Inductive Principles for Learning from Data

Vladimir Cherkassky

Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, Minnesota 55455, U.S.A.
cherkass@ece.umn.edu

Abstract. Face detection/recognition applications involve ill-defined concepts (such as 'face' or 'person') that cannot be specified a priori in terms of a small set of features. This implies the need for learning unknown class decision boundaries from data (i.e., images with known class labels). This task is a special case of a generic problem of predictive classification or pattern recognition, where the goal is to estimate class decision boundaries using available (training) data. There are many learning methods (i.e., constructive algorithms) for predictive classification. However, most approaches are heuristic, due to inherent complexity of estimation with finite data, and the lack of conceptual framework. This paper describes several principled approaches (called *inductive principles*) for estimating dependencies from data. The focus is on the general conceptual framework and on the major issues related to learning, rather than on specific learning algorithms. The following inductive principles for learning with finite data are described: penalization, structural risk minimization, Bayesian inference and minimum description length. Finally, we briefly describe a new powerful learning algorithm called Support Vector Machine, and its applications to face detection.

1 Introduction

As evidenced by many presentations at this ASI, existing approaches to face recognition fall into two groups:

(1) *Understanding / modeling of human recognition*, where the focus is on psychological models of human perception and (face) recognition process.

(2) *Statistical Learning models*, where the goal is to estimate unknown dependency from available data (samples). Specifically, the problem face detection/recognition is often formulated as the problem of *predictive classification*, where the goal is to estimate class decision boundaries (i.e., to discriminate between face and non-face images) in some input (feature) space.

The subject of this paper is the latter approach, even though successful face recognition systems are likely to combine both approaches. Face recognition applications place particularly stringent requirements on the learning problem, since the data is high-dimensional (i.e., 100 to 1,000 -dimensional vectors corresponding to image pixels) and the number of samples (images) is large (at least a few thousand). Numerous learning approaches have been proposed and used for face recognition. These include, for example, neural network, statistical pattern recognition, wavelet, genetic optimization and fuzzy methodologies (see examples elsewhere in this volume). However, it is difficult to objectively compare these methodologies, because most are heuristic, i.e. they lack general conceptual framework and often define the problem in terms of the solution approach. In this paper we present first the mathematical formulation of the learning problem and then describe several inductive principles. It is argued that a clear understanding of the basic issues and concepts related to learning is necessary for developing sound constructive learning methods. Finally, we describe a new constructive method called Support Vector Machine (SVM) for implementing structural risk minimization inductive principle. This method is based on Statistical Learning Theory (aka Vapnik-Chervonenkis or VC-theory). Several recent SVM applications to face detection are also reviewed.

2 Formulation of the Learning Problem

Learning is the process of estimating an unknown (input, output) dependency or structure of a System using a limited number of observations (see Fig.1). The general learning scenario involves three components: a Generator of random input vectors, a System which returns an output for a given input vector, and the Learning Machine which estimates an unknown (input, output) mapping of the System, from the observed (input, output) samples. This formulation describes many practical learning problems such as interpolation, regression, classification, clustering and density estimation.

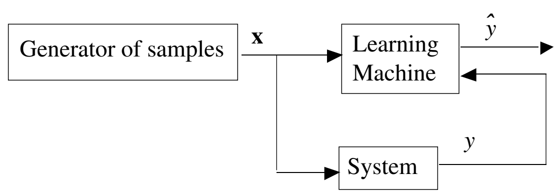

Fig. 1. A Learning Machine uses observations of the System to form an approximation of its output.

Each component of a learning system in Fig.1 can be further described in mathematical terms:

The Generator (or sampling distribution) produces random vectors $\mathbf{x} \in \Re^d$, drawn independently from a fixed probability density $p(\mathbf{x})$ which is unknown.

The System produces an output value y for every input vector $\mathbf{x}$ according to the fixed conditional density $p(y|\mathbf{x})$, which is also unknown.

The Learning Machine is capable of implementing a set of approximating functions $f(\mathbf{x}, \omega)$, $\omega \in \Omega$ where Ω is a set of parameters. The set of approximating functions of the Learning Machine should be chosen *a priori*, before the formal inference (learning) process is begun. Ideally, the choice approximating functions reflects a priori knowledge about the System (unknown dependency). This choice is application-dependent and is outside the scope of the learning problem. Common sets of approximating functions include: multilayer perceptron networks, radial basis functions, wavelets, Fourier (harmonic) expansion etc.

There is an important distinction between two types of approximating functions: linear in parameters vs nonlinear in parameters. Learning (estimation) procedures using the former are also refered to as *linear* , whereas those using the latter are called *nonlinear* .

The Learning problem can be now stated as follows:
Given a set of n training samples

$$(\mathbf{x}_i, y_i),\ (i = 1, \ldots, n) \tag{1}$$

produced according to (unknown) joint probability density function (pdf)

$$p(\mathbf{x}, y) = p(\mathbf{x})\, p(y|\mathbf{x}) \tag{2}$$

select a function (from the set of approximating functions $f(\mathbf{x}, \omega)$) which best approximates the System's response.

The quality of an approximation produced by the Learning Machine is measured by the loss or discrepancy measure $L(y, f(\mathbf{x}, \omega))$ between the output produced by the System and the Learning Machine for a given point $\mathbf{x}$. The expected value of the loss is called the (expected or prediction) *risk functional*:

$$R(\omega) = \int L(y, \mathrm{f}(\mathbf{x}, \omega))\ \mathrm{p}(\mathbf{x}, y) d\mathbf{x} dy \tag{3}$$

Learning is the process of estimating the function $f(\mathbf{x}, \omega_0)$ which minimizes the risk functional over the set of functions supported by the Learning Machine, using only the training data ($p(\mathbf{x}, y)$ is not known). We cannot expect to find $f(\mathbf{x}, \omega_0)$ exactly, so we denote $f(\mathbf{x}, \omega^*)$ as the estimate of the optimal solution obtained with finite training data using some learning procedure.

The generic learning problem can be subdivided into four classes of common problems: classification, regression, density estimation, and clustering/vector quantization [Vapnik, 1995; Cherkassky and Mulier, 1998]. For each of these problems, the nature of the loss function and the output (y) differ. However, the goal of minimizing the risk functional based only on training data is common to all learning problems. Next we briefly describe classification and density estimation tasks which are directly relevant to face recognition/detection applications.

Classification

In a (two-class) classification problem, the output of the System takes on only two (symbolic) values $y = \{0, 1\}$, corresponding to two classes (i.e., face and non-face). Hence, the output of the Learning Machine need only take on two values as well, so the set of functions $\boldsymbol{f}(\mathbf{x}, \omega)$, $\omega \in \Omega$ becomes a set of *indicator* functions. A commonly used loss function for this problem measures the classification error:

$$L(y, \mathrm{f}(\mathbf{x}, \omega)) = \begin{cases} 0 & \text{if } y = \mathrm{f}(\mathbf{x}, \omega) \\ 1 & \text{if } y \neq \mathrm{f}(\mathbf{x}, \omega) \end{cases} \tag{4}$$

Using this loss function, the risk functional (3) quantifies the probability of misclassification. Learning then becomes the problem of finding the indicator function $\boldsymbol{f}(\mathbf{x}, \omega_0)$ (classifier) which minimizes the probability of misclassification using only the training data.

Density Estimation

For estimating the density of $\mathbf{x}$, the output of the System is not used. The output of the Learning Machine now represents density, so $\boldsymbol{f}(\mathbf{x}, \omega)$, $\omega \in \Omega$ becomes a set of densities. Using the loss function

$$L(\boldsymbol{f}(\mathbf{x}, \omega)) = -\boldsymbol{ln}\, \boldsymbol{f}(\mathbf{x}, \omega) \tag{5}$$

in the risk functional (3) gives

$$R(\omega) = \int -ln f(\mathbf{x}, \omega) p(\mathbf{x}) d\mathbf{x} \tag{6}$$

which is a common risk functional (called Maximum Likelihood) used for density estimation. Minimizing (6) using only the training data $\mathbf{x}_1, \ldots, \mathbf{x}_K$ leads to the density estimate $\boldsymbol{f}(\mathbf{x}, \omega_0)$.

Note that a given learning task (i.e., classification or regression) can be solved by minimizing (3) directly if the density $\boldsymbol{p}(\mathbf{x}, y)$ were known. Moreover, if the density is known, the training data is not even needed.This implies that density estimation is the most general and hence the most difficult type of learning problem. It leads to the fairly obvious principle for estimation problems with finite data [Vapnik, 1995]: *Do not solve a given learning problem by indirectly solving a harder problem (of density estimation) as an intermediate step.* This principle is rarely followed under statistical and neural network approaches where classification problem is solved via class density estimation, as in [Bishop, 1995; Ripley, 1996].

So far, we provided the formulation of the learning problem without describing any learning methods for obtaining an estimate $\boldsymbol{f}(\mathbf{x}, \omega^*)$ of the "true dependency". The next issue is: how should a Learning Machine use training data? - The answer is given by the concept known as an *inductive principle*. An inductive principle is a general prescription for obtaining an estimate $f(\mathbf{x}, \omega^*)$ of the "true dependency" in the class of approximating functions, from the available (finite) training data. An inductive principle tells *what* to do with the data, whereas the learning method specifies *how* to obtain an estimate. Hence, a learning method is a constructive implementation of an inductive principle for selecting an estimate $f(\mathbf{x}, \omega^*)$ from a particular set of functions $f(\mathbf{x}, \omega)$. For a given inductive principle, there are (infinitely) many learning methods, corresponding to different sets of approximating functions.

There are two groups of inductive principles, depending on the amount of a priori knowledge about the true dependency. *Classical (parametric)* methods assume strong a priori knowledge, reflected in a pre-specified parametric model (comprising a set of approximating functions), so that only model parameters need to be estimated using training data. *Adaptive (or flexible)* methods use weak a priori knowledge, so that both the model structure (complexity) and its parameters need to be estimated from data.

3 Inductive Principles in Classical Statistics

The classical approach, as proposed by Fisher [1952], divides the learning problem into two portions: specification and estimation. *Specification* consists of determining the parametric form of the unknown underlying distributions, while *estimation* is the process of determining parameters which characterize the specified distributions. Classical theory focuses on the problem of estimation and sidesteps the issue of specification.

Note that the learning problem formulation does not assume that the true dependency $f(\mathbf{x}, \omega_0)$ belongs to a set of approximating functions $f(\mathbf{x}, \omega)$, $\omega \in \Omega$. Classical approaches to the learning problem depend on much stricter assumptions than those posed in the general learning formulation since they assume the true parametric form of the model is known (specified). Then available data is used to estimate the values of a parameter vector $\mathbf{w}$. The two inductive principles commonly used in the classical learning process are *Empirical Risk Minimization* (ERM) and *Maximum Likelihood* (ML). Maximum likelihood is a specific form of the more general ERM principle obtained when using particular loss function (5). ERM approach assumes that the training data is sufficiently representative of the future data (i.e., unknown distribution $p(\mathbf{x}, y)$). Hence, an estimate $f(\mathbf{x}, \mathbf{w}^\bullet)$ minimizing the empirical risk (training error)

$$R_{emp}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, f(\mathbf{x}_i, \mathbf{w})\right) \tag{7}$$

would also minimize the prediction risk (3). This leads to the following ERM prescription for estimating the (parametric) model from data:

$$\mathbf{w}^{\bullet} = \arg\min_{\mathbf{w}} R_{emp}(\mathbf{w}) \tag{8}$$

As an example, consider the classical formulation of classification, based on the following restricted learning model: The conditional densities for each class, $\boldsymbol{p}(\mathbf{x}|y=0)$ and $\boldsymbol{p}(\mathbf{x}|y=1)$ are estimated from data using classical (parametric) density estimation using the ML inductive principle. These estimates are denoted as $\boldsymbol{p}_0(\mathbf{x}, \alpha^*)$ and $\boldsymbol{p}_1(\mathbf{x}, \beta^*)$, respectively. The probability of occurrence of each class, called *prior* probabilities, $P(y=0)$ and $P(y=1)$, are assumed to be known or estimated, i.e. as a fraction of samples from a particular class in the training set. Using Bayes theorem it is possible to use these quantities to determine for a given observation $\mathbf{x}$, the probability of that observation belonging to each class. These probabilities, called *posterior* probabilities can be used to construct a discriminant rule that describes how an observation $\mathbf{x}$ should be classified, so as to minimize the probability of error. This rule chooses the output class which has the maximum posterior probability, as described next.

First, Bayes rule is used to calculate the posterior probabilities for each class:

$$\begin{aligned} \boldsymbol{p}(y=0|\mathbf{x}) &= \frac{\boldsymbol{p}_0(\mathbf{x}, \alpha^*) P(y=0)}{\boldsymbol{p}(\mathbf{x})} \\ \boldsymbol{p}(y=1|\mathbf{x}) &= \frac{\boldsymbol{p}_1(\mathbf{x}, \beta^*) P(y=1)}{\boldsymbol{p}(\mathbf{x})} \end{aligned} \tag{9}$$

There is no need to compute the denominator (normalizing constant) in (9), since the decision rule is a comparison of the relative magnitudes of the posterior probabilities. Once the posterior probabilities are estimated, the following rule (discriminant function) is used to classify input $\mathbf{x}$:

$$\boldsymbol{f}(\mathbf{x}) = \begin{cases} 0 & \text{if } \boldsymbol{p}_0(\mathbf{x}, \alpha^*) P(y=0) > \boldsymbol{p}_1(\mathbf{x}, \beta^*) P(y=1) \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

Note that the parameters α^* and β^* used in (10) were estimated via ML. Therefore, the classical approach applies the ERM inductive principle *indirectly* to first estimate the densities, which are then used to formulate the decision rule. This differs from applying the ERM inductive principle *directly* to minimize the empirical risk

$$R_{emp}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} I(y_i = \boldsymbol{f}(\mathbf{x}_i, \mathbf{w})) \tag{11}$$

by estimating the expected risk functional for classification (3) using the empirical risk (11).

There are two problems with using ML for learning problems with finite data. First, with finite data, the classification problem should be solved directly, rather than via density estimation. Posing the problem directly will require less data samples for a given level of solution accuracy. The second problem is that with finite data the knowledge of the parametric form of a model may lead to sub-optimal solutions. For example, Fisher [1952] suggested to use a linear discriminant function even when an optimal decision boundary is known to be quadratic. The intuitive explanation is that a simpler (linear) model has fewer parameters and hence can be estimated more accurately than a more complex (quadratic) model, with small number of samples. This suggests the need for model *complexity control* for learning problems with finite data. The lack of complexity control is the main deficiency of ERM. All inductive principles for adaptive learning described later in this paper have provisions for complexity control.

4 Concepts for Adaptive Learning

All learning methods use a priori knowledge to choose a class of approximating functions of a Learning Machine, $f(\mathbf{x},\omega),\ \ \omega \in \Omega$. For example, *parametric methods* use a very *restricted set* of approximating functions of pre-specified parametric form, so that only a fixed number of parameters need to be determined from data. *Adaptive methods* use a *wide set* of functions (universal approximators) capable of approximating any continuous mapping. This class of approximating functions is very flexible (overparameterized) and allows for multiple solutions when a model is estimated with finite data. Hence, additional constraints (penalty) need to be imposed on a potential of a function (within a class $f(\mathbf{x},\omega),\ \ \omega \in \Omega$) to be a solution to the learning problem.

In summary, in order to form a *unique* generalization (model) from finite data, *any learning process* requires:

(a) a (wide, flexible) set of approximating functions $f(\mathbf{x},\omega),\ \ \omega \in \Omega$ chosen a priori.

(b) *complexity penalty* (or assumptions) used to impose constraints on a potential of each function from the class (a) to be a solution. Usually, these assumptions provide (explicitly or implicitly) ordering of the functions according to some measure of their flexibility to fit the data.

(c) *an inductive principle (or inference method)*, i.e. a general prescription for combining assumptions (b) with available training data in order to produce an estimate of (unknown) true dependency. An inductive principle specifies *what* needs to be done; it does not say *how* to do it.

(d) *a learning method,* i.e. constructive (computational) implementation of an inductive principle for a given class of approximating functions.

The distinction between inductive principles and learning methods is crucial for understanding and further advancement of the methods. For a given inductive principle, there may be (infinitely) many learning methods, corresponding to different classes of approximating functions and/or different optimization techniques. For example, under Empirical Risk Minimization (ERM) inductive principle one seeks to find a solution $f(\mathbf{x},\omega^*)$ that minimizes the empirical risk(training error), as a substitute for (unknown) expected risk(true error).

Depending on the chosen loss function and the chosen class of approximating functions, ERM inductive principle can be implemented as a variety of methods (i.e., Maximum Likelihood estimators, linear regression, polynomial methods, fixed-topology neural networks etc.). ERM inductive principle is typically used in a classical (parametric) setting when the model is given (specified) first and then its parameters are estimated from data. This approach works well only when the number of training samples is large relative to the (pre-specified) model complexity (or the number of free parameters).

Adaptive methods do not make restrictive assumptions about the unknown dependency, so they use a very flexible class of approximating functions. For example, neural network classifiers (with variable number of hidden units) can form arbitrary class decision boundaries. Hence, the problem is to estimate both the *model flexibility or complexity* (i.e., the number of hidden units) and its parameters (neural network weights). The problem of choosing (optimally) the model complexity from data is also called *model selection*. Adaptive methods differ mainly on the basis of a class of approximating functions used by a method. Most practical adaptive methods developed in statistics and neural networks use classes of functions *nonlinear* in parameters. Hence in adaptive methods estimation of model parameters is a difficult task of *nonlinear optimization*.

5 Inductive principles for learning with finite data

The main issue in adaptive learning with finite data is choosing the candidate model of the right complexity to describe the training data. Various inductive principles differ in terms of:

- representation (encoding) of a penalty (or complexity ordering) on a set of approximating functions;
- applicability (of a principle) when the true model does not belong to the set of approximating functions;
- the mechanism for combining a complexity penalty with training data;
- availability of constructive procedures (learning algorithms) for a given principle.

This section provides an overview of inductive principles for the learning problem given in section 2.

Penalization (regularization) inductive principle. Under this approach, one assumes a very flexible (i.e. with many 'free' parameters) class of approximating functions $f(\mathbf{x},\omega),\ \ \omega \in \Omega$. However, in order to restrict the solutions, a penalization (regularization) term is added to the Empirical Risk to be minimized:

$$R_{pen}(\omega) = R_{emp}(\omega) + \lambda\phi\left[f(\mathbf{x},\omega)\right] \qquad (12)$$

Here $R_{emp}(\omega)$ denotes the empirical risk (training error), and the penalty $\phi\left[f(\mathbf{x},\omega)\right]$ is a non-negative functional associated with each possible estimate $f(\mathbf{x},\omega)$. Parameter $\lambda > 0$ controls the strength of the penalty relative to the term $R_{emp}(\omega)$. Note that the penalty term $\phi[f(\mathbf{x},\omega)]$ is independent of the training data. Under this framework, a priori knowledge is included in the form of the penalty term, and the strength of such knowledge is controlled by the value of regularization parameter λ. For example, if λ is very large, then the result of minimizing $R_{pen}(\omega)$ does not depend on the data; whereas for small λ the final model does not depend on the penalty functional. For many common classes of approximating functions, it is possible to develop functionals $\phi[f(\mathbf{x},\omega)]$ which measure complexity . The 'optimal' value of λ (providing smallest prediction risk) is usually chosen using resampling methods. Thus under this approach, the optimal model estimate is found as a result of a trade-off between fitting the data and a priori knowledge (i.e., penalty term).

Many heuristic procedures can be cast (at least conceptually) in the penalization framework. For example, neural networks employ gradient-descent (stochastic optimization) techniques for minimizing the empirical risk functional. One way to avoid overfitting with over-parameterized models (such as neural networks) is to stop the training early, i.e. before reaching minimum. Such *early stopping* can be interpreted as a form of penalization, where a penalty is defined on a path (in the space of model parameters) corresponding to the successive model estimates obtained during gradient-descent training. The solutions are penalized according to the number of gradient descent steps taken along this curve, i.e. the distance from the starting point (initial conditions) in the parameter space.

Structural Risk Minimization (SRM). Under SRM, approximating functions of a Learning Machine are ordered according to their complexity, i.e. form a *nested structure:*

$$S_0 \subset S_1 \subset S_2 \subset \ldots \tag{13}$$

For example, in the class of polynomial approximating functions, the elements of a structure are polynomials of a given degree. Condition (13) is satisfied since polynomials of degree m are a subset of polynomials of degree $(m+1)$. The goal of learning is to choose an optimal element of a structure (i.e. polynomial degree) and estimate its coefficients from a given training sample. For approximating functions *linear* in parameters (such as polynomials), the complexity is given by the number of free parameters. For functions nonlinear in parameters, the complexity is defined as VC-dimension [Vapnik, 1995]. By definition, a structure provides ordering of its elements according to their complexity (i.e., VC dimension denoted by h):

$$h_1 \leq h_2 \ldots \leq h_k \ldots$$

The optimal choice of model complexity provides the minimum of the Prediction Risk (3). Statistical Learning Theory [Vapnik, 1995] provides *analytic* upper-bound estimates for Prediction Risk. These estimates are used for complexity control, i.e. choosing an optimal element of a structure, under SRM inductive principle.

Here we only describe *constructive distribution-independent* bounds for classification. Other types of VC generalization bounds can be found in [Vapnik, 1995]. For two-class classification problems, the following bound for generalization ability of a learning machine holds with probability of at least $1-\eta$ simultaneously for all approximating functions of a learning machine:

$$R(\omega) \le R_{emp}(\omega) + \frac{\varepsilon}{2}\left(1+\sqrt{1+\frac{4R_{emp}(\omega)}{\varepsilon}}\right) \qquad (14)$$

where

$$\varepsilon = \varepsilon\left(\frac{n}{h}, \frac{-\boldsymbol{ln}\,\eta}{n}\right) = a_1 \frac{h\left(\boldsymbol{ln}\frac{a_2 n}{h}+1\right) - \boldsymbol{ln}(\eta/4)}{n} \qquad (15)$$

Statistical Learning Theory [Vapnik,1982, Vapnik,1995] proves that the values of constants a_1, a_2 must be in the range $0 < a_1 \le 4$, $0 < a_2 \le 2$. The values $a_1 = 4$, $a_2 = 2$ have been derived for the worst case distribution (discontinuous density function). For practical applications, smaller values for constants a_1 and a_2 should be chosen empirically. According to [Vapnik, 1995; Cherkassky and Mulier, 1998], the *confidence level* $1-\eta$ is chosen depending on the training set size as:

$$\eta = \boldsymbol{min}\left(\frac{4}{\sqrt{n}}, 1\right) \qquad (16)$$

The bound (14) can be conveniently presented as:

$$R(\omega) \le R_{emp}(\omega) + \Phi\left(R_{emp}(\omega), n/h, -\boldsymbol{ln}\,\eta/n\right) \qquad (17)$$

where the second term in the right-hand side is called the *confidence interval*, as it estimates the difference between the training error and the true error. The confidence interval Φ should not be confused with the confidence level $1-\eta$. Let us analyze the behavior of Φ as a function of sample size n, with all other parameters fixed. It can be readily seen that the confidence interval mainly depends on ε, which monotonically decreases (to zero) with n according to (15). Hence, Φ also monotonically decreases with n, as can be intuitively expected. Moreover, detailed analysis shows strong dependency of the confidence interval Φ on the ratio n/h, and we can distinguish two main regimes [Vapnik, 1995]:

- small (or finite) sample size, when the ratio of the number of training samples to the VC-dimension of approximating functions is small (say, less than 20);
- large sample size, when this ratio is large.

With small samples, there is a need to match complexity (capacity) of approximating functions to the available data. Under SRM, this is accomplished by choosing the (optimal) structure element providing minimum of the bound (14), as shown in Fig. 2.

Finally, we note that the structure (13) can be defined in various ways, i.e. via

- *dictionary representation*

$$\boldsymbol{f}_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^{m} w_i \, \boldsymbol{g}(\mathbf{x}, \mathbf{v}_i) \qquad (18)$$

where $\boldsymbol{g}(\mathbf{x}, \mathbf{v}_i)$ is a set of basis functions with adjustable parameters $\mathbf{v}_i$, and w_i are linear coefficients. Both w_i and $\mathbf{v}_i$ are estimated to fit the training data. Representation (18) defines a structure, since $\boldsymbol{f}_1 \subset \boldsymbol{f}_2 \subset \ldots \subset \boldsymbol{f}_k \subset \ldots$. For example, feedforward neural networks indexed by the number of hidden units m represent a dictionary structure (18).

- *penalization formulation* (12);
- *input preprocessing*;
- *initialization* of parameters of a (nonlinear) optimization algorithm (i.e., backpropagation);
- *stopping* conditions of an optimization algorithm.

See [Cherkassky and Mulier, 1998] for examples of these structures.

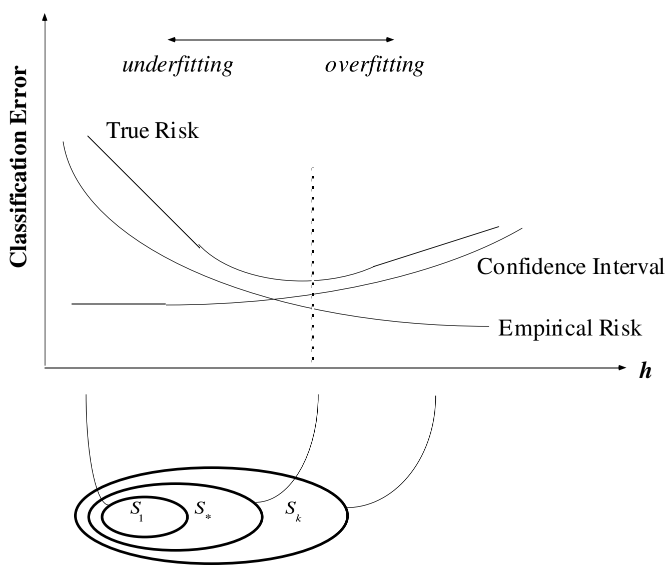

Fig. 2. An upper bound on the true (expected) risk and the empirical risk as a function of h (for fixed n).

Bayesian Inference. This type of inference uses additional a priori information about approximating functions in order to obtain a unique predictive model from finite data. This knowledge is in the form of the so-called *prior probability distribution*, i.e. the probability of any function (from the set approximating functions) to be the true (unknown) function. This adds subjectivity to the design of a learning machine, since the final model depends largely on a good choice of priors. Moreover, the very notion that the prior distribution adequately captures prior knowledge may not be acceptable in many situations, i.e. when we need to estimate a constant (but unknown) parameter. On the other hand, Bayesian approach provides an effective way of encoding prior knowledge, and can be a powerful tool when used by experts.

Bayesian inference is based on the classical Bayes formula for updating prior probabilities using the evidence provided by the data:

$$P[\text{model}|\text{data}] = \frac{P[\text{data}|\text{model}]P[\text{model}]}{P[\text{data}]} \tag{19}$$

where

$P[\text{model}]$ is *prior* probability (before the data is observed)

$P[\text{data}]$ is the probability of observing training data

$P[\text{model}|\text{data}]$ is *a posterior* probability of a model given the data

$P[\text{data}|\text{model}]$ is the probability that the data is generated by a model, also known as *likelihood* .

Let us consider the general case of (parametric) density estimation, when the class of density functions supported by the learning machine is a parametric set, i.e. $\boldsymbol{f}(\mathbf{x},\mathbf{w})$, $\mathbf{w}\in\Omega$ is a set of densities where $\mathbf{w}$ is an m-dimensional vector of 'free' parameters (m is fixed). It is also assumed that the unknown density $\boldsymbol{f}(\mathbf{x},\mathbf{w}_0)$ belongs to this class. Given a set of i.i.d. training data $\mathbf{X}=[\mathbf{x}_1,\ldots,\mathbf{x}_n]$, the probability of seeing this particular data set as a function of $\mathbf{w}$ is

$$P[\text{data}|\text{model}] = P(\mathbf{X}|\mathbf{w}) = \prod_{i=1}^{n} \boldsymbol{f}(\mathbf{x}_i,\mathbf{w}) \tag{20}$$

Recall that choosing the model (parameter $\mathbf{w}^*$) maximizing likelihood $P(X|\mathbf{w})$ amounts to ML.

A priori density function

$$P[\text{model}] = \boldsymbol{p}(\mathbf{w}) \tag{21}$$

gives the probability of any (implementable) density $\boldsymbol{f}(\mathbf{x},\mathbf{w})$, $\mathbf{w}\in\Omega$ to be the true one. Then Bayes formula gives:

$$\boldsymbol{p}(\mathbf{w}|\mathbf{X}) = \frac{P(\mathbf{X}|\mathbf{w})\boldsymbol{p}(\mathbf{w})}{P(\mathbf{X})} \tag{22}$$

Usually, the prior distribution is taken rather broad, reflecting general uncertainty about 'correct' parameter values. Having observed the data, this prior distribution is converted into posterior distribution according to Bayes formula. This posterior distribution will be more narrow, reflecting the fact that it is consistent with the observed data.

There are two distinct ways to use Bayes formula for obtaining an estimate of unknown p.d.f. The true Bayesian approach is to average over all possible models (implementable by a learning machine), which gives the following p.d.f. estimate:

$$\Theta(\mathbf{x}|\mathbf{X}) = \int f(\mathbf{x},\mathbf{w})\, p(\mathbf{w}|\mathbf{X})\, d\mathbf{w} \tag{23}$$

where $p(\mathbf{w}|\mathbf{X})$ is given by the Bayes formula (22). Equation (23) provides an example of an important technique in Bayesian inference called *marginalization*, which involves integrating out redundant variables (such as parameters $\mathbf{w}$). The estimator $\Theta(\mathbf{x}|\mathbf{X})$ has many attractive properties [Bishop, 1995]. In particular, the final model is a weighted sum of all possible predictive models, with weights given by the evidence (or posterior probability) that each model is correct. However, multi-dimensional integration (due to the large number of parameters $\mathbf{w}$) presents a challenging problem. Standard numerical integration is impossible, whereas analytic evaluation is feasible only under certain restrictive assumptions, i.e. when the posterior density has the same form as a prior (typically assumed to be Gaussian). When Gaussian assumptions do not hold, various forms of random sampling also known as Monte Carlo methods have been proposed to evaluate integrals (23) directly [Bishop, 1995].

Another way to implement Bayesian approach is to choose an estimate $f(\mathbf{x},\mathbf{w}^*)$ maximizing posterior probability $p(\mathbf{w}|\mathbf{X})$. This is known as the maximum a posterior probability (MAP) estimate. MAP formulation can be related to penalization formulation (12) with an explicit form of regularization parameter [Friedman,1994]. Choosing the optimal value of regularization parameter is equivalent to finding a 'good' prior. There has been some work done to tailor priors to the data, i.e. the so-called MLII techniques [Berger, 1985]. However, such empirical Bayes approach contradicts the original notion of data-independent prior knowledge.

Although the penalization inductive principle can be interpreted in terms of Bayesian formulation, they have a different motivation. The Bayesian methodology is used to encode a priori knowledge about multiple, general, user-defined characteristics of the target function. The goal of penalization is to perform complexity control by encoding a priori knowledge about function smoothness in terms of a penalty functional. Bayesian model selection tends to penalize more complex models, by choosing the model with largest evidence; but this does not guarantee the best generalization performance (or minimum prediction risk). On the other hand, formulations provided by Penalization framework and Statistical Learning Theory are based on the explicit minimization of the prediction risk.

Minimum Description Length (MDL). This principle is based on the information-theoretic analysis of the randomness concept. In contrast to all other inductive principles which use statistical distributions to describe an unknown model, the MDL approach regards *models as codes*, i.e. encodings of the training data. The main idea is that any data set can be appropriately encoded, and its *code length* represents an inherent property of the data, which is directly related to the generalization capability of the model (i.e., code).

Kolmogorov [1965] introduced the notion of *algorithmic complexity* for characterization of randomness of a data set. He defined the algorithmic complexity of a data set to be the shortest binary code describing this data. Further, the randomness of a data set can be related to the length of the binary code, i.e. the data is random if it cannot be compressed significantly. Rissanen [1978; 1989] proposed to use Kolmogorov's characterization of randomness as tool for inductive inference, known as MDL principle.

In order to illustrate the MDL principle, consider the training data set:

$$(\mathbf{x}_i, y_i),\ (i = 1, \ldots, n)$$

where samples $(\mathbf{x}_i, y_i)$ are drawn randomly and independently from some (unknown) distribution. Let us further assume that training data corresponds to classification problem, where the class label $y = \{0, 1\}$ and $\mathbf{x}$ is d-dimensional feature vector.

The problem of estimating dependency between $\mathbf{x}$ and y can be formulated under MDL inductive principle as follows:

Given a data object $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, is a binary string $y_1, \ldots, y_n$ random?

The binary string $\mathbf{y} = (y_1, \ldots, y_n)$ can be encoded using n bits. However, if there are systematic dependency in the data captured by the model $y = \boldsymbol{f}(\mathbf{x})$, we can encode the output string $\mathbf{y}$ by a possibly shorter code that consists of two parts:

- the model having code length $L(\text{model})$
- the error term specifying how the actual data differs from the model predictions, with a code length $L(\text{data}|\text{model})$.

Hence, the total length l of such a code for representing binary string $\mathbf{y}$ is:

$$l = L(\text{model}) + L(\text{data}|\text{model}) \quad (24)$$

and the coefficient of compression for this string is:

$$K(\text{model}) = \frac{l}{n} \quad (25)$$

If the coefficient of compression is small, then the string is not random, and the model captures significant dependency between $\mathbf{x}$ and y

Note that the optimal (MDL) model achieves balance between the complexity of the model and the error term in (24). It can be intuitively expected that the shortest description length model provides accurate representation of the unknown dependency and hence minimum prediction risk. Vapnik [1995] gives formal proof of the theorem that justifies the MDL principle (for classification problems).

Informally, this theorem states that minimizing the coefficient of compression corresponds to minimizing the probability of misclassification (for future data).

The MDL approach provides very general conceptual framework for learning from samples. In fact, the notion of compression coefficient (responsible for generalization) does not depend on the knowledge of the codebook structure, the number of tables in the codebook, the number of training samples etc. Moreover, the MDL inductive principle does not even use the notion of a statistical distribution and thus avoids the controversy between the Bayesian and frequentist interpretation of probability. Unfortunately, the MDL framework does not tell us how to construct 'good' codebooks with a small number of tables yet accurate representation of the training data. Developing effective constructive procedures under MDL principle remains an open research problem. In practice, MDL may be used for model selection for restricted types of models which allow simple characterization of the model description length, such as decision trees [Rissanen, 1989]. However, application of MDL to other types of model, i.e. feedforward neural networks has not been successful, due to difficulty in developing optimal encoding of the network in a data-dependent fashion.

We conclude this section by summarizing properties of various inductive principles (see Table 1). All inductive principles use a (given) class of approximating functions. In adaptive methods, this class is typically flexible (overparameterized) and allows for multiple solutions when a model is estimated with finite data. A priori knowledge effectively constrains functions in this class, in order to produce a unique predictive model. Different ways to represent a priori knowledge and model complexity are indicated in the first row of the Table. The second row describes constructive procedures for complexity control. Various mechanisms for model selection are described in the third row. Finally, the last row of the table indicates applicability of each inductive method when there is a mismatch between a priori knowledge and the truth, i.e. in a situation when the set of approximating functions does not include the true dependency. In the case of a mismatch, the Bayesian inference is not applicable (since the prior probability of the truth is zero), while all other inductive principles still apply.

Major inductive principles described above form a basis for various adaptive learning procedures. An obvious question is: which inductive principle is best for learning with finite data? There seems to be no clear answer. Each major inductive principle has its own school of followers, who claim its superiority and generality over the others. For example, Bishop [1995] suggests that MDL can be viewed as an approximation to the Bayesian inference. On the other hand, Rissanen[1989] claims that MDL approach 'provides a justification for the Bayesian techniques, which often appear as innovative but arbitrary and sometimes confusing'. Vapnik [1995] suggests superiority of SRM over Bayesian inference and shows close connection between SRM and MDL inductive principle. This situation is rather surprising and clearly unsatisfactory. Meaningful (empirical) comparisons could be certainly helpful, but are not readily available, mainly because each inductive approach comes with its own set of assumptions and terminology. In this respect, the comparison in Table 1 may be helpful for developing future comparisons. Each inductive principle, when reasonably applied, usually yields an acceptable solution for practical applications. Hence, experts in a particular approach tend to promote it as the best. It should be also emphasized that in most practical problems the success is mainly due to proper encoding / utilization of the application domain knowledge, rather than learning itself. However, the use of domain knowledge is, strictly speaking, outside the formal framework of predictive learning.

In our experience, Structural Risk Minimization appears to be the most powerful inductive principle for learning problems with finite data [Cherkassky and Mulier, 1998]. In particular, SRM provides useful conceptual framework for understanding existing statistical and neural network learning methods, and for developing new methodologies such as Support Vector Machines described next.

	Penalization	SRM	Bayesian	MDL
representation of a priori knowledge or complexity	penalty term	structure	prior distr.	codebook
constructive procedure for complexity control	min. of penalized risk	opt. element of a structure	aposteriori distribution	not defined
method for model selection	resampling	analytic bound on pred. risk	marginaliza-tion	min code length
applicability when the true model does not belong to the set of approx. functions	yes	yes	no	yes

Table 1 Comparison of inductive principles.

6 Support Vector Machines with Applications to Face Detection

There are two known strategies for minimizing the VC-bound (17) for classification, corresponding to two constructive implementations of the SRM inductive principle:

(1) *Keep the confidence interval fixed, and minimize the empirical risk.* This is done by specifying a structure where the value of the confidence interval is fixed for a given element S_m. Examples include all statistical and neural network methods using dictionary representation, where the number of basis functions (features) m specifies an element of a structure. For a given m, the empirical risk is minimized using numerical optimization. For a given amount of data, there is an optimal element of a structure (value of m) providing smallest estimate of expected risk.

(2) *Keep the value of the empirical risk fixed (small) and minimize the confidence interval.* This approach requires a special structure, such that the value of the empirical risk is kept small (say, zero misclassification error) for all approximating

functions. Under this strategy, an optimal element of a structure would minimize the value of the confidence interval. Implementation of the second strategy leads to a new class of learning methods called Support Vector Machines (SVM).

Next we present an overview of SVM for classification problems. The goal is to explain (informally) SVM conceptual framework. For complete description of SVM, see [Vapnik, 1995; Cherkassky and Mulier, 1998]. Let us describe first a structure on a set of linear indicator functions that keeps the value of the empirical risk fixed (i.e., zero) for all approximating functions. Such a structure implements the second strategy for SRM by minimizing the confidence interval.

A linear indicator function capable of separating the training data without error is called a *separating hyperplane* (see Fig. 3). Suppose training data consisting of n samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, $\mathbf{x} \in \Re^d$, $y \in \{+1, -1\}$ can be separated by the hyperplane decision function:

$$D(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + w_0 \tag{26}$$

with appropriate coefficients $\mathbf{w}$ and w_0. The assumption about linearly separable data will be later relaxed; however it allows clear explanation of the SVM approach. A separating hyperplane satisfies the constraints which define the separation of the data samples:

$$y_i\left[(\mathbf{w} \cdot \mathbf{x}_i) + w_0\right] \geq 1 \qquad i = 1, \ldots, n \tag{27}$$

For a given training data set, all possible separating hyperplanes can be represented in the form (27). Note that the formulation of the separating hyperplane allows us to solve the classification problem directly. It does not require estimation of density as an intermediate step.

The minimal distance from the separating hyperplane to closest data point is called the *margin* and denoted by τ. A separating hyperplane is called *optimal* if the margin is the maximum possible. It is intuitively clear that a larger margin corresponds to better generalization. The larger the margin, the more separation between the classes. The SVM framework presented next shows how to formally describe an optimal hyperplane in terms of the training data. The distance between the separating hyperplane and a sample $\mathbf{x}'$ is $|D(\mathbf{x}')| / \|\mathbf{w}\|$ as shown in Fig. 4. Assuming that a margin τ exists, all training samples obey the inequality:

$$\frac{y_k D(\mathbf{x}_k)}{\|\mathbf{w}\|} \geq \tau, \; k = 1, \ldots, n \tag{28}$$

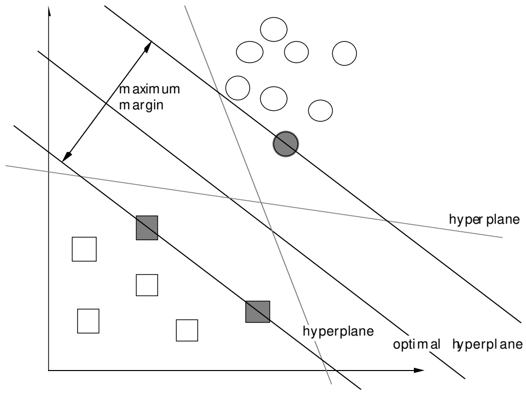

Fig. 3. Separating hyperplanes in a two dimensional space. An optimal hyperplane has maximal margin. The data points at the margin (indicated in gray) are called the support vectors since they define the optimal hyperplane.

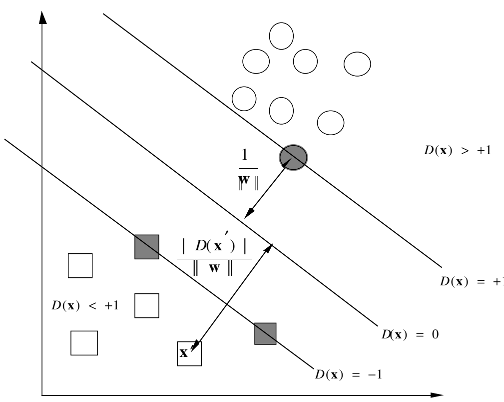

Fig. 4. The decision boundary of the optimal hyperplane is defined by points $\mathbf{x}$ for which $D(\mathbf{x}) = 0$. The distance between a hyperplane and any sample $\mathbf{x}'$ is $|D(\mathbf{x}')| / \|\mathbf{w}\|$. The distance between a support vector and the optimal hyperplane is $1/\|\mathbf{w}\|$.

Determining the optimal hyperplane amounts to finding the $\mathbf{w}$ which maximizes the margin τ. Note that there are an infinite number of solutions that differ only in scaling of $\mathbf{w}$. In order to limit solutions, fix the scale on the product of τ and norm of $\mathbf{w}$: $\tau\|\mathbf{w}\| = 1$.

Thus, maximizing the margin τ is equivalent to minimizing the norm of $\mathbf{w}$. An *optimal separating hyperplane* is one that satisfies condition (27) above and additionally minimizes

$$\eta(\mathbf{w}) = \|\mathbf{w}\|^2 \qquad (29)$$

with respect to both $\mathbf{w}$ and w_0. The data points that exist at the margin, or equivalently, the data points for which (27) is an *equality* are called the *support vectors* (Figure 4). Since the support vectors are data points closest to the decision surface, they determine the location of the optimal separating hyperplane. The decision surface of the optimal hyperplane can be written as a linear combination of the support vectors:

$$D(\mathbf{x}) = \sum_{i=1}^{N_s} \alpha_i^* y_i^s \left(\mathbf{x} \cdot \mathbf{x}_i^s\right) + w_0^* \qquad (30)$$

where $\left(\mathbf{x}_i^s, y_i^s\right)$, $(i = 1, \ldots, N_s)$ are the support vectors.

The generalization ability of the optimal separating hyperplane can be directly related to the number of support vectors. Assuming an optimal hyperplane can be constructed with a small number of support vectors (relative to the training set size), it will have good generalization ability, *even in high dimensional space*. This can be related to the MDL inductive principle: in order to define the optimal margin hyperplane one needs only to specify the support vectors and their class labels. Then according to [Vapnik, 1995], the test error is bounded by the compression coefficient, i.e. the ratio of the number of bits needed to encode support vectors to the number of bits to encode all training samples.

Since the hyperplane is used to implement SVM approximating functions, its VC dimension need to be estimated. The VC dimension for a set of (linear) hyperplanes is $d+1$. However, the hyperplanes with a given maximum margin form a subset of this set, and may have a smaller VC dimension. Vapnik [1995] provides the following bound:

For the hyperplane functions (27) satisfying the constraint $\|\mathbf{w}\|^2 \le c$, the VC dimension is bounded by

$$h \le \boldsymbol{min}\left(r^2 c, d\right) + 1 \qquad (31)$$

where r is the radius of the smallest sphere that contains the training input vectors $\left(\mathbf{x}_1, \ldots, \mathbf{x}_n\right)$.

Now it is possible to directly control the complexity of the hyperplane (i.e., VC dimension) *independent of dimensionality of the sample space*. Specifically, consider a structure on the set of separating hyperplanes according to increasing complexity by the norm of the weights $\|\mathbf{w}\|^2$:

$$S_k = \left\{(\mathbf{w} \cdot \mathbf{x}) + w_0 : \|\mathbf{w}\|^2 \le c_k\right\} \text{ where } c_1 < c_2 < c_3 \ldots \qquad (32)$$

Recall that the guaranteed risk for classification problems is the sum of the empirical risk and the confidence interval:

$$R(\mathbf{w}) \le R_{emp}(\mathbf{w}) + \Phi \qquad (33)$$

Since the separating hyperplane always has zero empirical risk, the guaranteed risk is minimized by *minimizing the confidence interval*. The confidence interval is minimized by minimizing the VC dimension h, which according to (31), corresponds to minimizing the norm of the weights $\|\mathbf{w}\|^2$.

Finding an optimal hyperplane for the separable case is a *quadratic optimization* problem of minimizing (29) with linear constraints (27) given the training data $(\mathbf{x}_i, y_i)$, $i = 1,\ldots,n$, $\mathbf{x} \in \Re^d$. The solution to this problem consists of $d+1$ parameters. For data of moderate dimension d, this problem can be solved using quadratic programming. For very high dimensional spaces, it is not practical to solve the problem in the present form. However, this problem can be translated into a *dual form* which is is easier to solve because in the dual formulation the problem size depends on the number of samples rather than dimension d. For precise dual SVM formulation see [Vapnik, 1995; Cherkassky and Mulier, 1998].

The SVM model described above is still not very useful, as it cannot handle *nonseparable* training data and *nonlinear* decision boundaries. So we briefly describe next SVM modifications addressing these issues.

First, consider linear separating hyperplanes for nonseparable data. In this case, some of the data samples fall inside the margin, and the SVM attempts to achieve two (conflicting) goals:

(a) maximize the margin;

(b) minimize the number of nonseparable samples, i.e. samples falling within the margin. This is (approximately) achieved by minimizing the total deviation (of nonseparable samples) from the margin boundary.

The trade-off between these two goals leads to SVM formulation called the *soft margin hyperplane* [Vapnik, 1995]. This hyperplane is defined by parameters $\mathbf{w}, w_0$ which minimize the functional:

$$\frac{C}{n}\sum_{i=1}^{n}\xi_i + \frac{1}{2}\|\mathbf{w}\|^2 \tag{34}$$

where the first term is the sum of deviations ξ_i of nonseparable samples from the margin boundary, and the second term defines the margin (as in the original formulation (27) (29)). The constant C effectively controls the trade-off between SVM complexity and the proportion of nonseparable samples. Thus C is a regularization parameter which must be set by a user or selected via resampling.

The second issue (i.e., *nonlinear* decision boundary) is solved by:

(1) Mapping the input data to high-dimensional feature space via nonlinear mapping $\mathbf{X} \rightarrow \mathbf{Z}$ specified a priori (see Fig. 5). Let us denote $\boldsymbol{g}_j(\mathbf{x}),\ j = 1,\ldots,m$ as a set of nonlinear transformation functions defined apriori. These functions map the vector $\mathbf{x}$ into an m-dimensional feature space $\mathbf{z} = \{\boldsymbol{g}_1(\mathbf{x}),\ldots,\boldsymbol{g}_m(\mathbf{x})\}$.

(2) Constructing an optimal (linear) hyperplane in this feature space. The complexity of SVM decision boundary is independent of the feature space dimensionality, which can be very large (or even infinite).

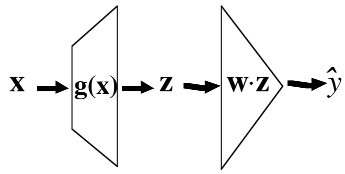

Fig. 5. The SVM maps input data $\mathbf{x}$ into a high dimensional feature space $\mathbf{z}$ using a nonlinear function $\boldsymbol{g}$. A linear approximation in the feature space (with coefficients $\mathbf{w}$) is used to estimate the output.

Notice that SVM formulation solves the conceptual problem of constructing an *optimal linear decision boundary* in a high-dimensional $\mathbf{z}$-space. Moreover, the high-dimensional mapping $\mathbf{X} \rightarrow \mathbf{Z}$ ensures that nonlinear decision boundaries (in $\mathbf{x}$-space) become linear boundaries (in $\mathbf{z}$-space). For example, the functions $\boldsymbol{g}_j(\mathbf{x}),\ j = 1,\ldots,m$ could correspond to polynomial terms of the components of $\mathbf{x}$ up to a certain order (including interaction terms). Linear decision boundaries in the feature space would then map to polynomial decision boundaries in the input space. For instance, a two dimensional input vector $\mathbf{x} = (x_1, x_2)$ mapped using third order polynomials, will be transformed to a 16-dimensional feature space. In practical applications, the dimensionality of the feature space is quite large. For example, mapping 100-dimensional input vectors (i.e., 10 by 10 pixel images) using third-order polynomials would yield $\mathbf{z}$-space dimensionality over 10^6.

Hence, there is a *technical obstacle* of solving a large-size quadratic optimization problem in $\mathbf{z}$-space. This technical problem is addressed next. The original SVM optimization formulation requires only calculation of the inner product between vectors in the $\mathbf{x}$ space. Similarly, following mapping $\mathbf{X} \rightarrow \mathbf{Z}$, solving the SVM optimization problem would require only determining inner products in the feature space defined by the basis functions $\boldsymbol{g}_j(\mathbf{x}),\ j = 1,\ldots,m$. It turns out, that for many mappings $\mathbf{X} \rightarrow \mathbf{Z}$ satisfying certain mathematical conditions [Vapnik, 1995], the inner product in a high-dimensional $\mathbf{z}$-space can be calculated via inner product kernel in $\mathbf{x}$-space:

$$(\mathbf{z} \cdot \mathbf{z}') = H(\mathbf{x}, \mathbf{x}') \tag{35}$$

where the vectors $\mathbf{z}$ and $\mathbf{z}'$ are the images of vectors $\mathbf{x}$ and $\mathbf{x}'$ in the input space. So instead of computing inner products in a high-dimensional $\mathbf{z}$-space one can use their kernel representation (35) in $\mathbf{x}$-space. In the dual form, the SVM decision function has the form:

$$D(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i H(\mathbf{x}_i, \mathbf{x}) \tag{36}$$

Notice that this representation is same as (30) except that the kernel H takes the place of the inner product in (30).

The kernel functions corresponding to different choices of basis functions can be analytically derived [Vapnik, 1995; Vapnik et al, 1996; Cherkassky and Mulier, 1998]. Below are a few common examples:

Polynomial expansion of degree q have inner product kernel:

$$H(\mathbf{x},\mathbf{x}') = \left[(\mathbf{x}\cdot\mathbf{x}') + 1\right]^q \tag{37}$$

Radial basis functions of the form:

$$f(x) = \boldsymbol{sign}\left(\sum_{i=1}^{n} \alpha_i\, \boldsymbol{exp}\left\{\frac{|\mathbf{x}-\mathbf{x}_i|^2}{\sigma^2}\right\}\right) \tag{38 }$$

where σ defines the width have the inner product kernel:

$$H(\mathbf{x},\mathbf{x}') = \boldsymbol{exp}\left\{-\frac{|\mathbf{x}-\mathbf{x}'|^2}{\sigma^2}\right\} \tag{39}$$

Note that the number of radial basis functions, the center parameters, which correspond to the support vectors, and the coefficients α_i are all automatically determined via the optimal hyperplane. All radial basis functions have the same width parameter which is specified a priori.

Next we review several SVM applications to pattern recognition.The first study (conducted by a team of researchers at Bell Labs) is a popular handwritten character recognition problem, where ten handwritten digits represent zipcodes [LeCun, et al., 1990]. The U.S. Postal data set contains 7,300 training patterns and 2,000 test patterns from real-life postal zipcodes. Following segmentation, each character is recorded as a grey-scale image with a resolution of 16x16 pixels, so the input vector has dimensionality 256. The support vector method with three different classes of approximating functions was applied: third-degree polynomials, radial basis functions, and neural network. The best generalization performance of the SVM model, i.e. the classification error observed on the test set is around 4% [Vapnik, 1995]. This generalization performance of SVM for this application does not depend on the type of approximating (or kernel) functions used by SVM. For comparison, the percentage error for humans classifying this data set is 2.5% [Bromley & Sackinger, 1991]. Note that SVM is a general purpose learning approach and it does rely on application-specific preprocessing. In fact, SVM would produce the same classification results following a random permutation of pixels in the original data.

Huang et al [1998] applied SVM to eye detection. They used 16x16 image vectors corrsponding to eye / non-eye images. All image data was preprocessed so that each pixel is normalized to a [-1, +1] range before training. The training data consisted of 186 eye images and 185 non-eye images. Independent test data was 100 images of each class. The best generalization performance of 4% (test error) was achieved with polynomial kernels of second degree.

Osuna et al [1997] applied SVM to face detection in natural images. That is, given an arbitrary image (i.e., scanned photo), determine whether it contains faces and locate them. The training data consisted of 19x19 images of faces and non-faces. Some generic preprocessing (illumination gradient correction, histogram equalization) was applied to this data prior to SVM training. The SVM model used second-order polynomial kernel. Following training, their SVM classifier is applied to 19x19 window subimages of an input image. The input image is presented at various scales, in order to achieve scale independent classification. Their test data used two test sets A and B. Set A contained 313 high-quality images containing exactly 1 face. Set B contained 23 'difficult' images containing multiple faces in each image. Note that even though the number of test images fairly small, they generate several million of pattern windows, i.e. overlapping subimages at different scales [Osuna et al, 1997]. Osuna et al [1997] report the following generalization performance of their SVM implementation:

for test set A face detection rate is 97.1% (with 4 false alarms);

for test set B face detection rate is 74.2% (with 20 false alarms).

These results are similar to generalization performance achieved by Sung and Poggio [1994] on the same image data. Sung and Poggio's approach relies on custom preprocessing of the input data to model the distribution of the face and non-face clusters in the feature space, followed by the neural network classifier.

Acknowledgement: this work was supported, in part, by NSF grant IRI--9618167 and by the IBM Partnership Award.

References

Berger, J.(1985) *Statistical Decision Theory and Bayesian Analysis*, Springer

Bishop, C.(1995) *Neural Networks for Pattern Recognition*, Oxford Univ. Press

Bromley, J., Sackinger, E.(1991) Neural network and k nearest neighbor classifiers, Technical Report 11359-910819-16TM, AT&T

Cherkassky, V., Mulier, F.(1998) *Learning From Data: Concepts, Theory and Methods*, Wiley (to appear)

Fisher, R.A.(1952) *Contributions to Mathematical Statistics*, Wiley, New York

Friedman, J.H.(1994) An overview of predictive learning and function approximation, in *From Statistics to Neural Networks*, Cherkassky, V., Friedman, J.H., and Wechsler, H. (Eds.), NATO ASI Series F, 136, Springer Verlag

Huang, J., Ii, D., Shao, X., and Wechsler, H.(1998) Eye Detection and Pose Estimation Using Support Vector Machines (SVM), *Face Recognition: From Theory to Applications*, Wechsler, H., Phillips, J., Bruce, V., Huang, T., and Fogelman Soulie, F. (Eds.), Springer Verlag (to appear)

Kolmogorov, A. N.(1965) Three approaches to the quantitative definitions of information, Problem of Information Transmission, 1, (1), 1-7

Le Cun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.J.(1990) Handwritten digit recognition with a back-propagation neural network, *Advances in Neural Information Processing Systems*, 2, 396-404

Osuna, E., Freund, R., Girosi, F., Training Support Vector Machines: an application to face detection(1997) in Proc. of CVPR'97, Puerto Rico

Rissanen, J.(1978) Modeling by shortest data description, Automatica,14, 465-471

Rissanen, J.(1989) *Stochastic Complexity and Statistical Inquiry*, World Scientific

Sung, K. and Poggio, T., Example-based learning for view-based human face detection(1994) AI memo 1521, MIT AI Lab

Vapnik, V., Golowich, S., Smola, A., Support vector method for function approximation, regression estimation, and signal processing(1996) *Advances in Neural Information Processing Systems*, 10

Vapnik, V. (1982) *Estimation of Dependencies Based on Empirical Data,* Springer Verlag, New York

Vapnik, V. (1995) *The Nature of Statistical Learning Theory*, Springer Verlag, New York

Wechsler, H. (1997) Personal communication