Deep Support Vector Machines for Regression Problems

Marten Schutten, s1777475, m.h.schutten@student.rug.nl
supervisor: Marco Wiering*

January 24, 2013

1 Introduction

Machine learning is one of the more actual topics within modern Artificial Intelligence. One of the most popular algorithms used for machine learning is the support vector machine (SVM) [3, 10, 4, 2]. SVMs provide a robust method to solve classification and regression problems.

In order to project data on the featurespace SVMs make use of kernels [4, 8]. Traditionally only single kernels were used, but in the past decade several methods have been proposed to apply multiple kernel learning [1, 7] to support vector machines. However, all of these kernels are designed a priori, and can not be decomposed into subkernels.

This paper will propose a new method for support vector machines using a new layer of support vector machines instead of a-priori defined kernels. This new method will be dubbed the 'Deep Support Vector Machine' (DSVM). The deep support vector machine will use an extra layer of support vector machines in order to learn to extract features from a set of inputs. These features will be propagated to the 'main' SVM, which will use those to construct a decision function. Once this decision function has been created, the outcome values produced by the main SVM, will be used to adapt to the different features. This method may be used for both classification and regression problems, however the focus of this paper will only be on regression problems.

Through the essence of the structure of the deep support vector machine it is also possible to concatenate several layers of support vector machines, so that not only the main SVM will use support vector machines to learn its features, but also feature SVMs might make use of SVMs that extract subfeatures. When using an extra layer of SVMs the parent support vector machine does not need an a-priori defined kernel to find its decision function, but uses the properties of its desired output in order to find a decision function.

This paper will present the new application of Deep Support Vector Machines. The main research question to be answered is: How well will the deep support vector machine work compared to regular support vector machines?

The hypothesis to be tested is: The deep support vector machine, using a single output unit, will be able to make more accurate predictions than regular support vector machines. In this context more accurate can be defined as having a smaller error.

Before presenting the deep support vector machine, the workings of support vector machines themselves will be explained in section 2. After that the DSVM itself will be presented in section 3. In section 4 the setup of the experiments that were run will be given. In section 5 the results of the experiments will be discussed, and finally, in section 6 we will give an overview of the implications of the research.

2 Support Vector Machines

Support Vector Machines were first introduced in the sixties by Vapnik [10] (also see [3]) as a solution to binary classification problems. As stated before Support Vector Machines try to place a hyperplane on a featurespace in order to separate two classes. Now a brief explanation on the workings of SVMs will be given. Since we will only focus on regression problems we will limit this explanation to (ϵ-insensitive) regression SVMs. For a more thorough explanation on Support Vector Machines, see Burges [2].

*University of Groningen, Department of Artificial Intelligence

2.1 Support Vector Regression

Consider a set of trainingdata containing the data $\mathfrak{D} = \{(\mathbf{x}_1, y_1) \ldots (\mathbf{x}_l, y_l)\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$. Here $\mathbf{x}_i$ denotes an input pattern with corresponding output value y_i.

We wish to find a function $f(\mathbf{x})$ that maps the inputvalues $\mathbf{x}_i$ to y_i with a deviation of at most ϵ (hence the name ϵ-insensitive-regression). This function $f(\mathbf{x})$ can take several forms, but we will begin explaining the linear form yielding our target function

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b, \quad \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}. \tag{2.1}$$

In this function $\mathbf{w}$ is a weight vector. The parameter $\frac{b}{\|\mathbf{w}\|}$describes the offset of the hyperplane from the origin, perpendicular to $\mathbf{w}$. We wish this function to be as *flat* as possible, meaning we want $\mathbf{w}$ to be as small as possible. Considering this we wish to minimize

$$\frac{1}{2}\|\mathbf{w}\|^2 \tag{2.2}$$

subject to

$$y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \leq \epsilon \tag{2.3}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \epsilon \tag{2.4}$$

Using these constraints we assume that there is a feasible solution $f(\mathbf{x})$ which maps all input patterns $\mathbf{x}_i$ to $y_i \pm \epsilon$. However, this may not always be the case. Hence we wish to add two slack variables ξ and ξ^* in order to allow some errors in our solution. Adding these relaxation constraints yields the following problem:

$$\min_{\mathbf{w},\xi^*,b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{2.5}$$

subject to

$$y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \leq \epsilon + \xi_i \tag{2.6}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \epsilon + \xi_i^* \tag{2.7}$$

$$\xi_i, \xi_i^* \geq 0 \tag{2.8}$$

In this equation C is a constant determining the trade-off between the flatness of f and the deviations from y_i.

Problem (2.5) can be solved by introducing Lagrangian Multipliers $\alpha_i^{(*)}$and $\eta_i^{(*)}$ (by $\alpha_i^{(*)}$ we mean α_i and α_i^*). We then obtain a new objective function which we want to minimize with our primal variables ($\mathbf{w}, \xi^*$and b) and maximize with our newly added Lagrangian multipliers. Doing so we obtain the Lagrange function [9]:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \tag{2.9}$$

$$- \sum_{i=1}^{l}(\alpha_i(\epsilon + \xi_i - y_i + \mathbf{w} \cdot \mathbf{x}_i + b))$$

$$- \sum_{i=1}^{l}(\alpha_i^*(\epsilon + \xi_i^* + y_i - \mathbf{w} \cdot \mathbf{x}_i - b))$$

Since we are dealing with a saddle point the partial derivatives of L with respect to the primal variables will have to vanish in order to obtain optimality, hence

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

which yields

$$\frac{\delta L}{\delta \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i = 0 \tag{2.11}$$

$$\frac{\delta L}{\delta \xi_i} = C - \alpha_i^{(*)} - \eta_i^{(*)} \tag{2.12}$$

$$\frac{\delta L}{\delta b} = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0 \tag{2.13}$$

When we substitute (2.11), (2.12) and (2.13) into (2.9) we obtain the following dual problem:

$$\text{maximize } W(\alpha^{(*)}) = \tag{2.14}$$

$$- \frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$- \epsilon\sum_{i=1}^{l}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{l}y_i(\alpha_i^* - \alpha_i)$$

subject to

$$\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \tag{2.15}$$

$$\alpha_i, \alpha_i^* \in [0, C] \tag{2.16}$$

Finally we can rewrite formula (2.11) in such a way that we can reformulate the regression estimate so that it can be described as a linear combination of all training patterns [9].

$$\mathbf{w} = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)\mathbf{x}_i \tag{2.17}$$

hence

$$f(\mathbf{x}) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)(\mathbf{x}_i \cdot \mathbf{x}) + b \tag{2.18}$$

2.2 Non-linear SVMs

In the explanation above we have shown the workings of SVMs trying to find a linear function $f(\mathbf{x})$ to map the input patterns to their corresponding output values. However, often other types of functions are used in order to obtain better estimations of y. In order to obtain these other forms we make use of so called kernel functions.

In our case this means we have to rewrite formula (2.14) to replace the linear form of $f(\mathbf{x})$ with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$. This general form of the dual problem looks like:

$$\begin{aligned} &\text{maximize } W(\alpha^{(*)}) \qquad (2.19)\\ &-\frac{1}{2}\sum_{i,j=1}^{l} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(\mathbf{x}_i, \mathbf{x}_j)\\ &-\epsilon\sum_{i=1}^{l}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{l} y_i(\alpha_i^* - \alpha_i) \end{aligned}$$

which is also subject to

$$\sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0$$
$$\alpha_i, \alpha_i^* \in [0, C]$$

Now that we have replaced the linear element in the dual form within the dual form for a more general one, allowing us to use kernels, we will have to do the same for the regression estimate (2.18). This gives us the more general estimate

$$f(\mathbf{x}) = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i)K(\mathbf{x}_i, \mathbf{x}) + b \tag{2.20}$$

In the above function $K(\mathbf{x}_i, \mathbf{x})$ represent any type of kernel function we would like to use. For example the kernel function for the linear function would be $\mathbf{x}_i \cdot \mathbf{x}$, yielding the dual problem and regression estimate as shown in section 2.1.

For our DSVM we use the gaussian kernel type. We will discuss this kernel later on in the article. For now it suffices to know that the gaussian dual form and regression estimate are given by

$$\begin{aligned} &\text{maximize } W(\alpha^{(*)}) = \qquad (2.21)\\ &-\frac{1}{2}\sum_{i,j=1}^{l} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)e^{-\Sigma_k \frac{(x_i^k - x_j^k)^2}{\sigma}}\\ &-\epsilon\sum_{i=1}^{l}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{l} y_i(\alpha_i^* - \alpha_i) \qquad (2.22) \end{aligned}$$

and

$$f(\mathbf{x}) = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i)e^{-\Sigma_k \frac{(x_i^k - x^k)^2}{\sigma}} + b \tag{2.23}$$

Where we use Σ_k to describe the sum over all input nodes. Note that equation (2.21) is still subject to the same constraints as before.

3 Deep Support Vector Machines

Now that we have seen how regular SVMs work we will examine the Deep Support Vector Machine. First let's take a look at the way Deep Support Vector Machines are constructed. For simplicity we will look at a case with only one layer of feature SVMs. It is possible to use multiple layers so there could be learned subfeatures, however this will not be further examined in this article. After having shown the architecture, the training procedure will be explained, followed by the testing procedure.

3.1 DSVM Architecture

The Deep Support Vector Machine (using a single feature layer) consists of the following parts:

- an input layer containing D inputnodes;
- a feature layer containing n featurenodes;
- n feature SVMs trained to extract features from the input;

- a single SVM to estimate the output using the extracted features which we will call the main SVM;
- an output layer consisting of 1 node.

Figure 1 presents a graphical overview of the DSVM.

The input layer receives an input pattern $\mathbf{x}_i$ with $1 \leq i \leq l$, which are processed by the feature SVMs. The resulting values $f(\mathbf{x}_k)_a$ with $1 \leq k \leq l$ and $1 \leq a \leq n$ will be stored in the feature layer. The main SVM will then use the vector $f(\mathbf{x}_k)$ as its input to determine the final output.

3.1.1 Kernels

One thing to note about the Deep Support Vector Machine is that all different layers of SVMs may make use of different types of kernels. For example, while the feature layer makes use of a gaussian kernel, the main SVM might use a linear one. It is even possible to use different type of kernels within one layer. One could use a feature layer which uses both linear and gaussian kernels to extract different features. For this article we built a DSVM using only gaussian kernels.

3.2 Training phase

Now we will explain the training procedure of the DSVM. In order to do this we will first make some small adjustments to the dual problem and regression estimate as formulated in section 2.

Since our main SVM no longer takes $\mathbf{x}_i$ as its input, but now uses $f(\mathbf{x}_k)$ the new dual problem and regression estimate are now defined as:

$$\begin{aligned}\min_{f(\mathbf{x}_i)}\max_{\alpha,\alpha^*} W(f(\mathbf{x}_i,\alpha^{(*)}) =& \\ -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^*-\alpha_i)(\alpha_j^*-\alpha_j)K(f(\mathbf{x}_i),f(\mathbf{x}_j))& \qquad (3.1)\\ -\epsilon\sum_{i=1}^{l}(\alpha_i^*+\alpha_i)+\sum_{i=1}^{l}y_i(\alpha_i^*-\alpha_i)&\end{aligned}$$

subject to

$$0 \leq a_i^{(*)} \leq C, 1 \leq i \leq n \qquad (3.2)$$

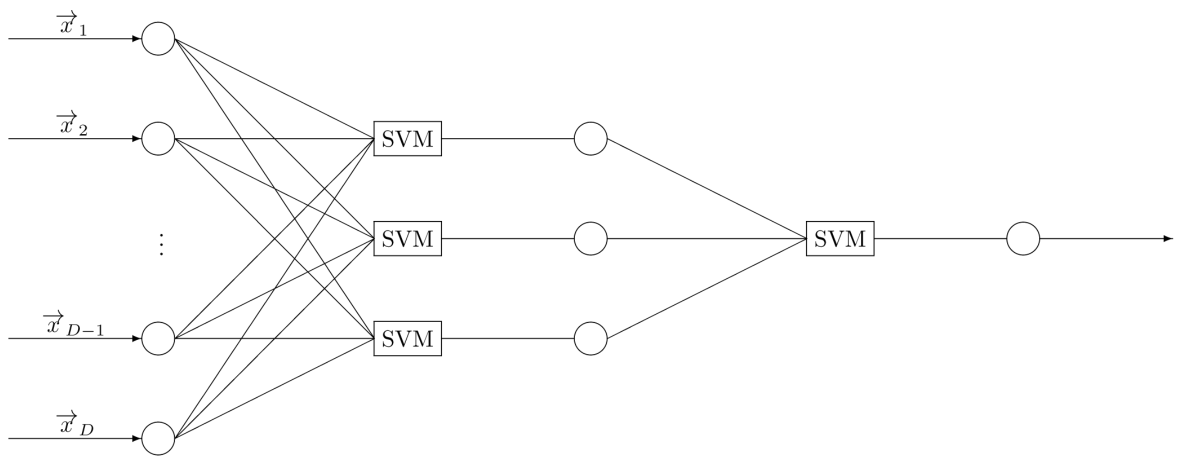
$$\sum_{i=1}^{n}(\alpha_i^*-\alpha_i) = 0 \qquad (3.3)$$

and

$$f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_i^*-\alpha_i)K(f(\mathbf{x}_i),f(\mathbf{x}))+b \qquad (3.4)$$

Besides this change there is also another difference. Instead of simply wanting to find the $\alpha^{(*)}$ which maximizes (3.1). We also want the feature SVMs to find the correct featurevectors $f(\mathbf{x}_i)$ that minimize equation (3.1).

Figure 1: A graphical overview of the DSVM



The training phase consists of the following steps, which will each be further examined individually.

1. intialize the feature layer with pseudorandom values;
2. train the main SVM with the current feature layer;
3. propagate the results of the main SVM back to its features;
4. train the feature layer SVMs;
5. repeat from step 2;

3.2.1 Initialize feature layer

In order to be able to run the DSVM we will have to initialize the feature SVMs. We do this in a pseudorandom way, depending on y_i.

First we initialize the values of $\alpha_i^{(*)}$. We do this by initializing them as usual, and then multiplying them with a pseudorandom number. This number is determined by two parameters, ι_1 and ι_2. We then multiply our intial values of $\alpha_i^{(*)}$ with $\iota_1 + \iota_2 R$, where R is a random number, $0 < R < 1$. This gives us:

$$\alpha_i^{(*)} = (\iota_1 + \iota_2 R) \tag{3.5}$$

After the SVMs have been initialized with their values of $\alpha_i^{(*)}$ they are trained with the output values y_i. In this way we obtain some sensible features which actually have something to do with y_i, and are not all the same, since they are pseudorandom.

3.2.2 Train the main SVM

The main SVM is trained with the current values of $f(\mathbf{x}_i)$. This works just like training a regular SVM, except that we do not use the input pattern provided by the data directly, but use the extracted features.

3.2.3 Propagate results back to features

In order to make sure that the feature SVMs extract the correct features from the input patterns they will need a little help from the main SVM. We do this by propagating a part of the results from the main SVM back to the feature SVMs. We do this by decreasing the output of $f(\mathbf{x}_i)_a$ with the derivative of W from the main SVM with respect to $f(\mathbf{x}_i)_a$.

$$f(\mathbf{x}_i)_a^{new} = f(\mathbf{x}_i)_a - \beta \frac{\delta W}{\delta f(\mathbf{x}_i)_a} \tag{3.6}$$

where β is a predefined constant determining the learning rate of the feature SVMs.

As we know, W depends on the type of kernel we use in the main SVM, hence the backpropagation depends on the same kernel. This gives us the following formula for backpropagation:

$$\frac{\delta W}{\delta f(\mathbf{x}_i)_a} = (\alpha_i^* - \alpha_i) \sum_{j=1}^{l} (\alpha_j^* - \alpha_j) \frac{\delta K(f(\mathbf{x}_i), f(\mathbf{x}))}{\delta f(\mathbf{x}_i)_a} \tag{3.7}$$

For the gaussian formula we use we have

$$\frac{\delta K(f(\mathbf{x}_i), f(\mathbf{x}))}{\delta f(\mathbf{x}_i)_a} = - 2 \frac{f(\mathbf{x}_i)_a - f(\mathbf{x}_j)_a}{\sigma_m} K(f(\mathbf{x}_i), f(\mathbf{x}_j)) \tag{3.8}$$

giving us the backpropagation formula:

$$\frac{\delta W}{\delta f(\mathbf{x}_i)_a} = (\alpha_i^* - \alpha_i) \sum_{j=1}^{l} (\alpha_j^* - \alpha_j) \tag{3.9}$$

$$\frac{f(\mathbf{x}_i)_a - f(\mathbf{x}_j)_a}{\sigma_m} K(f(\mathbf{x}_i), f(\mathbf{x}_j)) \tag{3.10}$$

3.2.4 Train the feature layer SVMs

After the result is backpropagated to the feature layer we create a new dataset for each SVM in the feature layer, containing one of the features of all instances of $\mathbf{x}_i$. We then train the feature SVMs on this database, mapping $\mathbf{x}_i$ to $f(\mathbf{x}_i)_a^{new}$. Then we propagate the input vectors back, to get new feature outputs, which will replace the values of $f(\mathbf{x}_i)_a$. The number of features that is used (and thus, the number of feature SVMs) is given by a parameter which can be set.

3.2.5 Repetition

After having extracted the new features the process will be repeated from step 2 for a fixed number of times. This number of times can be set as a parameter for the system. In the last repetition steps three and four will be ignored, as the new features will no longer be trained upon after the last time the main SVM is trained.

3.3 Testing phase

When processing previously unseen cases the DSVM first extracts the features from the input pattern. When the features are extracted they are used as input for the main SVM which computes the final output value for the unseen case.

4 Experimental Setup

Now we will describe how we tested the DSVM. We will first discuss the datasets we used, followed by our parameter selection. Finally we will show which statistics we used to determine the success of the DSVM.

4.1 Datasets

In order to test the DSVM, its results were compared to the results of a regular gradient ascent SVM. To make this comparison both programs were ran with eight datasets, which were obtained from [5]. These were the following datasets: 'Stock' (04), 'Boston Housing' (16), 'Diabetes' (17), 'Machine-CPU' (18), 'Wisconsin Breast Cancer' (19), 'Baseball' (22), 'Mortgage' (28) and 'Concrete Strength' (29) The numbers between brackets are the numbers the datasets have been given in [5].

Baseball Contains the salaries of Major League Baseball players in 1992. The dataset has 337 instances with 6 real-valued features.

Boston Housing This dataset contains information on the house pricing in Boston. It contains 461 instances, with 4 real-valued features.

Concrete Strength A dataset used for predicting concrete's compressive strength. It contains 72 instances, having 5 real-valued attributes.

Diabetes A dataset concerning the dependence of the level of serum C-peptide. It contains 43 instances with 2 features. Both features are real-valued.

Machine-CPU Predicting relative CPU performance, using 6 real-values features. It contains 188 instances.

Mortgage Predicts the 30-year mortgage rate, using 6 real valued instances. The dataset contains 1049 instances.

Stock This dataset is used for predicting daily stock prices. It contains 950 instances, with 4 real valued features.

Wisconsin Breast Cancer Predicting the time before Breast Cancer recurs in patients. The dataset uses 6 real-values features and contains 152 instances.

4.2 Experiments

In order to compare the regular Support Vector Machine with the Deep Support Vector Machine with only gaussian kernels, all datasets were presented 1000 or 4000 (depending on the size of the dataset) times to all three methods. From these runs the mean squared error and its standard error are computed and compared.

For each combination of dataset and method a new set of optimal learning parameters has to be found in order to find the optimal results. In order to find this set of parameters, we used the Particle Swarm Optimization (PSO) method [6]. We first conducted a global PSO search, followed by a fine-tuning search (also using PSO). The parameters which came out the best of the latter search were finally used.

4.3 Statistics

In order to give an objective view on the results of the DSVM its results will be compared properly to the results of the gradient ascent SVM. To make a proper comparison for every dataset, a student t-test is conducted between the results of each method, with $\alpha = 0.05$.

5 Experimental Results

In table 1 the Mean Squared Error (MSE) and standard error for each dataset of both the regular SVM and DSVM are shown. The results from [5] are

Table 1: Mean Squared Errors for the SVM and DSVM for each dataset

Dataset	#inst.	#feat.	N	SVM results	DSVM results	results from [6]
Baseball	337	6	4000	0.02413 ± 0.00011	**0.02294 ± 0.00010**	0.02588
Boston Housing	461	4	1000	0.00684 ± 0.000095	*0.00656 ± 0.000094*	0.007861
Concrete Strength	72	5	4000	0.00706 ± 0.000070	**0.00621 ± 0.000054**	0.008509
Diabetes	43	2	4000	0.02719 ± 0.000263	**0.02327 ± 0.000219**	0.025154
Machine-CPU	188	6	1000	0.00805 ± 0.000181	**0.00638 ± 0.000123**	0.007766
Mortgage	1049	6	1000	0.000080 ± 0.000001	0.000080 ± 0.000001	0.000081
Stock	950	5	1000	0.00086 ± 0.000006	**0.00076 ± 0.000005**	0.002385
Breast Cancer	152	6	4000	0.06947 ± 0.000297	0.06910 ± 0.000295	**0.068615**

shown as well, however, we have no standard error for these values. The best scores are printed in boldface. When these differences are rather small, they are printed in italics.

As can be seen, the DSVM scores better than the gradient descent SVM on all datasets except for the Mortgage set, which is the largest. However, since the error for this dataset is very small this might be caused by the rounding of the MSE and SE.

The results for the statistical analysis that was conducted can be found in table 2. As expected, the results on each dataset were significant (with $\alpha = 0.05$), except for the Mortgage dataset.

6 Discussion

Now that we have the results, it is time to draw some conclusions. First we will answer our research question and discuss some of the things that could have been done different during the experiment. After that some of the possible follow-up work will be discussed.

Table 2: Statistical results for each dataset

Dataset	N	P(X)
Baseball	4000	$< 2.2e-16$
Boston Housing	1000	0.01861
Concrete Strength	4000	$< 2.2e-16$
Diabetes	4000	$< 2.2e-16$
Machine-CPU	1000	$1.586e-15$
Mortgage	4000	0.5
Stock	1000	$< 2.2e-16$
Wisc. Breast Cancer	4000	0.0283

6.1 Conclusion

The question we posed at the introduction of this paper was whether the DSVM would outperform the regular SVM in regression problems. As we have shown above, on almost all of our datasets the DSVM had a better performance than the regular SVM. The only exception is the Mortgage dataset, which is the largest. This could be because feature extraction is not necessary when enough training examples are presented, or just because of rounding, since the mean squared error is really small.

Despite not scoring significantly better on one of the datasets it is safe to presume that the DSVM outperforms the regular SVM on most datasets. The number of instances does not really seem to influence performance, since we have datasets with a number of instances ranging from less than 50 to around 1000 instances. However, since the largest dataset was insignificant, it might be that from some number of instances the difference may diminish. The rest of our datasets do not really support this theory though, but this might be further looked into.

All of our datasets concerned problems using only 2 to 6 features. However, datasets exist using much more features. We can not say whether the DSVM will also perform better on problems using a high number of features.

6.2 Future Work

For this research we just used a quite simple form of the DSVM. However, the DSVM could be extended in several ways. Firstly, we have limited ourselves to regression problems. However, the DSVM could also be used for classification problems. To continue this thought, we can even extend these classification problems to problems concerning multiple classes and autoencoding.

Secondly we only used a single feature layer. However, in theory we could concatenate much more feature layers, extracting sub-features. We do not know whether multiple layers yield better results. Further research might be able to be more conclusive on this.

Finally, the number of training cycles is fixed, determined upfront. However, other methods could be used to determine this amount. For example, the training phase could continue until the final output created by the main SVM changes an insignificant amount, or until the features which are extracted are optimized.

Since this was only a first implementation of the DSVM it was used for solving simple problems. The next step is finding some more challenging problems, like object or facial recognition. Also, the theory of the DSVM has to be studied more thoroughly, as it has merely begun its development as a successful machine learning algorithm.

References

[1] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, New York, NY, USA, 2004. ACM.

[2] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[3] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[4] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines.* Cambridge University Press, 2000.

[5] M. Graczyk, T. Lasota, Z. Telec, and B. Trawiski. Nonparametric statistical analysis of machine learning algorithms for regression problems. In R. Setchi, I. Jordanov, R. Howlett, and L. 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.

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

[7] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 775–782, New York, NY, USA, 2007. ACM.

[8] A. Rakotomamonjy, F. R. Bach, S. Canu, and Y. Grandvalet. Simplemkl. *Journal of Machine Learning Research*, 9(11):2491 – 2521, 2008.

[9] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[10] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.