ФИЗИКА ЭЛЕМЕНТАРНЫХ ЧАСТИЦ И АТОМНОГО ЯДРА
2007. Т. 38. ВЫП. 2

MULTIVARIATE ANALYSIS METHODS IN PHYSICS

M. Wolter
The Henryk Niewodniczański Institute of Nuclear Physics,
Polish Academy of Sciences, Krakow, Poland

In this article a review of multivariate methods based on statistical training is given. Several popular multivariate methods useful in high-energy physics analysis are discussed. Selected examples from current research in particle physics are discussed, both from the on-line trigger selection and from the off-line analysis. Also statistical training methods, not yet applied in particle physics, are presented and some new applications are suggested.

В данной работе представлен обзор многомерных методов, основанных на статистическом «облучении». Обсуждается несколько популярных методов, используемых при анализе данных экспериментов по физике высоких энергий. Рассматриваются несколько примеров текущих исследований по физике частиц, касающихся как on-line селекции на уровне триггера, так и off-line анализа. Также представлены новые методы статистического «облучения», не использовавшиеся до сих пор в физике частиц, и предложено несколько новых приложений.

PACS: 02.50.Sk.

INTRODUCTION

High-energy physicists started to use multivariate techniques and have had considerable success in using them. It seems likely that as analysis becomes more challenging these methods will be used more routinely. Typical areas of application are background suppression (classification) and parameter estimation (regression), where a physical quantity is extracted from a set of directly measured observables. The reason to apply statistical training multivariate methods is, in most cases, simply the lack of knowledge about the mathematical dependence of the quantity of interest on the relevant measured variables. Either there is no mathematical model at all and an exhaustive search is the only possibility of finding the correct dependence, or the known models are insufficient and statistical training provides a better description of data.

Each of the multivariate methods is usually well characterized mathematically, but they seem to be less different than it might appear. Indeed, it seems that many of them are simply different algorithms to approximate the same mathematical object. Moreover, the problems they solve, appear to be relatively few in number, while many new methods are invented every year. A typical list of problems they address is:

• signal-to-background discrimination,

• variable selection (e.g., finding variables which give the maximum signal/background discrimination),

• dimensionality reduction of the feature space and simplification (by reducing the number of variables),

• finding regions of interest in data,

• measuring parameters (regression).

These algorithms can be also classified according to the character of the training process into two main classes:

• supervised training — where a set of training events with correct outputs is given,

• unsupervised training — where no outputs are given and the algorithm has to find them by itself (for example, a classification of input data into few classes of similar events).

In this article, an overview of multivariate methods is given together with the selected examples of applications. The differences and similarities between the methods are discussed.

1. LINEAR CLASSIFIERS

A linear classifier is a classifier that uses a linear function of its inputs to base its decision on. That is, if the input feature vector to the classifier is a real vector $\mathbf{x}$, then the estimated output score (or probability) is:

$$y = f(\mathbf{w} \cdot \mathbf{x}) = f\left(\sum_j w_j x_j\right), \tag{1}$$

where $\mathbf{w}$ is a real vector of weights and f is a function that converts the dot product of the two vectors into the desired output. Often f is a simple function that maps all values above a certain threshold to «yes» and all other values to «no».

For a two-class classification problem, one can visualize the operation of a linear classifier as splitting of a high-dimensional input space with a hyperplane: all points on one side of the hyperplane are classified as «yes», while the others are classified as «no».

A linear classifier is often used in situations where the speed of classification is an issue, since it is often the fastest classifier, especially when $\mathbf{x}$ is sparse. Also, linear classifiers often work very well when the number of dimensions in $\mathbf{x}$ is large, as in document classification, where each element in $\mathbf{x}$ is typically the number of counts of a word in a document.

All of the linear classifier algorithms can be converted into nonlinear algorithms operating on a different input space $\varphi(\mathbf{x})$, using the kernel trick (see Subsec. 4.2), by which the observables $\mathbf{x}$ are effectively mapped into a higher dimensional nonlinear space. Linear classification in this nonlinear space is then equivalent to nonlinear classification in the original space.

1.1. Fisher Linear Discriminant. The design of a good classifier becomes rapidly more difficult as the dimensionality of the input space increases. One way of dealing with this problem is data preprocessing in which the data dimensionality is reduced. Fisher discriminants [1] aim to achieve an optimal linear dimensionality reduction from N dimensions to one. A simple cut placed on the output works as a discriminating hyperplane.

Suppose two classes of observations have means $\boldsymbol{\mu}_{y=0}$, $\boldsymbol{\mu}_{y=1}$ and covariances $\sigma_{y=0}$, $\sigma_{y=1}$, where $y = 0, 1$ denotes the signal and background, respectively. Then the linear combination of features $\mathbf{w} \cdot \mathbf{x}$ will have means $\mathbf{w} \cdot \boldsymbol{\mu}_{y=i}$ and variances $\mathbf{w}^T \sigma_{y=i} \mathbf{w}$ for $i = 1, 2$. Fisher defined the separation between these two distributions to be the ratio of the variance between the classes to the variance within the classes:

$$S = \frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}} = \frac{(\mathbf{w} \cdot \boldsymbol{\mu}_{y=1} - \mathbf{w} \cdot \boldsymbol{\mu}_{y=0})^2}{\mathbf{w}^T \sigma_{y=1} \mathbf{w} + \mathbf{w}^T \sigma_{y=0} \mathbf{w}} = \frac{(\mathbf{w} \cdot (\boldsymbol{\mu}_{y=1} - \boldsymbol{\mu}_{y=0}))^2}{\mathbf{w}^T (\sigma_{y=0} + \sigma_{y=1}) \mathbf{w}}. \quad (2)$$

This measure is, in some sense, a measure of the signal-to-noise ratio for the class labelling. It can be shown that the maximum separation occurs when

$$\mathbf{w} = \frac{(\boldsymbol{\mu}_{y=1} - \boldsymbol{\mu}_{y=0})}{(\sigma_{y=0} + \sigma_{y=1})}. \quad (3)$$

The projected data can subsequently be used to construct a discriminant, by choosing a threshold f_0 so that a new point $\mathbf{x}$ is classified as belonging to the first class if $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} > f_0$ and to the second class otherwise.

Fisher discriminants are successfully used in HEP for a long time for the signal and background separation, one of the recent examples is the use of this technique by BELLE Collaboration [2]. The use of Fisher discriminants, as a linear methods, requires that data are separable by a hyperplane. If a nonlinear separation is needed, nonlinear techniques are required. As already mentioned, Fisher discriminants can be converted to the nonlinear kernel Fisher discriminant [3] using a kernel trick.

1.2. Principal Component Analysis. Principal Component Analysis (PCA) [4, 5] is a technique that can be used to simplify a dataset. It is a linear transformation that chooses, by a rotation of the coordinate system, new coordinates such that the greatest variance by any projection of the data set comes to lie on the first axis (then called the first principal component); the second greatest variance, on the second axis, and so on. PCA can be used for reducing

dimensionality in a dataset while retaining those characteristics of the dataset that contribute most to its variance by eliminating the later principal components. PCA can be regarded as a form of unsupervised training, since it relies entirely on the input data.

Principal component analysis is based on the statistical representation of a random variable. Suppose we have a vector population $\mathbf{x}_i$, where the mean of that population is denoted by

$$\boldsymbol{\mu}_x = \frac{\sum_i \mathbf{x}_i}{n} \tag{4}$$

and the covariance matrix of the same data set is

$$V_x = \frac{\sum_i (\mathbf{x}_i - \boldsymbol{\mu}_x) \cdot (\mathbf{x}_i - \boldsymbol{\mu}_x)^T}{n}. \tag{5}$$

The variance of a component indicates the spread of the component values around its mean value. If two components x_i and x_j of the data are uncorrelated, their covariance is zero $v_{ij} = 0$. The covariance matrix is, by definition, always symmetric. We assume that data are normalized, i.e., $v_{11} = v_{22} = \ldots = 1$.

From a symmetric matrix such as the covariance matrix, one can calculate an orthogonal basis by finding its eigenvalues and eigenvectors. The eigenvectors $\mathbf{e}_i$ and the corresponding eigenvalues λ_i are the solutions of the equation:

$$V_x \mathbf{e}_i = \lambda_i \mathbf{e}_i, \qquad i = 1, \ldots, M. \tag{6}$$

For simplicity we assume that λ_i are distinct. These values can be found by finding the solutions of the characteristic equation

$$|V_x - \boldsymbol{\lambda} I| = 0, \tag{7}$$

where I is the identity matrix. By ordering the eigenvectors in the order of descending eigenvalues (largest first), one can create an ordered orthogonal basis with the first eigenvector having the direction of the largest variance of the data.

Suppose that the first l eigenvalues have significantly larger values than the remaining $n - l$ eigenvalues. This tells us that the data can be represented to a relatively high accuracy by projection onto the first l eigenvectors (Fig. 1). However PCA is limited by being a linear technique and therefore not being able to capture more complex nonlinear correlations. As a dimensionality reduction technique it also fails, when the data are better separable along the eigenvector corresponding to the lower eigenvalue.

Despite the limitations mentioned above, PCA is widely used in pattern recognition to reduce the data dimensionality. One of the examples is the human

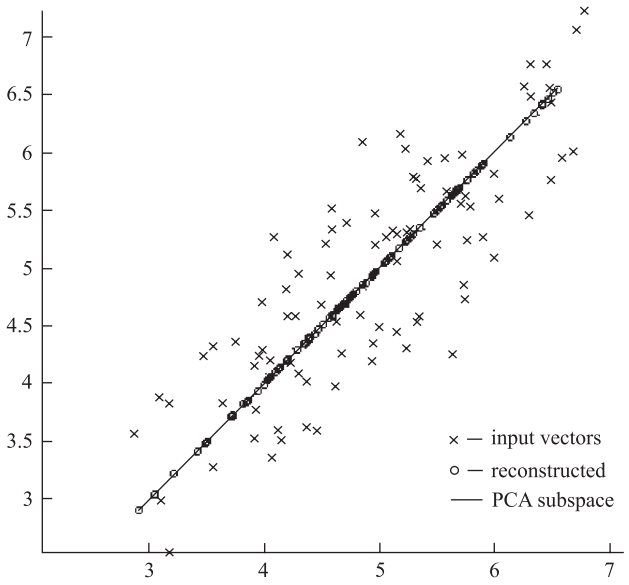

Fig. 1. Example of linear principal component analysis in two dimensions

face recognition procedure, in which each face is reduced to a smaller set of linear combinations of eigenvectors called eigenfaces [6]. PCA can also be used in the analysis of data from high energy physics experiments to reduce the data dimensionality and extract the useful information. Principal Component Analysis can be, as well as other linear techniques, converted to the nonlinear methods using a kernel trick or using a Neural Network as a nonlinear transformation (see Subsec. 3.5).

1.3. Independent Component Analysis. Independent Component Analysis (ICA) is a technique that recovers a set of independent non-Gaussian source signals from a set of measured signals. For a detailed review on ICA see, for example, [7].

In ICA it is assumed that each measured signal is a linear combination of independent input signals, and that there is an equal number of measured signals and input signals. The separation is performed by making the signals as statistically independent as possible, i.e. ICA is performing a blind source separation. The components x_i of the observed random vector $\mathbf{x}$ are generated as a sum of the independent components s_k: $\mathbf{x} = A \cdot \mathbf{s}$ weighted by the mixing weights matrix A. The aim is to find the demixing matrix A^{-1} and to recover the original sources by multiplying the observed signals by this matrix $\mathbf{s} = A^{-1} \cdot \mathbf{x}$.

A first step in the most of the ICA algorithms is to whiten (sphere) the data. In this operation any correlations in the data are removed, i.e., the signals are forced to be uncorrelated. We seek a linear transformation V such that when $\mathbf{y} = V \cdot \mathbf{x}$, than $\sum_i (\mathbf{y}_i \cdot \mathbf{y}_i^T) = I$. This is accomplished by setting $V = C^{-1/2}$, where $C = \sum_i (\mathbf{x}_i \cdot \mathbf{x}_i^T)$ is the correlation matrix of the data. After whitening, the separated signals can be found just by an orthogonal transformation (rotation) of the whitened signals $\mathbf{y}$. The rotation is performed to maximize the non-Gaussianity of the signals, since a linear mixture of independent random variables is necessarily more Gaussian than the original variables (see Fig. 2). This implies that in ICA at most one input signal can be Gaussian.

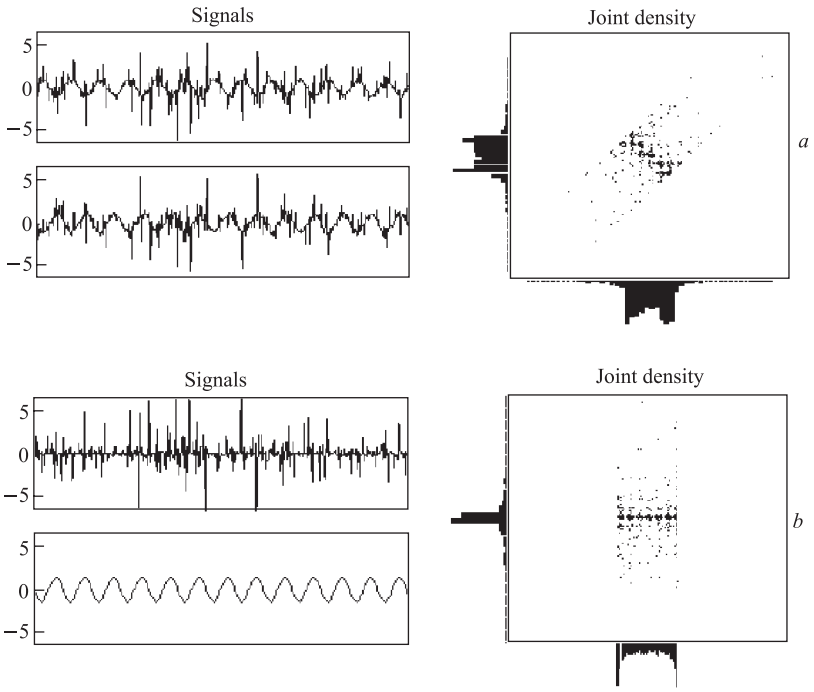

Fig. 2. Two input signals (*a*) and the signals separated by ICA (*b*). The two independent signals appeared to be a sinusoid and impulsive noise

The most efficient to date algorithm performing ICA is the FastICA [8]. It is using kurtosis to measure the deviation of the signals from the Gaussian distribution. Kurtosis, the fourth standardized moment, is defined as μ_4/σ^4,

where μ_4 is the fourth moment about the mean and σ is the standard deviation. Higher kurtosis means more of the variance to be due to the extreme deviations.

Beside the linear ICA there is also a nonlinear algorithm developed [9], in which the transformation $\mathbf{x} = \Phi(\mathbf{s})$ can be an arbitrary, nonlinear function. The main difficulty of the nonlinear ICA is that it tends to provide the nonunique solutions.

ICA has many practical applications: filtering a single source of sound from other sources (a cocktail-party problem, requires as many microphones as many sources we want to separate), separation of brain activity from artifacts in magnetoencephalography (MEG) recordings [10], in telecommunication for signal separation [11] and many others.

ICA is used in astrophysics to separate various signal sources [12, 13], also it was used to decompose the Fermilab booster beam monitor data into independent source signals [14]. It seems that in particle physics ICA could be a usefull tool for separating different processes contributing to signals. It could also be used to move to the new set of variables, which are less dependent than the original ones, therefore easier to handle.

2. PROBABILITY DENSITY ESTIMATION

Probability Density Estimation (PDE) estimates $P(\mathbf{x}|S)$ and $P(\mathbf{x}|B)$ for the signal and background classes. The idea, introduced by Parzen in the 1960s [43], is very simple: one approximates an unknown probability density as a sum of probability densities of known functional forms, usually with a few adjustable parameters. The Parzen method:

$$p(\mathbf{x}) = \sum_i \phi(\mathbf{x}, \mathbf{z}_i) \tag{8}$$

estimates the unknown $P(\mathbf{x})$ by placing a density function $\phi(\mathbf{x}, \mathbf{z}_i)$, called a kernel, at each point $\mathbf{z}_i, i = 1, \ldots, n$ of a sample of points, one sample from the signal class and one from the background class. There is considerable freedom in the choice of kernel function. The main mathematical requirement is that $\phi(\mathbf{x}, \mathbf{z}_i) \to \delta(\mathbf{x} - \mathbf{z}_i)$ as the sample sizes grow indefinitely. Note, if n is the sample size of each class, this method requires evaluating the kernel function n times for each class separately.

A Gaussian kernel is a typical application for high-energy physics, since nearly all variables analyzed have been Gaussian smeared by detector resolution or other effects. A Gaussian kernel PDE method estimates the probability at point $\mathbf{x}$ by the sum of Gaussians centered at the Monte Carlo generated points:

$$p(\mathbf{x}) = \frac{1}{N\sqrt{|V|}(2\pi h)^{k/2}} \sum_{i=1}^{N} \exp\left[-\frac{\mathbf{d}_i^T V^{-1} \mathbf{d}_i}{2h^2}\right], \tag{9}$$

where $\mathbf{d}_i = \mathbf{x} - \mathbf{z}_i$; V is a covariance matrix; k is the dimensionality of the parameter space and h is an additional scaling factor.

The matrix V might be determined from the covariance matrix of the overall sample (the static kernel method), and parameter h set to $h = N^{-1/(k+4)}$ [44] or V could be calculated from a subsample of points which are spatially closest to $\mathbf{x}$ (Gaussian expansion method [45]).

The kernel methods have a straightforward interpretation and are conceptually simple. Also in most cases they handle signal/background discrimination problems equally well as neural network, for some classes of problems they appear to be even superior [45]. Also they avoid the problem of falling into local minima, which is typical for neural networks. The drawback is that they need a lot of CPU for evaluating the kernel function and need to store in memory all the vectors from the training sample.

Kernel methods were successfully used by $D\emptyset$ experiment [46]. The algorithm, together with other methods implemented in the QUAERO package, was used to search for standard model WW, ZZ, and $t\bar{t}$ production, and to search for the above objects produced through a new hypothetical heavy resonance (see Fig. 3).

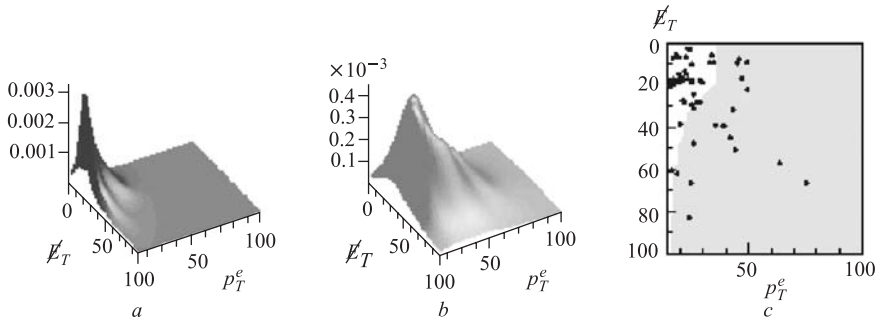

Fig. 3. The background density (*a*), signal density (*b*), and selected region (shaded) (*c*) determined by QUAERO for the standard model process $WW \to e\mu E_{T(\mathrm{miss})}$. The dots in the plot *c* represent events observed in the data collected by $D\emptyset$ experiment [46]

2.1. PDE-RS Method. Recently a new, faster probability estimation algorithm based on hypercubical kernels was developed [47]. The PDE-RS method is based on sampling the signal and background densities in a multidimensional phase space. Since the kernel functions have finite and short range, usually only few entries from the training sample are needed to calculate the probability density function. The event counting is done using a fast range-searching algorithm. Its speed makes it possible to use very large data samples required in analysis, where a high reduction of background is necessary, or to scan a large number of observables for those which give the best separation of signal. This technique

has been successfully used by ZEUS experiment at HERA for identification of isolated charged leptons [48] and also by the τ recognition package at ATLAS experiment [49].

3. NEURAL NETWORK

The development of Artificial Neural Networks (ANN) was inspired by the research on the central nervous system and the neurons (also their axons, dendrites and synapses) which constitute their information processing elements. Currently, the term ANN tends to refer mostly to mathematical models, which do not need to be an emulation of the central nervous system and have little in common with the brain research. The approach stimulated by biological research has been abandoned for an approach based on statistics, mathematics and optimization theory.

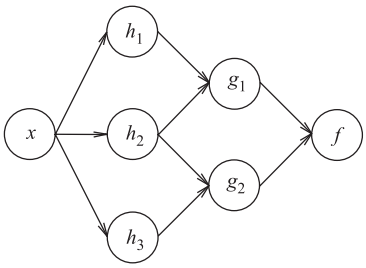

Fig. 4. Artificial Neural Network dependence graph

Neural network is a nonlinear function $f : \mathbf{x} \to \mathbf{y}$ which can model relationships between inputs and outputs. It is trained on examples to find the best fit between inputs and outputs. A detailed description of neural network algorithms can be found in [15, 16], while in this article only a brief overview is given.

The word «network» arises from the fact, that function f is a composition of other functions g_i which can be themselves compositions of other functions h_i, etc. This structure can be represented as a network (see Fig. 4), in which each function is represented by a node and arrows are showing dependences between functions. In most applications the functions are the nonlinear weighted sums, i.e.:

$$f(\mathbf{x}) = K\left(\sum_i \left(\omega_i g_i(\mathbf{x}')\right)\right), \tag{10}$$

where $\mathbf{x}'$ might be again a composition of functions acting on input vector $\mathbf{x}$ and K is a predefined function called the activation function. Commonly used activation functions are:

$$\begin{aligned} &f(x) = wx + w_0 && \text{linear function;} \\ &f(x) = \tanh\left(wx + w_0\right) && \text{hyperbolic tangent function;} \\ &f(x) = \frac{1}{1 + \exp\left(wx + w_0\right)} && \text{logistic function.} \end{aligned} \tag{11}$$

3.1. Feedforward Networks. Networks such as shown in Fig. 4 are called feedforward, since the information is feeded from input to output without any loops. In contrast, the networks with loops are called recurrent.

The most interesting feature of neural networks is their ability of training, which means finding an optimal function f using a set of observations. This requires defining the cost function $C : F \to \Re$ (F is the space of all functions f), which for the optimal solution f^* fulfills the relation $C(f^*) \leqslant C(f) \forall f \in F$. Training algorithms search through the solution space in order to find a function that has the smallest possible cost.

Since training is based on a set of observations the cost function itself must depend on observations, otherwise it would not be modelling anything related to the data. In practical cases only a limited number of observations is available, the cost function is itself only an approximation. Therefore the cost is minimized over a sample of the observations rather than over the true data distribution. In many applications the cost function is based on the χ^2 minimization:

$$C(f) = \frac{\sum_{i=0}^{N} \left(f(\mathbf{x}_i) - \mathbf{y}_i\right)^2}{N}, \tag{12}$$

where $\mathbf{y}_i$ is the desired output.

Beside being a classifier, the neural network can be used also for regression, i.e., for estimating real-valued functions.

3.1.1. The Backpropagation Algorithm. In supervised training we are given a set of observation pairs $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ is a known input vector and $\mathbf{y}$ is the expected output. This could be taken from the Monte Carlo simulation, where the true output is known (for example, a type of particle in case of particle identification). The aim of the training procedure is to train the network, i.e., to find a function $f(\mathbf{x})$ which has the minimal cost function.

In the process of training, patterns are presented to the network which generates an output. The output is compared with the desired output from the training sample and the cost function is calculated. Then the weights in nodes should be adjusted to decrease the value of the cost function. An algorithm is needed, which propagates the errors back in the neural network from the output units to the hidden units. This problem was unresolved for 30 years before the error backpropagation algorithm popularized a way to train hidden units, leading to a new wave of neural network research and applications. The algorithm was first proposed by Paul Werbos in 1974 [17]. However, it was not used until it was rediscovered in 1986 by Rumelhart, Hinton, and Williams [18].

At the output layer, the output vector is compared to the expected output. If the difference is not zero, the error is calculated from the delta rule and is propagated back through the network. Under the so-called delta rule, the change

in weight of the link between nodes i and j is given as

$$dw_{ij} = rx_i(t_j - y_j), \tag{13}$$

where r is the learning rate (an arbitrary, small parameter); t_j is the target output; y_j is the actual output at unit j, and x_i is the actual output of the node i of the preceding layer.

The delta rule changes the weight vector in a way that minimizes the error, the difference between the target output and the actual output. It can be shown mathematically that the delta rule provides a very efficient way to modify the initial weight vector toward the optimal one (the one that corresponds to minimum error) and that a network using the delta rule can learn associations whenever the inputs are linearly independent [18]. Such networks can learn arbitrary associations by using differentiable activation functions.

3.1.2. Supervised Training and Overtraining. In the beginning we should recall our experience from the ordinary function fitting. If there are too many free parameters in the function, then the function tends to follow the experimental points, but would not fit to the other, statistically independent set of points. The function doesn't generalize well in unseen examples but simply trains a set of training points. This effect is called overtraining.

One of the methods to check, whether overtraining occurs is to divide data into training and validation sets and to perform training on the training set only. The cost function for both training and validation sets should be calculated periodically during the training. When the cost for the validation sample becomes greater than for the training sample, the training should be stopped [19]. The remedy for overtraining is either to stop training early or to simplify the network by removing part of the nodes from the hidden layers.

For small data sets cross-validation [20] might be superior to the split-sample procedure described above. In k-fold cross-validation the data set is divided into k subsets of (approximately) equal size. The network is trained k times, each time leaving out one of the subsets from training, but using only the omitted subset to compute whether the error criterion is fulfilled.

Other method of dealing with overtraining is, for example, jittering [21], in which an artificial noise is added deliberately to the inputs during training. Training with jitter is a form of smoothing. Other techniques are based on regularization or a Bayesian approach with some penalty for overtraining (see [15]).

3.2. Applications in Data Analysis. Neural networks have been used in many experiments for separating signal from background. A good example could be the search and later measurements of the top quark by two experiments, $D\emptyset$ and CDF working at the Tevatron proton–antiproton collider at Fermilab. In the Run-I analysis, when the top quark was discovered, the $D\emptyset$ experiment started to use the multivariate methods: probability density estimation and neural

network. Both the cross-section measurement in the all-jets channel [22] and the mass measurement in the lepton+jets channel [23] used feedforward neural networks. The results were superior to the ones obtained by cut methods, since the neural network allowed to fully exploit the correlations that exist among several variables providing a discriminating boundary between signal and background in a multidimensional space. This can yield discrimination close to the theoretical maximum.

In the Run-II, both experiments, $D\emptyset$ and CDF, used the multivariate techniques increasing their top-quark sensitivity. The CDF analysis includes a measurement of $t\bar{t}$ in the lepton + jets channel using a neural net to distinguish signal and background events [24] (Fig. 5 shows the NN output). $D\emptyset$ experiment presented a search for electroweak production of single top quarks in the s channel and t channel using neural networks for signal and background separation [25].

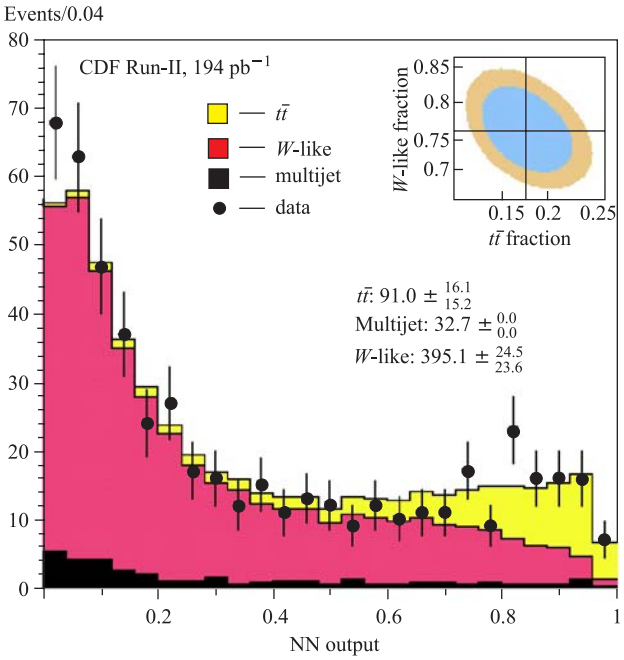

Fig. 5. Distribution of CDF experiment NN output in the $W^+ \geqslant 3$ jets sample, compared with the result of the fit [24]

One of the applications of neural networks is the reconstruction of physical quantities. In an example described in [26], Monte Carlo events are generated with various values of the quantity to be measured (in this case it is Higgs boson

mass) and used to determine the probability distribution for each value of the quantity. The probability density functions are fitted using a neural network and the final value of the measured quantity is obtained as a product of the single event distributions. In this method no knowledge about the functional dependence of the measured quantities on the observables is needed.

3.3. Data Selection at the Trigger Level. When implemented in hardware, neural networks can take advantage of their inherent parallelism and run orders of magnitude faster than software simulations. They are therefore suitable to be implemented for event selection on the trigger level using the dedicated hardware, either commercial NNW chips and neurocomputers, VME boards, etc., as well as systems built by HEP groups for on-line trigger applications.

The CDF experiment at the Tevatron had several calorimeter neural network hardware triggers [27]. Their trigger, built out of three ETANN VLSI neural network chips, receive analog signals from 50 trigger towers. One of the cards identifies isolated photon showers in the central calorimeter and the other looks for isolated electron showers in the plug calorimeter. Differences in the weighted sums of central and surrounding towers are calculated by the chips and then thresholds cuts on the outputs provide the triggers. The third chip implements a trained network to identify b jets in the central calorimeter.

Also the CPLEAR experiment at CERN was one of the first implementing a hardware neural network trigger [28]. The hardware networks were counting and locating tracks in the tracking detector within 75 ns using NN implemented in 16 cards with commercial ECL chips and doing on-line event selection within 40 ms, using information from the tracking network.

Later the neural network hardware triggers were implemented also in other experiments, a good example could be a DIRAC experiment at CERN [29]. In 210 ns the hardware neural network selects events with two particles having low relative momentum using the fast plastic scintillator information. The algorithm appeared to be more efficient and faster than the traditional methods.

The H1 experiment at DESY successfully uses since 1996 two independent fast pattern recognition systems operating as second level triggers (L2). The decision time is 20 μs. One of the two is the neural network trigger [30] running an array of VME-boards with CNAPS 1064 chips (20 MHz, 128 Mcps) by Adaptive Solutions.

3.4. Hopfield Network for Track Reconstruction. A Hopfield net is a form of recurrent artificial neural network invented by John Hopfield [31]. Hopfield nets serve as content-addressable memory systems with binary threshold units.

In the Hopfield model, each neuron is in general interacting with every other neuron. All interactions are symmetric, and the state of each neuron, expressed by its activation a_i, can only be either active «1» or inactive «0». The interaction is simulated by updating the state of a neuron according to the activations of all other neurons. The update rule in the Hopfield model sets the new activation a_i

of a neuron to

$$a_i = \begin{cases} 1 & \text{if } \sum_j w_{ij} s_j > \theta_i, \\ 0 & \text{otherwise,} \end{cases} \tag{14}$$

where w_{ij} is the connection weight from unit j to unit i; s_j is the state of unit j, and θ_i is the threshold of unit i. The connections in a Hopfield net have two restrictions on them:

$$\begin{aligned} w_{ii} &= 0 \quad \forall_i \quad \text{no unit has a connection with itself,} \\ w_{ij} &= w_{ji} \quad \forall_{i,j} \quad \text{all connections are symmetric.} \end{aligned} \tag{15}$$

It can be shown that such interactions characterize a system with an energy function:

$$E = -\sum_j \sum_{i,i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i. \tag{16}$$

This value is called the «energy» because the definition ensures that if units are randomly chosen to update their activations, the network will converge to states which are local minima in the energy function.

In the training process the patterns the network should memorize are presented to it. When the energy of states which the network should remember are local minima (trained network) and an unknown pattern is presented to the network, it will converge to the closest minimum, i.e., the most similar known pattern.

Hopfield networks were successfully used for pattern recognition in tracking algorithms assigning track segments to the tracks. A detailed description of these tracking algorithms can be found in [32], here only a brief description is given. An adaptation of Hopfield networks to track finding has been developed by Denby [33], and Peterson [34] and later modified by Ohlsson, Peterson, and Yuille (the elastic arms algorithm [35]). The basic idea of their method is to associate each possible connection between two hits with a neuron. Activation of such a neuron means that both hits are part of the same track. It is then essential to define an energy function such that in the global energy minimum only neurons corresponding to valid connections will be active. Interaction is only meaningful with neurons that have one hit in common.

It is remarkable that the Denby–Peterson method works without actual knowledge of a track model, favors series of hits that can be connected by a line as straight as possible, but also allows small bending angles from one segment to the next. Thus also curved tracks can be found, provided that a sufficient number of intermediate measurements exists which split the track into a large number of almost collinear segments. The Denby–Peterson algorithm is, in particular, indifferent about the global shape of the track — a circle and a wavy track with the same local bending angles but alternating directions are of equal value.

The Denby–Peterson method was first explored on track coordinates measured by the ALEPH experiment TPC chamber [36]. The algorithm found tracks in hadronic Z^0 decays rather accurately, which may be at least partially attributed to the 3D nature of the hits measured in the TPC and also the clean event structure and the low occupancy. Later it was applied to the tracks in the forward tracker of the H1 detector at HERA [37], where it was found that using Hopfield network one should be careful in cases of nearby parallel tracks. The elastic arms algorithm was also applied to tracks from DELPHI experiment TPC chamber [35], characterized by the moderate hit density and clean event structure and also later to the simulated events from the barrel part of ATLAS TRT detector [38]. Here much more dense events with 2D measurements have been targeted and the Hough transform was used for initialization. The efficiency found for fast tracks completely contained in the barrel TRT was practically identical to the one of the Hough transform itself, indicating that the elastic arms part did not find any new tracks that had not been properly covered by the initialization.

3.5. Nonlinear Principal Component Analysis. Multilayer neural networks can themselves be used to perform nonlinear dimensionality reduction overcoming some of the limitations of linear principal component analysis. Consider first a multilayer network having n input, n output units and one hidden layer with m nodes, where $m < n$. The targets used to train the network are simply the input vectors themselves, so the network is attempting to map each input vector onto itself. Such a network forms an auto-associative mapping. Since the hidden layer has a number of units smaller than the number of input units, a perfect reconstruction of all input vectors in general is not possible.

If the hidden units have linear activation functions, then it can be shown that the error function has a global unique minimum and that at this minimum the network performs a projection onto the m-dimensional subspace which is spanned by the first m principal components of the data [39]. This result is not surprising, since both principal component analysis and neural network are using linear dimensionality reduction and are minimizing the same sum of squares error function.

The situation is different if additional hidden layers are permitted and they are nonlinear activation functions. The network can be considered as two mappings F_1 and F_2, where the first mapping projects the original n-dimensional space into $m < n$ dimensions and F_2 projects again m dimensions back to the n-dimensional space. The mapping is essentially arbitrary, since additional layers of nonlinear units are present.

A network effectively performs a nonlinear principal component analysis. Also, the dimensionality of the subspace must be specified in advance of training, so that in practice it may be necessary to train and compare several networks having different values of m, which connected with nonlinear minimization techniques might be computationally expensive.

3.6. Self-Organizing Map. The self-organizing map algorithm is an unsupervised technique which can be used for model independent exploration of data by finding cluster patterns in data. The idea of SOM was first introduced and developed by Kohonen in the 1980s [40, 41] (hence also called Kohonen maps). The algorithm maps multidimensional feature space onto, usually, a two-dimensional space with a lattice of nodes. Events with similar features will be placed close to each other (clustered) in the output space. This method allows one to identify and separate groups of events with similar features.

One begins with n randomly directed vectors, usually modeled as neural network nodes whose weights are the components of the vector $\boldsymbol{\omega}_i$ and whose inputs are the components of the input vector $\mathbf{x}$. Then a distance measure between the vectors (either the Euclidean distance $|\boldsymbol{\omega}_i - \mathbf{x}|$ or the vector product $\boldsymbol{\omega}_i \cdot \mathbf{x}$) is defined. In the next step for each input vector $\mathbf{x}$ the vector $\boldsymbol{\omega}_i$ closest to it is found (the so-called «winning» vector). The winning vector $\boldsymbol{\omega}_i$ is adjusted to be closer to $\mathbf{x}$. Other vectors are adjusted as well, but by lesser amounts, depending on their distance from the winning vector. This procedure is repeated for each vector $\mathbf{x}$ and is continued until the vectors $\boldsymbol{\omega}_i$ are sufficiently stable, that is, their directions change by less than a specified amount per cycle.

Self-organizing maps are used in astrophysics as an automatic classifier (see, for example, [42], where Kohonen networks are used for classification of singly periodic astronomical lightcurves). In high-energy physics analysis this technique might also be useful for automated track, cluster, or event classification.

4. SUPPORT VECTOR MACHINE

In the early 1960s the linear support vector method was developed to construct separating hyperplanes for pattern recognition problems [50, 51]. It took 30 years that the method was generalized for constructing nonlinear separating functions [52, 53] and for estimating real-valued functions (regression) [54]. At that moment it became a general purpose algorithm performing data classification and regression which can compete with neural networks and probability density estimators. Typical applications of SVMs include text categorization, character recognition, bioinformatics and face detection. The main idea of the SVM approach is to build a separating hyperplane which maximizes the margin. The position of the hyperplane is defined by the subset of all training vectors called support vectors. The extension into nonlinear SVM is based on the «kernel trick»: input vectors are mapped into a high-dimensional feature space in which data can be separated by a linear procedure using the optimal separating hyperplane. Computations in the feature space are avoided by using kernel functions.

4.1. Linear Support Vector Machine. A detailed description of SVM formalism can be found, for example, in [55], here only a brief introduction is given. Consider a simple two-class classifier with oriented hyperplanes. If the training data is linearly separable, then a set of $(\mathbf{w}, b)$ pairs can be found such that the following constraints are satisfied:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geqslant 0 \quad \forall_i, \tag{17}$$

where $\mathbf{x}_i$ are the input vectors; y_i are the desired outputs ($y_i = \pm 1$), and $(\mathbf{w}, b)$ define a hyperplane. The decision function of the classifier is $f(x_i) = \operatorname{sign}(\mathbf{x}_i \cdot \mathbf{w} + b)$, which is $+1$ for all points on one side of the hyperplane and -1 for the points on the other side.

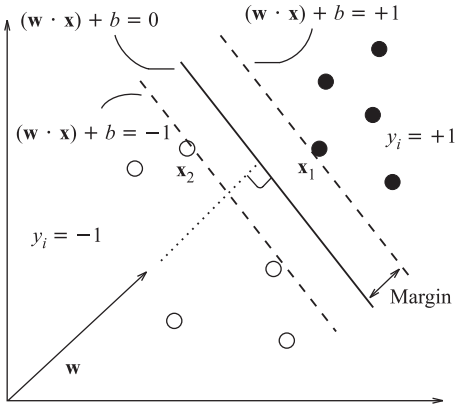

Fig. 6. Hyperplane classifier in two dimensions. Points $\mathbf{x}_1$, $\mathbf{x}_2$ define the margin, i.e., they are the support vectors

Intuitively, the classifier with the largest margin will give better generalization. The margin for this linear classifier is just $2/|\mathbf{w}|$. Hence to maximize the margin, one needs to minimize the $|\mathbf{w}|$ with the constraints in Eq. (17). At this point it would be beneficial to consider the significance of different input vectors $\mathbf{x}_i$. Those training data points laying on the margins are the data that contribute to defining the decision boundary (see Fig. 6). These data points are called the support vectors (SV). If the other data are removed and the classifier is retrained on the remaining data, the training will result in the same decision boundary.

To solve this constrained quadratic optimization problem, we first reformulate it in terms of a Lagrangian:

$$\mathcal{L}(\mathbf{w}, b, \alpha_i) = \left(\frac{1}{2}\right) |\mathbf{w}|^2 - \sum_i \alpha_i \left(y_i \left((\mathbf{x}_i \cdot \mathbf{w}) + b\right) - 1\right), \tag{18}$$

where $\alpha_i \geqslant 0$ and the condition from Eq. (17) must be fulfilled. Lagrangian $\mathcal{L}$ should be minimized with respect to $\mathbf{w}$ and b and maximized with respect to α_i. The solution has an expansion in terms of a subset of input vectors for which $\alpha_i \neq 0$ (these are the support vectors):

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{19}$$

since at extremum $\partial\mathcal{L}/\partial b = 0$ and $\partial\mathcal{L}/\partial\mathbf{w} = 0$. The optimization problem becomes one of finding the α_i which maximize

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{20}$$

Both the optimization problem and the final decision function depend only on dot products between input vectors, which is crucial for the generalization to the nonlinear case.

4.1.1. Nonseparable Data. The above algorithm can be extended to nonseparable data. The correct classification constraints in Eq. (17) are modified by adding a slack variable ξ_i to it ($\xi_i = 0$ if the vector is properly classified, otherwise ξ_i is a distance to the decision hyperplane)

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geqslant 0, \quad \xi_i \geqslant 0 \ \forall_i. \tag{21}$$

This will allow some points to be misclassified. The training algorithm will need to minimize the cost function shown in Eq. (22), i.e., a trade-off between maximum margin and classification error

$$W = \left(\frac{1}{2}\right)|\mathbf{w}|^2 + C\sum_i \xi_i. \tag{22}$$

The selection of C parameter defines how much a misclassification increases the cost function.

4.2. Nonlinear Support Vector Machine. The formulation of SVM presented above can be further extended to build a nonlinear SVM which can classify nonlinearly separable data. Consider a mapping Φ which maps the training data from $\Re^n$ to some higher dimensional space $\mathcal{H}$. In this high-dimensional space, the data can be linearly separable, hence the linear SVM formulation above can be applied to these data (see Fig. 7).

In the SVM formulation, data appear only in the form of dot products $\mathbf{x}_i \cdot \mathbf{x}_j$ (see Eq. (20)). The dot product $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ appears in the high dimension feature space. It can be replaced by a kernel function

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j). \tag{23}$$

By computing the dot product directly using a kernel function, one avoids the mapping $\Phi(\mathbf{x})$. This is desirable because $\Phi(\mathbf{x})$ can be tricky or impossible to compute. Using a kernel function, one does not need to know explicitly what the mapping is. Most frequently used kernel functions are:

$$\begin{aligned}
&K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d && \text{(polynomial of degree } d\text{)}, \\
&K(\mathbf{x}, \mathbf{y}) = \exp\left(-(1/2)\left|\mathbf{x} \cdot \mathbf{y}\right|^2 / 2\sigma^2\right) && \text{(Gaussian Radial Basis Function)}, \\
&K(\mathbf{x}, \mathbf{y}) = \tanh\left(\mathbf{x} \cdot \mathbf{y} - \Theta\right) && \text{(sigmoid)}.
\end{aligned} \tag{24}$$

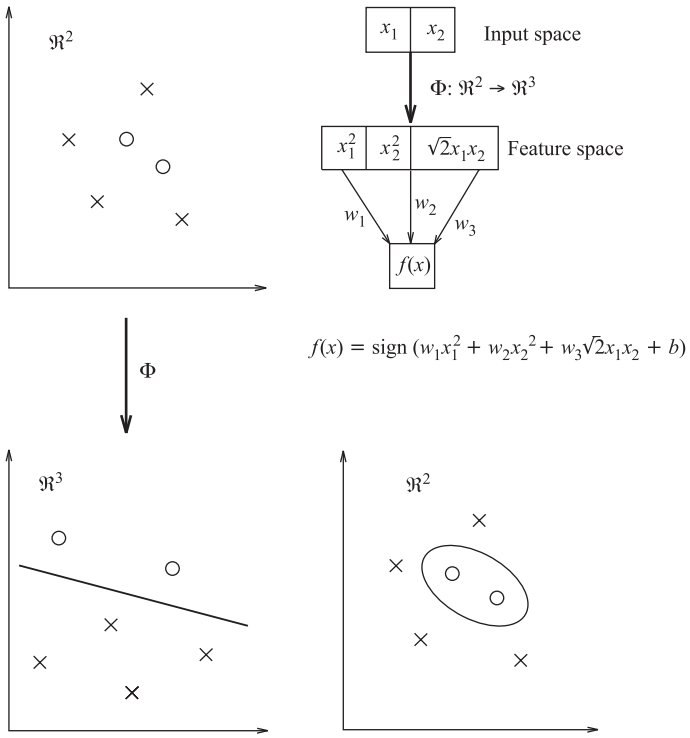

Fig. 7. Example of data separable by an elliptic curve in $\Re^2$, but linearly separable in the feature space $\Re^3$. The mapping to Φ transforms $(x_1, x_2) \to (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

A question arises, whether there is any constraint on the type of kernel function suitable for this task. It was shown [54] that a function must fulfill Mercer's condition to be suitable to be used as a kernel:

$$\int K(\mathbf{x}, \mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geqslant 0 \tag{25}$$

for any function g such that $\int g(\mathbf{x})^2 d\mathbf{x}$ is finite.

To extend the methodology described for linear case to nonlinear problems, one substitutes $\mathbf{x}_i \cdot \mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j)$ in Eq. (20). Due to Mercer's conditions on the kernel, the corresponding optimization problem is a well-defined convex quadratic programming problem which means there is a global minimum. This is an advantage of SVMs compared to neural networks, which may find only one of the local minima.

4.3. Regression SVM. A version of a SVM which can perform regression (called SVR) was proposed in 1997 by Vapnik, Steven Golowich, and Alex Smola [56]. The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin and are properly classified. Analogously, the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data that is close (within a threshold ϵ) to the model prediction:

$$|y - f(x)|_\epsilon := \max\{0, |y - f(x) - \epsilon|\}, \tag{26}$$

$$W = \left(\frac{1}{2}\right)|\mathbf{w}|^2 + \frac{C}{m}\sum_{i=1}^{m}|y_i - f(x_i)|_\epsilon . \tag{27}$$

The ϵ-insensitive loss function is more robust to small changes in data and in model and less sensitive to outliers while compared to the least squares loss function used by neural network (Fig. 8).

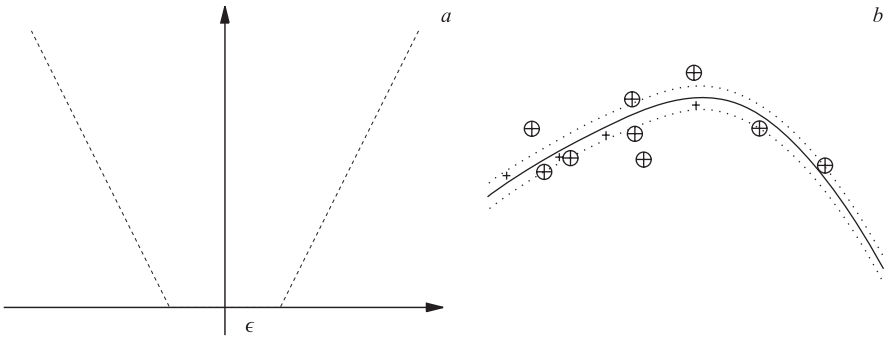

Fig. 8. *a*) ϵ-insensitive $|y - f(x)|_\epsilon$ function. *b*) The fit is sensitive only to points outside the $\pm\epsilon$ bounds

4.4. Comparison of SVM and Neural Networks. For Support Vector Machines, in contrast to neural networks, the capacity is independent of dimensionality of the data, thus avoiding curse of dimensionality. The algorithm is statistically well motivated, it can get bounds on the error, can use the theory of structural risk minimization (theory which characterizes generalization abilities of training machines). Finding the weights is a quadratic programming problem guaranteed to find a minimum of the error surface. Thus the algorithm is efficient and SVMs generate near optimal classification and obtain good generalization performance due to high dimensionality of the feature space.

It has also very few free parameters, in case of classification these are the type of the kernel function, kernel parameters (frequently one parameter only, like in case of Gaussian kernel) and the C parameter. Therefore it is in general possible to perform a grid search to identify an optimal set of parameters. In contrast, in case of neural network the entire network architecture has to be optimized.

On the other hand, the training of SVM is definitely slower due to computationally intensive solution of minimization problem especially for large amounts of training data. SVM generates complex solutions (frequently more than 60% of training points are used as support vectors) especially for large amounts of training data.

4.5. Applications of SVM. Support Vector Machine algorithms are widely used for classification problems, however up to now they were rarely used in particle physics. In 1999, there was an attempt to use SVM on simulated LEP data for tagging events of the type $e^+e^- \to c\overline{c}$ and the identification of muons produced in multihadronic e^+e^- annihilation events [57]. The authors compared neural networks and SVM methods and found that they both give similar results. Later SVM was proposed to be used in the top quark analysis at CDF experiment at Fermilab [58], but the actual analysis was never performed. However, since SVM appeared to be useful in solving many classification problems outside particle physics, it seems to be worthwhile to use them in HEP as well.

CONCLUSIONS

PCA, ICA, and SOM are focused on feature extraction and can be used for finding better sets of variables or some regions of interest. With the exception of these three methods, each of the methods presented above effects signal-to-background discrimination by trying to minimize the probability of misclassification. Consequently, although in practice each method minimizes a different empirical risk function, all such functions are equivalent to minimizing an approximation to the function [59]:

$$\epsilon(D) = C(S) \int\limits_{D(\mathbf{x})<0} p(\mathbf{x}|S)P(S)d\mathbf{x} + C(B) \int\limits_{D(\mathbf{x})\geqslant 0} p(\mathbf{x}|B)P(B)d\mathbf{x} \qquad (28)$$

with respect to the function $D(\mathbf{x})$, which is such that $D(\mathbf{x}) \geqslant 0$ leads to the assignment of event to signal S and $D(\mathbf{x}) < 0$ to the background B. Quantities $p(\mathbf{x}|S)$ and $p(\mathbf{x}|B)$ are estimated probability densities and $P(S)$ and $P(B)$ are the signal and background prior probabilities. $C(S)$ and $C(B)$ are the cost parameters, they quantify the cost of misclassification of events.

Minimization of $\epsilon(D)$ leads to a set of inequalities:

$$\begin{aligned} \frac{p(\mathbf{x}|S)P(S)}{p(\mathbf{x}|B)P(B)} &\geqslant \frac{C(B)}{C(S)} \text{ when } D(\mathbf{x}) \geqslant 0, \\ \frac{p(\mathbf{x}|S)P(S)}{p(\mathbf{x}|B)P(B)} &< \frac{C(B)}{C(S)} \text{ when } D(\mathbf{x}) < 0, \end{aligned} \tag{29}$$

where the ratio

$$\frac{p(\mathbf{x}|S)P(S)}{p(\mathbf{x}|B)P(B)} = \frac{p(S|\mathbf{x})}{p(B|\mathbf{x})} \tag{30}$$

is called the Bayes Discriminant Function. The classification rule it determines is called the Bayes rule [15]. The conditions from Eq. (29) hold when we chose:

$$D(\mathbf{x}) = \frac{p(S|\mathbf{x})}{p(B|\mathbf{x})} - \frac{C(B)}{C(S)}. \tag{31}$$

The various multivariate methods described in this article are different ways of approximating $D(\mathbf{x})$. Probability density estimation methods calculate $p(\mathbf{x}|S)$ and $p(\mathbf{x}|B)$, artificial neural network finds a function of D: $D(\mathbf{x})/(1 + D(\mathbf{x}))$. Also the Fisher discriminants can be described in these terms: they can be constructed as an estimate of $D(\mathbf{x})$ in which each density is approximated by a Gaussian, whose covariance matrix is the quadrature sum of those of the signal and background classes. The support vector machine method can be regarded as an algorithm to make the distributions Gaussian by projecting them into a space of sufficiently high dimension wherein the Fisher criterion can be applied to good effect.

There is much debate about which method is superior. Experience suggests that none is superior in every circumstance. For a given problem a reasonable definition of «best method» is that which yields the most accurate classification or estimation of the physical quantity for a given computational budget. Unfortunately, finding the best method can sometimes be a computationally demanding task. Fortunately, for the problems addressed in the field of particle physics the nonlinear methods presented above give similar performance.

Rather than engage in abstract debate, it is more useful to try a few methods and see how they perform. It should be kept in mind that many of these methods approximate the same mathematical object. Therefore, to the degree that the methods are properly applied and sufficiently nonlinear, they must give about the same or very similar results.

Acknowledgements. I would like to thank all the collaborators from the ATLAS group from IFJ PAN Institute in Krakow, in particular Prof. M. Turala and Prof. E. Richter-Was, for their encouragement, guidance and fruitful discussions. I am also grateful the Tufts University group led by Prof. K. Śliwa for their

generous support which enabled me visiting Fermilab and CERN and working for both ATLAS and CDF experiments.

This work was partially supported by Polish Government grant 620/E-77/SPB/CERN/P-03/DZ 110/2003-2005.

REFERENCES

1. *Fisher Ronald A.* The Use of Multiple Measurements in Taxonomic Problems // Ann. of Eugenics. 1936. V. 7. P. 179–188.
2. *Abe K. et al. (Belle Collab.).* Moments of the Photon Energy Spectrum from $B \to X/s$ Gamma Decays Measured by Belle. hep-ex/0508005.
3. *Mika S. et al.* Fisher Discriminant Analysis with Kernels // IEEE Conf. on Neural Networks for Signal Processing IX. 1999.
4. *Karhunen K.* Über lineare Methoden in der Wahrscheinlichkeitsrechnung // Am. Acad. Sci., Fennicade. Ser. A, I. 1947. V. 37. P. 3–79.
5. *Loeve M.* Probability Theory. Van Nostrand, 1955.
6. *Kirby M., Sirovich L.* Application of the Karhunen–Loeve Procedure for the Characterization of Human Faces // IEEE Trans. on Pattern Analysis and Machine Intelligence. 1990. V. 12 (1). P. 103–108.
7. *Hyvärinen A.* Survey on Independent Component Analysis // Neural Comp. Surveys. 1999. V. 2. P. 94–128; see also an ICA web page: http://www.cs.helsinki.fi/u/ahyvarin/whatisica.shtml
8. *Hyvärinen A.* Fast and Robust Fixed-Point Algorithms for Independent Component Analysis // IEEE Trans. on Neural Networks. 1999. V. 10(3). P. 626–634; see also FastICA web page: http://www.cis.hut.fi/projects/ica/fastica/
9. *Jutten C., Karhunen J.* Advances in Blind Source Separation (BSS) and Independent Component Analysis (ICA) for Nonlinear Mixtures // Intern. J. Neural Syst. 2004. V. 14, No. 5. P. 267–292; see also: http://www.cis.hut.fi/projects/ica/nonlinearica/
10. *Vigário R. et al.* Independent Component Analysis for Identification of Artifacts in Magnetoencephalographic Recordings // Adv. Neural Inform. Proc. Syst. 1998. V. 10. P. 229–235.
11. *Ristaniemi T., Joutsensalo J.* On the Performance of Blind Source Separation in CDMA Downlink // Proc. of Intern. Workshop on Independent Component Analysis and Signal Separation (ICA'99), Aussois, France, 1999. P. 437–441.
12. *Lu H. et al.* Ensemble Learning Independent Component Analysis of Normal Galaxy Spectra. astro-ph/0510246.
13. *Maino D. et al.* All-Sky Astrophysical Component Separation with Fast Independent Component Analysis (FastICA). astro-ph/0108362.
14. *Huang X. B. et al.* Application of Independent Component Analysis to Fermilab Booster // Phys. Rev. ST Accel. Beams. 2005. V. 8. P. 064001.
15. *Bishop C. M.* Neural Networks for Pattern Recognition. Oxford: Oxford Univ. Press, 1995.
16. *Zell A.* Simulation neuronaler Netze. Munich: R. Oldenbourg Verlag, 2000.
17. *Werbos P. J.* Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences. Ph.D. Thesis. Boston: Harvard Univ., 1974.

18. *Rumelhart D. E., Hinton G. E., Williams R. J.* Learning Internal Representations by Error Propagation // Computational Models of Cognition and Perception. Cambridge: MIT Press, 1986. V. 1, Ch. 8. P. 319–362.

19. *Sarle W. S.* Stopped Training and Other Remedies for Overfitting // Proc. of the 27th Symp. on the Interface of Comp. Science and Statistics. 1995. P. 352–360.

20. *Goutte C.* Note on Free Lunches and Cross-Validation // Neural Computation. 1997. V. 9. 1211–1215.

21. *Holmström L., Koistinen P.* Using Additive Noise in Back-Propagation Training // IEEE Trans. on Neural Networks. 1992. V. 3. P. 24–38.

22. *Abbott B. et al. (D0 Collab.).* Measurement of the Top Quark Pair Production Cross Section in the All-Jets Decay Channel // Phys. Rev. Lett. 1999. V. 83. P. 1908; hep-ex/9901023.

23. *Abachi S. et al. (D0 Collab.).* Direct Measurement of the Top Quark Mass // Phys. Rev. Lett. 1997. V. 79. P. 1197; hep-ex/9703008.

24. *Acosta D. et al. (CDF Collab.).* Measurement of the Cross Section for t anti-t Production in p Anti-p Collisions Using the Kinematics of Lepton + Jets Events // Phys. Rev. D. 2005. V. 72. P. 052003; hep-ex/0504053.

25. *Abazov V. M. et al. (D0 Collab.).* Search for Single Top Quark Production in p Anti-p Collisions at $s^{**}(1/2) = 1.96$ TeV // Phys. Lett. B. 2005. V. 622. P. 265; hep-ex/0505063.

26. *Wolter M.* Measurement of Physical Quantities in the Bayesian Framework Using Neural Networks. Prepared for Conf. on Advanced Statistical Techniques in Particle Physics, Durham, England, March 18–22, 2002.

27. *Denby H. et al.* Performance of the CDF Neural Network Electron Isolation Trigger // Nucl. Instr. Meth. A. 1995. V. 356. P. 485.

28. *Leimgruber F. R. et al.* Hardware Realization of a Fast Neural Network Algorithm for Real Time Tracking in HEP Experiments // Ibid. V. 365. P. 198.

29. *Kokkas P. et al.* The Neural Network First Level Trigger for the DIRAC Experiment // Nucl. Instr. Meth. A. 2001. V. 471. P. 358.

30. *Kohne J. K. et al.* Realization of a Second Level Neural Network Trigger for the H1 Experiment at HERA // Nucl. Instr. Meth. A. 1997. V. 389. P. 128.

31. *Hopfield J. J.* Neural Networks and Physical Systems with Emergent Collective Computational Abilities // Proc. of Nat. Acad. of Sci. 1982. V. 79, No. 8. P. 2554–2558.

32. *Mankel R.* Pattern Recognition and Event Reconstruction in Particle Physics Experiments // Rep. Prog. Phys. 2004. V. 67. P. 553; physics/0402039.

33. *Denby H.* Neural Networks and Cellular Automata in Experimental High-Energy Physics // Comp. Phys. Commun. 1988. V. 49. P. 429.

34. *Peterson C.* Track Finding with Neural Networks // Nucl. Instr. Meth. A. 1989. V. 279. P. 537.

35. *Ohlsson M., Peterson C., Yuille A. L.* Track Finding with Deformable Templates: The Elastic Arms Approach // Comp. Phys. Commun. 1992. V. 71. P. 77.

36. *Stimpfl-Abele G., Garrido L.* Fast Track Finding with Neural Nets // Comp. Phys. Commun. 1991. V. 64. P. 46.

37. *Bui D. L., Greenshaw T. J., Schmidt G.* A Combination of an Elastic Net and a Hopfield Net to Solve the Segment Linking Problem in the Forward Tracker of the H1 Detector at HERA // Nucl. Instr. Meth. A. 1997. V. 389. P. 184.

38. *Lindstrom M.* Track Reconstruction in the ATLAS Detector Using Elastic Arms // Nucl. Instr. Meth. A. 1995. V. 357. P. 129.

39. *Bourlard H., Kamp Y.* Auto-Association by Multilayer Perceptrons and Singular Value Decomposition // Biol. Cybernetics. 1988. V. 59. P. 291–294.

40. *Kohonen T.* Self-organized Formation of Topologically Correct Feature Maps // Biol. Cybernetics. 1982. V. 43. P. 59–69.

41. *Kohonen T.* Self-Organizing Maps // Springer Series in Inform. Sci. Berlin; Heidelberg; N. Y.: Springer, 1995; 1997; 2001. 3rd extended ed. V. 30. 501 p.

42. *Brett D. R., West R. G., Wheatley P. J.* The Automated Classification of Astronomical Lightcurves Using Kohonen Self-Organizing Maps. astro-ph/0408118.

43. *Parzen E.* Estimation of a Probability Density Function and Its Mode // Ann. Math. Statistics. 1962. V. 33. P. 1065–1076.

44. *Knuteson B., Miettinen H., Holmstrom L.* alphaPDE: A New Multivariate Technique for Parameter Estimation // Comp. Phys. Commun. 2002. V. 145. P. 351; physics/0108002.

45. *Towers S.* Kernel Probability Density Estimation Methods. Prepared for Conf. on Advanced Statistical Techniques in Particle Physics, Durham, England, March 18–22, 2002.

46. *Abazov V. M. et al. (D0 Collab.).* Search for New Physics Using QUAERO: A general Interface to D0 Event Data // Phys. Rev. Lett. 2001. V. 87. P. 231801; hep-ex/0106039.

47. *Carli T., Koblitz B.* A Multivariate Discrimination Technique Based on Range-Searching // Nucl. Instr. Meth. A. 2003. V. 501. P. 576; hep-ex/0211019.

48. *Chekanov S. et al. (ZEUS Collab.).* Search for Lepton-Flavor Violation at HERA // Eur. Phys. J. C. 2005. V. 44. P. 463; hep-ex/0501070.

49. *Janyst L., Richter-Was E.* Hadronic Tau Identification with Track Based Approach: Optimization with Multivariate Method. ATL-COM-PHYS-2005-028. Geneva: CERN, 2005.

50. *Vapnik V., Lerner A.* Pattern Recognition Using Generalized Portrait Method // Automation and Remote Control. 1963. V. 24.

51. *Vapnik V., Chervonenkis A.* A Note on One Class of Perceptrons // Automation and Remote Control. 1964. V. 25.

52. *Boser B. E., Guyon I. M., Vapnik V. N.* A Training Algorithm for Optimal Margin Classifiers // Proc. of the 5th Annual ACM Workshop on Comp. Learning Theory. ACM Press, 1992. P. 144–152.

53. *Cortes C., Vapnik V.* Support Vector Networks // Machine Learning. 1995. V. 20. P. 273–297.

54. *Vapnik V.* The Nature of Statistical Learning Theory. Springer Verlag, 1995.

55. *Burges C. J. C.* A Tutorial on Support Vector Machines for Pattern Recognition // Data Mining and Knowledge Discovery. 1998. V. 2(2). P. 1–47.

56. *Vapnik V., Golowich S., Smola A.* Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing // Adv. Neural Inform. Processing Syst. 1997. V. 9. P. 281–287.

57. *Vannerem P. et al.* Classifying LEP Data with Support Vector Algorithms. hep-ex/9905027.

58. *Vaiciulis A.* Support Vector Machines in Analysis of Top Quark Production // Nucl. Instr. Meth. A. 2003. V. 502. P. 492; hep-ex/0205069.

59. *Prosper H. B.* Multivariate Methods: A Unified Perspective. Prepared for Conf. on Advanced Statistical Techniques in Particle Physics, Durham, England, March 18–22, 2002.