Deep Support Vector Classification and Regression

David Díaz-Vico[1,2(✉)], Jesús Prada[1], Adil Omari[3], and José R. Dorronsoro[1,2]

[1] Dpto. Ing. Informática, Universidad Autónoma de Madrid, Madrid, Spain
david.diazv@estudiante.uam.es
[2] Instituto de Ingeniería del Conocimiento,
Universidad Autónoma de Madrid, Madrid, Spain
[3] Signal Theory and Communications Department,
Universidad Carlos III, Madrid, Spain

Abstract. Support Vector Machines, SVM, are one of the most popular machine learning models for supervised problems and have proved to achieve great performance in a wide broad of predicting tasks. However, they can suffer from scalability issues when working with large sample sizes, a common situation in the big data era. On the other hand, Deep Neural Networks (DNNs) can handle large datasets with greater ease and in this paper we propose Deep SVM models that combine the highly non-linear feature processing of DNNs with SVM loss functions. As we will show, these models can achieve performances similar to those of standard SVM while having a greater sample scalability.

1 Introduction

Support Vector Machines (SVM; [17]) are one of the state of the art methods for supervised classification and regression and, as such, widely used. One key fact for this is their ability to work implicitly with kernels such as the Gaussian one, which map the initial features to a possibly infinite dimensional reproducing kernel Hilbert space. But, on the other hand, this capability also hinders their ability to cope with large datasets, as the handling of the kernel matrix becomes too costly or, sometimes, unfeasible and, even if a model is finally built, the so called "kernelization curse", i.e., the fact that the number of Support Vectors grows linearly with sample size, implies that the model may be too costly in memory or time to exploit.

Many proposals have appeared in the literature to overcome these problems, usually for Support Vector classification (SVC) problems. Among them we can mention the incremental learning of SVMs [1], ensemble learning of SVMs [6] or cutting planes [15] but, nevertheless, it can be said that, unless substantial hardware resources are committed, current kernel SVM training methods are not competitive for datasets with more than about 100,000 patterns.

These problems are largely mitigated when working with linear SVMs [21] for which efficient algorithms, such as Pegasos [19] or dual coordinate descent [14], exist. However, efficient linear SVM models can only be expected when the original features have very large dimension so that projections are no longer needed. When sample dimension is just moderate, linear SVM models are usually less powerful that their Gaussian counterparts.

In principle, the main difficulty when working with SVMs is the non differentiable nature of the SVC hinge loss that, at first sight, forces a dual probem to be solved. However, its non differentiability is rather mild and subgradient descent is used, for instance, in the Pegasos algorithm. In fact, the SVM losses have the same non-differentiable behavior of the ReLU loss [11] routinely used in Deep Neural Networks (DNN) to compute gradients by backpropagation and which are easily handled automatically by DNN backends such as TensorFlow [12]. In fact, the hinge loss is already predefined in the Keras wrapper [5] for TensorFlow. Also, and although not predefined in Keras, the ϵ-insensitive loss used in Support Vector regression (SVR) has the same nondifferentiable behavior.

This suggests to consider an alternative to non linear SVMs by means of DNNs for which the standard cross entropy for classification or squared error for regression are replaced by the hinge and ϵ-insensitive losses respectively. This approach has been already applied in [20], where it is compared with standard softmax DNNs on classification problems, and by [22] in speech recognition. Here we will also extend it to Deep Support Vector Regression models and we will compare these Deep SV Classifiers and Regressors with their standard Gaussian SVC and SVR counterparts in a number of medium to large classification and regression problems.

The paper is organized as follows. In Sect. 2 we will briefly review standard kernel based SVC and SVR models while their deep counterparts are explained in Sect. 3. Numerical experiments are presented in Sect. 4 and the paper ends with a discussion, conclusions and pointers to further work.

2 Support Vector Machines

Given a sample $S = \{(x^p, y^p), p = 1, \dots, N, y^p = \pm 1\}$, if we have a linear model $\xi = f(x) = w \cdot x + b$ and h denotes the hinge loss

$$h(y, f(x)) = \max\{0, 1 - yf(x)\} = \max\{0, 1 - y\xi\} = h(y, \xi),$$

the optimization problem to be solved for a linear SVC is

$$\begin{aligned} &\arg\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_1^N h(y^p, w \cdot x^p + b) \\ &\equiv \arg\min_{w,b} \left(\frac{1}{2CN}\|w\|^2 + \frac{1}{N}\sum_1^N h(y^p, w \cdot x^p + b) \right) \\ &\equiv \arg\min_{w,b} \left(\frac{1}{N}\sum_1^N h(y^p, w \cdot x^p + b) + \alpha\|w\|^2 \right) \end{aligned} \tag{1}$$

with $\alpha = \frac{1}{2CN}$. In general, SVCs are usually built solving the dual problem of (1), namely

$$\max_{\alpha_i} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{2}$$

subject to $0 \leq \alpha_i \leq C,\ i = 1, \ldots, N$, $\sum_{i=1}^{N} \alpha_i y_i = 0$.

Notice that problems (1) and (2) only involve dot products; this is also the case of the final SVM model. As a consequence, all can be written as just done but replacing x with a kernel-related projection $\Phi(x)$. The map Φ can be implicit, as is the case with standard kernel SVCs, where the dot product only enters through some kernel, or it can be an explicit one, as it will be the case here, where $\Phi(x) = F(x, \mathcal{W}_h)$ will be the vector of last hidden layer values of a MLP with input to last hidden layer weighs $\mathcal{W}_h$. Moreover, if the kernel representation $\Phi(x^p)$ is known explicitly, the primal problem (1) can be directly solved, as done by the Pegasos algorithm [19] (see also [3], where the primal problem is solved for the squared hinge loss). We will also exploit the explicit kernel knowledge in the Deep SVC approach that we describe in the next section.

Given now a sample $S = \{(x^p, y^p), p = 1, \ldots, N\}$ with the y^p here being numerical targets, and a linear model $\xi = f(x) = w \cdot x + b$, the ϵ-insensitive loss, $\epsilon > 0$, would now be

$$\ell_\epsilon(y, f(x)) = \ell_\epsilon(y, \xi) = \max\{0,\ |\xi| - \epsilon\}.$$

The optimization problem to be solved now for linear SV Regression (SVR) is

$$\arg\min_{w,b} \frac{1}{N} \sum_{1}^{N} \ell_\epsilon(y^p, w \cdot x^p + b) + \alpha \|w\|^2. \tag{3}$$

As it was the case in SVC, the ϵ-insensitive loss is convex but only piecewise differentiable and because of this, here again, the task solved in practice is the following dual problem derived through the Lagrangian formalism

$$\max_{\alpha_i, \alpha_i^*} \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i) - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) x_i \cdot x_j - \epsilon \sum_{i=1}^{N} (\alpha_i^* + \alpha_i) \tag{4}$$

subject now to $0 \leq \alpha_i, \alpha_i^* \leq \frac{1}{2\alpha N}, i = 1, \ldots, N$, $\sum_{i=1}^{N} (\alpha_i^* - \alpha_i) = 0$.

The kernel trick can also be applied here and, if it is known explicitly, gradient or subgradient descent can be used. We describe next how to exploit this for Deep SVR.

3 Deep Support Vector Machines

3.1 Deep Learning

Artificial Neural Networks (ANNs) can be seen as an extension of the classical linear and logistic regression models. They can generate an arbitrarily good latent representation of the data [8,13] that can be efficiently used to build powerful models. Starting with the multi layer perceptrons (MLPs), whose basic theory was already well established in the 80s, and the backpropagation algorithm for gradient computation during ANN training, they can be considered as the first example of modern machine learning algorithms that could be applied to both regression and classification problems with minimal conceptual variations. However, technical difficulties, essentially due to knowledge gaps about their training plus the lack of computing power at the time, led to their relative decline in the late 90's and the rise of competing methods, particularly SVMs, for classification and regression.

But about 2010, the wide availability of powerful computing facilities, advances on the theoretical underpinnings of MLPs, many refinement of their training procedures and a better understanding of the difficulties related to many layered architectures [10] have produced a spectacular expansion under the deep neural network (DNN) paradigm. To all this we can add the appearance of several development frameworks such as TensorFlow [12], CNTK [18], MXNET [4] or Torch [7], as well as wrappers for them such as Keras [5], that have allowed the practitioners to experiment with different architectures, non-differentiable activations and, even, non-differentiable loss functions. Last, but not least, the iterative nature of backpropagation and its linear cost growth with respect to sample size imply that DNNs are much less affected from the bad scalability of kernel methods, and can be applied with relative ease to datasets with hundreds of thousands of patterns. As mentioned, mildly non-differentiable functions such as ReLUs or the hinge loss, can be handled within the DNN framework and are relatively simple to incorporate into DNN tools. We discuss next how the hinge and ϵ-insensitive loss can be introduced in a DNN set up.

3.2 Deep SVC and SVR

Consider a more or less standard DNN architecture where an input layer is followed by a number of hidden layers that can be either convolutional or fully connected or of any other type, and, finally, linear output activations. The transformation of such a network can be thus written as

$$f(x, w, b, \mathcal{W}_h) = w \cdot F(x, \mathcal{W}_h) + b = w \cdot \Phi(x) + b,$$

where $\mathcal{W}_h$ denotes the set of weights and biases up to the last hidden layer, w, b denote the linear weights and bias acting on this last hidden layer, and $F(x, \mathcal{W}_h)$ the last hidden layer outputs. We also denote these outputs as $\Phi(x)$ when we want to emphasize a kernel perspective.

Considering first SVCs, the hinge loss is differentiable, as mentioned, everywhere except at $\xi = 0$; the same is true for the ReLU activations commonly used in deep networks. As it is the case of ReLUs, the hinge loss can be implemented using the primitives of backends such as TensorFlow, and loss gradients are derived automatically when network models are compiled. Here, the optimal weights are to be obtained minimizing the following regularized cost

$$J(w, b, \mathcal{W}_h) = \frac{1}{N} \sum_{1}^{N} h\left(y^p, w \cdot F(x^p, \mathcal{W}_h) + b\right) + \alpha_S \|w\|^2 + \alpha_H \mathcal{R}(\mathcal{W}_h); \quad (5)$$

here $\|w\|^2$ is the squared norm of the linear output weights and $\mathcal{R}(\mathcal{W}_h)$ can be any regularizer function acting on the weights of the hidden layers. The preceding formulation points for instance to the regularization given by the Frobenius norm of the weights of fully connected or convolutional layers. Notice that, as explained below, we would keep in principle different weight penalties α_S and α_H for the linear output weights and for the hidden layer ones; if dropout was to be used, α_H would be the dropout probability.

The minimization of (5) can be achieved by standard DNN solvers such as stochastic gradient descent, Adam [16], Adagrad [9] or others. The regularization parameters α_S, α_H are to be selected by some form of cross validation (as well as the DNN architecture, if desired). If we work with a predefined DNN architecture, we will have to optimize here two hyperparameters, α_S, α_H, the same number than in Gaussian SVC, where the C penalty and γ kernel width have to be adjusted.

Notice that after optimization, the w^*, b^* weights correspond to a linear SVC acting on the hidden layer outputs $F(x^p, \mathcal{W}_h^*)$, for they must solve

$$\arg\min_{w,b} \frac{1}{N} \sum_{1}^{N} h(y^p, w \cdot z^p + b) + \alpha_S \|w\|^2 \quad (6)$$

with $z^p = F(x^p, \mathcal{W}_h^*)$. This is just the problem solved by a linear SVC on the z^p patterns; in particular, its margin is given by $\frac{1}{\|w^*\|}$. Since $\|w^*\|$ will be controlled by α_S while the other layers have different goals, this seems to point to the convenience of working with separate penalties α_S and α_H. Observe also that although the network's outputs $\xi = w^* \cdot F(x, \mathcal{W}_h^*) + b^*$ are continuous real numbers, the class prediction $\widehat{y}$ is obtained as $\widehat{y} = \text{sign}\ (\xi)$. Therefore, it is straightforward to obtain discrete classification scores such as accuracy, precision or recall, but some further work would be needed to achieve a more fine-grained prediction that makes possible to compute, for instance, ROC curves or their AUC. Possible ways of doing so would be to consider moving classification thresholds or to have a `predict_proba` method where the raw outputs of the Deep SVC were calibrated in some way.

Turning our attention to SVR, we again consider a more or less standard DNN architecture with linear outputs that defines a nonlinear transformation $f(x, w, b, \mathcal{W}_h) = w \cdot F(x, \mathcal{W}_h) + b = w \cdot \Phi(x) + b$ and a loss

$$J_\epsilon(w, b, \mathcal{W}_h) = \frac{1}{N}\sum_{1}^{N} \ell_\epsilon\left(y^p, w \cdot F(x^p, \mathcal{W}_h) + b\right) + \alpha_S \|w\|^2 + \alpha_H \mathcal{R}(\mathcal{W}_h) \quad (7)$$

where the regularization parameters α_S, α_H (and, if desired, the DNN architecture) are selected again by some form of cross validation. The optimal w^*, b^* will now coincide with those obtained by a linear SVR model acting on the last hidden layer outputs $z^p = F(x^p, \mathcal{W}_h^*)$. In Deep SVR we will have to optimize now three hyperparameters, α_S, α_H and the ϵ-insensitivity value, again the same number than in Gaussian SVRs.

In summary, essentially the same network structure is used for Deep SVC and SVR and the only differences are in the targets (± 1 for classification, real numbers for regression) and the loss used in each situation. In principle, we could expect classification accuracies or regression errors to be more or less the same when standard Gaussian or Deep SVC/SVR models are used, although the greater flexibility of the Deep SVC/SVR architectures should also result in a better performance in some problems, such as for instance, those dealing with images. In any case, the main advantage of the Deep SVC/SVR networks is likely to be manageable training times and, perhaps, more importantly, the ability to build models over large samples. Current kernel SV models have substantial difficulties with sample sizes above 100,000 and sizes above 500,000 are often outside their present reach unless really substantial computing resources are available. On the other hand, Deep SV models will be costly but still possible to be trained on those sample sizes. Moreover, prediction times will certainly be much faster for Deep SV models than for kernel based ones.

4 Experiments

4.1 Classification Experiments

We compare the performance of Kernel and Deep SVC over the datasets `a4a`, `a8a`, `australian`, `cod-rna`, `diabetes`, `german.numer`, `ijcnn1`, `w7a` and `w8a` from the LibSVM repository [2]. Their training and, when available, test sample sizes and dimensions are given in Table 1. In the Deep SVC (DSVC) models we will use ReLU hidden layer activations and Adam over minibatches as the loss optimizer. To lighten the hyper-parameterization costs we will use the default values in the Keras implementation of Adam for the initial learning rate (0.001) and the β_1 (0.9) and β_2 (0.999) parameteres. We shall also use in all cases a minibatch size of 200. Notice that network architecture, i.e., the number of hidden layers and of their units, could also be considered as hyper-parameters to be optimized. However we will work with DSVC models with 1 to 5 hidden layers and 100 units in all layers but the last one, which will have $0.1 \times |S|$ units, with $|S|$ denoting sample size, with a lower bound of 100 and an upper bound of 1000. The rationale for this is to enlarge the dimension of the last hidden layer projections of the sample patterns, so that the linear SVC models that act on them can have a large enough representation power.

Table 1. Number of train and test patterns and dimensions in the two-class problems.

	No. patterns train	No. patterns test	Dimension
a4a	4781	27780	123
a8a	22696	9865	123
australian	690		14
cod-rna	59535	271617	8
diabetes	768		8
german.numer	1000		24
ijcnn1	49990	91701	22
w7a	24692	25057	300
w8a	49749	14951	300

Table 2. Accuracies in the two-class problems.

	SVC	DSVC1	DSVC2	DSVC3	DSVC4	DSVC5	Best DSVC
a4a	**84.32 (1)**	84.19 (5.5)	84.20 (4)	84.19 (5.5)	84.29 (2)	84.27 (3)	84.29
a8a	84.92 (6)	**85.18 (1)**	84.95 (5)	85.14 (2.5)	85.08 (4)	85.14 (2.5)	**85.18**
australian	85.50 (5)	85.51 (4)	**87.09 (1)**	86.82 (2)	86.09 (3)	85.21 (6)	**87.09**
cod-rna	96.58 (5)	**96.66 (1)**	96.62 (2)	96.61 (3)	96.60 (4)	96.45 (6)	**96.66**
diabetes	**77.60 (1)**	76.95 (4)	77.47 (2)	76.43 (5)	76.43 (6)	77.21 (3)	77.47
german.numer	76.10 (2)	75.80 (3.5)	75.20 (5)	75.10 (6)	**76.60 (1)**	75.80 (3.5)	**76.60**
ijcnn1	97.93 (6)	98.88 (4)	99.03 (2)	**99.07 (1)**	98.84 (5)	98.99 (3)	**99.07**
w7a	**98.87 (1)**	98.82 (5)	98.85 (2)	98.83 (3.5)	98.79 (6)	98.83 (3.5)	98.85
w8a	99.04 (2)	98.99 (5)	98.92 (6)	**99.16 (1)**	99.04 (3)	99.03 (4)	**99.16**
rank mean	3.2222	3.6667	3.2222	3.2778	3.7778	3.8333	

We thus will hyperparameterize the L_2 (or Tikhonov) regularization penalties. We will consider two different such penalties one α_S, for the output weights and another one α_H which is the same for all the hidden layer weights. In both cases we will explore 5 values evenly spaced on a log scale in the interval $[2^{-30}, 2^{10}]$ selecting the optimal one by 4-fold cross validation as described below. As customary with neural networks, the Deep SVC inputs have been normalized feature-wise to 0 mean and 1 standard deviation.

The SVC models require two hyperparameters, the C regularization term and the width γ of the Gausian kernels $\exp\left(-\gamma\|x - x'\|^2\right)$. For C we will explore 5 values in the interval $[10^{-3}, 10^6]$. In order to select the Gaussian kernel width γ, we scale feature-wise the SVC inputs to a $[0, 1]$ range; the rationale for this is that after this normalization, we have $\|x - x'\|^2 \leq d$ with d pattern dimension. Because of this we will explore γ values of the form $\frac{2^k}{d}$, with k in the $[-3, 6]$ range.

Table 3. Number of patterns and dimensions in the regression problems.

	No. patterns	Dimension
abalone	4177	8
bodyfat	252	14
cpusmall	8192	12
housing	506	13
mg	1385	6
mpg	392	7
pyrim	74	27
space_ga	3107	6

Turning now to the cross validation (CV) procedure, when there is a separate test set, we find model hyperparameters by 4-fold stratified CV over the train set and then report the performance of the best hyperparameter model over the test set. When there is only one dataset, we use a nested two loop CV approach in order to assess model performance. More precisely, we apply stratified 4-fold CV on both loops. In the outer loop each one of the four outer folds is set apart for testing and the dataset made of the other three folds is passed to the inner loop, where again 4 fold CV is applied to determine the best hyperparameters, which are then tested on the test fold set apart. Model performance is measured as the average over these 4 test folds. Notice that while in the first case a single best parameter set is found, in the nested procedure optimal model hyperparameters may be different for each one of the outer test folds. In all cases the score used is model accuracy.

We report in Table 2 the accuracies of the Gaussian SVCs (SVC) and of Deep SVCs with 1 (DSVC1) to 5 (DSVC5) layers. For a better reading the table also gives in parenthesis the ranking of these accuracies. Similarly, we also give in the last column the accuracy of the best performing deep SVC model. This is done on a descriptive basis, as we have not performed a statistical analysis of the accuracy table, given the relatively small number of datasets considered. Gaussian SVCs give the largest accuracies on four datasets, `a4a`, `cod-rna`, `diabetes` and `w7a`. For the other five datasets the highest accuracy is achieved by a Deep SVC model. The row at the bottom of the table gives the average rankings. The better performing models appear to be SVC, DSVC2 and DSVC3. But, in any case, notice that all accuracy values are quite similar; this seems to imply that a statistical analysis of these accuracies would show them to be essentially the same. As mentioned before, the difference would lie in training and, particularly, test times, much lower for the deep models over the larger datasets.

4.2 Regression Experiments

Turning our attention to the regression problems, the datasets used to assess the performance of Kernel SVR and Deep SVR are `abalone`, `bodyfat`, `cpusmall`, `housing`, `mg`,

Table 4. MAEs in the regression problems.

	SVR	DSVR1	DSVR2	DSVR3	DSVR4	DSVR5	Best DSVR
abalone	1.48 (2)	1.49 (4)	1.49 (3)	1.50 (5)	**1.48 (1)**	1.51 (6)	**1.48**
bodyfat (×100)	**0.05 (1)**	0.42 (4)	0.31 (3)	0.51 (5)	0.53 (6)	0.28 (2)	0.28
cpusmall	2.13 (2)	2.21 (4)	**2.12 (1)**	2.19 (3)	2.31 (5)	2.40 (6)	**2.12**
housing	**2.28 (1)**	2.58 (6)	2.51 (4)	2.34 (3)	2.30 (2)	2.57 (5)	2.30
mg (×100)	9.26 (2)	9.68 (6)	9.65 (5)	**9.13 (1)**	9.36 (3)	9.58 (4)	**9.13**
mpg	**1.91 (1)**	2.39 (4)	2.28 (2)	2.49 (6)	2.42 (5)	2.36 (3)	2.28
pyrim (×100)	**5.62 (1)**	6.78 (3)	8.39 (5)	6.45 (2)	8.81 (6)	8.02 (4)	6.45
space_ga (×100)	9.67 (6)	9.14 (4)	9.22 (5)	**8.63 (1)**	8.70 (2)	8.86 (3)	**8.63**
rank mean	2	4.375	3.5	3.25	3.75	4.125	

`mpg`, `pyrim` and `space_ga`, again taken from the LibSVM repository [2]. Their sample sizes and dimensions are given in Table 3; notice that in this case there are no test sets.

The experimental setup is now exactly the same as in the classification setting, with the exception of the extra hyper-parameter needed for the ϵ-insensitive loss used in regression. This hyper-parameter ϵ is searched in a grid of 5 values distributed in a log scale between $2^{-10} \times std(y)$ and $2^{-1} \times std(y)$. Since there are no separate test sets, model performance in all cases is estimated by nested 4 fold CV.

We report now in Table 4 the mean absolute errors (MAEs) of the Gaussian SVRs (SVR) and of Deep SVRs with 1 (DSVR1) to 5 (DSVR5) layers. Again, the table also gives in parenthesis the ranking of these MAEs for a better reading; no statistical analysis has been performed here, again because of the relatively small number of datasets involved. We also give here the MAE of the best performing deep SVR model in the last column. Gaussian SVRs give the smallest MAE on the four smaller datasets, `bodyfat`, `housing`, `mpg` and `pyrim`, and give the second smallest MAE in three others. A Deep SVR model gives the smallest MAE for the other four datasets. Looking at the bottom row of the table, which has the average rankings, the SVR is clearly in the first place but, nevertheless, here again all MAE values are quite similar, something that a statistical test would probably confirm.

5 Conclusions and Further Work

In this paper we have shown that Deep SVC or SVR models, i.e., more or less standard DNNs with linear outputs and either hinge or ϵ-insensitive losses can give classification accuracies or mean absolute errors similar or even slightly better than those of Gaussian SVC or SVR models, but with much more manageable computational training and, particularly, test costs.

However, our experimental results open the way to more questions. A first one is which margin structure arises on the last hidden layer representations of

the sample patterns. Moreover, we have also seen that each problem seems to have its own optimal DNN architecture. Notice that Gaussian (or other kernel) SVMs also define a particular kind of architecture in terms of the kernel width hyper-parameter γ and the concrete support vectors found during training. This suggests that possible Deep SVM architectures should also be considered as hyperparameters to be adequately found. If this is done, it is likely that we would find a tie between the SVC/SVR performance and that of the best deep counterpart. Finally, when used for classification, DNNs with softmax outputs and cross entropy loss automatically yield posterior probabilities. However, this is not the case with standard SVC and, thus, neither with the deep SVMs proposed here. These and other related questions are currently under study.

Acknowledgments. With partial support from Spain's grants TIN2016-76406-P and S2013/ICE-2845 CASI-CAM-CM. Work partially supported also by project FACIL-Ayudas Fundación BBVA a Equipos de Investigación Científica 2016, and the UAM-ADIC Chair for Data Science and Machine Learning. We also gratefully acknowledge the use of the facilities of Centro de Computación Científica (CCC) at UAM.

References

1. Bordes, A., Ertekin, S., Weston, J., Bottou, L.: Fast kernel classifiers with online and active learning. J. Mach. Learn. Res. **6**, 1579–1619 (2005)
2. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 27:1–27:27 (2011). http://www.csie.ntu.edu.tw/~cjlin/libsvm
3. Chang, K., Hsieh, C., Lin, C.: Coordinate descent method for large-scale L2-loss linear support vector machines. J. Mach. Learn. Res. **9**, 1369–1398 (2008)
4. Chen, T., et al.: MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. CoRR abs/1512.01274 (2015)
5. Chollet, F.: Keras: deep learning library for Theano and TensorFlow (2015). https://github.com/fchollet/keras
6. Claesen, M., Smet, F.D., Suykens, J.A.K., Moor, B.D.: Ensemblesvm: a library for ensemble learning using support vector machines. J. Mach. Learn. Res. **15**(1), 141–145 (2014)
7. Collobert, R., Kavukcuoglu, K.: Torch7: a matlab-like environment for machine learning. In: BigLearn, NIPS Workshop (2011)
8. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Sig. Syst. (MCSS) **2**(4), 303–314 (1989)
9. Duchi, J.C., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
10. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010), vol. 9, pp. 249–256, May 2010
11. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011), April 2011

12. Google: Tensorflow, an open source software library for machine intelligence. https://www.tensorflow.org/
13. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Netw. **4**(2), 251–257 (1991)
14. Hsieh, C., Chang, K., Lin, C., Keerthi, S.S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, 5–9 June 2008, pp. 408–415 (2008)
15. Joachims, T., Yu, C.J.: Sparse kernel SVMs via cutting-plane training. Mach. Learn. **76**(2–3), 179–193 (2009)
16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2014)
17. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge (2002)
18. Seide, F., Agarwal, A.: CNTK: Microsoft's open-source deep-learning toolkit. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016, p. 2135 (2016)
19. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: primal estimated sub-gradient solver for SVM. Math. Program. **127**(1), 3–30 (2011)
20. Tang, Y.: Deep learning using support vector machines. CoRR abs/1306.0239 (2013). http://arxiv.org/abs/1306.0239
21. Yu, H., Hsieh, C., Chang, K., Lin, C.: Large linear classification when data cannot fit in memory. TKDD **5**(4), 23:1–23:23 (2012)
22. Zhang, S., Liu, C., Yao, K., Gong, Y.: Deep neural support vector machines for speech recognition. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, 19–24 April 2015, pp. 4275–4279 (2015)