KERNEL MACHINES, NEURAL NETWORKS, AND GRAPHICAL MODELS

PAOLO FRASCONI

Dipartimento di Sistemi e Informatica – Università degli Studi di Firenze

ALESSANDRO SPERDUTI

Dipartimento di Matematica Pura ed Applicata – Università degli Studi di Padova

ANTONINA STARITA

Dipartimento di Informatica – Università degli Studi di Padova

1 Statistical and Probabilistic Learning

Learning is fundamental to intelligence as it allows to acquire knowledge from the environment by using data. Several paradigms and approaches are available for defining the learning problem. In this paper we mainly focus on methods based on *statistical* and *probabilistic* assumptions.

In the statistical setting, a learning algorithm is given a set of examples obtained by sampling from a fixed but unknown distribution. Most often, for the sake of developing simple models, examples are also assumed to be independently sampled, although it is not always reasonable to believe that this is actually the case. The purpose of learning is to use the available data to characterize the unknown distribution from which the examples were sampled. Several paradigms can be defined for this purpose.

In the case of *supervised learning*, training data consists of input-output pairs $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. The set $\mathcal{X}$ is called the input (or instance) space and can be any set. The set $\mathcal{Y}$ is called the output (or target) space. The two simplest forms of supervised learning are *binary classification*, where $\mathcal{Y} = \{-1, 1\}$, and *regression*, where $\mathcal{Y}$ is the set of real numbers. The case where $\mathcal{Y}$ is itself structured has received less attention until recently. Many pattern recognition and related prediction tasks can be formulated according to this paradigm. The fundamental underlying assumptions is that a *joint distribution* characterizes the relation between inputs and outputs, naturally allowing for noise in the examples and their labels. This would be impossible in the so-called *reduced learning model* where it is assumed that examples are sampled in the input space and labels are deterministically assigned by a fixed but unknown function. The general goal of supervised learning is to estimate the conditional density $p(y|x)$, although very often it is sufficient to find a function $f : \mathcal{X} \mapsto \mathcal{Y}$ that well approximates the probabilistic dependency between inputs and outputs (e.g. by making sure that a given loss function is minimized). The class of functions that is searched for a solution is called the *hypothesis space*.

Unsupervised learning aims at directly estimating the distribution that underlies the data. The goal in this case is to estimate $p(x)$. Unsupervised learning is typically useful to obtain interesting and compressed representations of the data. This may be achieved, for example, by identifying clusters, sub-manifolds or other regions of interest in high-dimensional spaces, or by determining the hidden structure of dependencies between causes and effects. Unsupervised learning algorithms can be sometimes used also for solving classification tasks. This basically involves modeling *class conditional densities*, i.e. the generative distributions $p(x|y)$ behind each class y. Note that such a generative direction can be seen as reversed with respect to the *discriminant* direction taken by the supervised learning approaches mentioned above.

In the case of *semi-supervised learning*, some examples are labeled with an output target and some are not. This setting is typically useful to take advantage of unlabeled data that may be available at lower cost than labeled data. When this is the case, semi-supervised learning takes the form of learning from labeled and unlabeled data. In other situations, the set of instance on which predictions are required might be known during the training phase, leading to a formulation called *transductive learning*.

2 State-of-the-art Methods

Statistical and probabilistic learning is a very active research area. Here we restrict our focus to some of the most popular methods.

2.1 Kernel Methods

A Mercer kernel is a positive semi-definite symmetric function that generalizes the notion of inner product to arbitrary domains—see e.g., [38] for details. Kernels are used in machine learning to measure the similarity between

two instances so that we can make predictions about future instances by taking into account their similarity to training examples. It can be shown that for any Mercer kernel K there exists a map $\Phi : \mathcal{X} \mapsto \ell^2$ from the instance space to the Hilbert space of square-summable sequences, such that $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$. This map is usually called the *feature* map. Intuitively, the similarity between two examples will increase with the number of features they have in common. Note that we do not need to explictly represent $\Phi(x)$ if we know how to compute the kernel. This nice property is generally referred to as the *kernel trick*.

2.2 Empirical Risk Minimization and Regularization

A simple and straightforward way of solving the supervised learning problem is based on empirical risk minimization (ERM). In this approach, each hypothesis is evaluated by observing its behavior on the available training data. Hence, an ERM learner will search the hypothesis space to find a solution h by minimizing the value of a real valued *loss function* on the training set, where the loss on a single example measures the cost incurred in predicting $h(x)$ when the target was in facts y. Although intuitively appealing, ERM is fundamentally flawed by its lack of uniqueness and stability. In other words, it generally attempts to solve a mathematically ill-posed problem. To convince, think about examples consisting of pairs of real numbers and suppose that the hypothesis space is the set of polynomials in x. The loss could be in this case the squared error between y and $h(x)$. There are clearly infinite solutions if we allow polynomials of degree higher than the number of examples. In addition, some of these solution may be very unstable in the sense that they may change significantly when examples are changed by small perturbations, or when a small fraction of examples is deleted from the training set [30]. Common sense in this case tells us that to avoid overfitting we should avoid this kind of solutions that need to oscillate wildly to perfectly fit the examples. We can penalize these unstable solutions if we have a way of measuring their complexity. Towards this goal, Vapnik [43] proposed the abstract idea of *structural risk minimization* by which an algorithm should search a minimizer of the empirical risk over a sequence of hypothesis spaces of increasing complexity.

When using kernel methods in supervised learning, the hypothesis space is the so-called reproducing kernel Hilbert space (RKHS) associated with K. The good news is that we now have a very simple measure for the *complexity* of an hypothesis h via its norm in the RKHS, $\|h\|_K$. This allows us to modify ERM into a so-called Tikhonov regularized problem by adding $\mu \|h\|_K$ to the average training loss. μ is a positive regularization constant that should be chosen according to the noise that is believed to affect the data. Popular algorithms in this framework include support vector machines (SVM) [43] and kernel ridge regression. The representer theorem shows that the solution to the regularized problem can be expressed as $f(x) = \sum_{i=1}^{m} c_i K(x, x_i)$. In particular, in the case of SVM classification, each term in the sum takes on the form $\alpha_i y_i K(x, x_i)$, i.e. the solution is a weighted average of data targets, where the weights are formed by multiplying a coefficient α_i that measures the importance of the i-th example and a term $K(x, x_i)$ that measures the similarity between the input portion of each example and the new data point x. Other supervised kernel methods (such as the kernelized voted perceptron [10]) output solution that can be expressed in this way.

2.3 Artificial Neural Networks

Artificial neural networks constitute a family of nonlinear data modeling tools that can be used both in supervised and unsupervised settings. The computation of a neural network is traditionally described in terms of a data-flow process performed by an interconnected set of computing units in a form of network. A neural unit computes a simple nonlinear function $\sigma()$ of the weighted sum, according to weights w_i, of its inputs x_i, $out = \sigma(\sum_i w_i x_i + \theta)$, where θ is a threshold value. Adopting different designs for the neural units, i.e. using a $\sigma(\cdot)$ with a "step" shape, or a non-linear squashing function like the sigmoidal-logistic function, or a linear function, we obtain different neural models. Specifically, the sigmoidal-logistic function has the property to be a smoothed differentiable threshold function, and thus amenable to gradient based learning procedures. The *architecture* of a neural network defines the topology of the connections among the units. In the context of supervised learning, the most renewed model is the well-know MLP (feedforward multi-layer perceptron) architecture: the units are connected by weighted links and they are organized in the form of layers. The input layer is simply the source of the input that projects onto one or more hidden layers of units. The last hidden layer projects onto the output layer. According to the definition of hypothesis space given above, a MLP realizes a mathematical parameterized hypothesis in the form $h(x) = g_{\mathcal{W}}(x)$, where $x \in \mathbb{R}^n$ is the input pattern in the form of fixed-size vector, $\mathcal{W}$ is a set of real-valued parameters (*the weights*) whose values determine the specific computed function, and $g_{\mathcal{W}} : \mathbb{R}^n \rightarrow \mathbb{R}^z$, were z is the number of output components. For instance, a two-layer feedforward neural network with a single output (i.e., $z = 1$) and m "hidden" units, computes a function of the type $h(u) = \tilde{\sigma}(\sum_{j=1}^{m} w_j^{out} \tilde{\sigma}(\sum_{q=1}^{n} w_{jq}^{hidden} x_q)) = \tilde{\sigma}(\boldsymbol{w}^{out} \cdot \tilde{\sigma}(W^{hidden} x))$ where $W^{hidden} \in \mathbb{R}^m \times \mathbb{R}^n$ is the weight matrix from the input to the hidden layer, $\boldsymbol{w}^{out} \in \mathbb{R}^m$ is the weight vector from the hidden layer to the output layer (recall that in general we have $z > 1$), $\tilde{\sigma}(\cdot)$ is a function that applies component-wise a sigmoidal nonlinear *activation function* $\sigma(u)$, e.g. $\sigma(u) = \frac{1}{1+\exp^{-u}}$, and $y = \tilde{\sigma}(W^{hidden} x) \in \mathbb{R}^m$ is the so called hidden activity

vector. In this case, the set of parameters $\mathcal{W}$ is constituted by both the elements of W^{hidden} and the elements of $\boldsymbol{w}^{out}$.

Hence, the hypothesis space that characterizes the neural model is the space of all functions that can be represented by assigning specific values to the parameters in $\mathcal{W}$, i.e. the weight values of the given neural network architecture. This allows the model to represent a rich space of non-linear functions, making neural networks a good choice for learning discrete and continuous functions whose general form is unknown in advance. In particular, approximation theory (Theorem by Cybenko, see [16]) formally supports the generality of this class of hypotheses: this class of functions is sufficient to approximate continuous and derivable functions with arbitrary small error, provided that a sufficient number of hidden units are available.

Note that, depending on the class of values produced by the network output units, discrete or continuous, the model can deal, respectively, with classification or regression tasks. Given a loss or error function which is differentiable with respect to the weights learning can be based on the minimization of the error function via gradient descent techniques. For multi-layer neural networks, an efficient implementation of gradient descent is obtained by *back-propagation* [35], the most popular among the supervised training algorithms for this class of networks, where the values of the errors are back propagated from the output neural units to the input units, changing the weights proportionally to the influence they had on the total error R_{emp}, i.e. at each iteration the weight values w_{ij} are updated according to the learning rule $w_{ij} = w_{ij} + \Delta w_{ij}$, where $\Delta w_{ij} = -\eta \frac{\partial R_{emp}}{\partial w_{ij}}$ and $\eta > 0$ is the gradient descent step size, or learning rate.

Despite of its implementation simplicity and usefulness, back-propagation has the classical disadvantages of any gradient descent method (local minima, dependency of the final hypothesis on the initial weights condition, choice of the "right" value for the gradient descent step, etc.). Another problem with learning is how to chose the "right" topology of the network. The problem of optimizing the architecture has been addressed by different approaches, such as the use of constructive learning algorithms, which grow the network topology during training, or pruning techniques (see e.g., [34]), which after training try to prune "useless" units or connections. Various techniques (such as the Cascade Correlation algorithm [7]) have been proposed to dynamically modify the network topology and its dimension as part of the learning algorithm. Constructive methods, also called growing methods, permit to start with a minimal network configuration and to add units and connections progressively, allowing automatic adaptation of the architecture to the computational task. Bayesian learning has also been defined for neural networks where the output is interpreted as a probability distribution [28].

Another relevant class of neural networks for supervised learning, i.e. radial basin function (RBF) neural networks [31], is defined by considering units with Gaussian activation functions. The use of stochastic activation functions has also been studied, leading to models such as Boltzmann machines [1].

In the context of unsupervised learning, winner-take-all networks are the most typical. These models are based on units $k - WTA_n(x_1, \ldots, x_n)$ that compute a mapping $k - WTA_n : \mathbb{R}^n \rightarrow \{0,1\}^n$, where the ith output returns 1 iff the number of input values greater than x_i is at most $k-1$. The most used unit is the one defined for $k = 1$, where the output of the unit indicates which of the n inputs has maximal value. Very popular models in this family are Self-Organizing Maps (SOMs) [21] and Adaptive Resonance Thory (ART) (e.g., [4]).

The aim of a SOM map $\mathcal{M}$ is to represent high-dimensional input patterns with prototype vectors w (weights of neurons) which can be visualized in a low dimensional lattice structure. The lattice structure defines the network topology, specifying for each neuron a set of neighbor neurons. The weights of the neurons are randomly initialized. Then, when an input vector is presented to the map $\mathcal{M}$, the neuron whose weight vector with the smallest Euclidean distance from the input vector (i.e. the winning neuron) is selected, together with the neurons in its neighbor, for training according to a Hebbian-based learning rule. The strength of learning is regulated by a function that both decreases with the distance (over the lattice topology) of a neuron from the winning neuron, and with the number of training iterations. In this way, the hope is to capture in the map the main topographic relationships among input patterns. In fact, SOMs are (basically) able to perform input space density approximation, independently of the order of presentation of the input patterns, even if they may over-represent areas of low pattern density and under-represent areas of high pattern density.

Adaptive Resonance Theory is a theory from which a family of networks able to learn arbitrary input patterns in a stable, fast, and self-organizing fashion has been derived. The basic architecture is constituted by two layers of neurons linked by inter-layers connections. The bottom layer receives the input patterns and activates prototypical winners-take-all neurons in the top layer. The winning neuron projects an expectation pattern back to the bottom layer so to compare it with the input pattern. A third component controls the mismatch between the input pattern and the expectation pattern, and if this is above a given threshold (vigilance), a new prototype (i.e., cluster) is generated into the top layer, and the input assigned to it. Both networks for dealing with discrete and continuous variables have been developed, as well as supervised versions, where an ART module representing the output patters is added and correlated with the original ART module.

Among the many other models, both deterministic and stochastic, that have been developed in the connectionist area (see e.g. [16]), it is worth to mention networks of spiking neurons (also called artificial *pulsed neural*

networks)[27], which are supposed to be more biologically plausible. In these networks, computational states are encoded as temporal differences between *spikes*, or *firing times* of neurons.

2.4 Probabilistic Graphical Models

The interest in belief networks (a.k.a. graphical models) lies at the intersection between several research areas including at least reasoning and uncertain knowledge representation, statistics, and machine learning [29, 20]. In their simplest formulation, these models are graphs with vertices that consist of random variables taking on values in a given propositional domain. Missing edges encode probabilistic independence relations among these variables. For example, in Markov networks, two variables A and B are independent if no path links vertices A and B, and they are *conditionally independent* given C if all paths linking A and B are intercepted by C. Bayesian network are a directed form of belief networks that allow to naturally model causation by linking causes to their effects by means of directed arcs.

Learning in graphical models usually takes two important and different forms. In *parameter* (or quantitative) learning, the edge set of the network is given and the problem is to estimate the parameters that characterize the distribution defined by the network. This is usually achieved by means of either maximum-likelihood or Bayesian statistics approaches, maybe using an expectation-maximization algorithm or gradient descent [3] in the presence of hidden or missing variables. In *structure* (or qualitative) learning, the aim is to search the graph space for an edge set that encode independence relations compatible with the observed data. For example, structure learning in Bayesian networks could be used to learn cause-effect relations [17].

Several popular models used in machine learning are special cases of probabilistic graphical models. We briefly mention some interesting cases in the following.

2.4.1 Naive Bayes

The Naive Bayes classifier can be seen as the simplest model for the class conditional densities. For simplicity, let us assume that the instance space is a set of tuples and denote an instance by $x = (x^1, \ldots, x^n)$. For each class y the model prescribes that attributes are conditionally independent given the class: $p(x|y) = \prod_{j=1}^{n} p(x^j|y)$. Classification is obtained by straightforward application of the Bayes rule $p(y|x) \propto p(y)p(x|y)$ where $p(y)$ is estimated e.g. by class relative frequencies. In many cases the conditional independence assumption of Naive Bayes will be violated, leading to a poor modeling of $p(x|y)$. The problem can be mitigated by adding dependencies between attributes in the form of additional edges in the Bayesian network representation. However, it is important to remark that using inaccurate class conditional densities does not necessarily lead to classification errors. In practice, Naive Bayes is known to work well for small data sets but since it does not converge to the optimal Bayes classifier, other discriminant methods (such as logistic regression or SVM) may be preferable when enough training examples are available.

2.4.2 Hidden Markov Models

Hidden Markov models (HMM) were originally introduced by researchers working in speech recognition and signal processing [32]. An HMM is a generative model of sequences based on the underlying assumption that temporal context is summarized into a finite set of states that are not observed in the data (hence hidden). States evolve in time based on transition probabilities. In this sense the model can be seen as a kind of probabilistic finite state automaton and in facts an HMM can also be interpreted as a description of a probabilistic regular grammar. Sequence elements are generated at each time position according to a probability distribution conditioned on the state. The model is Markovian because of its fundamental conditional independence assumption that future is conditionally independent of the past given the present. HMM are usually trained by variants of the Expectation-Maximization algorithm for maximum likelihood estimation of parameters with incomplete data (where incompleteness here is due to hidden states). Although EM and conditional independence are two central notions also for probabilistic graphical model, the fact that HMMs are a special case of belief network was not recognized until it was presented in a number of papers in the early 1990's (see e.g. [8, 39]). Besides speech recognition, HMMs have gained popularity in many application domains, like modeling biological sequences and natural language.

2.5 Statistical Relational Learning

Research in machine learning has historically evolved by following in parallel several mainstream approaches. In particular, the statistical, probabilistic and connectionist views of learning discussed so far have strong links to statistics but less strong links to some of the other mainstream views of learning that are based on symbolic representations. These have been primarily discussed in [6] and include rule learning, program induction, and relational learning. Recently, these views have begun to converge, although a full synthesis is challenging and likely to require many additional years of research. Statistical relational learning (sometimes also referred to by the even more ambitious term "probabilistic logic learning") is about machine learning exploiting at the same time rich data representations (e.g. logical and relational) and probability theory. In this context, a wealth of formalisms have been introduced, including probabilistic relational models (a relational extension of Bayesian networks), stochastic logic programs, logical Markov networks and Bayesian logic programs (see e.g. [12]). All these models aim at lifting the expressive power of belief networks that are propositional

in their original formulations. Probabilistic modeling, (especially when endowed with first-order logical expressiveness) is also appealing to solve one of the most crucial limitations of present supervised learning algorithms, i.e. the assumption that examples are independently sampled. Probabilistic inference, in facts, allows us to propagate evidence among different examples thus modeling various form of linkage that occur in most data sets and naturally achieve collective classification. Quoting a suggestive sentence of T. Dietterich at a recent Dagstuhl seminar, "i.i.d. in machine learning is dead, it's time to relationalize our data."

2.6 Dealing with Structured Domains

There are many interesting application tasks in Bioinformatics, Chemistry, and Natural Language Processing, just to name a few, where data can naturally be represented in a structured form such as sequences, trees, or graphs. Although good results have been obtained by the application of statistical learning techniques to "flat" representations of these structured objects, i.e. vectors of real numbers representing structural features extracted by a preprocessing stage, both computational and generalization concerns have motivated some researchers to develop new techniques to directly deal with structured information.

From a computational point of view, representing a structured object by a flat representation poses the dilemma between preserving universality of representation, which implies high space/time complexity and almost sure overfitting during training, and dropping structural details in the representations, with the potential risk, if there is lack of solid a priori knowledge, to incur in severe underfitting through loss of information. When there is no a priori knowledge, however, an approach which tries to preserve in a natural way as much structural information as possible in the representations , and develops suitable efficient procedures to process these representations, seems to be more sensible. Two different but highly interrelated streams of research follow this philosophy: Recurrent/Recursive Neural Networks (see for example, [22, 40, 9]) and Kernels for Structures (see [11] for a short survey).

Recurrent/Recursive Neural Networks are based on the following strategy: temporal/structural relationships are represented explicitly and concisely according to the current input structure, although with some limitations in the case of structural data (e.g. cyclic graphs); then an internal task-dependent and task-efficient representation is developed via learning, and concurrently used, in supervised learning, to learn the classifier/regressor of interest. This is obtained by jointly training an encoding function for the structural data, and an output function for classification and/or regression. The problem of variance in size of input structures is solved by weights sharing. Very interesting results from the computational point of view have already emerged for this approach, both concerning supervised and unsupervised learning (e.g. [13, 14]). There is still the need, however, to develop learning procedures which guarantee, at least in probability, the generalization error to be below a specified threshold.

Kernels for Structures, on the other hand, try to exploit the variety and success of kernel methods such as SVMs and use the kernel trick to avoid an explicit representation of the structural features into a vectorial form: since only comparisons among structures is actually required, string/structure matching procedures are directly defined in the structured input domain, without explicitly constructing the (often large) vectorial feature space. A difficulty of this approach is the a priori definition of the kernel so to fit the application domain: for many domains a structured kernel cannot preserve all structural information unless solving NP-hard problems (see [33]). There have been several approaches which address the problem of designing domain-specific structure kernels. Fisher kernels developed by [18] use the Fisher-score vectors of Markov-model parameters as their feature space. Convolution kernels for discrete structures were introduced in [15], where kernels are based in turn on smaller kernels which compare specific structure parts. At the same time, Watkins [44] independently proposed a kernel for strings based on comparing all (possibly non-contiguous) k-length subsequences for two input strings. The connection between Fisher kernels and other discrete kernels was highlighted by Saunders et. al [37], where it was shown that string-type kernels have a probabilistic interpretation and equivalent Fisher kernels for the resulting HMMs can be defined. There now exist several general frameworks for building kernels for discrete structures, most notably rational kernels [5] and probability product kernels [19]. Perhaps the most successful applications of structure kernels has been in the field of bioinformatics, where several structure kernels such as profile kernels [24], mismatch kernels [25] and local-alignment kernels [36] have been shown to achieve state-of-the-art performance on tasks such as protein-homology detection.

Kernel-based approaches for other types of structures (rather than individually structured training examples) have also been developed. These include diffusion kernels [23], for when the training examples themselves form part of a structure (e.g. web pages are often related by an ontology). Recently, proposals on generating structured outputs rather than a single label have also been presented and have received a great deal of interest (e.g. [2, 42, 41]).

3 Relationships to Other Areas

Although strictly speaking Machine Learning is a subset of Artificial Intelligence and Statistical Learning a subset of Machine Learning, there are several other areas of Artificial Intelligence in which statistical and probabilistic data driven algorithms have been investigated. In these cases, research mainly aims at solving specific problems where

statistical learning tools have rapidly been included among the preferred methods of choice.

One of these areas is Natural Language Processing (see [26]). A common characteristic of many NLP problems is the relational and structured nature of the representations that describe data and that are internally used by various algorithms, offering interesting applications for the structured learning methods outlined in Section 2.6. Tasks are numerous and diverse and include for example refinement of statistical parsers, tagging named entities, syntactic chunking, extraction of relations between entities.

A second important areas that has strong intersections with statistical and probabilistic learning is machine perception. We mentioned that Hidden Markov models were introduced in the 1970's within the speech recognition community. More in general, speech and vision have originated a large number of important pattern recognition tasks for which statistical learning is clearly a natural solution.

Machine learning methods have been applied for many years to several important problems related to molecular biology and chemistry and very often provide state-of-the-art solutions to prediction problems in these disciplines.

Since the beginning of the genomics era and the introduction of high-throughput technologies, biology has evolved into a massively data oriented discipline. Machine learning is well suited here because of the noisy nature of the domain and the intrinsic difficulty in devising models that are accurate and computationally tractable. Examples of prediction tasks that are more or less routinely solved with various machine learning algorithms (especially neural networks and kernel machines) include protein structure, protein function, gene expression analysis and inference of gene networks.

Machine learning research is also active in developing methods for Chemoinformatics, in particular for predicting the biological activity of small molecules. Early methods developed since the 1960's in the QSAR community are based on linear regression applied to numerical descriptors of molecules, obtained from physico-chemical properties, from topological information (regarding the molecule as a graph), and from 3D atom coordinates. Recent research has shown that several families of graph kernels can be successfully applied to large-scale screening tasks (e.g. the small molecule data sets made available by the National Cancer Institute).

REFERENCES

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[2] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *ICML03*, pages 3–10, 2003.

[3] J.L.A. Binder, D.L.A. Koller, S.L.A. Russell, and K.L.A. Kanazawa. Adaptive Probabilistic Networks with Hidden Variables. *Machine Learning*, 29(2):213–244, 1997.

[4] G. A. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[5] C. Cortes, P. Haffner, and M. Mohri. Rational kernels. In S. Becker, S. Thrun, and A. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, 2003.

[6] F. Esposito, A. Giordana and L. Saitta. Machine Learning and Data Mining. *Intelligenza Artificiale*, 3(1) (this issue).

[7] 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. San Mateo, CA: Morgan Kaufmann, 1990.

[8] P. Frasconi and Y. Bengio. An EM approach to grammatical inference. In *Proc. 12th International Conference on Pattern Recognition (ICPR94)*, volume 2, pages 289–294, 1994.

[9] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5):768–786, 1998.

[10] Y. Freund and R.E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[11] T. Gartner. A survay of kernels for structured data. *ACM SIGKDD Explorations Newsletter*, 5(1):49–58, 2003.

[12] L. Getoor and B. Taskar, editors. *An Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, 2006.

[13] B. Hammer. *Learning with Recurrent Neural Networks*, volume 254 of *Springer Lecture Notes in Control and Information Sciences*. Springer-Verlag, 2000.

[14] B. Hammer, A. Micheli, A. Sperduti, and M. Strickert. Recursive self-organizing network models. *Neural Networks*, 17(8-9):1061–1085, 2004.

[15] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, July 1999.

[16] S. Haykin. *Neural Networks, A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1999.

[17] D. Heckerman, D. Geiger, and D. M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[18] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1,2):95–114, 2000.

[19] T. Jebara, R. Kondor, and A. Howard. Probability product kernels. *Journal of Machine Learning Research*, 5:819–844, 2004.

[20] M.I. Jordan. *Learning in graphical models*. MIT Press Cambridge, MA, USA, 1999.

[21] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995.

[22] J.F. Kolen and S.C. Kremer, editors. *A Field Guide to Dynamical Recurrent Networks*. IEEE Press, Inc., New York, 2001.

[23] R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. 2002.

[24] R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C.S. Leslie. Profile-based string kernels for remote homology detection and motif extraction. In *3rd International IEEE Computer Society Computational Systems Bioinformatics Conference (CSB 2004)*, pages 152–160, 2004.

[25] C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. Stafford Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–76, 2004.

[26] L. Lesmo and M.T. Pazienza. Natural Language Processing. *Intelligenza Artiþciale*, 3(1) (this issue).

[27] W. Maass and C.M. Bishop. *Pulsed neural networks*. MIT Press, 1998.

[28] D.J.C. MacKay. Bayesian interpolation. *Neural Comput.*, 4(3):415–447, 1992.

[29] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[30] T. Poggio, R. Rifkin, S. Mukherjee, and P. Niyogi. General conditions for predictivity in learning theory. *Nature*, 428(6981):419–422, Mar 2004.

[31] M.J.D. Powell. Radial basis functions for multivariable interpolation: A review. In J. C. Mason and M. G.Cox, editors, *Proceedings of the IMA Conference on Algorithms for the Approximation of Functions and Data*, pages 143–167. Oxford Science Publications, 1985.

[32] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[33] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. pages 65–74. ECML/PKDD'03 workshop proceedings, September 2003.

[34] R. Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[35] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. MIT Press, 1986.

[36] H. Saigo, J.P. Vert, T. Akutsu, and N. Ueda. Protein homology detection using string alignment kernels. *Bioinformatics*, 20:1682–1689, 2004.

[37] C. Saunders, J. Shawe-Taylor, and A. Vinokourov. String Kernels, Fisher Kernels and Finite State Automata. In S. Becker, S. Thrun, and A. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, 2003.

[38] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[39] P. Smyth, D. Heckerman, and M.I. Jordan. Probabilistic Independence Networks for Hidden Markov Probability Models. *Neural Computation*, 9:227–269.

[40] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[41] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. In *Neural Information Processing Systems*, 2003.

[42] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In Carla E. Brodley, editor, *ICML Õ04: Twenty-Þrst international conference on Machine learning*, New York, NY, USA, 2004. ACM Press.

[43] V.N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[44] C. Watkins. Dynamic alignment kernels. Technical Report CSD-TR-98-11, Royal Holloway, University of London, January 1999.