values may be unknown for some objects. We exclude this possibility from our discussion and suppose that missing values have been replaced in some ad hoc way before the learning problem is formulated.

2.1.3 Supervised learning problem

The general problem of supervised learning is formally stated as follows:

> For any value of N and a learning set ls^N (a realization of LS^N) and without any a priori knowledge of the functions $P(.)$, $Y(.)$, or, $\underline{A}(.)$, find a function $f(.)$ defined on $\underline{\mathcal{A}}$ which minimizes the expected prediction error defined by[6]:
>
> $$Err(f) = E_{\underline{A},Y}\{L(Y, f(\underline{A}))\}, \tag{2.2}$$
>
> where $L(.,.)$ is a loss function which measures the discrepancy between its two arguments.

In other words, the goal of learning is to find a function of the input attributes which predicts at best the outcome of the output attribute. The goodness of the fit is evaluated according to a predefined loss function. In practice, this loss function depends on the type of values of the output attribute:

- **Classification problem:** Y is a symbolic variable taking its values in $\mathcal{Y} = \{c_1, \ldots, c_m\}$. The most common loss function is the 0-1 loss function $L(Y, f(\underline{A})) = 1(Y \neq f(\underline{A}))$. The error becomes the probability of mis-classification:

$$Err(f) = E_{\underline{A},Y}\{1(Y \neq f(\underline{A}))\} = P(Y \neq f(\underline{A})) \tag{2.3}$$

- **Regression problem:** Y is a real-valued variable, $\mathcal{Y} \subset \mathbb{R}$. The most commonly used loss function is the square loss $L(Y, f(\underline{A})) = (Y - f(\underline{A}))^2$ and the corresponding error is the mean square error:

$$Err(f) = E_{\underline{A},Y}\{(Y - f(\underline{A}))^2\}, \tag{2.4}$$

To distinguish a classification problem from a regression one and following the machine learning literature, we will often denote a discrete output attribute by C while Y will be reserved to a regression output or a general output attribute whatever its type.

2.1.4 Bayes model and residual error

For a given problem and loss function the best possible model, i.e. the target of a learning algorithm, can be derived in the following way (assuming complete knowledge of the conditional probability distribution $P(Y|\underline{A})$).

Since :

$$Err(f) = E_{\underline{A},Y}\{L(Y, f(\underline{A}))\} = E_{\underline{A}}\{E_{Y|\underline{A}}\{L(Y, f(\underline{A})\}\}, \tag{2.5}$$

the function f_B which minimizes $Err(f_B)$ is obtained by minimizing the inner expectation at each point $\underline{A}$ of the input space, i.e. taking:

$$f_B(\underline{a}) = \arg\min_{y'} E_{Y|\underline{a}}\{L(Y, y')\}. \tag{2.6}$$

[6] $E_X\{f(X)\}$ denotes the expectation of a function $f(.)$ with respect to the distribution of a random variable X; it is defined by

$$E_X\{f(X)\} = \sum_{x \in \mathcal{X}} P(X = x) f(x) = \sum_{o \in \mathcal{U}} P(o) f(X(o)),$$

where X is some random variable.

This function is called the *Bayes model* in the statistical pattern recognition literature. For the 0-1 loss function, it becomes the *Bayes classifier* :

$$\begin{aligned} f_B(\underline{a}) &= \arg\min_c E_{C|\underline{a}}\{1(C \neq c)\} &(2.7)\\ &= \arg\min_c P(C \neq c|\underline{a}) &(2.8)\\ &= \arg\max_c P(C = c|\underline{a}). &(2.9)\end{aligned}$$

The Bayes classifier thus predicts the most probable class at each point of the input space. For the square loss, a derivation of the inner expectation of 2.5 gives the following minimizer:

$$f_B(\underline{a}) = E_{Y|\underline{a}}\{Y\}. \tag{2.10}$$

The best prediction at the point $\underline{a}$ of the input space is thus the conditional expectation of Y given $\underline{a}$.

In both cases, the minimal error vanishes if and only if there exists a functional relationship g between Y and $\underline{A}$, i.e. $P(Y = y|\underline{A} = \underline{a}) = 1(y = g(\underline{a}))$. In this case, g and f_B coincide. In general, however, there exists some random deviation (or noise) around the Bayes model which translates into a non deterministic probability distribution $P(Y|\underline{A})$ and thus there exists a non null minimal attainable error, $Err(f_B)$, which is independent of the learning algorithm. This minimal achievable error is also called the *residual error* in supervised learning.

2.2 Learning algorithms

A learning algorithm is a (computable) function which takes as sole input a learning sample and outputs a (computable) classification or regression model. From the previous consideration, it appears that to solve the supervised learning problem, it would be sufficient to estimate, from the learning sample, the distribution $P(C|\underline{A})$ in classification or $P(Y|\underline{A})$ in regression and then define a model based on this estimate (e.g. by formula (2.9) or (2.10)). In the context of finite universes, a trivial solution would be to use frequency counts to estimate these probabilities. In practice, however, an accurate estimation of this distribution would require a very large learning sample which indeed needs to contain a lot of samples at each point of the input space. Actually, as the number of input variables grows the number of observations required by this trivial approach would grow in an exponential way.

Thus, to work with reasonable sample sizes and in large dimensional input spaces, learning algorithms must make some "smoothness" hypothesis of the function $f(.)$ over the input space and use neighbor points to make a prediction at a point $\underline{a}$. One classical way to do this is to restrict candidate models to a specified form and do optimization in this family of models in order to minimize the error.

2.2.1 Hypothesis space

The design of a learning algorithm thus begins by the definition of a hypothesis space $\mathcal{H}$ which contains potential candidate models h among which this algorithm will select one on the basis of the learning sample ls. In the sequel, we will provide several examples of hypothesis spaces, e.g. linear models, neural networks, or decision trees.

Note that for a particular learning problem, the best possible model belonging to a particular hypothesis space may or may not be close to the Bayes model of this problem.

The *approximation error* of a hypothesis space for a given learning problem can be defined by:

$$\min_{h \in \mathcal{H}} E_{\underline{A}}\{L(h(\underline{A}), f_B(\underline{A}))\}. \tag{2.11}$$

Thus, in practice it is often necessary to iterate in order to adjust the hypothesis space a posteriori to the problem specifics, so as to reduce this approximation error.

Of course, the larger the hypothesis space, the smaller the approximation error, and, in particular, if the hypothesis space contains models which are arbitrarily "close" to any function within a given class, then the hypothesis space is said to have the *universal approximation capability* in this class. Unfortunately, as was already suggested above in the context of the trivial frequency counting method, the larger the hypothesis space the more difficult it is to identify the optimal hypothesis from a finite sample, and the larger the so-called *estimation error*. This latter phenomenon is also called *overfitting*, or *bias/variance tradeoff* in the automatic learning literature. The design of good hypothesis spaces through the study of the tradeoff between approximation error and estimation error in the context of finite sample size is one of the main themes of modern automatic learning theory. It is also one of the main themes of this thesis, and is further elaborated in chapter 3 where the notions of *bias* and *variance* will be defined more precisely. For the sake of the present discussion, however, we assume that the hypothesis space is already fixed.

2.2.2 Empirical risk minimization

As the only available data is a finite sample of the universe $\mathcal{U}$, the value of the error $Err(h)$ for a hypothesis h can not be computed exactly. This error is thus first approximated by a summation over the objects of the learning set:

$$\hat{Err}(h, ls) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, h(\underline{a}_i)), \tag{2.12}$$

which is also called the *empirical risk* in the automatic learning literature.

Then, learning could consist in selecting a function in $\mathcal{H}$ minimizing this estimation of error, which is the so-called *empirical risk minimization principle.* A learning algorithm is thus fully characterized by its hypothesis space and by the way it optimizes the learning set error. The optimization can be deterministic (i.e. output always the same function of the hypothesis space in response to a fixed learning set) or random (i.e. may output different hypotheses for the same learning set).

In regression, the mean square error (2.4) is smooth with respect to variations of the output of the model being evaluated. Hence, if the hypothesis space is a parametric family and if the output of a hypothesis is smooth with respect to variations of the parameters, traditional optimization techniques (for example, gradient descent) can be used to find the model which minimizes the learning set error. In classification, the output of the model is discrete and direct optimization of the error is often difficult. So, many algorithms first compute an estimate $\hat{P}(C|\underline{a})$ of the conditional class probability distribution $P(C|\underline{a})$ and then use this estimate to make a prediction at each point according to:

$$f(\underline{a}) = \arg\max_{c} \hat{P}(C|\underline{a}). \tag{2.13}$$

There are several ways to find an estimate of $P(C|\underline{a})$. Some methods find a region $R(\underline{a})$ around a point $\underline{a}$ and then estimate probabilities by relative frequency counts of objects from the learning set appearing in these regions. As we will see below, decision trees and the K-nearest neighbors work that way. Another solution is to use a regression algorithm and the square error loss to find a model for a new series of numerical outputs derived from the classification output by:

$$Y_C^i = 1(C = c_i), i = 1, \dots, m. \tag{2.14}$$

Indeed, We have seen that, under square loss, a regression method tries to find an estimation of $E_{Y|\underline{a}}\{Y\}$. With Y_i^C as output, this amounts to estimating:

$$E_{Y_C^i|\underline{a}}\{Y_C^i\} = P(Y_C^i = 1|\underline{a}) = P(C = c_i|\underline{a}), \tag{2.15}$$

which yields an estimation of conditional class probabilities. This is a possible way to learn classification models with linear regression and neural networks.

2.2.3 Problem specific background knowledge

In our formalization of the supervised learning problem, the sole problem specific information which is used by the algorithm is the learning sample. There is presently research in automatic learning about methods able to exploit other problem specific information in the form of some constraints about the relation among attributes and/or output variable. For example, inductive logic programming (ILP) does this in a logical setting in the form of first order logic statements. Bayesian networks incorporate this information in a probabilistic setting, in the form of conditional independence assumptions. We will not consider these aspects in this thesis as we are focusing on methods which are not exploiting such side information explicitly.

2.3 Main classes of supervised learning algorithms

In this section, we briefly describe five well-known learning algorithms which will be used later in this dissertation. Although there exist of course much more than five learning algorithms, they cover a large spectrum of basic automatic learning algorithms. Linear models and neural networks are representative of parametric techniques. Parametric techniques make assumptions on the hypothesis spaces by means of a fixed number of parameters and optimize these parameters from the learning set. On the other hand, decision trees and k-NN are termed non parametric techniques since they do not make any assumption on the form of the Bayes model and can adapt the shape of the model they produce on the basis of the learning set. Decision trees are also the main representative of the family of symbolic machine learning techniques. The initial motivation behind this method is the induction of logical rules and while all other methods are statistical in nature, decision tree induction has been largely studied within the symbolic artificial intelligence community. Finally, Naive Bayes is a very simple instance of probabilistic models used for classification of which Bayesian networks are a generalization. The main characteristic of these latter probabilistic models with respect to other algorithms is that they provide full descriptive models (as opposed to predictive models) which induction is not directed by constraints of good performance in prediction.

Besides these basic automatic learning algorithms, most of the recent advances in machine learning are devoted to meta-machine learning or wrapper techniques. These are general techniques which are applied on the top of existing learning algorithms so as to improve their accuracy. For example, the variance reduction techniques discussed in subsequent chapters of this dissertation belong to this family of wrapper techniques. As they will be tackled latter, they will not be discussed here.

2.3.1 Linear models

$\mathcal{H}$ contains all linear combinations of input attributes:

$$h(\underline{a}) = w_0 + \sum_{i=1}^{n} w_i \cdot a_i. \tag{2.16}$$

This model can be extended to give a linear discriminant model for two-class problems in the following way:

$$h(\underline{a}) = \begin{cases} c_1 & \text{if } w_0 + \sum_{i=1}^{n} w_i \cdot a_i > 0 \\ c_2 & \text{otherwise} \end{cases}. \tag{2.17}$$

Several alternatives are possible to combine such binary linear discriminant functions to handle the multi-category case.

Learning consists in searching for the value of the vector of parameters $\underline{w} = (w_0, w_1, \ldots, w_n)$ which minimizes $\hat{Err}(h, ls)$. Although the optimal solution can be computed analytically in the regression case, there exist numerous gradient descent procedures which iteratively modify the vector of parameters in a direction which optimizes the error criterion. These algorithms differ in the loss function and in the way they correct $\underline{w}$ at each iteration steps (see [DHS00] for a good review of linear models).

The expressiveness of linear models can be simply improved by the introduction of new attributes which are non-linear functions of existing attributes:

$$h(\underline{a}) = \sum_{i=1}^{d} w_i \cdot f_i(\underline{a}), \tag{2.18}$$

where d is the number of such attributes and f_i can be arbitrary functions. For example, quadratic models can be represented by using functions f_i which are products of pairs of attributes. Since model (2.18) is still linear in the transformed input space, optimization of parameters w_i, $i = 1, \ldots, d$, is done similarly as the optimization of non generalized linear models.

2.3.2 Artificial neural networks

Artificial neural networks are a generalization of linear models inspired by analogies with the biological brain. A perceptron is a simple unit which computes the following function:

$$h(\underline{a}) = f(w_0 + \sum_{i=1}^{n} w_i \cdot a_i),$$

where the activation function f is a non linear function of its argument. Examples are the sign function:

$$f(x) = \text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases},$$

or the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Note that the first one gives a classification rule similar to (2.17). A single perceptron is thus essentially equivalent to a linear model.

The idea of neural networks is to combine such simple units in order to improve expressiveness. In feed-forward multilayer architectures, the neural networks are organized into successive layers of perceptrons where the outputs of a layer are only connected to the

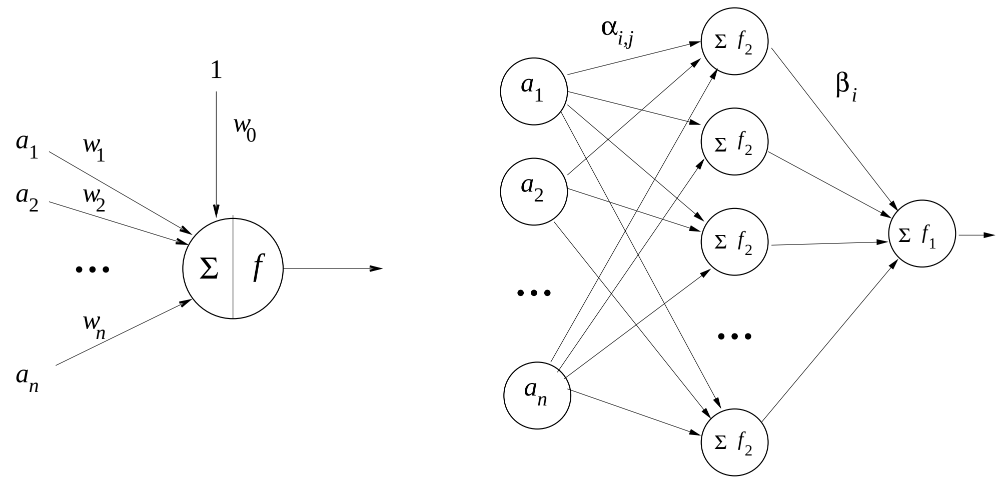


Figure 2.1: Left, a perceptron, right, a three layer neural network with one output

inputs of the next layers. A perceptron and an example of a three-layer neural networks is depicted in Figure 2.1. The first layer is the input layer which transmits the input attribute values to the second layer. The last layer is the output layer which computes the output value. Layers between the input and output layers are called hidden layers (since their outputs do not correspond to observed variables). A neural network with a single hidden layer like the one of Figure 2.1 represents the following parametric family of functions:

$$h(\underline{a}) = f_1(\beta_0 + \sum_{i=1}^{L} \beta_i f_2(\alpha_{i,0} + \sum_{j=1}^{n} \alpha_{i,j} \cdot a_i), \tag{2.19}$$

where L is the number of units in the hidden layer, f_1 and f_2 are respectively the activation functions of the output and hidden layers (which do not need to be identical), and $\alpha_{i,j}, \beta_i$ are the parameters. It has been shown that a neural network with sufficient number of hidden layers and units is able to represent any function with arbitrary accuracy (this is called the universal approximation property, see e.g. [Bar94]).

Learning in neural networks is also an optimization in a parameters space. When the activation and the error functions are smooth, there exists an efficient algorithm to compute the gradient of the error with respect to the vector of parameters (the back-propagation algorithm). This gradient is then used to iteratively modify the parameters in the direction of a minimization of error:

$$\underline{w}^{new} = \underline{w}^{old} - \eta \cdot \nabla_{\underline{w}} \hat{Err}(h, LS), \tag{2.20}$$

where η is called the learning rate, and $\underline{w}$ denotes the vector of parameters. The learning algorithm starts with random initial weights and then iteratively improves them using (2.20) until a local minimum of the error is reached. Several variants of this algorithm have been introduced to accelerate convergence (by introduction of second derivatives in (2.20) for example). In general, (2.20) may have several local minima and thus the hypothesis returned by the learning algorithm may depend to some extent on the (randomized) initial conditions.

As mentioned previously, classification problems are handled with neural networks by the introduction of the numerical class indicator variables Y_C^i defined by (2.14).

2.3.3 K-Nearest Neighbors

For fixed k and fixed learning sample, the hypothesis produced by this method predicts at each point an aggregation of the outputs of its k nearest neighbors in the learning set according to some a priori defined distance measure. When attributes are numerical, the distance is often a weighted Euclidian distance where weights are usually chosen as the inverse variance of the corresponding attribute so as to give every attribute equal importance:

$$d(\underline{a}, \underline{a}') = \sum_{i=1}^{n} \frac{1}{\sigma_i^2}(a_i - a_i')^2,$$

where σ_i is the standard deviation of attribute A_i (estimated from the learning set). The usual aggregated predictions are the average output for regression:

$$h(\underline{a}) = \frac{1}{k} \sum_{(\underline{a},y)\in NN(\underline{a},LS,k)} y,$$

and the majority class for classification:

$$h(\underline{a}) = \arg\max_{c_i \in \mathcal{Y}} \sum_{(\underline{a},y)\in NN(\underline{a},LS,k)} 1(y = c_i),$$

where the set $NN(\underline{a}, LS, k)$ contains the k nearest neighbors of $\underline{a}$ in LS.

In practice the optimal value of k will depend on the learning sample size N and on the problem specifics (dimensionality of the input space and smoothness of the Bayes model). Note that when N tends towards infinity and k is appropriately adjusted to N, the hypothesis produced by this method converges to the Bayes model under fairly unrestrictive assumptions [DHS00].

2.3.4 Decision and regression trees

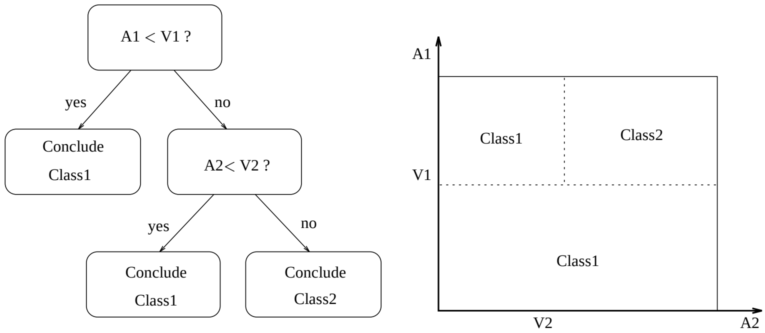
As this method will be studied in detail in the sequel of the thesis, the tree induction algorithm that we have been using in our experiments will be explained precisely in later chapters. We therefore restrict the following presentation to a very brief introduction of the general principles of the so-called TDIDT family of algorithms (for Top Down Induction of Decision Trees).

The hypothesis space of decision/regression trees is the set of piecewise constant functions $h(.)$ defined on the input space by:

$$h(\underline{a}) = \begin{cases} h_1 & \text{if } \underline{a} \in R_1 \\ h_2 & \text{if } \underline{a} \in R_2 \\ \dots & \\ h_L & \text{if } \underline{a} \in R_L \end{cases}, \tag{2.21}$$

where $R_1, \dots, R_L$ is a recursive partition of the input space into L regions and h_i is the constant prediction associated to each region R_i (a class or a number).

In order to represent the structure of the recursive partition and how it is defined in terms of the input attributes, a decision tree structure is used. Each interior node of such a tree is labeled with a test defined as a function of the attribute values and each terminal node defines a region R_i. To find the region associated to a point $\underline{a}$, it is sufficient to propagate it into the tree according to test issues until it reaches a terminal node. Each region R_i is thus described by a sequence of tests and the whole tree simply translates into a set of "If ... then ..." rules. In the classical method, tests are binary and involve only



$$\begin{array}{c} \text{if } a_1 < v_1 \text{ then } C = c_1 \\ \text{if } a_1 \geq v_1 \text{ and } a_2 < v_2 \text{ then } C = c_1 \\ \text{if } a_1 \geq v_1 \text{ and } a_2 \geq v_2 \text{ then } C = c_2 \end{array}$$

Figure 2.2: A decision tree in a two-dimensional input space, the corresponding rule, and the resulting partition of the input space

one attribute at the same time: for numerical attributes they compare the attribute value to a threshold, resulting in a partition of the input space into hyper-rectangular regions. Figure 2.2 shows an example tree and the resulting partition of the input space.

As it is often practically infeasible to compute the best tree of a given complexity for a given problem (finding a tree of fixed complexity minimizing (2.12) is NP-complete), decision tree induction uses a hill-climbing algorithm composed of two stages, namely tree growing and tree pruning. The growing algorithm starts with a one node tree and tries to split this node by selecting a test among a set of candidate tests. The algorithm then proceeds recursively to split the left and right successors of this node. Predictions h_i at leaves are estimated from the subset of learning samples which belong to the corresponding region R_i, by conditional class frequency counts and majority vote in classification and by averaging the output values of these objects in regression. Typically, tree growing proceeds until the terminal nodes correspond to subsamples in which the output variable is (almost) constant. In order to reduce overfitting it is followed by a pruning stage which aims at determining the correct complexity of the tree by replacing subtrees of the fully grown tree by terminal nodes. The pruning procedure operates in a bottom up fashion.

The decision tree induction method used in our experiments is described in full detail in Section 5.2 (growing) and Section 6.2 (pruning).

2.3.5 Naive Bayes and Bayesian networks

The previously described family of algorithms seeks for a direct approximation of $P(C|\underline{A})$ or $E_{Y|\underline{A}}\{Y\}$ and so are termed discriminative or predictive approaches. Another class of automatic learning methods proceed by first modeling the joint distribution $P(\underline{A}, Y)$ and then inferring the conditional distribution $P(Y|\underline{A})$ from the former according to the Bayes rule, i.e.

$$P(Y|\underline{A}) = \frac{P(\underline{A}, Y)}{P(\underline{A})},$$

in order to eventually make predictions according to (2.9) or (2.10). In order to make the learning of $P(\underline{A}, Y)$ easier, some assumptions about its structure must be made. Since the early eighties a large body of work has been carried out on the appropriate modeling of modularity in the context of probabilistic reasoning, which has led to various classes

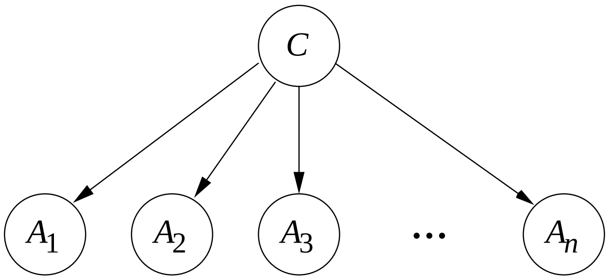


Figure 2.3: The Bayesian network for the Naive Bayes method. Each node represents a random variable and is independent of its non descendants given its parents in the graph.

of modular probabilistic models. In this context, modularity essentially corresponds to conditional independence assumptions among sets of random variables, and this modularity is expressed by the factorization of the joint distribution of all variables into a product of functions involving each one only a small number of variables [CDLS99]. The most well-known type of graphical models are the so-called Bayesian networks, where the factorization is graphically represented by a directed acyclic graph (DAG) each node of which corresponds to one of the variables involved in the modeled joint distribution, and where the component functions correspond to conditional probability distributions of each variable given the parents of this variable in the DAG [Pea88].

The naive Bayes method in classification corresponds to a very simple type of Bayesian network which assumes that:

$$P(\underline{A}, C) = P(C) \cdot P(A_1|C) \cdots P(A_n|C), \tag{2.22}$$

which translates the conditional independence of the attributes given the output variable. In terms of graphical structure, this factorization is represented in Figure 2.3 by a shallow directed graph the top-node of which corresponds to variable C and where the nodes corresponding to the input attributes A_i have this node as single parent.

Once this independence hypothesis is assumed, the automatic learning problem reduces to the estimation of $P(C)$ and of the $P(A_i|C)$ for each input attribute on the basis of the learning sample. Thus, in the standard Naive Bayes method the prior class probabilities $P(C = c_i)$ are estimated by counting the frequency of occurrence of class c_i in the learning set (i.e. the maximum likelihood estimates). For a symbolic attribute,, $P(A_i|C = c_i)$ usually is also estimated by direct counting. For numerical attributes, several parametric and non parametric techniques are possible. For example, the distribution $p(A_i|c_i)$ can be assumed to be Gaussian and the parameters of this distribution, the mean and standard deviation, can be estimated from the learning sample. Among non parametric techniques, the most simple one is the histogram. Histogram consists in dividing the range of the attribute A_i into subregions (or bins) and counting the relative number of examples (of a fixed class) falling into each subregion.

Of course, the hypothesis of conditional independence of the attributes given the class is too strong in many problems. Actually, in the special case where distributions $P(A_i|C_i)$ are assumed to be Gaussian, the naive Bayes model is nothing but a quadratic discriminant. More general Bayesian network structure have been also used for classification (see [FGG97] for example).

2.4 Evaluation of learning algorithms

Three main criteria are used to compare learning algorithms:

- **Accuracy**: the way it succeeds in reducing the error.

- **Computational efficiency**: the time needed to learn a model but also the time to apply this model to new cases.

- **Interpretability**: if it gives comprehensible models or not.

Large experiments comparing the accuracy of several learning algorithms (see for example [MS94] or more recently [LLS00]) show that the exact ranking among algorithms depends on the problem and how well the basic hypotheses of the learning algorithm (for example, linearity) are satisfied by this problem. However, neural networks and nearest neighbors are often among the most accurate methods. Although competitive in some problems (for example, when there are redundant or useless attributes), tree based models are often not as accurate as these two methods. Of course, the strong assumptions behind linear models and the Naive Bayes prevent them to reach high accuracy in many problems. Nevertheless, in some cases, these methods are competitive with other more complex algorithms such as decision trees (see for example [DP97] for the Naive Bayes). In the next chapter, we will show that this apparent inconsistency can be explained by analyzing the variance of these methods.

The precise ranking of learning algorithms in terms of computational efficiency would be possible only in the context of specific implementations (the general methods that we have described can be implemented in different ways, with quite significant impacts on their efficiency). Also, the problem size in the context of automatic learning is determined by two parameters characterizing the database size, namely the number of objects N and number of attributes n, which may have a different impact on computational efficiency. Nevertheless, for medium problem sizes (say a few thousand objects and a few hundred attributes) a crude ranking of these methods can be done. For example, because of their simplicity, the learning step of linear models and Naive Bayes are very fast. Although slower, decision tree induction remains very efficient even for large sample sizes and large numbers of variables. The basic K-NN method does not require any learning phase (except for the optimization of K). At the very end of this ranking are neural networks. Indeed, especially complex neural networks can take substantial time to be optimized. On the other hand, new predictions are computed very quickly with all models except for K-NN which requires to compute the distances to each element of the learning set.

Interpretability is highly subjective and also problem dependent. However, it is generally accepted that neural networks are black-box type of models and that on the other side, decision trees are very comprehensible. Linear model and Naive bayes because of their simplicity are also termed interpretable. Nearest neighbors techniques, although they do not provide any synthetic model, can still give useful information by retrieving similar cases in a data set.

In general, there is a tradeoff between these three criteria. An accurate method is likely to be resource demanding or induces non comprehensible models, while interpretable models are often not very accurate. Also, the relative importance assigned to each criterion is highly application-dependent. Sometimes, it will be acceptable to sacrifice some accuracy in order to gain interpretability or computational efficiency. In other cases, accuracy has to be optimized as far as possible, whatever the price to pay in terms of interpretability or computation times. This makes existing algorithms complementary and none of them can be claimed to be globally superior to all other ones.

In this thesis, new improvements or algorithms will be discussed having in mind these three criteria.

2.5 Evaluation of accuracy of models

The evaluation of a model is important first in order to predict its future performance on new data but also in order to provide the learning algorithm with a criterion to choose a model in its hypothesis space. Suppose we want to evaluate the true error (see eqn. (2.2)) of a model f_{ls} induced from a learning set ls by a given learning algorithm and let us denote an error estimate for this model by $\hat{Err}(f_{ls})$.

The *resubstitution estimate* (which is equivalent to the empirical risk) directly evaluates the accuracy on the learning set used to induce the model:

$$\hat{Err}(f_{ls}) = \hat{Err}(f_{ls}, ls), \tag{2.23}$$

where $\hat{Err}(f_{ls}, ls)$ is given by (2.12). While the resubstitution estimate is useful to guide learning algorithms, the error estimation it gives for the induced model is likely to be very optimistic since most of the learning algorithms explicitly try to reduce this estimate.

A more reliable estimate would be obtained from a data set independent of the sample of data used during induction. For example, a *hold-out estimate* consists in partitioning ls into two disjoint subsets vs and $ls \setminus vs$. A new model is induced from $ls \setminus vs$ and the error is estimated by:

$$\hat{Err}(f_{ls}) = \hat{Err}(f_{ls \setminus vs}, vs), \tag{2.24}$$

If vs is too small, this estimate is not stable and if vs is too large, the estimate will be pessimistic as the truly assessed model, f_{ls}, is induced from a larger learning set. A common practice is to save one third of the data for validation.

In cases where the learning sample is very small, even one third of the data will yield an unstable estimate of the error. *Cross-validation* is a way to stabilize error estimates with small validation set vs. In k-fold cross-validation, we divide the learning set into k disjoint subsets of the same size, $ls = ls_1 \cup ls_2 \cup \ldots \cup ls_k$. A model is built from every set $ls \setminus ls_i, i = 1, \ldots, k$ and the error is estimated either by:

$$\hat{Err}(f_{ls}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_{ls \setminus ls_{k(i)}}(\underline{a}_i)), \tag{2.25}$$

where $k(i)$ is the index of the subset ls_i of ls that includes the instance $(\underline{a}_i, y_i)$, or by:

$$\hat{Err}(f_{ls}) = \frac{1}{k} \sum_{i=1}^{k} \hat{Err}(f_{ls \setminus ls_i}, ls_i). \tag{2.26}$$

When the number of objects is an exact multiple of the number of folds, equations (2.25) and (2.26) are in fact equivalent. When this is not the case, the first one should be preferred, especially in the case of small samples, as it gives identical weights to each instance. When k is equal to the number of samples in the learning set, this method is called leave-one-out. Except for some learning algorithms like K-NN, small values of k (between 10 and 20) are preferred for computational efficiency reasons. Stratified cross-validation is a variant of cross-validation where the class proportions in the learning set are kept into each fold.

When comparing different learning algorithms or variants of the same algorithm, it is not crucial to use the whole data set to build models and so the fact that hold-out estimates give pessimistic estimates of the error of a model induced from the whole dataset is not important as long as it gives a reliable estimate of the error of the model induced from the reduced learning set. So in what follows, when comparing algorithms, we will first divide the available data into two disjoint subsets: one subset devoted to the induction of models and the other set devoted to their validation. Occasionally, when the available data set will be too small to allow reliable estimates from hold-out estimate, 10-fold cross-validation will be preferred.

At this point, it is important to note that the methods explained here do not give an idea of the general accuracy of a learning algorithm, but only of a model induced from a particular learning set in the context of one particular problem. If learning algorithms have to be compared, other criteria have to be taken into account. For example, the next chapter will give some guide to study the general behavior of a learning algorithm when it is faced with different learning sets. Another criterion is its behavior on a large spectrum of problems. To this end, there exist several repositories of data sets which are used as benchmarks by many researchers to compare algorithms (for example [BM98]).