Learning Machines

Norbert Jankowski and Krzysztof Grąbczewski

Department of Informatics, Nicolaus Copernicus University, Toruń, Poland
norbert,kgrabcze@phys.uni.torun.pl

1 Introduction

Learning from data may be a very complex task. To satisfactorily solve different problems, many different kinds of algorithms must be appropriately combined or selected and applied. *Feature extraction* algorithms are valuable tools, which prepare data for other learning methods. To estimate their usefulness one must examine the whole complex processes they are parts of.

The goal of the chapter is to present a short survey of different approaches to learning from data with a special emphasis on solving classification (and approximation) problems, where feature extraction plays especially important role. We address this review to the readers who know the basic ideas of the field and would like to get quick acquaintance with numerous techniques of *computational intelligence*. Some familiarity with the foundations of statistics and the ease of understanding mathematical formalism is certainly very advantageous.

For novice readers we recommend textbooks by Duda et al. [2001], Mitchell [1997], Bishop [1995], Haykin [1994], Cherkassky and Mulier [1998], Schalkoff [1992], de Sá [2001], Hastie et al. [2001], Ripley [1996], Schölkopf and Smola [2002].

Our tutorial starts with the mathematical statement of the learning problem. Then, it presents two general induction principles: risk minimization and Bayesian learning, that are widely applied in the algorithms being the subject of this review and of the next chapters. A discussion of classification task and their connections with the decision borders induced by classification models precedes the survey of the learning machines including: Naive Bayes, Linear Discriminant Analysis, kernel methods, Neural Networks, similarity based approaches and Decision Trees.

2 The learning problem

The term *learning machines* encompasses many kinds of computational intelligence systems capable of gathering knowledge by means of data analysis. The algorithms are sometimes divided into different groups (not necessarily disjoint) such as *machine learning*, *soft computing* or more uniform types: *neural networks*, *decision trees*, *evolutionary algorithms* etc. Learning from data may be treated as searching for the most adequate model (hypothesis) describing given data. A *learning machine* is an algorithm which determines a *learning model*, which can be seen as a function:

$$f_M : \mathcal{X} \to \mathcal{Y}. \tag{1}$$

The function transforms objects from the data domain $\mathcal{X}$ to the set $\mathcal{Y}$ of possible target values. The data domain and the set of target values are determined by the definition of the problem for which the f_M is constructed.

The very important point of learning is that the learning model f_M depends on some *adaptive parameters* sometimes also called *free parameters*. The purpose of *learning* of the model f_M can be seen as a process in which a *learning algorithm* searches for parameters of the model f_M, which solves given task.

The learning algorithm learns from a *sequence* $\mathcal{D}$ of data, defined in the space $\mathcal{X}$ or in $\mathcal{X} \times \mathcal{Y}$:

$$\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} = X \tag{2}$$

$$\mathcal{D} = \{\langle \mathbf{x}_1, y_1\rangle, \langle \mathbf{x}_2, y_2\rangle, \ldots, \langle \mathbf{x}_m, y_m\rangle\} = \langle X, Y\rangle \tag{3}$$

Usually X has a form of a sequence of multidimensional vectors. An alternative statement of the problem defines X as a sequence of object names and provides a matrix of values describing similarity between the objects.

The definition (2) states an *unsupervised learning* problem (learning without teacher), where learning algorithms may base only on values $\mathbf{x}_i$ (called *inputs*) from the data domain. Unsupervised learning is used for example in clustering, self-organization, auto-association and some visualization algorithms.

In the case of definition (3) learning algorithms use pairs $\langle \mathbf{x}_i, y_i\rangle$ where y_i is the desired *output* value for $\mathbf{x}_i$. Such learning is called *supervised* (with teacher). When $\mathcal{Y}$ is a set of several symbols (the number $|\mathcal{Y}|$ of elements of $\mathcal{Y}$ is usually significantly smaller than the number of vectors in the training data set), the learning problem is called a *classification task* and $\mathcal{Y}$ is called the set of *class labels*. If $|\mathcal{Y}| = 2$, then we deal with *binary classification* and if $|\mathcal{Y}| > 2$ — with multi–class problems. For convenience it is often assumed that $\mathcal{Y} = \{-1, +1\}$, $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{1, \ldots, c\}$. Other examples of supervised learning are *approximation* (or *regression*) and *time series prediction*. In such cases $\mathcal{Y} = R$ or $\mathcal{Y} = R \times \ldots \times R$.

Risk minimization and Bayesian learning

Many learning algorithms perform a minimization of *expected risk* (or more precisely: its approximation), defined by:

$$R[f] = \int_{\mathcal{X}\times\mathcal{Y}} l(\mathbf{x}, y, f(\mathbf{x}))\ dP(\mathbf{x}, y), \tag{4}$$

where $l(\cdot)$ is a *loss function*, and P is the data distribution. Note that in the case of unsupervised learning above equation does not depend on y and $\mathcal{Y}$.

The loss function may be defined in one of several ways. In the case of classification it may be

$$l_c(\mathbf{x}, y, f(\mathbf{x})) = \begin{cases} 0 & f(\mathbf{x}) = y, \\ 1 & f(\mathbf{x}) \neq y. \end{cases} \tag{5}$$

For binary classification with $\mathcal{Y} = \{-1, +1\}$ the *soft margin loss* proposed by Bennett and Mangasarian [1992] may be used:

$$l_b(\mathbf{x}, y, f(\mathbf{x})) = max\{0, 1 - yf(\mathbf{x})\}. \tag{6}$$

The most popular loss function designed for regression (however it is also commonly used for classification tasks) is the *squared loss*:

$$l_s(\mathbf{x}, y, f(\mathbf{x})) = (f(\mathbf{x}) - y)^2. \tag{7}$$

Another popular loss function dedicated for regression is called ϵ-*insensitive loss*:

$$l_\epsilon(\mathbf{x}, y, f(\mathbf{x})) = max\{0, |f(\mathbf{x}) - y| - \epsilon\}. \tag{8}$$

It may be seen as an extension of the soft margin loss.

In the case of $\mathcal{Y} = \{0, 1\}$ and $f(\mathbf{x}) \in (0, 1)$, a possible solution is the *cross-entropy* [Kullback and Leibler, 1951] loss function

$$l_{ce}(\mathbf{x}, y, f(\mathbf{x})) = -y \log f(\mathbf{x}) - (1 - y) \log(1 - f(\mathbf{x})). \tag{9}$$

In practice, the distribution $P(\mathbf{x}, y)$, crucial for the integration of (4), is usually unknown. Thus, the expected risk is replaced by *empirical risk*

$$R_{emp}[f] = \frac{1}{m} \sum_{i=1}^{m} l(\mathbf{x}_i, y_i, f(\mathbf{x}_i)), \tag{10}$$

which is the average of errors over the set of data pairs $\langle \mathbf{x}_i, y_i \rangle$. The empirical risk used with the squared loss function is the well known *mean squared error* (MSE) function:

$$R_{MSE}[f] = \frac{1}{m} \sum_{i=1}^{m} l_s(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) = \frac{1}{m} \sum_{i=1}^{m} (f(\mathbf{x}_i) - y_i)^2. \tag{11}$$

The *sum squared error* (SSE) is equal to $m \cdot R_{MSE}[f]$. Hence, minimization of MSE is equivalent to minimization of SSE (up to the constant m). In practice, yet another formula is used ((11) with m replaced by 2 in the denominator), because of a convenient form of its derivative.

Minimization of the empirical risk measured on a training data sample does not guarantee good *generalization* which is the ability to accurately estimate the target values for *unseen data*[1]. Even if the empirical risk $R_{emp}[f]$ is small, f may provide poor generalization i.e. for a representative sample $\mathcal{D}' = \{\langle \mathbf{x}'_1, y'_1\rangle, \langle \mathbf{x}'_2, y'_2\rangle, \ldots, \langle \mathbf{x}'_{m'}, y'_{m'}\rangle\}$ of test patterns the empirical risk

$$R_{test}[f] = \frac{1}{m'} \sum_{i=1}^{m'} l(\mathbf{x}'_i, y'_i, f(\mathbf{x}'_i)) \tag{12}$$

may be much higher. The $R_{test}[f]$ is commonly called *test error*.

Poor generalization is often caused by *overfitting*. One of possible ways to protect the model against overfitting is to add a *regularization term*. Regularization was originally proposed for ill-posed problems by Tikhonov [1963], Tikhonov and Arsenin [1977]:

$$R_{reg}[f] = R_{emp}[f] + \lambda \Omega(f). \tag{13}$$

$\Omega(f)$ is a *regularizer* designated to control the complexity of f. More details on regularization are presented in section 3.7.

The problem of finding the best model from some space H can be analyzed from the Bayesian point of view. The Bayes theorem

$$P(f|\mathcal{D}) = \frac{P(\mathcal{D}|f)P(f)}{P(\mathcal{D})} \tag{14}$$

defines the relationship between the *posterior probability* $P(f|\mathcal{D})$ and *prior probability* $P(f)$ of given hypothesis f. This formula is fundamental in *Bayesian learning*, which aims in finding the hypothesis which maximizes the *a posteriori* probability[2] (MAP):

$$f_{MAP} = \arg\max_{f \in H} \; P(f|\mathcal{D}) = \arg\max_{f \in H} \; P(\mathcal{D}|f)P(f). \tag{15}$$

Assuming equal *a priori* probabilities for all the hypotheses $f \in H$ leads to the definition of the *maximum likelihood* (ML) hypothesis:

$$f_{ML} = \arg\max_{f \in H} \; P(\mathcal{D}|f). \tag{16}$$

With additional assumptions that the training examples are identically and independently distributed and correspond to a target function g with some

[1] *Unseen* means not used in the learning process.

[2] In the case of discrete distribution P denotes the probability, otherwise it is the density function.

normally distributed noise $\epsilon \sim N(0, \sigma)$ (i.e. $\mathbf{y} = g(\mathbf{x}) + \epsilon$), the maximization of $P(\mathcal{D}|f)$ is equivalent to maximization of $\prod_{i=1}^{m} P(y_i|x_i, f)$ and to minimization of the negation of its logarithm which is exactly the minimization of MSE.

Equivalent formulation of (15):

$$f_{MAP} = \arg\min_{f \in H} \left[-\log_2 P(\mathcal{D}|f) - \log_2 P(f) \right], \tag{17}$$

lets us use the information theory language and state that the optimal model is the one which minimizes the sum of the description length of the hypothesis f (assuming the optimal hypotheses coding[3]) and the description length of the data $\mathcal{D}$ under the assumption that f holds (also assuming the optimal coding). If the symbol of $L_C(X)$ is used to denote the length of the description of X using coding C, then for a hypotheses coding C_H and data coding C_f (assuming the hypothesis f) the *Minimum Description Length* (MDL) principle [Rissanen, 1978] can be formulated as:

$$f_{MDL} = \arg\min_{f \in H} \left[L_{C_f}(\mathcal{D}) + L_{C_H}(f) \right]. \tag{18}$$

It confirms the expectations that shorter descriptions (simpler models) should be preferred over sophisticated ones. The function being minimized in (17) and (18) can be seen as a risk function ($L_{C_f}(\mathcal{D}) = -\log_2 P(\mathcal{D}|f)$) with a regularizer ($L_{C_H}(f) = -\log_2 P(f)$).

More detailed theoretical description of the problem can be found for example in [Mitchell, 1997].

In the case of regression (with MSE function) or classification the error (risk) function may be decomposed into *bias* and *variance* terms [Bishop, 1995, Duda et al., 2001]. The decomposition reveals the so-called *bias–variance trade-off* – flexible models with low bias show large variance of results and vice versa: highly biased models show small variance.

It is important to remember that regardless of the type of the error function being used, obtaining high level of generalization requires building as simple models as possible. Model complexity should be increased only when simpler models do not offer satisfactory results. This rule is consistent with the medieval rule called *Ockham's razor* and other ideas such as MDL principle or regularization.

3 Learning algorithms

There is no commonly used taxonomy of the algorithms that learn from data. Different point of views stress different aspects of learning and group the

[3]Shannon and Weaver [1949] proved that the optimal code assigns $\log_2(P(i))$ bits to encode message i.

methods in different ways. Learning algorithms have different theoretical background and model the knowledge with miscellaneous data structures. Their model building strategies may be founded on the achievements of statistics or different kinds of optimization methods. The optimal (or suboptimal) models can be determined with *search strategies* (from simple, blind search methods, through heuristically augmented search and evolutionary computation, to global optimization techniques such as simulated annealing), *gradient descent methods*, *mathematical programming* (linear and quadratic programming find especially numerous applications) or other tools including *fuzzy logic* (see chapter **??**) and *rough sets theory* [Pawlak, 1982].

To increase the adaptation capability of computational intelligence systems, ensembles of many models are used. Such *ensembles* may be *homogeneous* (multiple models of the same kind) or *heterogeneous* (taking advantage of different methodologies in a single learning process).

Classifier decision borders

Classification models divide the feature space into disjoint regions assigned to class labels. Different classifiers provide different kinds of borders between the regions (*decision borders*).

Figure 1 illustrates two examples of two-dimensional classification problems. Four different solutions for each of the data sets are depicted.

For both tasks, the top-left plot presents an example of linear decision borders. Many learning algorithms yield models which discriminate with linear functions only, e.g. linear discrimination methods (see section 3.2) and simple neural networks (see section 3.4).

The top-right plots show decision borders perpendicular to the axes of the feature space – most common solutions of decision tree algorithms (see section 3.6).

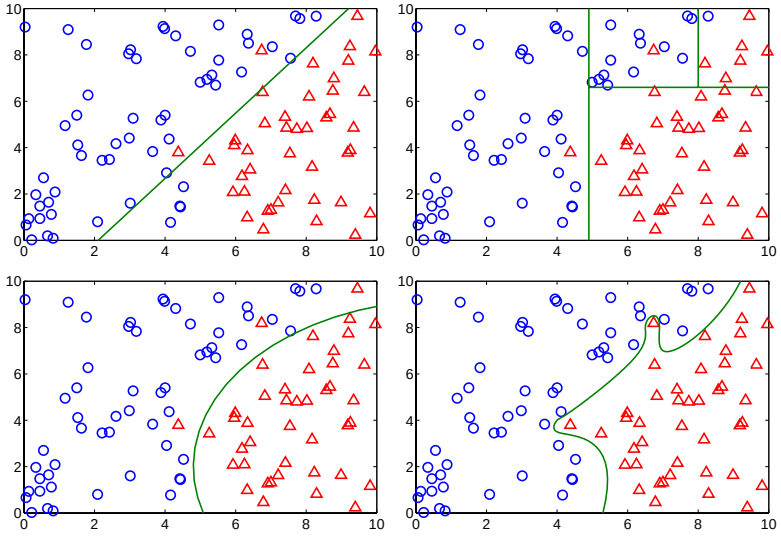
In the bottom-left plots the classes are embraced with quadratic curves. Such shapes can be obtained for example with simple neural networks with radial basis functions (see section 3.4) and other nonlinear methods.

The bottom-right plots present decision borders of maximum classification accuracy. Many learning algorithms are capable of finding models with so complicated borders, but usually it does not provide a desirable solution because of overfitting the training data – the accuracy reaches maximum for the sample used for training, but not for the whole of the data.

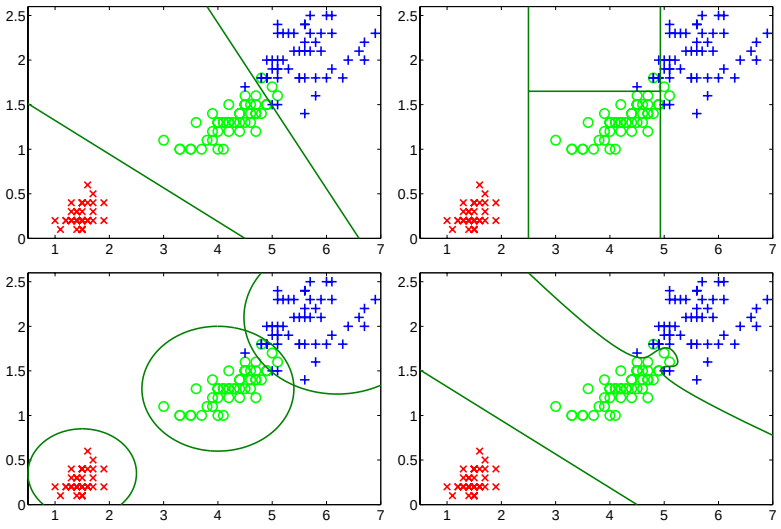
Some adaptive models can separate more than two classes in a single learning process. Other algorithms must be run several times and their results must be appropriately combined to construct a final classifier. The combination is not straightforward because of possible conflicts between the combined models – the bottom-left plot of figure 1(b) clearly shows such a clash.

Generalization and model selection

To determine the generalization ability of a model one would need to measure the average risk for the set of all possible data objects. In real life applications



(a) Artificial classification task



(b) Two of four dimensions of the well known iris data [Fisher, 1936]

Fig. 1. Example solutions to two classification tasks: top-left – with linear decision borders, top-right – specific for decision trees, bottom-left – with centered decision areas and bottom-right – of maximum accuracy.

it is not feasible, so we estimate the risk by counting it for a test set (equation (12)). When doing this we should be aware of the danger of testing models on a single test set (for example resulting from a rigid partition of the set of all available data to the training and test parts). Model selection based on testing trained models on a single test set does not get rid of the danger of overfitting. More accurate estimation of the empirical risk can be obtained with *n-fold cross-validation* (CV). In this technique we split the set of available data into n parts and perform n training and test processes (each time the test set is one of the parts and the training set consists of the rest of the data). Average test risk can be a good estimate of real generalization ability of the tested algorithm, especially when the whole cross-validation is performed several times (each time with different data split) and n is appropriately chosen[4]. To get a good estimate of generalization ability of a learning machine, it is important to analyze not only the average test error, but also its variance, which can be seen as a measure of *stability*.

The *Ockham's razor* and other issues discussed in section 2 suggest that accurate models which are simple should provide stability and good generalization. So, if we have several models of similar accuracy, we should prefer stable models and the simplest ones (linear over nonlinear, those with the smallest number of parameters etc.). Even if more complex models are more accurate it is often worth to resign from high learning accuracy and select less accurate but more stable or simpler models. The bottom-right images of figure 1 present highly accurate models for the training data, but providing poor generalization.

Another technique to obtain models of good generalization is regularization (see section 3.7). Preserving large classification margins may also be seen as a kind of regularization and has been successfully used in SVM methodology (see section 3.3). Models with optimized margins may reach high level of generalization despite the complexity of their decision borders.

In the case of high-dimensional feature spaces the information about the function being modelled is often contained within a small-dimensional subspace and the rest of the features play a role of a noise, making learning difficult. Hence, the methods of feature selection can be very helpful in the pursuit of high accuracy and good generalization.

3.1 Optimal and Naive Bayes Classifiers

The Bayesian framework presented in section 2 deals with the problem of selection of the optimal hypothesis describing given data set. Another problem is how to assign proper class labels to given data objects. The optimal choice is defined by the *Bayes Optimal Classifier*:

[4]Setting n to the number of data objects yields a special case of cross-validation called *leave-one-out* which, although sometimes used, is not a good estimate (see e.g. [Kohavi, 1995]).

$$BOC(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \; P(y|\mathbf{x}). \tag{19}$$

Given a set of possible hypotheses H and training data set D the most probable class of a new data vector $\mathbf{x}$ can be determined in accordance with the law of total probability as [Mitchell, 1997]:

$$BOC(\mathbf{x}|\mathcal{D}, H) = \arg\max_{y \in \mathcal{Y}} \sum_{h \in H} P(y|h, \mathbf{x}) P(h|D). \tag{20}$$

Anyways, the BOC formula is useless in most real applications. To calculate the probabilities for each of the classes one needs to know the probabilistic structure of the problem and to examine all the candidate hypotheses, which is usually impossible (even in quite simple cases).

Feasible Bayes classifiers must pay the price of losing the optimality guarantee. One of the well known simplifications of the method is *Naive Bayes Classifier*. Its reduced complexity is the result of the assumption that the random variables corresponding to particular features of the data space are independent. It makes the definition of the maximum *a posteriori* class as simple as:

$$\begin{aligned} NBC(\mathbf{x}) &= \arg\max_{y \in \mathcal{Y}} \; P(y|\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \; \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \\ &= \arg\max_{y \in \mathcal{Y}} \; P(y) \prod_i P(x_i|y). \end{aligned} \tag{21}$$

The formula can be easily used if we know the probabilities of observing each class and of observing a particular feature value among the populations of vectors representing the classes. In real world applications the probabilities can only be estimated on the basis of a given training data set.

In the case of discrete features the probabilities $P(x_i|y)$ can be evaluated as the relevant frequencies of observing x_i values among the vectors of class y. This is the most common application of Naive Bayes Classifiers. The frequencies are good estimators when the training data set is large enough to reflect real distributions. Otherwise some corrections to the frequencies calculations (e.g. *Laplace correction* or *m-estimate*) are strongly recommended [Provost and Domingos, 2000, Kohavi et al., 1997, Cestnik, 1990, Zadrozny and Elkan, 2001]. The corrections are also fruitfully used in decision tree algorithms.

The features with continuous values can be first discretized to use the same method of probability estimation. An alternative is to assume some precise distribution of the *a priori* class probabilities. Most commonly the normal distribution is assumed:

$$P(x_i|y) \propto N(\mu_i^y, \sigma_i^y), \tag{22}$$

where N is the normal density function while μ_i^y and σ_i^y are respectively: the mean value and the standard deviation of the i-th feature values observed among the training data vectors belonging to class y.

In practice the assumption of the independence of different features and of the normality of the distributions may be far from true – in such cases also the Naive Bayes Classifier may be far from optimal.

Some other approaches to the problem of estimation of the conditional probabilities have also been proposed. One of the examples [John and Langley, 1995] gets rid of the assumption of normal distributions by using a kernel density estimation technique.

3.2 Linear discriminant methods

Linear discriminant methods determine linear functions, which divide the domain space into two regions (see the top-left plot in figure 1(a)). Learning processes adjust the parameters of linear models to obtain an optimal correspondence between the half-spaces of the feature space and the data categories (classes). *Linear discriminant functions* are defined by linear combinations of the argument vector components:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b, \tag{23}$$

where $\mathbf{w}$ is a *weight vector* and $-b$ defines the *threshold*.

If a vector $\mathbf{x}$ satisfies $f(\mathbf{x}) > 0$, then the model assigns the label of the positive category to it, otherwise the label of the negative class is assigned. The instances for which $f(\mathbf{x}) = 0$ define the hyperplane which splits the whole space into the two regions.

For a given training data set, there can be no such hyperplane which separates the vectors representing different classes. In such a case we say that the problem is *linearly inseparable*.

In a multicategory case it is not possible to separate different classes with a single linear discriminant function. One of the solutions to this is to form a linear discriminant function for each class to separate the samples of the i-th class from the rest of the training data. Another solution is to compose one linear model for each pair of classes. Both ways may result in such a partition of the space that for some regions there is no simple way to determine the winner class, because the half-spaces corresponding to different classes overlap. A good combination of the linear discriminants is the *linear machine* which, given a linear discriminant function for each class $y \in \mathcal{Y}$:

$$f_y(\mathbf{x}) = \mathbf{w}_y^T\mathbf{x} + b_y, \tag{24}$$

provides a reasonable scheme of label assignment eligible for each point of the space (except for the decision borders):

$$k = \arg\max_{y\in\mathcal{Y}} \; f_y(\mathbf{x}). \tag{25}$$

The linear discriminant function can be determined in different ways. The idea of Fisher's linear discriminant [Fisher, 1936] lies in maximization of

$$m_{-1} - m_{+1} = \mathbf{w}^T(\mathbf{m}_{-1} - \mathbf{m}_{+1}), \tag{26}$$

where $\mathbf{m}_{\pm 1}$ are the means calculated for the two classes:

$$\mathbf{m}_{\pm 1} = \frac{1}{|\{j : y_j = \pm 1\}|} \sum_{\{j:y_j=\pm 1\}} \mathbf{x}_j. \tag{27}$$

Maximization of (26) can be seen as maximization of the distance between the projected averages ($m_{\pm 1}$). Such criterion could strongly depend on the directions of the largest spread of the data. This is why the final Fisher criterion is

$$J(\mathbf{w}) = \frac{(m_{-1} - m_{+1})^2}{s_{-1}^2 + s_{+1}^2}, \tag{28}$$

where $s_{\pm 1}^2$ is the within-class variance. It results that the $\mathbf{w}$ should be

$$\mathbf{w} \propto \mathbf{S}_{\mathbf{w}}^{-1}(\mathbf{m}_{-1} - \mathbf{m}_{+1}) \tag{29}$$

where

$$\mathbf{S}_{\mathbf{w}} = \sum_{\{j:y_j=-1\}} (\mathbf{x}_j - \mathbf{m}_{-1})(\mathbf{x}_j - \mathbf{m}_{-1})^T + \sum_{\{j:y_j=+1\}} (\mathbf{x}_j - \mathbf{m}_{+1})(\mathbf{x}_j - \mathbf{m}_{+1})^T \tag{30}$$

Equation (29) defines the direction of the discriminant:

$$f(\mathbf{x}) = \mathbf{w}^T(\mathbf{x} - \bar{\mathbf{x}}) \tag{31}$$

where $\bar{\mathbf{x}}$ is the average of all the training samples.

Most linear discriminant learning methods are based on the gradient descent procedure. Its general scheme is presented in the following algorithm. To simplify the notation we define $\mathbf{w} = [b, w_1, \ldots, w_n]^T$ and $\bar{\mathbf{x}} = [1\ \mathbf{x}]^T$.

Gradient descent procedure

```
k:=0; init w_0;
do
    w_{k+1} := w_k - η(k)∇J(w_k);
    k:=k+1;
while not stop-criterion(k, θ, ∇J(w_k), η(k))
return w;
```

A typical definition of the `stop-criterion` is $|\eta(k)\nabla J(\mathbf{w}_k) < \theta|$, where θ is a user-defined parameter. The $\eta(k)$ controls the speed of learning. Sometimes $\eta(k)$ is a constant scalar below 1 and sometimes a decreasing function, such as $\eta(k) = \eta(0)/k$. In the second order forms $\eta(k)$ can be equal to $\frac{||\nabla J(\mathbf{w}_k)||^2}{\nabla J(\mathbf{w}_k)^T \mathbf{H} \nabla J(\mathbf{w}_k)}$ ($\mathbf{H}$ is the Hessian of $J(\mathbf{w}_k)$). If $\eta(k)$ is defined as $\mathbf{H}^{-1}$, then the *Newton descent algorithm* is obtained.

Another source of flexibility of the gradient descent algorithm is the definition of $J(\mathbf{w}_k)$. Moreover, the algorithm may work in *batch* or *online* mode. In the batch mode the $\nabla J(\mathbf{w}_k)$ is calculated using the whole training data set, and in the online mode $\nabla J(\mathbf{w}_k)$ is defined for a single (current) vector. The *Least Mean Square* (LMS) algorithm defines the $J(\mathbf{w}_k)$ as:

$$J(\mathbf{w}_k) = \sum_i (\mathbf{w}_k^T \bar{\mathbf{x}}_i - y_i)^2, \tag{32}$$

and modifies the weights (in the online version) as follows:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \eta(k)(y_p - \mathbf{w}_k^T \bar{\mathbf{x}}_p)\bar{\mathbf{x}}_p. \tag{33}$$

The p index is chosen randomly or is equal to $(k \mod m) + 1$.

Another definition of $J(\mathbf{w}_k)$ is the perceptron criterion:

$$J(\mathbf{w}_k) = \sum_{p \in \mathcal{P}_k} -y_p \mathbf{w}^T \bar{\mathbf{x}}_p \tag{34}$$

where $\mathcal{P}_k$ is the set of indexes of misclassified vectors. Then the weight changes are:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \eta(k) \sum_{p \in \mathcal{P}_k} y_p \bar{\mathbf{x}}_p. \tag{35}$$

Relaxation procedures use squared versions of the perceptron criterion:

$$J(\mathbf{w}_k) = \sum_{p \in \mathcal{P}_k} (\mathbf{w}_k^T \bar{\mathbf{x}}_p)^2 \qquad \text{or} \qquad J(\mathbf{w}_k) = \sum_{p \in \mathcal{P'}_k} \frac{(\mathbf{w}_k^T \bar{\mathbf{x}}_p - \beta)^2}{||\bar{\mathbf{x}}||^2} \tag{36}$$

where $\mathcal{P'}_k = \{i : \mathbf{w}_k^T \bar{\mathbf{x}}_i \leq \beta\}$ and β is a constant defining a margin. Similar idea of margins plays a prominent role in Support Vector Machine algorithms which, used with linear kernels, are an efficient linear discrimination methodology capable of optimizing the margins (see below for more).

An exhaustive description of several linear discriminant methods can be found in [Duda et al., 2001, Guyon and Stork, 2000].

3.3 Support Vector Machines

Support Vector Machines (SVMs) were introduced by Boser et al. [1992]. Initially SVMs were constructed to solve binary classification and regression problems. Today, there are several more areas where the SVM *framework* has been successfully applied [Schölkopf and Smola, 2002], for example: novelty detection (data set consistency) [Schölkopf et al., 1999, Schölkopf and Smola, 2002], clustering [Ben-Hur et al., 2001], feature selection [Fung and Mangasarian, 2003, Guyon et al., 2002, Weston et al., 2001, Schölkopf and Smola, 2002], feature extraction (kernel PCA) [Schölkopf and Smola, 2002], kernel Fisher discriminant [Schölkopf and Smola, 2002].

In application to classification problems SVMs can produce models with different kinds of decision borders – it depends on the parameters used (especially on kernel type). The borders can be linear (like in top-left plots of figure 1) or highly nonlinear (may resemble the bottom-right images of figure 1). Here the complexity of the borders not necessarily announces poor generalization, because the margin optimization (described below) takes care for proper placement of the border.

SVMs minimize the empirical risk function (10) with soft margin loss function (6) for classification problems or with ϵ-insensitive loss function (8) for regression problems. For connections of risk function and regularization with SVMs compare section 3.7 and especially table 1.

Optimal hyperplane

The construction of the *optimal hyperplane* is the fundamental idea of SVM. The optimal hyperplane separates different classes with maximal margin (the distance between the hyperplane and the closest training data point). Such goal can be defined as maximization of the minimum distance between vectors and the hyperplane:

$$\max_{\mathbf{w},b} \quad \min\{||\mathbf{x}-\mathbf{x}_i|| \ : \ \mathbf{w}^T\mathbf{x}+b=0, \quad i=1,\ldots,m\}. \tag{37}$$

The $\mathbf{w}$ and b can be rescaled in such a way that the point closest to the hyperplane $\mathbf{w}^T\mathbf{x}+b=0$, lies on a hyperplane $\mathbf{w}^T\mathbf{x}+b=\pm 1$. Hence, for every $\mathbf{x}_i$ we get: $y_i[\mathbf{w}^T\mathbf{x}_i+b] \geq 1$, so the width of the margin is equal to $2/||\mathbf{w}||$. The goal (37) can be restated as the optimization problem of *objective function* $\tau(\mathbf{w})$:

$$\min_{\mathbf{w},b} \ \tau(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 \tag{38}$$

with the following constraints:

$$y_i[\mathbf{w}^T\mathbf{x}_i+b] \geq 1 \quad i=1,\ldots,m. \tag{39}$$

To solve it a Lagrangian is constructed:

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m}\alpha_i(y_i[\mathbf{x}_i^T\mathbf{w}+b]-1), \tag{40}$$

where $\alpha_i > 0$ are Lagrange multipliers. Its minimization leads to:

$$\sum_{i=1}^{m}\alpha_i y_i = 0, \qquad \mathbf{w} = \sum_{i=1}^{m}\alpha_i y_i \mathbf{x}_i. \tag{41}$$

According to the Karush-Kuhn-Thucker (KKT) conditions [Schölkopf and Smola, 2002]:

$$\alpha_i(y_i[\mathbf{x}_i^T\mathbf{w}+b]-1) = 0, \quad i=1,\ldots,m. \tag{42}$$

The non-zero α_i correspond to $y_i[\mathbf{x}_i^T\mathbf{w} + b] = 1$. It means that the vectors which lie on the margin play the crucial role in the solution of the optimization problem. Such vectors are called *support vectors*.

After some substitutions the optimization problem can be transformed to the *dual optimization problem*:

$$\max_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{43}$$

with constraints:

$$\alpha_i \geq 0 \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{44}$$

Using the solution of this problem the decision function can be written as:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \mathbf{x}^T \mathbf{x}_i + b \right). \tag{45}$$

The kernel trick

The dot product $\mathbf{x}^T\mathbf{x}'$ in (43) and (45) can be replaced by a kernel function $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$ [Boser et al., 1992]. It extends the linear discriminant SVM to a nonlinear machine. The new decision function is:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b \right). \tag{46}$$

The dot product is the simplest kernel and may be generalized to the polynomial kernel

$$k_p(\mathbf{x}, \mathbf{x}') = [\gamma(\mathbf{x}^T\mathbf{x}') + \theta]^q, \tag{47}$$

where q is an integer and $\theta = 0$ or $\theta = 1$. Probably the most powerful is the gaussian kernel

$$k_G(\mathbf{x}, \mathbf{x}') = \exp[-\gamma ||\mathbf{x} - \mathbf{x}'||^2]. \tag{48}$$

The SVM decision function (46) with gaussian kernel is equivalent to RBF networks (67). Another popular kernel is the hyperbolic tangent

$$k_t(\mathbf{x}, \mathbf{x}') = \tanh(\gamma[\mathbf{x}^T\mathbf{x}'] + \theta). \tag{49}$$

Soft margin hyperplane

The construction of optimal hyperplane is impossible if the data set (transformed by kernels if kernels are used) is not linearly separable. To solve this problem Cortes and Vapnik [1995] introduced the *soft margin hyperplane* technique using *slack variables* ξ_i ($\xi_i \geq 0$):

$$y_i[\mathbf{w}^T\mathbf{x}_i + b] \geq 1 - \xi_i \quad i = 1, \ldots, m. \tag{50}$$

This leads to a new optimization problem:

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i \tag{51}$$

with constraints (50). It defines a *Support Vector Classifier* (SVC) with the C parameter (C-SVC) controlling the balance between training accuracy and the margin width (C must be greater than 0).

The dual optimization problem for C-SVC is defined as

$$\max_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \tag{52}$$

with constraints:

$$0 \leq \alpha_i \leq C \quad i = 1, \ldots, m, \qquad \sum_{i=1}^{m}\alpha_i y_i = 0. \tag{53}$$

ν-SVC

Schölkopf and Smola proposed the ν-SVM [Schölkopf and Smola, 2002] justifying that the C parameter of C-SVC is not intuitive. They defined a new primary optimization problem as:

$$\min_{\mathbf{w},b,\boldsymbol{\xi},\rho} \quad \tau(\mathbf{w}, \boldsymbol{\xi}, \rho) = \frac{1}{2}||\mathbf{w}||^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i \tag{54}$$

with constraints:

$$y_i[\mathbf{x}_i^T\mathbf{w} + b] \geq \rho - \xi_i, \qquad \xi_i \geq 0, \qquad \rho \geq 0 \tag{55}$$

If the ρ after the optimization procedure is greater than 0 then ν has an interesting interpretation: it is the upper bound of the fraction of vectors within the margin and the lower bound of the fraction of vectors which are support vectors.

The ν-SVM has also been stated for regression tasks [Schölkopf et al., 2000].

Regression with SVM (ϵ-SVR)

The starting point to define the SVM for regression (SVR) is the $\epsilon-$*insensitive error function* which is defined by (8).

The goal of regression can be defined as minimization of

$$\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}|y_i - f(\mathbf{x}_i)|_\epsilon. \tag{56}$$

The primary optimization problem formulation is based on two types of slack variables $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^*$: the former for $f(\mathbf{x}_i) - y_i > \epsilon$ and the latter for $y_i - f(\mathbf{x}_i) > \epsilon$. It can be stated as:

$$\min_{\mathbf{w},\boldsymbol{\xi},\boldsymbol{\xi}^*,b} \quad \tau(\mathbf{w},\boldsymbol{\xi},\boldsymbol{\xi}^*) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \tag{57}$$

with constraints:

$$f(\mathbf{x}_i) - y_i \le \epsilon + \xi_i, \qquad y_i - f(\mathbf{x}_i) \le \epsilon + \xi_i^*, \qquad \xi_i, \xi_i^* \ge 0, \quad i = 1, \ldots, m. \tag{58}$$

Quadratic programming problems

Each of the above algorithms solves its dual optimization problem by means of quadratic programming (QP). The first implementations of QP were significantly less effective than the recent ones. The dual optimization problems defined for SVC (43), C-SVC (52), ν-SVM, ϵ-SVR can be generalized to:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \mathbf{p}^T\boldsymbol{\alpha}, \tag{59}$$

where $\mathbf{p}$ depend directly on dual optimization problem (for example in the case of optimization problem defined by (43) $\mathbf{p}$ is a vector of 1's).

The crucial point of the most effective implementations is the *decomposition* of the QP problem [Osuna et al., 1997, Joachims, 1998, Platt, 1998].

$$\max_{\boldsymbol{\alpha}_B} \quad W(\boldsymbol{\alpha}_B) = (\mathbf{p} - Q_{BR}\boldsymbol{\alpha}_R)^T\boldsymbol{\alpha}_B - \frac{1}{2}\boldsymbol{\alpha}_B^T Q_{BB}\boldsymbol{\alpha}_B, \tag{60}$$

where $[Q_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j))]$. The idea is that the vector $\boldsymbol{\alpha}$ is divided into: the working part $\boldsymbol{\alpha}_B$ and the fixed one $\boldsymbol{\alpha}_R$. At particular stage only the working part is being optimized while the fixed part does not change. During the optimization procedure the subset B is changed from time to time. The most interesting examples of decomposition algorithms are SVM^{light} [Joachims, 1998] and SMO [Platt, 1998] with modifications described in [Keerthi et al., 2001]. These two algorithms differ in the working set selection technique and in the stop criterion.

3.4 Artificial Neural Networks

Artificial neural networks (ANN) represent a very broad class of different algorithms designed for classification, regression, (auto-)associations, signal processing, time series prediction, clustering etc. A number of good books

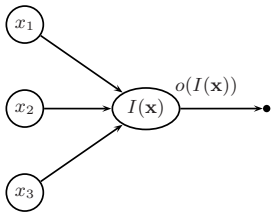


Fig. 2. A neuron. Transfer function $F(\mathbf{x}) = o(I(\mathbf{x}))$ is a superposition of output function $o(\cdot)$ and activation function $I(\cdot)$.

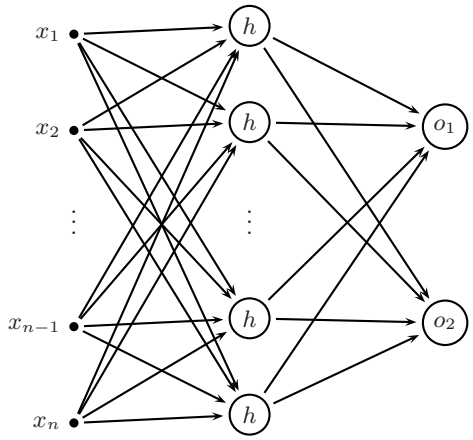


Fig. 3. An example of neural network. x_i denotes inputs, h – hidden neurons, o_j – outputs. This network is composed of input, one hidden and output layers.

on neural networks may be recommended [Bishop, 1995, Haykin, 1994, Kohonen, 1995, Ripley, 1996, Żurada, 1992]. Here only a few most popular concepts related to artificial neural networks, used in classification and regression problems, are presented.

First neural networks were proposed by McCulloch and Pitts [1943]. Neural networks are built from *neurons* which are grouped in layers. Neurons may be connected in a number of ways. For an example see figure 3. A single neuron can be seen as an operational element which realizes a *transfer function* based on incoming signals $\mathbf{x}$. Transfer function is a superposition of *output function* $o(\cdot)$ and *activation function* $I(\mathbf{x})$ – compare figure 2. If all the transfer functions realized by neurons are linear, then also the neural network realizes a linear transformation. There is a variety of transfer functions, and their type strongly determines the properties of the network they compose (see the review by Duch and Jankowski [1999]). Two best known activation functions are the *inner product* ($\mathbf{w}^t\mathbf{x}$) and the *Euclidean distance* ($||\mathbf{x} - \mathbf{t}||$). The best known output functions are the *threshold function* [McCulloch and Pitts, 1943]:

$$\Theta(I;\theta) = \begin{cases} -1 & I - \theta < 0, \\ 1 & I - \theta \geq 0, \end{cases} \tag{61}$$

the *logistic function*[5]: $\sigma(I) = 1/[1 + e^{-I}]$, and the *gaussian function* (48).

Perceptrons (or *threshold units*) are defined as the threshold output function with the inner product activation and were studied by Rosenblatt [1962]:

[5]Logistic function is a special case of sigmoidal function.

$$F(\mathbf{x};\mathbf{w}) = \Theta(\mathbf{x}^T\mathbf{w};\theta). \tag{62}$$

The perceptron learning goes according to the following update rule:

$$\mathbf{w} = \mathbf{w} + \eta[y_i - F(\mathbf{x}_i;\mathbf{w})]\mathbf{x}_i. \tag{63}$$

The most common and successful neural network is the *multilayer perceptron* (MLP). It is an extension of the concept of perceptron. MLP is a feedforward network with at least one hidden layer. The ith output of MLP (with l hidden layers) is defined by:

$$o_i(\mathbf{x};\mathbf{w}) = \tilde{o}\left(\sum_j w_{ij}^{l+1}\phi_j^l\right), \tag{64}$$

where $\tilde{o}$ may be a linear or a nonlinear function, w_{ij}^k denotes the weight connecting jth neuron in kth layer with ith neuron in layer $k+1$ (we assume that the output is the layer number $l+1$) and ϕ_i^k is the total output of ith neuron in kth layer:

$$\phi_j^0 = x_j, \qquad \phi_i^k = \sigma\left(\sum_j w_{ij}^k\phi_j^{k-1})\right) \quad k = 1,\ldots,l, \tag{65}$$

where $\sigma(\cdot)$ is a sigma–shaped function, typically the logistic function or the hyperbolic tangent function.

The MLP network may be trained with the *back-propagation algorithm* [Werbose, 1974]. The weights (connections between neurons) are adapted according to the gradient-based *delta-rule*:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}, \qquad \Delta w_{ji} = -\eta\frac{\partial E}{\partial w_{ji}} \tag{66}$$

where E is the mean squared error (11).

Neural networks with single hidden layer, using sigmoidal functions are universal approximators [Cybenko, 1989, Hornik et al., 1989], i.e. they can approximate any continuous function on a compact domain with arbitrary precision given sufficient number of neurons[6].

Decision borders of neural network classifiers using linear transfer functions only, are linear (like in top-left plots of figure 1). Nonlinear transfer function introduces nonlinearity to decision borders which can get different shapes (for example similar to the bottom-left or bottom-right plot of figure 1).

The back-propagation algorithm has been accelerated and optimized in a number of ways. For example *Quickprop* [Fahlman, 1988], *conjugate gradient*

[6]These mathematical results do not mean that sigmoidal functions always provide the optimal choice or that a good neural approximation is easy to find.

[Fletcher and Reeves, 1964] or Newton method (see section 3.2) or Levenberg-Marquardt [Levenberg, 1944, Marquardt, 1963]. Other types of algorithms are also in use. The cascade correlation (CC) is one of the best learning algorithms for the MLP networks. In CC the network is growing. For each new neuron the correlation between its output and the error of the network is maximized [Bishop, 1995].

Striving for as simple models as possible, we can eliminate the useless weights with miscellaneous *pruning techniques*, such as the *optimal brain damage* [Cun et al., 1990] and *optimal brain surgeon* [Hassibi and Stork, 1993] or with *weight decay* regularization [Hinton, 1987].

Another popular group of neural networks is the family of *radial basis function* (RBF) networks [Poggio and Girosi, 1990] applicable to both classification and regression problems. They are also universal approximators [Hartman et al., 1990, Park and Sandberg, 1991]. These networks substantially differ from the MLPs: the neurons are based on radial functions instead of sigmoidal functions, and quite different learning algorithms are used. RBF networks can be defined by:

$$f_{RBF}(\mathbf{x};\mathbf{w}) = \sum_{i}^{K} w_i G_i(\mathbf{x}) + b. \tag{67}$$

Here $G(\cdot)$ represents a radial function. The simplest type is the radial coordinate neuron ($||\mathbf{x}-\mathbf{t}||$, $\mathbf{t}$ is the center of the neuron), and the most often used one is the gaussian neuron which is nothing else but Gaussian kernel defined by (48). RBF network function (67) is, up to the sgn function, the same as the SVM model (46) with Gaussian kernel (48).

Typically the learning process of the RBF network is divided into two stages. The first one (usually unsupervised) determines the initial positions of radial neurons as centers of clusters obtained by a clustering algorithm (for example *k-means clustering* [Duda and Hart, 1973]) or as a random subset of the learning data. The second phase tunes the weights and (frequently) the centers and biases (γ in (48)) using a gradient descent algorithm [Poggio and Girosi, 1990, Bishop, 1995], *ortogonal least squares* [Chen et al., 1989] or EM algorithm [Bishop et al., 1996]. Often, some regularization terms (compare section 3.7) are added to the MSE error function [Poggio and Girosi, 1990, Bishop, 1995] to avoid the overfitting of the network.

Some RBF networks are able to automatically adjust their architectures (the number of neurons and weights in hidden layer) during the learning process, for example RAN [Platt, 1991] or IncNet [Jankowski and Kadirkamanathan, 1997].

There are many different transfer functions [Duch and Jankowski, 1999] which can be used in MLP or RBF networks. Trying different functions (especially combined into a single heterogeneous network [Jankowski and Duch, 2001]) can significantly increase the generalization capability and at the same time reduce the network complexity.

3.5 Instance based learning

The *instance based* or *similarity based* methods are a large branch of the machine learning algorithms that were developed by the pattern recognition community. A primary example of such models is the *k nearest neighbors* (kNN) algorithm [Cover and Hart, 1967], which classifies data vectors on the basis of their *similarity* to some memorized *instances*. More precisely, for a given data vector it assigns the class label that appears most frequently among its k nearest neighbors. This may be seen as an approximation of the Bayes Optimal Classifier (19), where the probabilities are estimated on the basis of the analysis of the vicinity of the classified example.

Methods restricting their analysis to a neighborhood of given object are called *local*. The locality causes that the shape of the decision border can be very complicated and has no simple and general characteristic. In the case of single neighbor classifier (1NN), the decision borders are the *Voronoi tesselation* of the feature space (see e.g. [Duda et al., 2001]).

There are many different possibilities of kNN algorithm implementation and application. For example, different distance measures can be used (possibly yielding quite different results and different decision borders), and the choice is not simple. There are some methods aiming at the optimal choice of the similarity measure [Wilson and Martinez, 1997]. Heterogenous measures are recommended when the classified objects are described by both ordered and unordered features.

Also the choice of *neighbors* is not an unambiguous task. One can select k nearest neighbors or neighbors within a given radius. Neighbors influence on the decision may be weighted according to the distance ($1/(1+dist)$ or $\max[1-dist, 0]$), flexible k depending on the region may be used, best k may be estimated by means of cross-validation, etc. (for a systematic presentation of different aspects of such methods see [Duch, 2000]).

A group of algorithms is dedicated to prototype selection (instances among which search for neighbors is done). Comparisons and reviews of these methods can be found in [Wilson and Martinez, 2000, Jankowski and Grochowski, 2004]. Algorithms that work well for prototype selection include DROP, Explore, LVQ, DEL, ICF and ENN.

3.6 Decision trees

Decision trees are hierarchical models, very popular in classification tasks. Figure 4 presents two simple examples of such trees displayed in upside-down manner – the root is placed at the top and branches grow down. The tree nodes are described by logical conditions using single features. This is the most common technique in decision trees implementations and results in decision borders perpendicular to the axes of the feature space (see figure 1). Another shapes of borders are possible when, for instance, linear combinations of features or distances from some reference points are used in logical

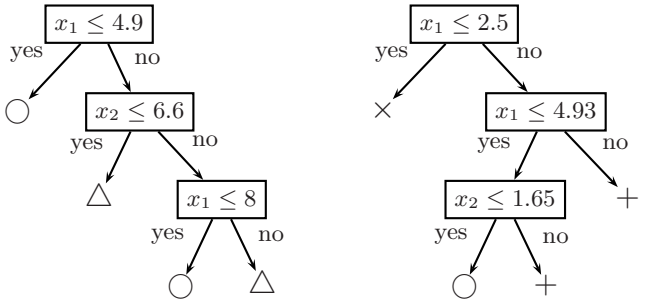


Fig. 4. Decision trees corresponding to decision borders presented in the top-left plots of figure 1.

expressions assigned to the nodes, however perpendicular borders are often preferred, because of their comprehensibility.

In some fields experts' knowledge can be directly used to specify decision rules, but more often all that is given, is a set of classified data vectors, so the best way to find decision rules is to use computational intelligence tools.

Tree growing algorithms start with a given training data set as the root node and recursively split the nodes into several disjoint parts to separate vectors assigned to different classes. Each split is described by a general logical rule, which in fact divides the whole feature space (not just the training set) into separate regions. Hence, decision trees can usually be presented in a form of logical rules. Such comprehensible models are advantageous in many fields (for instance in medical diagnosis) and provide information about particular feature relevance. The recursiveness of the processes makes feature selection a local task – useful features can be picked up even if they are valuable only in a subregion of the input space, while from the point of view of the whole space they seem to contain less information than other features.

Building optimal trees is a very complex problem, especially because the optimization of a particular split is not the same as maximization of classification accuracy for the whole model (de Sá [2001] shows that it is true even when dealing with a family of quite simple trees).

A number of different decision tree construction methods has already been published. Some of them are: *Classification and Regression Trees* (CART) [Breiman et al., 1984], ID3 [Quinlan, 1986], C4.5 [Quinlan, 1993], *Separability of Split Value* (SSV) Trees [Grąbczewski and Duch, 1999], *Fast Algorithm for Classification Trees* (FACT.

The methods are based on different ideas and use different model structures. Some of them use dichotomic splits (CART, SSV), others allow for more complex branching. Some algorithms assume a particular data distribu-

tion and use parametric statistical tests (FACT, QUEST, Cal5), others make no such assumptions.

Nevertheless, all trees can be described with the same terminology including nodes, subnodes, subtrees, leaves, branches, branch length, tree depth, class labels assigned to nodes, data vectors falling into a node, and others. The definitions are quite natural, so we do not provide them here.

There are three main aspects, where decision tree algorithms differ from each other: the way nodes are split into subnodes, the way of tree construction, which in most cases is just a search process[7], and the way of taking care for generalization (stopping criteria or pruning techniques).

Splitting criteria

Splitting nodes according to continuous and discrete inputs must be performed differently. This is why some methods (like ID3) can deal only with discrete features. Continuous attributes need to be converted to discrete before such algorithms can be applied. Good classification results can be obtained only in the company of good (external) discretization methods.

On the other side: methods like FACT and QUEST are designed to deal only with continuous input – here discrete features are converted to continuous by translating them into binary vectors of dimension equal to the number of possible discrete values and then, projected to a single dimension. The original Cal5 algorithm was also designed for continuous data, however it can be quite easily adapted to deal with discrete features.

The candidate splits of continuous features are usually binary (of the form $\{(-\infty, a],\ (a, \infty)\}$), however there exist some methods which split such input into several intervals (e.g. FACT and Cal5).

Most decision tree algorithms split the nodes with respect to the values of a single feature. Selecting the feature most eligible for the split is often closely bound up with the selection of the splitting points (CART, C4.5, SSV). An alternative strategy is applied in FACT, where the split feature is defined as the one that maximizes the value of F statistic known from the *analysis of variance* (ANOVA) method. The QUEST algorithm also calculates F statistic for continuous features, but for discrete ones it uses χ^2 statistic (to prevent from favoring the discrete attributes over continuous ones). Also Cal5 selects the split feature with a separate algorithms: either the amount of information about the classes is measured with entropy based formula (in this case the discretization must be performed first) or a coefficient is calculated for each

[7]Simple search techniques like *hill climbing* are most frequently used. The experience shows that increasing the computational efforts of the search method (e.g. using *beam search*) does not improve the generalization abilities of constructed trees [Quinlan and Cameron-Jones, 1995] – no reliable explanation of this fact is known, but it looks like more thorough search leads to solutions more specific to the training set.

feature to estimate their class separation abilities on the basis of mean squared distances within classes and between class centroids.

The most common split selection criterion is called *purity gain* or *impurity reduction*. For a split s and tree node N it is defined as:

$$\Delta I(s,N) = I(N) - \sum_i p_i I(N_i^s), \tag{68}$$

where I is a node impurity[8] measure, N_i^s is the i-th subnode of N resulting from split s, and p_i is the estimated probability of falling into N_i^s provided that the data vector falls into N (usually the quotient of the numbers of training vectors in the relevant nodes). This criterion is used in CART with the impurity measure called *Gini index*:

$$I_G(N) = 1 - \sum_{y\in\mathcal{Y}} [P(y|N)]^2. \tag{69}$$

Another impurity measure comes from the information theory and uses entropy:

$$I_E(N) = -\sum_{y\in\mathcal{Y}} P(y|N)\log_2 P(y|N). \tag{70}$$

Applied with (68) it gives the *information gain* criterion. It is the idea of ID3, also available in CART. C4.5 uses a modification of this criterion (information gain divided by *split information*) known as the impurity *gain ratio*:

$$\Delta I'(s,N) = \frac{\Delta I_E}{SI(s,N)}, \qquad SI(s,N) = -\sum_i p_i \log_2 p_i. \tag{71}$$

The idea of SSV criterion is to perform binary splits which separate as many pairs of training vectors belonging to different classes as possible, while separating possibly lowest number of pairs of vectors representing the same class:

$$SSV(s,N) = 2\cdot\sum_{y\in\mathcal{Y}} |N_1^s\cap N_y|\cdot|N_2^s\setminus N_y| - \sum_{y\in\mathcal{Y}} \min(|N_1^s\cap N_y|, |N_2^s\cap N_y|), \tag{72}$$

where N_y is the set of training vectors of class y falling into node N. Because of the second part of the formula, the SSV criterion does not fit to the impurity reduction scheme (68).

FACT implements completely different idea of nodes splitting – a linear discrimination analysis splits each node to the number of parts equal to the number of classes represented within the node. QUEST uses quadratic discriminant analysis and performs binary splits – the *two-means* clustering of the class centers is used to group classes into two *superclasses*.

[8] A node is regarded as pure if all the samples belonging to it, represent the same class.

The authors of CART also group the classes in order to make dichotomic splits. They call the technique *twoing* and group vectors striving for superclasses of possibly similar counts.

Although SSV Trees are also binary, there is no need for twoing, because the SSV criterion definition guarantees good behavior also for multiclass data.

Cal5 also applies statistical methods for splitting continuous features into intervals. A single process is responsible for both discretization and deciding whether to further split the subnodes or not. The data vectors are sorted by the values of particular feature, and the intervals are created from $-\infty$ to ∞ testing statistical hypotheses to decide the positions of interval borders. After discretization the adjacent intervals can be merged: leaf-intervals are merged if they share the majority class, and non-leaves if they consist of vectors from the same set of classes (after discarding infrequent class labels by means of a statistical test).

Generalization

Building decision trees which maximally fit the training data usually ends up in overfitted, large trees with leaves classifying only a few cases, and thus does not provide general knowledge about the problem. To keep the trees simpler and more general the methods may use *stopping criteria*. A simple idea is to keep the numbers of vectors in the nodes above a specified threshold. Another way is to extend the separability criterion with a penalty term conforming to the Minimum Description Length principle. In both cases it is difficult to define criteria which yield results close to optimal. A more reasonable way is to use statistical tests to verify the significance of the improvement introduced by a split (such technique is a part of C4.5).

Stopping tree growth can save much time, but makes optimum solutions less probable. Usually, better generalization is obtained when building overfitted trees and *pruning* them. There are many different pruning methods. They may concern tree depth, number of vectors in a node etc. Their parameters can be chosen on the basis of a cross-validation test performed within the training data (like in CART, QUEST or SSV Tree).

Missing values

The training data set can be incomplete (some feature values for some data vectors may be inaccessible). Putting arbitrary values in the place of the missing ones should be taken into consideration only if no other methods can be used. Much caution about it is advised.

CART introduced the idea of *surrogate splits*, which is to collect some spare splits, maximally similar to the main one but using different features. The surrogate splits are used when the classified data vector does not provide the value necessary to check the main split condition.

When C4.5 is trained on incomplete data, the gains are scaled with a factor equal to the frequency of observing the feature values among the vectors falling

into the node – each training sample has a weight associated with it, and the weights affect the p_i values of (71).

The SSV criterion simply does not include the vectors with missing values in calculations. Such vectors are regarded to fall into both subnodes. When an incomplete vector is to be classified by SSV Tree, all the branches of non-zero probability are checked and their leaves are treated as a single leaf to determine the dominating class.

Other decision tree ideas

There are many other decision tree algorithms, not mentioned here. Some of them are just slight modifications of the basic ones (e.g. NewID is an improved ID3), and some are quite original.

TDDT (*Top-Down Decision Trees*) is the algorithm available in the MLC++ library [Kohavi et al., 1996]. It is similar to C4.5 and introduces some changes to protect against generating small nodes which are created by the information gain based strategies when discrete features have multiple values.

There are also some decision thee approaches, where node membership is decided by more than one feature. *Dipol criteria* (similar to SSV) have been used [Bobrowski and Kręt owski, 2000] to construct decision trees where splitting conditions use linear combinations of features. *Linear Machine Decision Trees* (LMDT) [Utgoff and Brodley, 1991] use linear machines at tree nodes. They try some variable elimination, but in general such methods are not eligible for feature selection – instead they construct new features as combinations of the original ones.

Oblique Classifier (OC1) [Murthy et al., 1994] combines heuristic and non-deterministic methods to determine interesting linear combination of features. It searches for trees by *hill climbing*.

Option decision trees [Buntine, 1993] allow several alternative splits of the same node. The final classification is determined with relevant calculations and an analysis of probabilities.

On the basis of the SSV criterion heterogeneous decision trees have been proposed [Duch and Grąbczewski, 2002]. Their split conditions may concern distances from some reference vectors.

3.7 Regularization and complexity control

Regularization and model complexity control are very important, because they help us obtain most accurate and stable models for given data $\mathcal{D}$.

There are many possible definitions of the regularizer $\Omega(f)$ augmenting the empirical risk (13). One of them is the *weight decay* proposed by Hinton [1987], also called *ridge regression*, defined as square L_2-norm of parameters:

$$\Omega_{wd}[f] = ||\mathbf{w}||_2^2. \tag{73}$$

We assume that the model f is parameterized by $\mathbf{w}$ ($f = f(\mathbf{x};\mathbf{w})$) or its superset. The regularizer used with the empirical risk yields:

$$R_{wd}[f] = R_{emp} + \lambda||\mathbf{w}||^2, \tag{74}$$

but it may be used also with other risk functions. For example in the SV framework it is used in conjunction with loss function defined for classification (6), regression (8) or *empirical quantile function* [Schölkopf et al., 1999], to control the spread of the margin (see (38), (51) and (56)).

Regularization via $\Omega_{wd}[f]$ term is equivalent to assuming Gaussian prior distribution on the parameters of f in f_{MAP} (17). On the assumption of Gaussian priors for parameters w_i with zero mean ($P(w_i) \propto \exp[-w_i^2/\sigma_a^2]$) and independence assumption on parameters $\mathbf{w}$ we have

$$-\log_2 P(f) \quad \propto \quad -\ln \prod_i \exp[-w_i^2/\sigma_a^2] \quad \propto \quad ||\mathbf{w}||^2. \tag{75}$$

Ω_{wd} is also used with *penalized logistic regression* (PLR), where loss is defined as $\log(1+exp[-yf(\mathbf{x})])$. On the other side, PLR may also be used with $||\mathbf{w}||_1$ regularizer, which in *regularized adaboost* is used with loss $\exp[-yf(\mathbf{x})]$ and in *lasso regression* with squared loss.

A variant of ridge regression is the *local ridge regression*:

$$\Omega_{lrr}[f] = \sum_i \lambda_i w_i^2, \tag{76}$$

intended for local smoothing of the model f (compare [Orr, 1996]). Commonly, the regularization parameters are determined with a cross-validation.

Weigend et al. [1990, 1991] proposed a *weight elimination* algorithm which can be seen as another regularization penalty:

$$\Omega_{we}(f) = \sum_i \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2}, \tag{77}$$

where w_0 is a constant. This regularizer is not so restrictive as Ω_{wd} and allows for some amount of parameters of large magnitude. For the weight elimination the λ from (13) may become a part of learning [Weigend et al., 1990, 1991].

A regularization of the form

$$\Omega_{mlp2ln}[f] = \sum_i w_i^2(w_i - 1)^2(w_i + 1)^2 \tag{78}$$

was used to build and learn especial type of MLP networks (MLP2LN). Such regularizer forces the weights to become close to 0, +1 or −1, which is very advantageous from the point of view of logical rule extraction [Duch et al., 1998].

Very interesting property of regularization was shown by Bishop [1995]. He proved that learning with Tikhonov regularization [Tikhonov, 1963, Tikhonov and Arsenin, 1977] is equivalent to learning with noise.

Table 1 displays some well known algorithms in the context of loss functions and regularizers.

Algorithm	**Loss**	**Regularizer**
Weight decay/ridge regression	$(y - f(\mathbf{x}))^2$	$\|\|\mathbf{w}\|\|_2^2$
Original SV classifier	$\max\{0, 1 - yf(\mathbf{x})\}$	$\|\|\mathbf{w}\|\|_2^2$
SV for regression	$\max\{0, \|y - f(\mathbf{x})\| - \epsilon\}$	$\|\|\mathbf{w}\|\|_2^2$
Penalized logistic regression	$\log(1 + \exp[-yf(\mathbf{x})])$	$\|\|\mathbf{w}\|\|_2^2$, $\|\|\mathbf{w}\|\|_1$
Regularized adaboost	$\exp[-yf(\mathbf{x})]$	$\|\|\mathbf{w}\|\|_1$
Lasso regression	$(y - f(\mathbf{x}))^2$	$\|\|\mathbf{w}\|\|_1$
Local ridge regression	$(y - f(\mathbf{x}))^2$	$\sum_i \lambda_i w_i^2$
Weight elimination	$(y - f(\mathbf{x}))^2$	$\sum_i (w_i^2/w_0^2)/(1 + w_i^2/w_0^2)$
MLP2LN	$(y - f(\mathbf{x}))^2$	$\sum_i w_i^2(w_i - 1)^2(w_i + 1)^2$
Linear discrimination	miscellaneous	$\|\|\mathbf{w}\|\|_2^2$
Learning with noise	$(y - f(\mathbf{x}))^2$	noise

Table 1. Combinations of loss functions and regularizers.

Regularization may be used as embedded feature selection or a neuron pruning technique. Beside the regularization several other techniques were developed to control the complexity of learning machines. One of them uses the *cross-validation* technique (compare page 8) *for learning*: the submodels are learned and the influence of selected parameter(s) is measured and validated on the test part of the data. This type of complexity control is used for example in decision trees (see section 3.6) or (as mentioned above) for estimation of the adequate strength of regularization. A similar goal may be reached with Monte Carlo scheme used in place of the cross-validation randomization.

Controlling complexity of artificial neural networks may be done by adjusting the structure of neural network (the number of neurons and the weights of connections between neurons) to the complexity of considered problem. ANN's which can change their structure during learning are called *ontogenic*. For more information on ontogenic neural networks see [Fiesler, 1994, Platt, 1991, Jankowski and Kadirkamanathan, 1997, Adamczak et al., 1997, Cun et al., 1990, Hassibi and Stork, 1993, Finnoff et al., 1993, Orr, 1996, Mezard and Nadal, 1989, Frean, 1990, Campbell and Perez, 195, Fahlman and Lebiere, 1990].

3.8 Complex systems

Learning machines generate very different models, demonstrating large variability [Breiman, 1998]. Different models combined into a single, larger model facilitate data analysis from different points of view. A non-deterministic learning process may produce different models even when trained several times on the same data. A deterministic machine can also give different results when trained on different data samples (e.g. generated in a cross-validation manner or with different bootstrap methods). Quite different models should be expected when completely different learning techniques are applied. A number of different models can be used as a *committee* (or *ensemble*) – an averaged decision of several experts is likely to be more accurate and more stable. Averaging of results can be done in several ways, compare the strategies of known ensembles such as *bagging* [Breiman, 1998], *adaboost* [Freund and Schapire, 1996, 1997, Schapire et al., 1998], arcing [Breiman, 1996], regionboost [Maclin, 1998], *stacking* [Wolpert, 1992], *mixture of local experts* [Jacobs et al., 1991], *hierarchical mixture of experts* [Jordan and Jacobs, 1993] and *heterogenous committees* [Jankowski et al., 2003]. Cross-validation can be used to test the generalization abilities of models and to build committees at the same time. Sometimes (especially when the validation results are unstable) it is more reasonable to combine the validated models than to use their parameters to train a new model on the whole training data set. Chapter **??** presents more information on ensemble models.

Each expert has an area of *competence*, and the same applies to computational intelligence models. It is worth to analyze the competence of committee members and to reflect it in committee decisions [Duch et al., 2002].

Some learning algorithms should be applied only to appropriately prepared data (standardized, with preselected features or vectors, discretized etc.). In such cases, the data preparation stage of learning should always be regarded as a part of the system.

Recently a growing interest in *meta-learning* techniques may be observed, aimed at finding the most successful learning algorithms and their parameters for given data. The methods include simple search procedures and more advanced learning techniques for the meta level.

4 Some remarks on learning algorithms

No algorithm is perfect or best suited for all the applications. Selection of the most accurate algorithm for a given problem is a very difficult and complex task. Efficient exploration of the space of possible models in pursuit of optimal results requires a lot of knowledge about the advantages and dangers of applying different sorts of methods to particular domains.

First of all, the computational complexity of learning methods (with regard to the number of features, the number of vectors, data complexity and possibly other quantities) decides on their applicability to the task.

Learning processes should always be accompanied by validation tests. The subject is discussed in more detail in chapter **??**. Here, we will just point out one of the dangers of incorrect validation: when supervised feature selection (or other supervised data transformation) techniques are used as the first stage of classification (or regression) they must be treated as inseparable part of a complex model. Validation of classification models on pre-transformed data is usually much faster, but yields unreliable (and over-optimistic) results. Especially when the number of features describing data vectors is large (similar to the number of training vectors or bigger), it is easy to select a small number of features for which simple models demonstrate very good results of cross-validation tests, but their generalization abilities are illusory – this may be revealed when the feature selection is performed inside each fold of the cross-validation as a part of a complex system.

Another very important problem of learning methods is data incompleteness. Much care must be devoted to data analysis when missing data are replaced by some arbitrary values, some averages or even with techniques like *multiple imputation*, because the substitutions may strongly affect the results. The safest way to analyze incomplete data is to use methods which can appropriately exhibit the influence of the missing values on the model representation.

Acknowledgements

We are very grateful to Isabelle Guyon and Włodek Duch for their very precious suggestions and constructive comments.

References

R. Adamczak, W. Duch, and N. Jankowski. New developments in the feature space mapping model. In *Third Conference on Neural Networks and Their Applications*, pages 65–70, Kule, Poland, October 1997. Polish Neural Networks Society.

A Ben-Hur, D. Horn, H.T. Siegelman, and V. Vapnik. Suppor vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

C. M. Bishop, M. Svensén, and C. K. I. Williams. EM optimization of latent-variable density models. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, Cambridge, MA, 1996. MIT Press.

L. Bobrowski and M. Krętowski. Induction of multivariate decision trees by using dipolar criteria. In D. A. Zighed, J. Komorowski, and J. M. Żytkow, editors,

Principles of data mining and knowledge discovery: 5th European Conference: PKDD'2000, pages 331–336, Berlin, 2000. Springer Verlag.

B. E. Boser, I. M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, Pittsburgh, PA, 1992. ACM Press.

L Breiman. Arcing classifiers. Technical report, Statistics Department, University of California, Berkeley, 1996.

L. Breiman. Bias-variance, regularization, instability and stabilization. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, pages 27–56. Springer, 1998.

L. Breiman, J. H. Friedman, A. Olshen, and C. J. Stone. *Classification and regression trees*. Wadsworth, Belmont, CA, 1984.

W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall,London, 1993.

C. Campbell and C. V. Perez. Target switching algorithm: a constructive learning procedure for feed-forward neural networks. *Neural Networks*, pages 1221–1240, 195.

B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the Ninth European Conference on Artificial Intelligence*, pages 147–149, 1990.

S. Chen, S. A. Billings, and W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896, 1989.

V. Cherkassky and F. Mulier. *Learning from data*. Adaptive and learning systems for signal processing, communications and control. John Wiley & Sons, Inc., New York, 1998.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13(1): 21–27, January 1967.

Y. Le Cun, J. Denker, and S. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kauffman.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

J. P. Marques de Sá. *Pattern Recognition. Concepts, Methods and Applications*. Springer Verlag, 2001.

W. Duch. Similarity based methods: a general framework for classification, approximation and association. *Control and Cybernetics*, 29(4):937–968, 2000.

W. Duch, R. Adamczak, and K. Grąbczewski. Extraction of logical rules from backpropagation networks. *Neural Processing Letters*, 7:1–9, 1998.

W. Duch and K. Grąbczewski. Heterogeneous adaptive systems. In *Proceedings of the World Congress of Computational Intelligence*, Honolulu, May 2002.

W. Duch, Ł. Itert, and K. Grudziński. Competent undemocratic committees. In L. Rutkowski and J. Kacprzyk, editors, *Neural Networks and Soft Computing. Proceedings of the 6th International Conference on Neural Networks and Soft Computing (ICNNSC)*, Advances in Soft Computing, pages 412–417, Zakopane, Poland, June 2002. Springer-Verlag.

W. Duch and N. Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2:163–212, 1999.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

R. O. Duda, P. E. Hart, and D. G. Stork. *Patter Classification*. John Wiley and Sons, New York, 2001.

S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann, 1988.

S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

E. Fiesler. Comparative bibliography of ontogenic neural networks. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 793–796, 1994.

W. Finnoff, F. Hergert, and H. G. Zimmermann. Improving model detection by nonconvergent methods. *Neural Networks*, 6(6):771–783, 1993.

R. A. Fisher. The use of multiple measurements in taxonomic problems. 1936. Reprinted in *Contributions to Mathematical Statistics*, John Wiley & Sons, New York, 1950.

R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Computer journal*, 7:149–154, 1964.

M. Frean. *Small nets and short paths: optimizing neural computation*. PhD thesis, Center for cognitive science. University of Edinburgh, 1990.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55 (1):119–139, August 1997.

G. M. Fung and O. L. Mangasarian. A feature selection newton method for support vector machine classification. *Computational Optimization and Aplications*, 2003.

K. Grąbczewski and W. Duch. A general purpose separability criterion for classification systems. In *Proceedings of the 4th Conference on Neural Networks and Their Applications*, pages 203–208, Zakopane, Poland, June 1999.

I. Guyon and D. G. Stork. Linear discriminant and support vector classiers. In A. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classiers*, pages 147–169. MIT Press, 2000.

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 2002.

E. J. Hartman, J. D. Keeler, and J. M. Kowalski. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2(2): 210–215, 1990.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan Kaufmann, 1993.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2001.

S. Haykin. *Neural Networks - A Comprehensive Foundation*. Maxwell MacMillian Int., New York, 1994.

G. E. Hinton. Learning translation invariant recognition in massively parallel networks. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*, pages 1–13, Berlin, 1987. Springer-Verlag.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

R. A. Jacobs, M. L. Jordan, S. J. Nowlan, and J. E. Hinton. Adaptive mixtures of local exports. *Neural Computation*, 79(3), 1991.

N. Jankowski and W. Duch. Optimal transfer function neural networks. In *9th European Symposium on Artificial Neural Networks*, pages 101–106, Bruges, Belgium, April 2001.

N. Jankowski, K. Grąbczewski, and W. Duch. *GhostMiner 3.0*. FQS Poland, Fujitsu, Kraków, Poland, 2003.

N. Jankowski and M. Grochowski. Comparison of instances seletion algorithms: I. Algorithms survey. In *Artificial Intelligence and Soft Computing*, pages 598–603. Springer, June 2004.

N. Jankowski and V. Kadirkamanathan. Statistical control of RBF-like networks for classification. In *7th International Conference on Artificial Neural Networks*, pages 385–390, Lausanne, Switzerland, October 1997. Springer-Verlag.

T. Joachims. Advances in kernel methods — support vector learning. chapter Making large-scale SVM learning practical. MIT Press, Cambridge, MA., 1998.

G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, San Mateo, 1995. Morgan Kaufmann Publishers.

M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. Technical Report AIM-1440, 1993.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13: 637–649, 2001.

R. Kohavi. *Wrappers for performance enhancement and oblivious decision graphs*. PhD thesis, Stanford University, 1995.

R. Kohavi, B. Becker, and D. Sommerfield. Improving simple bayes. In *Proceedings of the European Conference on Machine Learning*, 1997.

R. Kohavi, D. Sommerfield, and J. Dougherty. Data mining using MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, pages 234–245. IEEE Computer Society Press, 1996. http://www.sgi.com/tech/mlc.

T. Kohonen. *Self-organizing maps*. Springer, Heidelberg Berlin, 1995.

S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:76–86, 1951.

K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly journal of applied mathematics II*, 2:164–168, 1944.

R. Maclin. Boosting classifiers regionally. In *Proceeding of AAAI*, 1998.

D. W. Marquardt. An algorithm for least squares estimation of nonlinear parameters. *Journal of the Society of industrial and applied mathematics*, 11(2):431–441, 1963.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

M. Mezard and J. P. Nadal. Learning in feedforward layered networks: the tiling algorithm. *Journal of physics*, 22:2191–2203, 1989.

T. Mitchell. *Machine learning*. McGraw Hill, 1997.

S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, August 1994.

M. Orr. Introduction to radial basis function networks. Technical report, Centre for Cognitive Science, University of Edinburgh, 1996.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *In Proceedings of CVPR'97*, pages 130–136, New York, NY, 1997. IEEE.

J. Park and I. W. Sandberg. Universal approximation using radial basis function networks. *Neural Computation*, 3(2):246–257, 1991.

Z. Pawlak. Rough sets. *International Journal of Computer and Information Sciences*, 11(5):341–356, 1982.

J. C. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3:213–225, 1991.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, Cambridge, MA., 1998.

T. Poggio and F. Girosi. Network for approximation and learning. *Proceedings of the IEEE*, 78:1481–1497, 1990.

F. Provost and P. Domingos. Well-trained PETs: Improving probability estimation trees. Technical Report IS-00-04, Stern School of Business, New York University, 2000.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

J. R. Quinlan. Programs for machine learning, 1993.

J. R. Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *IJCAI*, pages 1019–1024, 1995.

B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:445–471, 1978.

F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanics*. Spartan, Washington, DC, 1962.

R. Schalkoff. *Pattern Recognition: statistical, structural and neural approaches*. Wiley, 1992.

R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26 (5):1651–1686, 1998.

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report TR 87, Microsoft Research, 1999.

B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207–1245, 2000.

C. E. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois Press, Urbana, 1949.

A. N. Tikhonov. On solving incorrectly posed problems and method of regularization. *Doklady Akademii Nauk USSR*, 151:501–504, 1963.

A. N. Tikhonov and V. Y. Arsenin. *Solutions of ill-posed problems*. W. H. Winston, Washington, DC, 1977.

P. E. Utgoff and C. E. Brodley. Linear machine decision trees. Technical Report UM-CS-1991-010, Department of Computer Science, University of Massachusetts, 1991.

A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Back–propagation, weight elimination and time series prediction. In *Proceedings of the 1990 Connectionist Models Summer School*, pages 65–80, Los Altos/Palo Alto/San Francisco, 1990. Morgan Kaufmann.

A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight elimination with application to forecasting. In R. P. Lipmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 875–882, San Mateo, CA, 1991. Morgan Kaufmann.

P. J. Werbose. *Beyond regression: New tools for prediction and analysis in the bahavioral sciences*. PhD thesis, Harvard Univeristy, Cambridge, MA, 1974.

J. Weston, A. Elisseeff, and B. Schölkopf. Use of the ℓ_0-norm with linear models and kernel methods. Technical report, Biowulf Technologies, 2001.

D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38:257–286, 2000.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

B. Zadrozny and C. Elkan. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *Proc. 18th International Conf. on Machine Learning*, pages 609–616. Morgan Kaufmann, San Francisco, CA, 2001.

J.M. Żurada. *Artificial neural systems*. West Publishing Company, 1992.