A new approach to multilayer SVMs

Joan Acero-Pousa and Lluís A. Belanche-Muñoz

School of Computer Science
Jordi Girona, 1-3, 08034, Barcelona - Catalonia, Spain

Abstract. Despite the traditional high performance of Support Vector Machines (SVMs) in classification and regression tasks, modern data loads have introduced new efficiency challenges, rendering SVMs incapable of handling non-linear problems when the dataset size is large. On the other hand, neural architectures have shown excellent results when dealing with complex patterns in data. By leveraging kernel approximation techniques and linear optimizations, this work introduces a multilayer SVM architecture, presenting competitive performance against classical SVMs.

1 Introduction

Support Vector Machines (SVMs) have been a cornerstone in machine learning, particularly for classification and regression tasks, due to a solid mathematical foundation, use of the kernel trick and built-in regularisation [1]. In the era of big data, SVMs have become less favourable in specific scenarios due to their use of kernel matrices, resulting in prohibitive computational costs when dealing with large datasets –$O(n^3)$ time and $O(n^2)$ space, being n the training sample size. The exploration of multilayer SVMs, extending their capabilities by introducing multiple layers of decision boundaries seems a natural development, albeit the original SVM was not designed with this evolution in mind, and research in this area is relatively unexplored. Key reasons are that training multilayer SVMs could be even more computationally expensive than the classical single layer version, and effective optimization techniques are essential. Also, as the model complexity increases, there is a risk of overfitting the training data.

The earliest contribution [2] leverages the high-dimensional feature space of one SVM as input for subsequent "layers" but, despite the title, it is specifically designed to solve n-th order ODEs; it does not address scalability for large datasets or integrate modern kernel approximation techniques. In fact, [3] is the only meaningful reference. The training process employs a min-max optimisation strategy: the hidden-layer SVMs work to minimise the dual-objective function of the output-layer SVM, while the output-layer SVM seeks to maximise its dual-objective function. In spite of its innovative approach, the proposal still computes kernel matrices and uses backpropagation, hence the inefficiency issue when dealing with moderate to large datasets remains basically unresolved. Finally, the DHNKN [4] combines Random Fourier Features (RFFs) with perceptron layers; it requires gradient descent to optimize a large number of parameters, inheriting all the troubles with backpropagation (training complexity, vanishing gradients, local minima, etc).

This paper introduces a more workable proposal, based in the combination of two essential ingredients: the avoidance of kernel matrices via *approximation* techniques and the fact of solving only *linear* optimizations.

2 Multilayer SVMs

We first focus on a setting with a single hidden layer: in this setup, the ML-SVM consists of an input "layer", a hidden layer and an output layer –see Fig. 1. Consider an input matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, where $\mathbf{x}_i \in \mathbb{R}^d \quad i = 1, \ldots, n$. The number of features in the original space is d, and n is the number of observations. The hidden layer transforms the input vector using a Gaussian RBF kernel approximation based on RFFs [5]:

$$\mathbf{h}^1 = z^1(\mathbf{X}) = \sqrt{\frac{2}{D}} \left[\cos(\omega_1^T \mathbf{X} + b_1), \ldots, \cos(\omega_D^T \mathbf{X} + b_D)\right]^T \tag{1}$$

Here, D is the number of RFFs used to approximate the kernel function, and hence the dimension of the transformed data is $\mathbf{h}^1 \in \mathbb{R}^{n \times D}$. Moreover, $w_i \in \mathbb{R}$ $i = 1, \ldots, D$ are i.i.d. observations from the Fourier transform of a specific shift-invariant kernel function, and $b_i \in \mathbb{R} \quad i = 1, \ldots, D$ are i.i.d. observations from the Uniform $[0, 2\pi]$ distribution [5]. Next, $\mathbf{h}^1 = (\mathbf{x}_1^1, \ldots, \mathbf{x}_n^1)$ is used as the input for a linear SVM, which computes the output of the hidden layer. Note that $\mathbf{x}_i^1 \in \mathbb{R}^D \quad i = 1, \ldots, n$ are the input observations after the RFF transformation.

$$\mathbf{SVM}^1 = \mathbf{w}_1^T z^1(\mathbf{X}) + b_1 = \mathbf{w}_1^T \mathbf{h}^1 + b_1 \tag{2}$$

Where $\mathbf{w}_1 \in \mathbb{R}^D$ and $b_1 \in \mathbb{R}$ are the weight vector and the bias of the SVM in the hidden layer. The dimension of the output of the hidden layer depends on the problem. In a classification setting with P different classes, each observation has as an output a vector of dimension P, so $\mathbf{SVM}^1 \in \mathbb{R}^{n \times P}$. Instead, in a regression setting, each observation has a single value as output, being $\mathbf{SVM}^1 \in \mathbb{R}^n$. Next, $\mathbf{SVM}^1$ is used as the input of the output layer, the outcome $\boldsymbol{M}$ of the entire ML-SVM. The output layer consists of a linear SVM. In a binary classification problem where $\mathbf{SVM}^1 \in \mathbb{R}^{n \times 2}$, $\boldsymbol{M} = \text{sgn}(\mathbf{w}_2^T \mathbf{SVM}^1 + b_2)$, where $\mathbf{w}_2 \in \mathbb{R}^2$ and $b_2 \in \mathbb{R}$ are the weight vector and the bias of $\boldsymbol{M} \in \mathbb{R}^n$.

2.1 General Case

The architecture introduced in the previous section can be expanded to an unlimited number of hidden layers, an ML-SVM. The input of the first hidden layer is $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of observations and d the number of features in the original feature space. The input of the hidden layer k is the outcome of the previous layer, $\mathbf{SVM}^{k-1}$. In a classification problem, the input of the hidden layer k is $\mathbf{SVM}^{k-1} \in \mathbb{R}^{n \times P}$, where P is the total number of classes. Instead, the input of the hidden layer k in a regression problem is $\mathbf{SVM}^{k-1} \in \mathbb{R}^n$, as in Fig. 2. The k-th hidden layer follows the procedure:

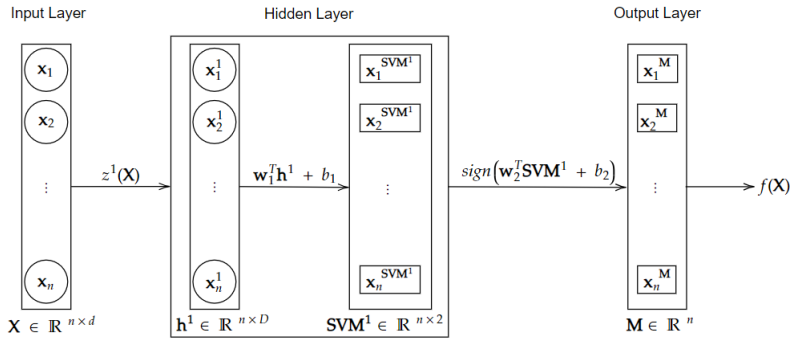

Fig. 1: Single hidden layer ML-SVM for a binary classification task. The number of observations is n, and the original feature space is d-dimensional. The number of RFFs used to approximate the kernel function is D.

$$\mathbf{h}^k = z^k(\mathbf{SVM}^{k-1}) = \sqrt{\frac{2}{D}}\left[\cos(\omega_1^T\mathbf{SVM}^{k-1} + b_1), \ldots, \cos(\omega_D^T\mathbf{SVM}^{k-1} + b_D)\right]^T$$

$$\mathbf{SVM}^k = \mathbf{w}_k^T\mathbf{h}^k + b_k$$

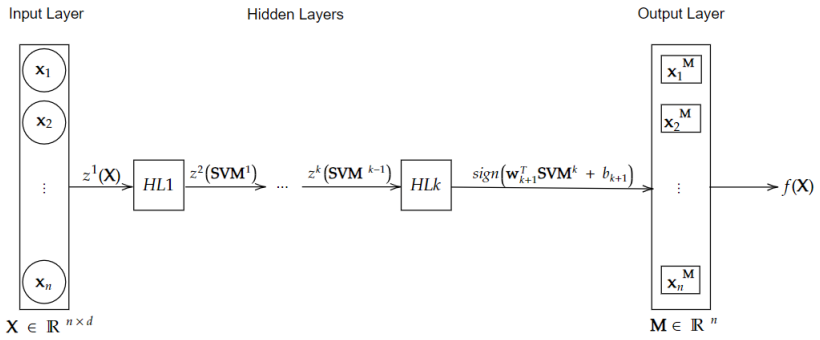

Fig. 2: An ML-SVM with k hidden layers. Each hidden layer is represented as $HLi, \quad i = 1, \ldots, k$. Refer to Fig. 1 to see the content inside hidden layer boxes.

2.2 Training Algorithm

The training algorithm of the ML-SVM focuses on optimising *linear* SVMs in the architecture. The training algorithm optimises the values of the Lagrange multipliers $\boldsymbol{\alpha}_i$ and bias b_i for each SVM_i dual optimisation problem. It has as hyperparameters: k**:** Number of hidden layers, D_i: Number of RFFs in layer i, and σ_i**:** stdev of the distribution from which RFFs are sampled, where $i = 1, \ldots, k$. Note that the RFF weights $\{\hat{\boldsymbol{\omega}}_i\}_{i=1}^k$ and biases $\{\hat{\mathbf{b}}_i\}_{i=1}^k$ that were sampled during

the training stage must be stored. This is crucial to reproduce the transformations in which SVMs of each layer have been optimised –see Algorithm 1. The overall time complexity of the training phase is $O(k \times n \times D(D+n))$, which reduces to $O(k \times n^2 \times D)$ when $n > d$.

Algorithm 1 Training Algorithm for an ML-SVM binary classifier

Input: Training data $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, where $\mathbf{x}_i \in \mathbb{R}^d$, labels $\mathbf{y} = (y_1, \ldots, y_n)$, number of RFFs $\{D_j\}_{j=1}^{k}$ and kernel parameter$\{\sigma_j\}_{j=1}^{k}$ for each layer, number of layers k
Output: Trained weights $\{\mathbf{w}_j\}_{j=1}^{k+1}$, biases $\{b_j\}_{j=1}^{k+1}$, RFF weights $\{\hat{\omega}_i^j\}_{i=1,j=1}^{D,k}$, RFF biases $\{\hat{b}_i^j\}_{i=1,j=1}^{D,k}$.
for $j = 1$ to k **do**
    Sample $\hat{\omega}_i^j \sim \mathcal{N}(0, \sigma_j \mathbf{I})$ for $i = 1, \ldots, D_j$
    Sample $\hat{b}_i^j \sim \mathcal{U}[0, 2\pi]$ for $i = 1, \ldots, D_j$
    if $j = 1$ **then**
        $\mathbf{h}^j = z^j(\mathbf{X})$
    else
        $\mathbf{h}^j = z^j(\mathbf{SVM}^{j-1})$
    end if
    Optimise the j-th linear SVM parameters $\mathbf{w}_j$ and b_j using $(\mathbf{h}^j, \mathbf{y})$
    $\mathbf{SVM}^j = \mathbf{w}_j^T \mathbf{h}^j + b_j$
end for
Optimise the final linear SVM parameters $\mathbf{w}_{k+1}$ and b_{k+1} using $(\mathbf{SVM}^k, \mathbf{y})$
Return $\{\mathbf{w}_j\}_{j=1}^{k+1}, \{b_j\}_{j=1}^{k+1}, \{\hat{\omega}_i^j\}_{i=1,j=1}^{D,k}, \{\hat{b}_i^j\}_{i=1,j=1}^{D,k}$

3 Experimental Work

3.1 Testing large-scale capabilities

Among the many experiments performed to study the empirical behaviour of the new proposal, due to space reasons, we show just some representative ones. We use the Supersymmetry (SUSY) dataset [6] to evaluate the ability of the ML-SVM to manage large datasets effectively. The task is to classify a signal process as one that generates supersymmetric particles or a background process that does not (5,000,000 observations, 18 features, 2 classes) – see Table 1. The ML-SVM models train faster than the SVM with RBF kernel while generally obtaining similar test accuracy when the number of observations is large (10^5, in this case). With an equivalent number of layers, ML-SVMs are also faster than MLPs; with such large size, each method obtains very close results across the 10 executions, causing no significant stdev (not shown).

Model	Obs.	Train error	Test error	Time (s)
ML-SVM 1 layer	10^5	0.19 ± 0.00	0.20 ± 0.00	53.78 ± 4.64
ML-SVM 2 layers	10^5	0.20 ± 0.00	0.20 ± 0.00	66.95 ± 4.95
ML-SVM 3 layers	10^5	0.20 ± 0.00	0.20 ± 0.00	78.18 ± 6.47
MLP 1 layer	10^5	0.19 ± 0.00	0.20 ± 0.00	60.49 ± 8.93
MLP 2 layers	10^5	0.16 ± 0.00	0.22 ± 0.00	191.54 ± 19.29
MLP 3 layers	10^5	0.11 ± 0.00	0.25 ± 0.00	291.72 ± 13.00
SVM-RBF	10^5	0.20 ± 0.00	0.20 ± 0.00	385.25 ± 3.12
ML-SVM 1 layer	10^6	0.20 ± 0.00	0.20 ± 0.00	981.53 ± 120.00
ML-SVM 2 layers	10^6	0.20 ± 0.00	0.20 ± 0.00	1089.28 ± 112.29
ML-SVM 3 layers	10^6	0.20 ± 0.00	0.20 ± 0.00	1179.36 ± 103.22
MLP 1 layer	10^6	0.20 ± 0.00	0.20 ± 0.00	217.53 ± 57.19
MLP 2 layers	10^6	0.19 ± 0.00	0.20 ± 0.00	974.18 ± 208.25
MLP 3 layers	10^6	0.19 ± 0.00	0.20 ± 0.00	2859.64 ± 639.77
SVM-RBF	10^6	N/P	N/P	N/P

Table 1: Performance comparison of various models using 10^5 and 10^6 observations of the SUSY problem to train alongside CPU time in seconds; N/P stands for "not possible" due to excessive sample size.

3.2 Testing Multi-Layer Efectiveness

In this experiment, ML-SVM models are trained on several classification datasets [6], summarised in Table 2. In all datasets, adding layers ended up, at some point, improving the accuracy of the model. A similar trend is observed in all cases, where the test error gets reduced by adding layers. Often, the major increase in accuracy is between the use of 1 and 2 hidden layers, but in some cases, the ML-SVM still gets more accuracy while adding layers. All experiments were conducted with results averaged over 10 training (90% of the data) and test (10%) to provide mean values and stdevs.

Dataset	N. of observations	N. of features	N. of classes
Magic	19,000	10	2
Cover Type (subset)	10,000	54	7
Spam Base	4,600	57	2
Breast Cancer	699	9	6
Ionosphere	351	34	2
E. Coli	336	7	8
Glass	214	9	7

Table 2: Summary of the dataset information. Datasets from [6].

In Table 3 we summarise the results obtained across all datasets. For every number of layers, results of the ML-SVMs are the best values among the runs, all with the RBF kernel approximation; it should be mentioned that the

σ and C valueswere optimised for the first layer only. The MLP in our study is constructed with three layers, each comprising 200 neurons.

Dataset	SVM	MLP	**ML-SVM**	W&S [3]
Glass	0.29 ± 0.30	0.18 ± 0.20	0.18 ± 0.01	0.26 ± 0.30
E. Coli	0.03 ± 0.11	0.03 ± 0.00	0.00 ± 0.00	0.13 ± 0.20
Breast Cancer	0.01 ± 0.00	0.03 ± 0.00	0.008 ± 0.01	0.03 ± 0.10
Ionosphere	0.11 ± 0.00	0.06 ± 0.01	0.05 ± 0.05	0.05 ± 0.10
Magic	0.16 ± 0.00	0.17 ± 0.00	0.15 ± 0.00	–
Spam Base	0.31 ± 0.00	0.08 ± 0.01	0.15 ± 0.01	–
Cover Type (subset)	0.31 ± 0.00	0.28 ± 0.00	0.21 ± 0.01	–

Table 3: Test error of several models in the datasets; – refers to tasks not studied in the reference.

4 Conclusions

The exponential growth of data availability in recent years has introduced new challenges for many learning algorithms. Traditional SVMs must often struggle with complex, high-dimensional, large datasets, armed only with the choice of an adequate kernel function. The ML-SVM proposed in this paper starts with a single hidden layer configuration, where the input data is transformed before being provided to a linear SVM. This process can be iteratively extended to multiple hidden layers, total output being computed by another linear SVM. Besides the weights and biases of the optimised linear SVMs, the algorithm stores sampled RFF weights and biases to ensure reproducibility in transformations during the evaluation phase. Our experiments reveal that adding layers to the ML-SVM generally produces positive effects. However, the correlation between adding layers and achieving higher test accuracy is not always met. For instance, the optimal accuracy is achieved with an addition of two layers rather than just one in some cases. Our future work is oriented to automatically adjust all ML-SVM parameters to each specific task, bearing in mind a trade-off between feasible computation time and needed complexity.

References

[1] Vladimir Naumovich Vapnik, Vlamimir Vapnik, et al. Statistical learning theory. *Statistical learning theory*, 1998.

[2] You-xi Wu, Lei Guo, Yan Li, Xue-qin Shen, and Wei-li Yan. Multi-layer support vector machine and its application. In *2006 International Conference on Machine Learning and Cybernetics*, pages 3627–3631. IEEE, 2006.

[3] Marco A Wiering and Lambert RB Schomaker. Multi-layer support vector machines. *Regularization, optimization, kernels, and support vector machines*, pages 457–475, 2014.

[4] Siamak Mehrkanoon and Johan AK Suykens. Deep hybrid neural-kernel networks using random fourier features. *Neurocomputing*, 298:46–54, 2018.

[5] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[6] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The UCI ML repository, 2024.