Many Layered Support Vector Machines
(Bachelorproject)

Jeroen van Asselt, s1810723, j.v.asselt@student.rug.nl,
M.A. Wiering*

August 14, 2013

Abstract

This thesis describes the expansion of the machine learning algorithm known as the two-layered Support Vector Machine (SVM) (Wiering, Schutten, Millea, Meijster, and Schomaker, 2013) in order to improve the performance of this system. First, a description of the standard SVM is given, followed by an overview of the two-layered SVM. Then another layer of SVMs is added to the two-layered SVM. The mathematical implications of this expansion will be described in detail. The metaparameters of the three-layered SVM are then found by using a combination of particle swarm optimisation and the UCB bandit algorithm. Experimental results show that the three-layered SVM outperforms a single-layered SVM. However, these results do not show an increase in performance compared to the two-layered SVM.

1 Introduction

In machine learning, there are many tools which can be called upon, all having their own set of pros and cons. If the analysis to be made is a regression analysis, a good choice of tool is using SVMs (Vapnik, 1995) (Cristianini and Shawe-Taylor, 2000).
Although SVMs are superior in regression tasks (Schölkopf and Smola, 2001), they have a limited flexibility when used as a single unit. A similar realisation came in 1969 with respect to the perceptron, when Minsky and Papert (1969) concluded that perceptrons were not able to learn certain functions such as the exclusive-or (XOR) function. The multi-layer perceptron proposed by Rumelhart, Hinton, and Williams (1986) demonstrated that more complex functions, like the XOR function, can be learned by placing multiple artificial neurons in a network. These neural networks have proven to be very useful in a variety of tasks such as classification and time series prediction (Altunkaynak, 2013).

In order to explore the power of network-like architectures, Wiering et al. (2013) introduced the Deep Support Vector Machine. In this thesis the Deep Support Vector Machine will be refered to as the two-layered SVM. The two-layered SVM has a similar architecture as the multi-layer perceptron, but instead of using artificial neurons, the units which compose this system are made of SVMs. This two-layered SVM was then compared to a single SVM in a regression task, which showed a better performance. Related research toward the usage of multiple kernel learning (MKL) instead of using a single kernel has shown that performances increase when using MKL (Gönen and Alpaydin, 2011). Also, MKL has proven to be a good choice when performing object recognition, which is concluded by Varma and Ray (2007).

This paper gives a description about an expansion of the two-layered SVM which in this paper will be referred to as the Many-Layered Support Vector Machine (three-layered SVM). Section 2 will first give a description of the single layered SVM. Then, the architecture and mechanics of the two-layered SVM are introduced. The three-layered SVM is constructed by taking the original two-layered SVM architecture and

*University of Groningen, Department of Artificial Intelligence

then adding another layer of SVMs to it. Section 3 will describe the expansion of the mathematical formulation used in the two-layered SVM to fit the three-layered SVM. Section 4 describes the conditions in which the three-layered SVM was tested and the outcome of these tests. Finally, section 5 will discuss the outcome.

2 Previous work

2.1 Support Vector Machines

For understanding the two-layered and the three-layered SVM, one must first understand the basic element: the SVM itself. The SVM is a tool which can be used for classification and regression analysis. When performing a regression analysis, the SVM tries to learn a function to generate an output to fit the training data properly. This description of the SVM below is based on a tutorial written by Smola and Schölkopf (2004).

The SVM itself is a supervised learning algorithm, and therefore the data which is presented to the SVM comes in the form of

$$(x_i, y_i) \quad where \quad x_i \in \mathbb{R}^s, \; i = 1 \dots l$$

pairs. Here, x_i is some vector of dimensionality s, where y_i is the desired output which comes in the form of a numerical value. The output function which has to be learned is of the form $\mathbf{f}(x) = w \cdot x_i + b$, where $w \cdot x_i$ is the dot product between vectors w and x_i, and b is the bias. The goodness of the function which has to be learned by the SVM must be represented in a variable. To make a distinction between wrong outputs and outputs with a certain error which are accepted as a good outcome, ϵ is introduced. This variable represents the maximal error which may be produced by the output function.
Previous research has shown that the regression fits the data better when w is minimised (Vapnik, 1995). The constraints which define a maximal error ϵ do not allow errors bigger than ϵ. Therefore, a non-negative slack variable known as $\xi^{(*)}$ is introduced, which tolerates errors larger then ϵ. The optimisation problem can now be defined as (2.1) and (2.2) as originally proposed by Vapnik (1995)

$$minimise \; \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \quad (2.1)$$

$$subject\ to \begin{Bmatrix} y_i - w \cdot x_i - b \leq \epsilon + \xi_i \\ w \cdot x_i + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{Bmatrix} \quad (2.2)$$

where $C > 0$.

For this optimisation problem to be written as a single mathematic equation, Lagrangian multipliers $\eta^{(*)}$ and $\alpha^{(*)}$ are introduced, both being larger or equal to zero. Equation L_p describes the Lagrangian primal objective function.

$$\begin{aligned} L_p = \frac{1}{2}\|w\|^2 + C\sum_{i}^{l}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ - \sum_{i=1}^{l}\alpha_i(\epsilon + \xi_i - y_i + w \cdot x_i + b) \\ - \sum_{i=1}^{l}\alpha_i^*(\epsilon + \xi_i^* + y_i - w \cdot x_i - b) \end{aligned} \quad (2.3)$$

In general, if one tries to find the global minimum of a function, the derivatives of this function with respect to all variables must be taken and set to 0. Then, these derivatives can be substituted back into the original function. Using this technique, the global minimum can be obtained.
The solution of L_p is found where the parameters b, w, and $\xi_i^{(*)}$ are minimised. To find a solution for L_p, one must take partial derivatives of L_p with respect to b, w, and $\xi_i^{(*)}$.

$$\begin{aligned} \partial_b L_p = & \quad \textstyle\sum_{i=1}^{l}(\alpha_i^* - \alpha_i) & = 0 \\ \partial_w L_p = & \quad w - \textstyle\sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i & = 0 \\ \partial_{\xi_i^{(*)}} L_p = & \quad C - \alpha_i^{(*)} - \eta_i^{(*)} & = 0 \end{aligned}$$

which then have to be substituted back into L_p, forming the dual objective function L_d:

$$\begin{aligned} L_d = -\frac{1}{2}\sum_{i,j=1}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(x_i \cdot x_j) \\ -\epsilon\sum_{i=1}^{l}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i \quad (2.4) \\ where\ C \geq \alpha^{(*)} \geq 0 \end{aligned}$$

This dual objective function L_d provides a good solution of the problem when the model to be learned is linear. This is due to the dot product in L_d. For the SVM to be able to learn non-linear models, the data must be mapped to a higher dimension. To map this data to a higher dimension, the kernel function is introduced. This function can be defined in a variety of ways, and is therefore denoted in general as $K(x_i, x_j)$, which replaces the dot product in the dual objective function (2.4).

2.2 Two-layered SVM

The two-layered SVM, as proposed by Wiering et al. (2013), is a structure consisting of two layers of SVMs. It very much resembles the architecture and mechanics of a feedforward neural network. The data flows through the system like in any feedforward neural network. First, the data is presented to the lowest layer. This layer then extracts hidden variables from this data. These hidden variables are then used by the highest layer, to give the final output in the form of a regression function. Once the output is given by the system, it can use a backpropagation-like method to propagate the objective function back into the system, so it can adjust the parameters for making a better hidden variable extraction, making the next datapropagation come to a better result. The algorithm is displayed in algorithm 2.1

Algorithm 2.1 Two-layered SVM algorithm

```
Initialise all SVMs within appropriate layers
Train all hidden SVMs on perturbed dataset
for all training epochs do
    Recompute main SVM kernel matrix
    Train main SVM
    for all Hidden SVMs do
        Perform backpropagation
        Train SVM to obtain α^(*)
    end for
end for
```

Certain parts of this algorithm need more explanation, which will be described next.
Throughout this thesis the SVMs which extract hidden variables will be refered to as hidden SVMs, and the layers in which they are located will be refered to as hidden layers. Note that the usage of the word 'backpropagation' in this thesis does not refer to the traditional backpropagation techniques. Instead it refers to the process which the two-layered SVM and the three-layered SVM uses for constructing new target pairs, which will be described in section 3.2.

3 Expansion of the two-layered SVM

In order to expand the two-layered SVM, another hidden layer is added to this system. All neighboring layers are fully connected, to keep the architecture of the three-layered SVM mathematically as simple as possible. Figure 1 shows schematically the architecture of the three-layered SVM used in this paper. Before the mathematical implications of this expansion can be described, a new parameter d is introduced, which will be used as an index for specifying layer index, where capital D is used for describing the top layer.

3.1 Propagation of data

For the hidden variables to be extracted from the original inputs, the output function as is used in the two-layered SVM is modified to match the three-layered SVM specifications, which results in equation (3.1).

$$\mathbf{f}(\mathbf{x})_a^1 = \sum_{i=1}^{l} (\alpha_i^*(a)^1 - \alpha_i(a)^1) \cdot K(\mathbf{x}_i, \mathbf{x}) + b_a^1 \tag{3.1}$$

In this equation, both α^* and α are the coefficients for the support vectors and b is the bias. All these parameters are bound to a single SVM a at layer depth 1.
The hidden variables extracted by (3.1) are used by higher layers as input vector. Superscript 1 as depicted in equation (3.1) determines the layer depth. This superscript is denoted by d, where $d = 1$ is set as the lowest hidden layer.
To take input from hidden variables extracted by a lower layer, minor modifications need to be made

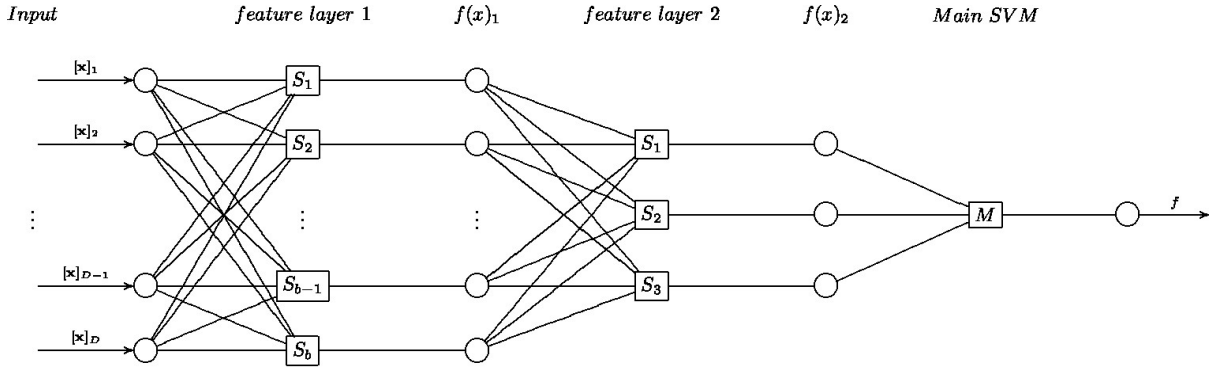

Figure 1: three-layered SVM architecture consisting of two hidden layers and one output layer, where hidden layer one contains b SVMs S, and hidden layer two contains three SVMs, and the main layer contains only a single SVM. Spheres represent extracted hidden variables.

to (3.1) which result in the following equation:

$$\mathbf{f}(\mathbf{f}(\mathbf{x}_j)^d)_b^{d+1} = \sum_{i=1}^{l}(\alpha_i^*(b)^{d+1} - \alpha_i(b)^{d+1}) \cdot K(\mathbf{f}(\mathbf{x}_i)^d, \mathbf{f}(\mathbf{x}_j)^d) + b_b^{d+1} \quad (3.2)$$

As the layer parameter indicates, equation (3.2) is on top of (3.1) which is indicated by $d+1$. Therefore, the arguments of the kernel function refer to the hidden variables extracted by lower layers. Equation (3.2) can be applied on any layer where $D > d > 1$ to calculate the output of all SVMs, including the main SVM.

In the two-layered SVM, Wiering et al. (2013) used the radial base function (RBF) kernel, because this kernel type produced the best results. Therefore, the kernel function that is used by the three-layered SVM is also a RBF kernel which is defined by

$$K(\mathbf{x}_i, \mathbf{x}) = \exp(-\sum_a \frac{(\mathbf{x}_i - \mathbf{x})^2}{\sigma_d})$$

for the layer which takes the input from the original data. The non-lowest layers are using

$$K(\mathbf{f}(\mathbf{x}_i)^d, \mathbf{f}(\mathbf{x})^d) = \exp(-\sum_a \frac{(\mathbf{f}(\mathbf{x}_i)_a^d - \mathbf{f}(\mathbf{x})_a^d)^2}{\sigma_d})$$

where the kernel applies to lower level extracted hidden variables.

3.2 Learning algorithm

Because of the SVM units which make up the system, a supervised way of learning is used in the three-layered SVM to construct new target pairs per SVM. This objective function is constructed by maintaining the following constraints, which are quite similar to the constraints in (2.2)

$$\begin{aligned} y_i - \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i) - b &\leq \epsilon + \xi_i \\ \mathbf{w} \cdot \mathbf{f}(\mathbf{x}_i) + b - y_i &\leq \epsilon + \xi_i^* \\ \xi_i^*, \xi_i &\geq 0 \end{aligned}$$

where ϵ is the error tolerance, ξ^* and ξ are slack variables. For the system to find the right coefficients for all SVMs, the following dual objective function is formulated (in roughly the same way as the dual objective was formulated in section 2) which applies to the main SVM located at layer D.

$$\begin{aligned} \min_{\mathbf{f}(\mathbf{x})^{D-1}} \max_{\boldsymbol{\alpha}^D, \boldsymbol{\alpha}^{*D}} W^D(\mathbf{f}(\mathbf{x})^{D-1}, \boldsymbol{\alpha}^{(*)D}) = \\ -\varepsilon \sum_{i=1}^{\ell}(\alpha_i^{*D} + \alpha_i^D) + \sum_{t=1}^{\ell}(\alpha_i^{*D} - \alpha_i^D)y_i \\ -\frac{1}{2}\sum_{i,j=1}^{\ell}(\alpha_i^{*D} - \alpha_i^D)(\alpha_j^{*D} - \alpha_j^D) \cdot \\ K(\mathbf{f}(\mathbf{x}_i)^{D-1}, \mathbf{f}(\mathbf{x}_j)^{D-1}) \end{aligned} \quad (3.3)$$

In this formulation, $\alpha^{(*)}$ maximizes W^D. This is done by using the following gradient ascent learning

rule to get $\alpha^{(*)}$ to a (local) maximum.

$$\alpha_i^D = \alpha_i^D + \lambda(-\epsilon - y_i + \sum_j(\alpha_j^{*D} - \alpha_j^D) \cdot K(\mathbf{f}(\mathbf{x}_i)^{D-1}, \mathbf{f}(\mathbf{x}_j)^{D-1})) \quad (3.4)$$

This system learns by constructing new target pairs $(\mathbf{f}_i^d, y_i^d)$, by adjusting y_i^d for each vector extracting SVM input vector $\mathbf{f}_i^d$. The magnitude of this adjustment is determined by a learning parameter μ, multiplied by the error of the SVM which is backpropagated through the network. For the topmost hidden layer, the mathematical formulation of this backpropagation is as follows:

$$\frac{\partial W^D}{\partial \mathbf{f}(\mathbf{x}_i)_a^{D-1}} = -(\alpha_i^{*D} - \alpha_i^D) \sum_{j=1}^{l}(\alpha_j^{*D} - \alpha_j^D) \cdot K(\mathbf{f}(\mathbf{x}_i)^{D-1}, \mathbf{f}(\mathbf{x}_j)^{D-1}) \quad (3.5)$$

The construction of a new example in the topmost hidden layer is therefore defined as $(x_i, \mathbf{f}(\mathbf{x})_a^{D-1} - \mu \cdot \partial W^D / \partial \mathbf{f}(\mathbf{x})_a^{D-1})$.

The backpropagation method as described above applies only to the layers which are located in layer $D-1$, because the objective function W^D applies to the main SVM. SVMs which aren't using $\partial W^D / \partial \mathbf{f}(\mathbf{x}_i)_a^{D-1}$ (such as D-2 and below) for constructing new datasets are never reached by this system. Therefore, partial derivatives must be included into equation $\partial W^D / \partial \mathbf{f}(\mathbf{x}_i)_a^{D-1}$. This can be done by taking the output equation (3.1) or (3.2) (depending on layer depth) of the SVM at layer d, and the partial derivative to the output function of the SVM at layer $d-1$, which is defined by the following equation.

$$\frac{\partial \mathbf{f}(\mathbf{x}_i)_a^{d+1}}{\partial \mathbf{f}(\mathbf{x}_j)_b^d} = -(\alpha_i^*(a)^{d+1} - \alpha_i(a)^{d+1}) \cdot \frac{\mathbf{f}(\mathbf{x}_i)_b^d - \mathbf{f}(\mathbf{x}_j)_b^d}{\sigma_d} \cdot K(\mathbf{f}(\mathbf{x}_i)^d - \mathbf{f}(\mathbf{x}_j)^d) \quad (3.6)$$

When $i{=}j$, the partial derivative of $\partial \mathbf{f}(\mathbf{x}_i)_a^{d+1} / \partial \mathbf{f}(\mathbf{x}_i)_b^d$ needs to be adjusted. In this case, the following derivative is used for backpropagation instead of (3.6).

$$\frac{\partial \mathbf{f}(\mathbf{x}_i)_a^{d+1}}{\partial \mathbf{f}(\mathbf{x}_i)_b^d} = -\sum_k(\alpha_k^*(a)^{d+1} - \alpha_k(a)^{d+1}) \cdot \frac{\mathbf{f}(\mathbf{x}_i)_b^d - \mathbf{f}(\mathbf{x}_k)_b^d}{\sigma_d} \cdot K(\mathbf{f}(\mathbf{x}_i)^d - \mathbf{f}(\mathbf{x}_k)^d) \quad (3.7)$$

To plug derivations 3.6 and 3.7 back into the existing equation, $\partial \mathbf{f}(\mathbf{x}_i)_a^d / \partial \mathbf{f}(\mathbf{x}_j)_b^{d-1}$ just needs to be multiplied by $\partial \mathbf{W} / \partial \mathbf{f}(\mathbf{x}_j)_b^{D-1}$, which yields the following equation:

$$\frac{\partial W^D}{\partial \mathbf{f}(\mathbf{x}_j)_b^{D-2}} = \sum_a \frac{\partial W^D}{\partial \mathbf{f}(\mathbf{x}_i)_a^{D-1}} \cdot \frac{\partial \mathbf{f}(\mathbf{x}_i)_a^{D-1}}{\partial \mathbf{f}(\mathbf{x}_j)_b^{D-2}} \quad (3.8)$$

Because the system has all neighboring SVMs fully connected per layer, the error of an SVM located in layer $D-x$ is backpropagated through all SVMs on layer $D-x+1$. Hence, the summation sign must be added to include all top influences.

Equation (3.8) describes the backpropagation of objective W^D from the top layer D. It is also possible for layers $D-x$ to propagate their objective function W^{D-x}. In this three-layered system, $\partial W_a^{D-1} / \partial \mathbf{f}(\mathbf{x}_j)^{D-2}$ describes the backpropagation of the objective at SVM a from layer $D-1$ to the lowest layer in this system $D-2$. The three-layered SVM uses both equation (3.8) and $\partial W_a^{D-1} / \partial \mathbf{f}(\mathbf{x}_j)^{D-2}$, and tries to find an optimum by multiplying these equations with certain parameters β_1 and β_2. The application of these two methods both yield a max-min-max formulation.

3.3 Algorithm description

The addition of more layers requires the addition of an additional for-loop in algorithm 2.1. The algorithm which then makes up the three-layered SVM is displayed in algorithm 3.1.

The number of training cycles which can be seen in algorithm 3.1 is based on an estimated optimal metaparameter. Note that the cross validation runs are not displayed in this overview of the algorithm.

Algorithm 3.1 Three-layered SVM algorithm

```
Initialise all SVMs within appropriate layers
Train all hidden SVMs on perturbed dataset
for all training epochs do
  Recompute main SVM kernel matrix
  Train main SVM
  for all hidden Layers do
    for all hidden SVMs do
      Recompute SVM kernel matrix
      Perform backpropagation
      Recalculate α^(*)
    end for
  end for
end for
```

4 Results

To decide if the three-layered version outperforms the two-layered SVM, the testing is done in three stages. First, the metaparameters have to be found in order for the three-layered SVM to perform to its upmost. Using these metaparameters, the system has to train on a dataset for then it has to be tested to get the error of this system. These errors then have to be compared to the error made by the two-layered SVM using statistical analysis. The datasets used in this thesis are the same datasets which Wiering et al. (2013) used, which are described in Graczyk, Lasota, Telec, and Trawiski (2010). They are shown in table 1.

4.1 Finding the correct metaparameters for the three-layered SVM

The search for the metaparameters of the three-layered SVM is compared to the two-layered SVM more complex. This is because the three-layered SVM has more metaparameters to be determined. These metaparameters determine the form of the system by varying the number of SVMs per layer. Also, the internal behavior of the SVMs are determined during this process. Consider the initialisation of σ . If this parameter is set to a high number, the kernel function would overreach itself, for every kernel distance will be a relevant one. The opposite is also true: if σ is set to low, none of the distances matter. One can do this parameter optimisation by hand, but this task would be tedious: the three-layered SVM relies on 36 adjustable metaparameters.
For the configuration of metaparameters to be judged, the three-layered SVM has to be executed with them and judge the configuration using the output. This iterative process can be done automatically by Particle Swarm Optimisation (PSO) (Kennedy and Eberhart, 1995). In this research, PSO ran 100.000 parameter configurations on the three-layered SVM, and using a UCB bandit (Auer, Cesa-Bianchi, and Fischer, 2002) algorithm to eliminate unpromising parameter configurations.
The differences between the datasets do not allow a general good parameter configuration, so the parameter optimisation as described above was done for every dataset.

4.2 Training the three-layered SVM

Once good metaparameters per dataset are found, they can be used to train the three-layered SVM. First, it divides the data into partitions consisting of 90% and 10% of all entries in the dataset, which are used for training and testing respectively. The results are then crossvalidated 1000 or 4000 times, which is determined by the times a dataset was crossvalidated by the two-layered SVM, to preserve the comparative nature of this research. For each run, the three-layered SVM calculates an error. After the crossvalidation is done, the three-layered SVM gives the Mean Squared Error (MSE) as output, which will be used to compare the performance to the performance of the two-layered SVM.

Dataset	Entries	Features	crossval repetitions
Diabetes	43	2	4000
Machine-CPU	188	6	1000
Baseball	337	6	4000
Ele1-2	495	2	1000

Table 1: Dataset properties and number of crossvalidation repetitions

4.3 Results with the three-layered SVM

The outcome of the regression analysis made by the three-layered SVM are shown in table 2, together with the associated score of single SVM and

	Mean Squared Error + Standard Error		
Dataset	single SVM	two-layered SVM	three-layered SVM
Diabetes	0.02719±0.00021	**0.02327±0.00022**	0.02694±0.00025
Machine-CPU	0.00805±0.00018	**0.00638±0.00012**	0.00743±0.00020
Baseball	0.02413±0.00011	**0.02294±0.00010**	0.02314±0.00016
Ele1-2-Electrical-Length	0.006382±0.000066	0.00641±0.00007	0.00653±0.00007

Table 2: Results per dataset. Bold results are results which indicate significant better performance with respect to the other two systems.

the two-layered SVM. The scores are denoted as $MSE \pm SE$, where the standard error SE is calculated by dividing the standard deviation by the number of crossvalidation repetitions, which can be found in table 1. The table is filled with four columns, where three are dedicated to the outcome of the single SVM, the two-layered SVM and the three-layered SVM. These outcomes represent a population of error values consisting of 1000 or 4000 members. For the difference between the performance of the two-layered SVM and the performance of the three-layered SVM to be viewed as significant, a students t-test was performed on both scores. In order to mark the difference in performance as significant, a p-value < 0.05 is chosen.

5 Discussion

As can be seen in table 2, the two-layered SVM convincingly outperforms the three-layered SVM (all resulting in a p-value < 0.0001). This could be the result of unforseen properties of the dataset. For example: it is highly possible that the three-layered SVM needs more data simply because the system is bigger in size than the two-layered SVM. Therefore, the full potential of the three-layered SVM has not been fully uncovered, so future work should be focussing on bigger datasets.
Although the three-layered SVM performs better on some datasets with respect to the use of a single SVM, one should think twice before chosing between the three-layered SVM or single layer SVM based on performance alone. During the training phase, the three-layered SVM performed the training cycles very slowly, as a result of the quantity of kernel matrices which had to be recalculated each time. One could therefore conclude that instead of the number of training cycles, a certain amount of training time should determine the choice between a single layer or three-layered SVM.
To improve the idea of building a layered support vector machine, future research should investigate the possibility of creating a system which dynamically adds more layers where needed. This way, a system can be found dynamically which suits the properties of the datasets best in order to prevent unnecessary usage of both high performance machines and time to train the three-layered SVM.

References

A. Altunkaynak. Prediction of significant wave height using geno-multilayer perceptron. *Ocean Engineering*, 58(0):144 – 153, 2013.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 1 edition, 2000.

M. Gönen and E. Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, pages 2211–2268, July 2011.

M. Graczyk, T. Lasota, Z. Telec, and B. Trawiski. Nonparametric statistical analysis of machine learning algorithms for regression problems. In Rossitza Setchi, Ivan Jordanov, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, volume 6276 of *Lecture Notes in Computer Science*, pages 111–120. Springer Berlin Heidelberg, 2010.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, 1995.

M. Minsky and S. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

A.J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14 (3):199–222, August 2004.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verslag, 1995.

M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.

M.A. Wiering, M. Schutten, A. Millea, A. Meijster, and L. Schomaker. Deep support vector machines for regression problems. In *International Workshop on Advances in Regularization, Optimization, Kernel Methods, and Support Vector Machines: theory and applications*, Leuven Belgium, 2013.