The Neural Support Vector Machine
(Bachelor's thesis)

Michiel van der Ree, s1574256, m.h.van.der.ree@student.rug.nl
Supervisor: Marco Wiering*

July 29, 2011

Abstract

In this paper we describe a new training algorithm which can be used for both classification and regression problems. The Neural Support Vector Machine (NSVM) is a hybrid system whose architecture includes both neural networks and support vector machines (SVMs). The output of the NSVM is given by SVMs who take a central feature layer as their input. This feature layer is in turn the output of a number of neural networks who are trained to minimize the objectives of the SVMs. Since the system is able to handle multiple outputs, it can also be used as a dimensionality reduction method. The results of the conducted experiments indicate that the system performs on par with state-of-the-art classification and regression algorithms. NSVMs with a large feature layer seem able to outperform autoencoders on dimensionality reduction.

1 Introduction

Neural networks are universal in the sense that they can approximate any continuous nonlinear function arbitrary well on a compact interval [10]. However, one of their major drawbacks is that in batch training of neural networks one usually solves a nonlinear optimization problem which has many local minima. Support vector machines [14] are a more robust method for both classification and regression tasks, but by design they are unable to handle multiple outputs. In this article, we introduce the Neural Support Vector machine (NSVM): a hybrid system consisting of both neural networks and SVMs. By combining multiple SVMs into one hybrid network we have aimed to develop a system which extends an SVM's generalization capability to multiple outputs. The output of the NSVM is given by support vector machines who take a small central feature layer as their input. This feature layer is in turn the output of a number of neural networks, trained through backpropagation of the derivatives of the dual objectives of the SVMs with respect to the feature node values.

Our system shows similarities with at least two other methods. Extreme learning machines (ELMs) [5] are two-layer neural networks in which the weight matrix between the input layer and the hidden layer stays fixed at its random initial values. Save for the optimization method, an ELM resembles an NSVM in which the neural networks weights are fixed. An even more similar system was introduced by Suykens [10]. His modified support vector method trains the weight matrix between the input layer and the hidden layer by minimizing an upper bound on the VC dimension. Our system uses another optimization method and we put more emphasis on how it can be extended to handle multiple outputs. Because of its ability to handle mutiple outputs, our system is able to act as a dimensionality reduction method.

With data sets routinely containing observational spaces of millions of dimensions the problem of finding low-dimensional structures in high-dimensional data takes on increasing importance [6]. The curse of dimensionality is clearly exhibited here: Often one needs to conduct meaningful inference with a limited number of samples in a very high-dimensional space. Conventional statistical and computational tools have become severely inadequate for processing and analyzing such high-dimensional data.

*University of Groningen, Department of Artificial Intelligence

Although the data might be presented in a very high-dimensional space, their instrinsic complexity and dimensions are typically much lower. Dimensionality reduction is the process of finding a more efficient way to represent the data. Principle Component Analysis (PCA) is a linear and by far the most popular unsupervised technique to do so, but more recently nonlinear techniques have been proposed [13]. Among them are multilayer neural networks with an odd number of hidden layers dubbed *autoencoders* [4],[13].

The input and the output layer of an autoencoder consist of the same number of nodes D. The middle layer has d nodes, where d is much smaller than D. The network is trained through backpropagation to reconstruct its input in its output layer. When the system is able to decently reconstruct its input, it can be said to have learned an efficient coding of the data. Recent applications of autoencoders include the reconstruction of handwritten digit images [11], missing sensor restoration [12], excitation modeling of text-to-speech synthesis [15], and HIV classification [1]. Because of its ablity to handle multiple output the NSVM can also be used as an autoencoder.

This paper attempts to answer the following research questions:

- How does the single-output NSVM compare to other machine learning algorithms such as neural networks and support vector machines?
- How does the NSVM autoencoder compare to a regular autoencoder?

Outline. In section 2 we will discuss the theory of support vector machines. Next, we present a single-output NSVM, discussing its architecture and the modified support vector objectives it utilizes. In section 4 we show how the system can be adapted to handle multiple outputs by presenting the NSVM autoencoder. Section 5 will cover the setup and the results of the experiments conducted with both the single-output NSVM on several benchmark datasets and the NSVM autoencoder on a dataset of images of faces. The results are discussed in section 6. A conclusion will be presented in section 7.

2 Support Vector Machines

The support vector machine algorithm [14] is a nonlinear generalization of the *Generalized Portrait* algorithm developed in Russia in the sixties [8]. It is based on statistical learning theory, or *VC theory* [14]. VC theory allows vague, handwaving notions such as "generalization" and "outliers" to be placed into a mathematical framework [7]. SV machines can be applied to both classification and regression tasks. Both algorithms are described in this section, starting with support vector classification.

2.1 Classification

In two-class pattern classification, we are given training data of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$, with $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \{-1, 1\}$. We seek to infer a function

$$f : \mathbb{R}^D \to \{-1, 1\}. \tag{2.1}$$

The Support Vector algorithm realizes such a function through the construction of a *decision hyperplane*

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.2}$$

corresponding to decision function

$$f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b). \tag{2.3}$$

In support vector classification, the capability of the decision function to generalize is obtained through maximization of the *margin* between the two classes. Since a large margin corresponds with a small $\mathbf{w}$, we have the problem of minimizing

$$\frac{1}{2}||\mathbf{w}||^2 \tag{2.4}$$

subject to constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1. \tag{2.5}$$

So far we have assumed that the two classes can be completely separated by the hyperplane (2.2). Of course, this might not be the case. We want to decrease the influence outliers might have on the algorithm by relaxing constraint (2.5). This can be done by introducing the slack variables $\boldsymbol{\xi}$. The problem now consists of finding

$$\min_{\mathbf{x}, \boldsymbol{\xi}, b} = \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{\ell} \xi_i \tag{2.6}$$

subject to constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \tag{2.7}$$

$$\xi_i \geq 0. \tag{2.8}$$

with C a constant determining the trade-off between margin maximization and training error minimization.

The function in (2.6) is called the *objective function* and the constraints in (2.7), (2.8) are called the *inequality constraints*. Together, they form a *constrained optimization problem* which is dealt with by introducing *Lagrange multipliers* and a *Lagrangian*

$$\begin{aligned} &L(\mathbf{w}, \boldsymbol{\xi}, b, \boldsymbol{\alpha}, \boldsymbol{\eta}) = \\ &\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{\ell}\xi_i - \sum_{i=1}^{\ell}\eta_i\xi_i \\ &-\sum_{i=1}^{\ell}\alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i) \end{aligned} \tag{2.9}$$

The Lagrangian L has to be minimized with respect to the *primal variables* $\mathbf{w}$, ξ_i and b and maximized with respect to the *dual* variables α_i and η_i, i.e. a saddle point has to be found.

Why is it that the saddle point provides a solution to the problem? Suppose a certrain training pattern $(\mathbf{x}_i, y_i)$ violates constraint (2.7), i.e. $y_i(\mathbf{w} \cdot \mathbf{x}_i) + b < 1 - \xi_i$. Then L can be increased by increasing the corresponding α_i. Simultaneously, $\mathbf{w}$ and b will have to change such that L decreases. To prevent $\alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) + 1 - \xi_i)$ from becoming an arbitrarly large negative number, the change in $\mathbf{w}$ and b will be such that the constraint will eventually be satisfied. Additionally, one can see from (2.9) that for training samples who don't precisely meet the constraint, i.e. $y_i(\mathbf{w} \cdot \mathbf{x}_i) + b > 1 - \xi_i$, the corresponding α_i must be 0, since that is the value by which L is maximized in that case.

Since we are looking for a saddle point, we are looking for a point at which the derivatives of L with respect to the primal variables vanish,

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\xi_i} = \frac{\partial L}{b} = 0 \tag{2.10}$$

which means that at the saddle point

$$\eta_i = C - \alpha_i \tag{2.11}$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0, \tag{2.12}$$

and

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i. \tag{2.13}$$

Substituting (2.13) into (2.3) the latter can be written as a linear combination of a subset of the training patterns $\mathbf{x}_i$, namely

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} y_i\alpha_i(\mathbf{x} \cdot \mathbf{x}_i) + b\right). \tag{2.14}$$

This is the so-called *Support Vector expansion.* The training patterns with a non-zero α value are called *support vectors*, and the algorithm takes its name from them.

Substituting (2.11), (2.12) and (2.13) into the Lagrangian (2.9), we eliminate the primal variables $\mathbf{w}$, ξ_i and b and arrive at the *dual optimization problem*, which is the problem that is usually solved in practice [7]:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \\ &\sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{\ell}\alpha_i\alpha_j y_i y_j(\mathbf{x}_j \cdot \mathbf{x}_j) \end{aligned} \tag{2.15}$$

subject to constraints

$$0 \leq \alpha_i \leq C, \tag{2.16}$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{2.17}$$

The offset b can be determined by initially setting it to zero and then computing

$$b = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i - f(\mathbf{x}_i). \tag{2.18}$$

2.2 Regression

Regression algorithms deal with training data of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$, with $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$. In ε-SV regression the desired function $f(\mathbf{x})$ has at most ε deviation from the target values y_i for all training data, and at the same time is as flat as possible [8].

Consider the case of linear functions f, taking the form

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \text{ with } \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}. \tag{2.19}$$

Desiring a *flat* function means that we seek a small $\mathbf{w}$. We can obtain such a small $\mathbf{w}$ by once again minimizing $\|\mathbf{w}\|^2$. Considering the fact that we don't care about deviations smaller than ε, this yields the optimization problem of minimizing

$$\frac{1}{2}\|\mathbf{w}\|^2 \tag{2.20}$$

subject to constraints

$$\begin{aligned} y_i - \mathbf{w} \cdot \mathbf{x}_i - b &\le \varepsilon, \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i &\le \varepsilon. \end{aligned} \tag{2.21}$$

Similar to the classification case, we relax the influence outliers might have on the task by introducing slack variables $\boldsymbol{\xi}$, $\boldsymbol{\xi}^*$, one for each of the two cases $f(\mathbf{x}_i) - y_i \le \varepsilon$ and $y_i - f(\mathbf{x}_i) \le \varepsilon$ respectively [7]. The problem now consists of finding

$$\min_{\mathbf{w}, \boldsymbol{\xi}^{(*)}, b} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} (\xi_i + \xi_i^*) \tag{2.22}$$

subject to constraints

$$y_i - \mathbf{w} \cdot \mathbf{x}_i - b \le \varepsilon + \xi_i, \tag{2.23}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b - y_i \le \varepsilon + \xi_i, \tag{2.24}$$

$$\xi_i, \xi_i^* \ge 0. \tag{2.25}$$

Analogous to support vector classification, we eventually obtain the dual objective

$$\begin{aligned} &\text{maximize } W(\boldsymbol{\alpha}^{(*)}) = \\ &- \varepsilon \sum_{i=1}^{\ell} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) y_i \\ &- \frac{1}{2} \sum_{i,j=1} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{x}_i \cdot \mathbf{x}_j) \end{aligned} \tag{2.26}$$

subject to constraints

$$0 \le \alpha_i, \alpha_i^* \le C, i = 1, \ldots, \ell, \tag{2.27}$$

$$\sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) = 0. \tag{2.28}$$

The regression estimate takes the form

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i)(\mathbf{x}_i \cdot \mathbf{x}) + b \tag{2.29}$$

2.3 Nonlinearity in SVMs

Barring outliers, the two algorithms described in the preceding sections both assume linearity. The SV classifier assumes that the two classes can be separated by a hyperplane, and the SV regression machine assumes that the relation between $\{\mathbf{x}_i, y_i\}$ is a linear one. Obviously, we want to make SV learning nonlinear.

This could be achieved by preprocessing the training patterns $\mathbf{x}_i$ by a map $\Phi : \mathcal{X} \to \mathcal{F}$ into some feature space $\mathcal{F}$ and then applying the standard SV algorithm [8]. For instance, consider the map $\Phi : \mathbb{R}^2 \to \mathbb{R}^3$ with $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$. Training a linear SV machine on data preprocessed by Φ would yield a quadratic function. While this approach seems reasonable in this particular example, it can easily become computationally infeasible for both polynomial features of higher order and higher dimensionality [8].

A key observation is that both the classifier decision function (2.14) and the regression estimate (2.29) only depend on dot products between patterns $\mathbf{x}_i$. Hence it suffices to know $k(\mathbf{x}_i, \mathbf{x}) := \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$, rather than Φ explicitly. It is this *kernel function* which is often used in both SV classification and SV regression to make the algorithms nonlinear. A number of kernel functions are used widely, including polynomial functions, radial basis functions and sigmoidal functions.

3 The NSVM

Here we introduce the Neural Support Vector Machine. This section concerns a single-output NSVM. First we will discuss its architecture. Next we describe the modifications made to the SVM objectives of section 2. The section is closed by a description of the procedure by which the system is trained.

3.1 Architecture

The NSVM consists of the following elements:

- An input layer consisting of D nodes
- A single-node output layer
- A central feature layer, consisting of d nodes

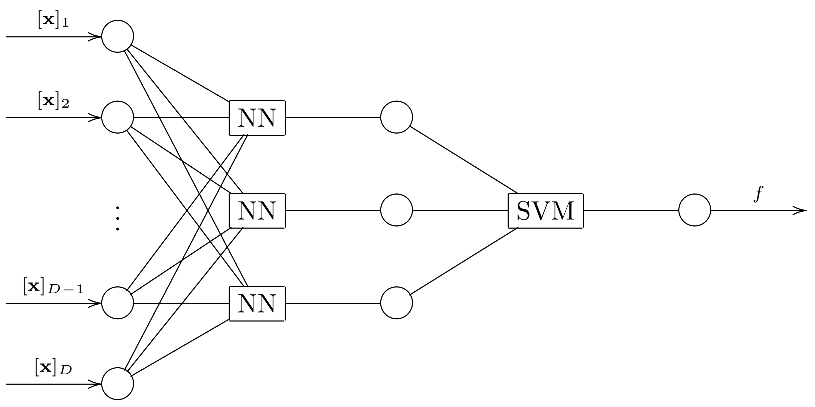

Figure 1: Architecture of an NSVM. In this example, the feature layer consists of three nodes.

- A total of d two-layer neural networks N, which each take the entire input layer as their input and determine the value of one of the feature nodes
- A support vector machine M, which takes the entire feature layer as its input and determines the value of the output node

Figure 1 depicts the architecture graphically. When a pattern $\mathbf{x}$ of dimension D is presented to system, it is propagated through the neural networks, determining the values of the feature layer. We use $\mathbf{f}(\mathbf{x})$ to denote the feature vector repesentation of input pattern $\mathbf{x}$, with $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^d$. The vector $\mathbf{f}(\mathbf{x})$ is then used as input for the Support Vector machine M which determines the value of the output node. In the NSVM classifier, M determines its output according to

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} y_i \alpha_i (\mathbf{f}(\mathbf{x}_i) \cdot \mathbf{f}(\mathbf{x})) + b\right). \tag{3.1}$$

In the NSVM regression estimator, it does so according to

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i)(\mathbf{f}(\mathbf{x}_i) \cdot \mathbf{f}(\mathbf{x})) + b. \tag{3.2}$$

3.2 Modified objectives

To obtain a suitable f, the system must find a representation of the input data in $\mathbf{f}(\mathbf{x})$ which codes the features most relevant to the property corresponding with the desired output. As described in section 2, the Support Vector algorithm obtains a suitable $f(\mathbf{x})$ through minimization of $||\mathbf{w}||^2$ with respect to its primal variables. The NSVM adapts the regular objective of both SV regression and SV classification by introducing $\mathbf{f}(\mathbf{x})$ as an additional primal variable. For classification, this yields the primal objective

$$\min_{\mathbf{x}, \boldsymbol{\xi}, b, \mathbf{f}(\mathbf{x})} = \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{\ell} \xi_i \tag{3.3}$$

subject to constraints

$$y_i(\mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i) + b) \geq 1 - \xi_i, \tag{3.4}$$

$$\xi_i \geq 0. \tag{3.5}$$

Correspondingly, our new 'dual objective' (the primal variable $\mathbf{f}(\mathbf{x})$ has not been eliminated!) for the classification problem is

$$\min_{\mathbf{f}(\mathbf{x})} \max_{\boldsymbol{\alpha}} W(\mathbf{f}(\mathbf{x}), \boldsymbol{\alpha}) =$$
$$\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j (\mathbf{f}(\mathbf{x}_i) \cdot \ \mathbf{f}(\mathbf{x}_j)) \tag{3.6}$$

subject to constraints

$$0 \leq \alpha_i \leq C, i = 1, \ldots, \ell, \tag{3.7}$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{3.8}$$

By substituting $\mathbf{x}$ by $\mathbf{f}(\mathbf{x})$ we have not only obtained a clue as to how a suitable $\mathbf{f}(\mathbf{x})$ might look like, but we have also made the algorithm nonlinear.

Analogously, the new dual objective for the regression problem is

$$\begin{aligned}
&\min_{\mathbf{f}(\mathbf{x})} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} W(\mathbf{f}(\mathbf{x}), \boldsymbol{\alpha}^{(*)}) = \\
&- \varepsilon \sum_{i=1}^{\ell} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) y_i \\
&- \frac{1}{2} \sum_{i,j=1}^{\ell} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{f}(\mathbf{x}_i) \cdot \ \mathbf{f}(\mathbf{x}_j)) \quad (3.9)
\end{aligned}$$

subject to constraints

$$0 \le \alpha_i, \alpha_i^* \le C, i = 1, \ldots, \ell, \quad (3.10)$$

$$\sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) = 0. \quad (3.11)$$

3.3 Training procedure

In this section, the training procedures for both the NSVM classifier and the NSVM regression estimator are described.

3.3.1 Classification

The goals of training the NSVM classifier are twofold:

1. We want to find the $\boldsymbol{\alpha}$ which maximizes (3.6)
2. We want to find the weights of the neural networks such that the output of each network N_a is the a-th entry of $\mathbf{f}(\mathbf{x})$ which minimizes (3.6)

The two goals are not obtained independently, i.e. an adjustment of $\boldsymbol{\alpha}$ requires an adjustment of the neural networks weights and vice versa.

Whenever the system is presented a training pattern $\mathbf{x}_i$, the NSVM adjusts each α_i towards a local maximum of W^+ by

$$\alpha_i \leftarrow \alpha_i + \lambda \frac{\partial W^+}{\partial \alpha_i} \quad (3.12)$$

where λ is a metaparameter controlling the learning rate of the values α_i and

$$W^+ = W - P(\sum_{j=1}^{\ell} \alpha_j y_j)^2 \quad (3.13)$$

where P is a parameter chosen beforehand. By using the derivative of W^+ in (3.12) we ensure that α_i is adjusted towards a satisfaction of constraint (3.8). The derivative with respect to α_i is given by

$$\begin{aligned}
\frac{\partial W^+}{\partial \alpha_i} &= 1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j (\mathbf{f}(\mathbf{x}_i) \cdot \mathbf{f}(\mathbf{x}_j)) \\
&\quad -2P y_i \sum_{j=1}^{\ell} \alpha_j y_j. \quad (3.14)
\end{aligned}$$

Similarly, if we want to adjust the output of neural network N_a such that it minimizes W, we want $[\mathbf{f}(\mathbf{x}_i)]_a$ to be decreased by

$$\frac{\partial W}{\partial [\mathbf{f}(\mathbf{x}_i)]_a} = -\alpha_i y_i \sum_{j=1}^{\ell} \alpha_j y_j [\mathbf{f}(\mathbf{x}_j)]_a \quad (3.15)$$

where we use $[\mathbf{f}(\mathbf{x}_j)]_a$ to denote the a-th entry of $\mathbf{f}(\mathbf{x}_j)$.

In a sense, we can consider (3.15) to be the *error* of N_a, since we want N_a's output to be decreased by it. Just like error backpropagation, we can backpropagate (3.15) through the network to find the weights that minimize (3.15). This is the method by which the weights of all neural networks are updated for every training pattern.

3.3.2 Regression

The two goals of training the NSVM regression estimator are:

1. Find the $\boldsymbol{\alpha}^{(*)}$ which maximizes (3.9)
2. Find the weights of the neural networks such that the output of each network N_a is the a-th entry of $\mathbf{f}(\mathbf{x})$ which minimizes (3.9)

Similar to the training of the classifier, both α_i and α_i^* are updated according to

$$\alpha_i^{(*)} \leftarrow \alpha_i^{(*)} + \lambda \frac{\partial W^+}{\partial \alpha_i^{(*)}} \quad (3.16)$$

with

$$W^+ = W - P_1(\sum_{j=1}^{\ell} (\alpha_j^* - \alpha_j))^2 - P_2 \alpha_i^* \alpha_i \quad (3.17)$$

where P_1, P_2 are parameters chosen beforehand. By using the derivative of W^+ in (3.16) we ensure that

$\alpha_i^{(*)}$ is adjusted towards a satisfaction of constraint (3.11) and towards a pair of $\{\alpha_i^*, \alpha_i\}$ of which at least one of the values equals zero. The two derivatives are given by

$$\begin{aligned}&\frac{\partial W^+}{\partial \alpha_i^*} = \\ &- \varepsilon + y_i - \sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j)(\mathbf{f}(\mathbf{x}_i) \cdot \ \mathbf{f}(\mathbf{x}_j)) \\ &- 2P_1 \sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j) - P_2\alpha_i \end{aligned} \tag{3.18}$$

and

$$\begin{aligned}&\frac{\partial W^+}{\partial \alpha_i} = \\ &- \varepsilon - y_i + \sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j)(\mathbf{f}(\mathbf{x}_i) \cdot \ \mathbf{f}(\mathbf{x}_j)) \\ &+ 2P_1 \sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j) - P_2\alpha_i^* \end{aligned} \tag{3.19}$$

and by backprogation, the output of $[\mathbf{f}(\mathbf{x}_i)]_a$ is decreased by

$$\begin{aligned}&\frac{\partial W}{\partial [\mathbf{f}(\mathbf{x}_i)]_a} = \\ &- (\alpha_i^* - \alpha_i) \sum_{j=1}^{\ell}(\alpha_j^* - \alpha_j)[\mathbf{f}(\mathbf{x}_j)]_a. \end{aligned} \tag{3.20}$$

4 The NSVM autoencoder

We will now show how the single-output NSVM can be adapted to handle multiple outputs by presenting the NSVM autoencoder. The NSVM autoencoder differs from the single-output NSVM in two respects:

1. The output layer consists of D nodes, the same number of nodes the input layer has
2. It utilizes a total of D support vector regression machines M_c, which each take the entire feature layer as their input and determine the value of one of the output nodes

Figure 2 depicts the architecture graphically. Just like the system described in section 3, the forward-propagation of a pattern $\mathbf{x}$ of dimension D determines the values of the feature layer $\mathbf{f}(\mathbf{x})$. The feature layer is then used as input for each Support Vector machine M_c which determines its output according to

$$f_c(\mathbf{f}(\mathbf{x})) = \sum_{i=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i)(\mathbf{f}(\mathbf{x}_i) \cdot \mathbf{f}(\mathbf{x})) + b_c. \tag{4.1}$$

Like the single-output NSVM, the NSVM autoencoder tries to find a representation of the input data in $\mathbf{f}(\mathbf{x})$ which codes the features most relevant to the property corresponding with the desired output. Since the desired output of an autoencoder is the same as the input data, it is trained to learn structural features of the data in general.

Since we are dealing with vectors of output values $\mathbf{y}_i$, the dual objective of each support vector machine M_c is

$$\begin{aligned}&\min_{\mathbf{f}(\mathbf{x})} \max_{\boldsymbol{\alpha}_c^*, \boldsymbol{\alpha}_c} W_c(\mathbf{f}(\mathbf{x}), \boldsymbol{\alpha}_c^{(*)}) = \\ &- \varepsilon \sum_{i=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i + [\boldsymbol{\alpha}_c]_i) + \sum_{i=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i)[\mathbf{y}_i]_c \\ &- \frac{1}{2}\sum_{i,j=1}^{\ell}([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i)([\boldsymbol{\alpha}_c^*]_j - [\boldsymbol{\alpha}_c]_j)(\mathbf{f}(\mathbf{x}_i) \cdot \ \mathbf{f}(\mathbf{x}_j))\end{aligned} \tag{4.2}$$

subject to constraints

$$0 \le \alpha_i, \alpha_i^* \le C, i = 1, \ldots, \ell, \tag{4.3}$$

$$\sum_{i=1}^{\ell}(\alpha_i^* - \alpha_i) = 0. \tag{4.4}$$

Recall from section 3.3 that the first goal of training the NSVM regression estimator is finding the $\boldsymbol{\alpha}^{(*)}$ that maximizes (3.9). Similarly, for every Support Vector machine M_c we want to find the $\boldsymbol{\alpha}_c^{(*)}$ that maximizes (4.2). Like for the single-output NSVM, this is done through gradient ascent.

The minimization of (4.2) with respect to $\mathbf{f}(\mathbf{x})$ is a different story however. Since all SVMs share the same feature layer, we can't just minimize $\mathbf{f}(\mathbf{x})$ for every SVM. It is actually this shared nature of the feature layer which enables the system to handle multiple value outputs. Still, for every neural network N_a, we can try to find a local minimum of (4.2) with respect to $[\mathbf{f}(\mathbf{x}_i)]_a$ by backpropagation of

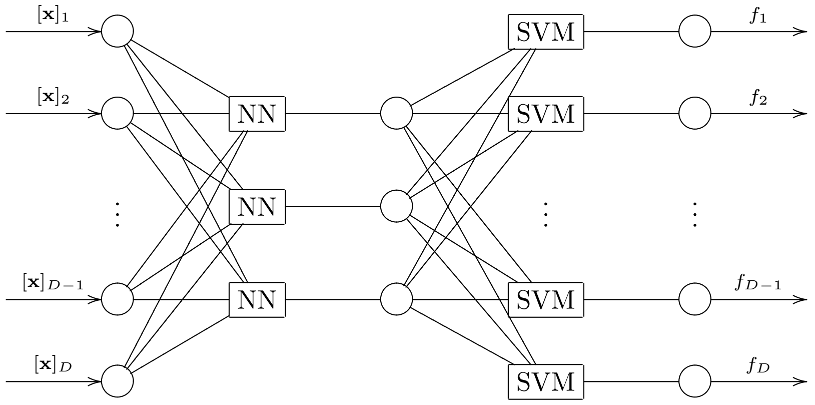

Figure 2: Architecture of the NSVM autoencoder. In this example, the feature layer consists of three nodes.

the *sum* of the derivatives of all W_c's with respect to the a-th feature node

$$\sum_{c=1}^{D} \frac{\partial W_c}{\partial[\mathbf{f}(\mathbf{x}_i)]_a} \tag{4.5}$$

with

$$\frac{\partial W_c}{\partial[\mathbf{f}(\mathbf{x}_i)]_a} = - ([\boldsymbol{\alpha}_c^*]_i - [\boldsymbol{\alpha}_c]_i) \sum_{j=1}^{\ell} ([\boldsymbol{\alpha}_c^*]_j - [\boldsymbol{\alpha}_c]_j)[\mathbf{f}(\mathbf{x}_j)]_a. \tag{4.6}$$

Observe that this situation resembles that of a single neural network with multiple outputs, in which a single hidden node's error is determined by summing the proportional errors of all the neurons it directly contributes to [3].

5 Experiments

Here the results of experiments conducted with the NSVM classifier, regression estimator and autoencoder are presented. The system features a large number of metaparameters, including the size of the feature layer, the learning rate of the neural networks and the value of P in (3.13). In tuning these values experimental parameter optimization software has been utilized. The values that were eventually used were obtained through interaction between the software and fine-tuning by hand. The used parameter sets for each problem can be found in Appendix A.

5.1 Classification

Here we compare NSVM classifier with a neural network and a support vector machine. We used four datasets from the UCI machine learning repository shown in Table 1. Of each dataset, $\frac{9}{10}$ of all the entries were randomly assigned to the training set, leaving the other $\frac{1}{10}$ to be used as test patterns. We have performed a total of 1000 experiments with a new partition of traindata and testdata for each experiment. Each experiment was run for a fixed number of training epochs. At each epoch, the accuracy on both the traindata and the testdata was computed. After all epochs, the epoch with the highest training accuracy was determined and the test accuracy at that epoch was taken to be the accuracy of the system.

Dataset	#Inst.	#Feat.
Ionosphere	351	34
Glass	214	9
Hepatitis	155	19
Iris	150	4

Table 1: The four datasets from the UCI repository that are used in the classification experiment.

The results for the neural networks and the SVM are taken from [16].

Experimental results The results on the four datasets are shown in Table 2. Whether differences in performance are statistically significant has been assessed through use of the Student's t-test. The

Dataset	NSVM	SVM	NN
Ionosphere	93.2 ± 4.1	94.0 ± 2.5$^+$	91.1 ± 4.7$^-$
Glass	66.4 ± 9.8	70.1 ± 10.2$^+$	64.5 ± 11.2$^-$
Hepatitis	80.8 ± 9.4	81.9 ± 9.6$^+$	84.3 ± 8.6$^+$
Iris	96.5 ± 4.8	96.5 ± 4.8	96.8 ± 3.4
Average	84.4	85.6	84.2
Wins/Losses		3-0	1-2

Table 2: The accuracies on the four datasets of the NSVM, a neural network and an SVM. +/- mean significant ($p = 0.05$) win/loss compared to the NSVM.

differences in accuracies of the systems are not very large. There is no dataset on which the NSVM performs significantly better than the support vector machine.

5.2 Regression

For the regression experiments, the three benchmark datasets shown in Table 3 were used.

Dataset	#Inst.	#Feat.
Baseball	337	6
Electric	495	2
Concrete	72	5

Table 3: The three datasets that are used in the regression experiment.

Like in the classification experiment, a total of 1000 experiments per set have been conducted with a new partition of traindata and testdata for each experiment. We compare the performance of the NSVM with those of an NN and an SVM obtained by Graczyk et al. [2] on the same datasets. The results by Graczyk were obtained by using 10-fold cross validation. The mean square errors (MSE) of the systems are used as a measure of their performance.

Dataset	NSVM	SVM	NN
Baseball	0.0233 ± 0.0065	0.0259$^-$	0.0283$^-$
Electric	0.0072 ± 0.0027	0.0064$^+$	0.0064$^+$
Concrete	0.0086 ± 0.0043	0.0085	0.0084
Wins/Losses		1-1	1-1

Table 4: The test MSEs on the three regression datasets of the NSVM, a neural network and an SVM. +/- mean significant ($p = 0.05$) win/loss compared to the NSVM.

Experimental results Table 4 shows the results for the regression experiments. Since the deviations of the SVM and NN results were not known, differences were checked to be significant through use of the one sample t-test. The NSVM outperforms the other systems on one set, but is in turn outperformed on another. The "Concrete" set yielded no significant differences, partly due to large variations in the MSE.

5.3 NSVM Autoencoder

The dataset we use contains 1300 gray-scale images of left eyes of 20 by 20 pixels, generated from the "Labeled Faces in the Wild Dataset". The 400 pixel values have been normalised such that their average value is $\mu = 0.0$ with a standard deviation of $\sigma = 0.3$. Figure 3 shows three examples of the data used.

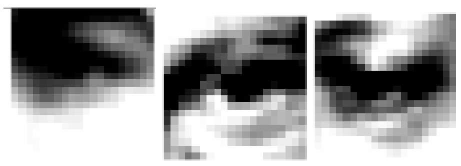

Figure 3: Three examples of the data used in the autoencoder experiment.

We compare the NSVM with a regular autoencoder presented in [9], using the root mean square error (RMSE) as a measure of the performance. The autoencoder is a two-layer neural network that uses backpropagation to adjust its weights. For each experiment, $\frac{2}{3}$ of the data is randomly assigned to the training set, leaving the other $\frac{1}{3}$ to be used as test data. We compare the two systems for three different sizes of the feature layer: 10, 20 and 50 nodes. The results of the regular autoencoder were obtained by averaging the RMSE of 20 experiments. Due to time constraints, we have not been able to run the NSVM for as many experiments. The number of experiments on the NSVM can be found in Table 5. For the autoencoder experiments, we implemented an adaptive learning rate of the neural networks β of the NSVM. Whenever the training error decreases, β is multiplied by 1.1. In the case of an increase of the error, β is multiplied by 0.7.

Experimental results Table 5 shows the results of the autoencoder experiments. With feature layer of 10 and 20 nodes there are no significant differ-

$\lvert\mathbf{f}(\mathbf{x})\rvert$	NSVM	AE	Exp.
10	0.1214 ± 0.0012	0.1209 ± 0.0009	10
20	0.0884 ± 0.0007	0.0881 ± 0.0009	5
50	0.0494	0.0543 ± 0.0007	1
Win/Loss			

Table 5: The test RMSE obtained by the NSVM and the regular autoencoder for different sizes of the feature layer. +/- mean significant ($p = 0.05$) win/loss compared to the NSVM.

ences between the NSVM and the autoencoder according to the Student's t-test. Since we managed to do only one run with an NSVM with a feature layer of 50 nodes we can't check for significance, but the RMSE obtained so far seems to indicate that the NSVM starts to outperform the autoencoder for greater feature layer sizes.

6 Discussion

In this section we will answer the research questions based on the results from section 5.

- **Question:** How does the single-output NSVM compare to other machine learning algorithms such as neural networks and support vector machines?
  Answer: The algorithm can be said to perform on-par with SVMs and NNs. It outperforms the latter two systems on some datasets. More experiments need to be performed to assess on what type of problems the NSVM performs best.

- **Question:** How does the NSVM autoencoder compare to a regular autoencoder?
  Answer: For small feature layers sizes, no significant differences exist between the performance of the NSVM and a regular autoencoder. Preliminary results indicate that the NSVM might have the advantage with larger feature layers. This does come at a price though: Training the NSVM autoencoder takes much more time than training a regular autoencoder.

7 Conclusion

In this paper we have presented the Neural Support Vector machine: A new machine learning algorithm that learns by finding a feature representation of the input data relevant to the task at hand. It does so through interaction between support vector machines and neural networks. We have shown that the system is able to perform on par with state-of-the-art machine learning techniques. It should be noted however that the system has a very large number of metaparameters, which can take a long time to be tuned. On the other hand, the large number of parameters implies that the results presented here still leave room for improvement.

The results on the autoencoding problem with a feature layer of 50 nodes seems to be the most promising. It would be interesting to see how the NSVM compares to a regular autoencoder when this layer is even larger. Apart from regression, classification and dimensionality reduction, we would also like to study how the NSVM can be used in reinforcement learning and time-series problems.

Acknowledgments

I am grateful to a number of people for their help on this project. I would like to thank Prof. Magdalena Graczyk for being so kind to send me the datasets she used in her review article on regression algorithms. Marijn Stollenga's fast reactions and results on requested autoencoder experiments were a great help. I thank Aleke Nolte and Egbert van der Wal for developing parameter optimization software which sped up the process of finding good parameter values greatly. And last but by no means least, I would like to express my gratitude towards Dr. Marco Wiering for his supervision on this project, his comments on my writing and his help with the programming. Also, all credits for the conception of the NSVM algorithm should go to Marco.

References

[1] B. L. Betechuoh, T. Marwala, and T. Tettey. Autoencoder networks for hiv classification. *Current science*, 91(11):1467–1473, 2006.

[2] Magdalena Graczyk, Tadeusz Lasota, Zbigniew Telec, and Bogdan Trawinski. Nonparametric statistical analysis of machine learning

algorithms for regression problems. In *KES 2010*, pages 111–120, 2010.

[3] Kevin Gurney. *An Introduction to Neural Networks.* CRC Press, 1997.

[4] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[5] Guang-Bin Huang, Dian Hui Wang, and Yuan Lan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, (2):107–122, 2011.

[6] Yi Ma, Partha Niyogi, Guillermo Sapiro, and René Vidal. Dimensionality reduction via subspace and submanifold learning. *IEEE Signal Processing Magazine*, 28(2):14–15.

[7] Alex J. Smola and Bernard Schölkopf. *Learning with Kernels.* MIT Press, 2002.

[8] Alex J. Smola and Bernard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[9] Marijn Stollenga. Using guided autoencoder on face recognition. Master's thesis, University of Groningen, May 2011.

[10] J.A.K. Suykens and J. Vandewalle. Training multilayer perceptron classifiers based on a modified support vector method. *IEEE Transactions on Neural Networks*, 1(4):907–911, 1999.

[11] C. C. Tan and C. Eswaran. Reconstruction of handwritten digit images using autoencoder neural networks. In *Canadian Conference on Electrical and Computer Engineering*, pages 465–469, 2008.

[12] B. B. Thompson, R. J. Marks II, and M. A. El-Sharkawi. On the contractive nature of autoencoders: Application to missing sensor restoration. In *Proceedings of the International Joint Conference on Neural Networks*, volume 4, pages 3011–3016, 2003.

[13] L.J.P. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality reduction: A comparative review, 2008.

[14] Vladimir V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[15] S. Vishnubhotla, R. Fernandez, and B. Ramabhadran. An autoencoder neural-network based low-dimensionality approach to excitation modeling for hmm-based text-to-speech. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 4614–4617, 2010.

[16] M.A. Wiering, H. van Hasselt, A.D. Pietersma, and L. Schomaker. Reinforcement learning algorithms for solving classification problems. In *Proceedings of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), Paris*, 2011.

A Parameter Sets

Dataset	$\lvert\mathbf{f}(\mathbf{x})\rvert$	γ_λ	γ_β	P	C	α-Iter.	Hid. Nodes	Epochs	NN outp.
Ionosphere	14	1	1	40	4	3	4	50	Tanh
Glass	10	1	1	0.5	8	8	4	80	Tanh
Hepatitis	20	1	1	1.5	8	10	3	25	Tanh
Iris	8	1	1	1	1024	5	5	100	Lin(-2,2)

Table 6: Parameter sets used in the classification experiments

Dataset	α	Weights NNs	λ	β
Ionosphere	1	(-0.5,0,5)	0.03	0.3
Glass	1	(-0.5,0.5)	0.05	0.05
Hepatitis	1	(-0.5,0.5)	0.012	0.0008
Iris	1	(-0.5,0.5)	0.2	0.09

Table 7: Initialization values of classifier parameters

Dataset	$\lvert\mathbf{f}(\mathbf{x})\rvert$	γ_λ	γ_β	P_1	P_2	C	α-Iter.	ε	Hid. Nodes	Epochs	NN outp.
Baseball	12	1	1	2.6	1.5	6	7	0.15	14	50	Tanh
Electric	16	1	1	3.8	2.0	11	6	0.021	30	50	Tanh
Concrete	40	1	1	0.0	4.0	6	4	0.0046	30	500	Tanh

Table 8: Parameter sets used in the regression experiments

Dataset	α	Weights NNs	λ	β
Baseball	3.0	(-0.5,0.5)	0.000011	0.000016
Electric	3.0	(-0.5,0.5)	0.0083	0.000063
Concrete	3.0	(-0.5,0.5)	0.0094	0.00050

Table 9: Initialization values of regression estimator parameters

$\lvert\mathbf{f}(\mathbf{x})\rvert$	γ_λ	γ_β	P_1	P_2	C	α-Iter.	ε	Hid. Nodes	Epochs	NN outp.
10	0.9993	0.7/1.1	3.5	-0.007	20	1	0.35	150	400	Lin(-3,3)
20	0.9993	0.7/1.1	3.5	-0.007	20	1	0.35	200	600	Lin(-3,3)
50	0.9993	0.7/1.1	3.5	-0.007	20	1	0.25	200	600	Lin(-3,3)

Table 10: Parameters set used in the NSVM autoencoder for different feature layer sizes

α	Weights NNs	λ	β
10.0	(-0.05,0.05)	0.08	0.0005

Table 11: Initialization values of autoencoder parameters