Support Vector Machines on Large Data Sets: Simple Parallel Approaches

Oliver Meyer, Bernd Bischl, and Claus Weihs

Abstract Support Vector Machines (SVMs) are well-known for their excellent performance in the field of statistical classification. Still, the high computational cost due to the cubic runtime complexity is problematic for larger data sets. To mitigate this, Graf et al. (Adv. Neural Inf. Process. Syst. 17:521–528, 2005) proposed the Cascade SVM. It is a simple, stepwise procedure, in which the SVM is iteratively trained on subsets of the original data set and support vectors of resulting models are combined to create new training sets. The general idea is to bound the size of all considered training sets and therefore obtain a significant speedup. Another relevant advantage is that this approach can easily be parallelized because a number of independent models have to be fitted during each stage of the cascade. Initial experiments show that even moderate parallelization can reduce the computation time considerably, with only minor loss in accuracy. We compare the Cascade SVM to the standard SVM and a simple parallel bagging method w.r.t. both classification accuracy and training time. We also introduce a new stepwise bagging approach that exploits parallelization in a better way than the Cascade SVM and contains an adaptive stopping-time to select the number of stages for improved accuracy.

1 Introduction

Support vector machines (e.g., Schoelkopf and Smola 2002) are a very popular supervised learning algorithm for both classification and regression due to their flexibility and high predictive power. One major obstacle in their application to larger data sets is that their runtime scales approximately cubically with the

O. Meyer (✉) · B. Bischl · C. Weihs
Chair of Computational Statistics, Department of Statistics, TU Dortmund, Germany
e-mail: meyer@statistik.uni-dortmund.de; bischl@statistik.uni-dortmund.de; weihs@statistik.uni-dortmund.de

M. Spiliopoulou et al. (eds.), *Data Analysis, Machine Learning and Knowledge Discovery*, Studies in Classification, Data Analysis, and Knowledge Organization,
DOI 10.1007/978-3-319-01595-8_10,

number of observations in the training set. Combined with the fact that not one but multiple model fits have to be performed due to the necessity of hyperparameter tuning, their runtime often becomes prohibitively large beyond 100.000 or 1 million observations. Many different approaches have been suggested to speed up training time, among these online SVMs (e.g., the LASVM by Border et al. 2005, sampling techniques and parallelization schemes. In this article we will evaluate two quite simple methods of the latter kind. All of our considered approaches break up the original data set into smaller parts and fit individual SVM models on these. Because of the already mentioned cubical time-scaling of the SVM algorithm w.r.t. the number of data points a substantial speed up should be expected.

We state two further reasons for our general approach: (a) Computational power through multiple cores and multiple machines is often cheaply available these days. (b) We would like to keep as much as possible from the original SVM algorithm (use it as a building block in our parallelization scheme) in order to gain from future improvements in this area. It is also of general interest for us, how far we can get with such relatively simple approaches.

In the following sections we will cover the basic SVM theory, describe the considered parallelization approaches, state our experimental setup, report the results and then summarize them with additional remarks for future research.

2 Support Vector Machines

In supervised machine learning, data for classification tasks can be represented as a number of observations $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, where the set $\mathcal{X}$ defines the space in which our feature vectors $\boldsymbol{x}_i$ live in (here assumed to be $\mathbb{R}^p$ as we will mainly discuss the Gaussian kernel later on) and $\mathcal{Y} = \{-1, 1\}$ is the set of binary class labels. The support vector machine (SVM) relies on two basic concepts:

(a) Regularized risk minimization: We want to fit a large margin classifier $f : \mathbb{R}^p \to \mathbb{R}$ with a low empirical regularized risk:

$$(\hat{f}, \hat{b}) = \arg \inf_{f \in \mathcal{H}, b \in \mathbb{R}} ||f||^2_{\mathcal{H}} + C \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i) + b) \,. \tag{1}$$

Here, b is the so-called bias term of the classifier and L is a loss function. For classification with the SVM, we usually select the hinge loss $L(y, t) = \max(0, 1 - yt)$. This is a convex, upper surrogate loss for the 0/1-loss $L(y, t) = I[yt < 0]$, which is of primary interest, but algorithmically intractable.

While the second term above (called the empirical risk) measures the closeness of the predictions $f(x_i) + b$ to the true class labels -1 and $+1$, respectively, by means of L, the first term $||f||^2_H$ is called a regularizer, relates to the maximization of the margin and penalizes "non-smooth" functions f.

The balance between these two terms is controlled by the hyperparameter C. For an unknown observation x the class label is predicted by $\text{sign}(\hat{f}(x) + \hat{b})$.

(b) Kernelization: In order to be able to solve non-linear classification problems we "kernelize" (1) by introducing a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which measures the "similarity" of two observations. Formally, k is a symmetric, positive semi-definite Mercer kernel. And $\mathcal{H}$ is now defined as the associated reproducing kernel Hilbert space for k, our generalized inner product in $\mathcal{H}$ is $\langle x, x' \rangle_{\mathcal{H}} = k(x, x')$ and $||x||^2 = \langle x, x \rangle_{\mathcal{H}}$ By using this so-called "kernel trick" we implicitly map our data into a usually higher-dimensional space, enabling us to tackle nonlinear problems with essentially linear techniques. The Gaussian kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\sigma ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right) \tag{2}$$

is arguably the most important and popular kernel function and we have therefore focused on it in all subsequent experiments. But note that all following parallelization techniques are basically independent of this choice.

The optimization problem (1) is usually solved in its dual formulation and leads to the following quadratic programming problem:

$$\begin{aligned} \max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle_{\mathcal{H}} \\ \text{s.t. } 0 \leq \boldsymbol{\alpha} \leq C \text{ and } \boldsymbol{y}^T \boldsymbol{\alpha} = 0 \,, \end{aligned} \tag{3}$$

where $\boldsymbol{\alpha}$ denotes the vector of Lagrange multipliers.

As we will usually obtain a sparse solution due to the non-differentiability of our hinge loss L, some α_i will be zero, and the observations x_i with $\alpha_i > 0$ shall be called support vectors (SVs). They are the samples solely responsible for the shape of our decision border $f(\boldsymbol{x}) = 0$. This is implied by the fact that if we retrain an SVM on only the support vectors, we will arrive at exactly the same model as with the full data set.

The SVM performance is quite sensitive to hyperparameter settings, e.g., the settings of the complexity parameter C and the kernel parameter σ for the Gaussian kernel. Therefore, it is strongly recommended to perform hyperparameter tuning before the final model fit. Often a grid search approach is used, where performance is estimated by cross-validation, but more sophisticated methods become popular as well (see e.g., Koch et al. 2012).

Multi-class problems are usually solved by either using a multi-class-to-binary scheme (e.g., one-vs-one) or by directly changing the quadratic programming problem in (3) to incorporate several classes.

3 Cascade Support Vector Machine

The Cascade SVM is a stepwise procedure that combines the results of multiple regular support vector machines to create one final model. The main idea is to iteratively reduce a data set to its crucial data points before the last step. This is done by locating potential support vectors and removing all other samples from the data. The method described here is essentially taken from the original paper by Graf et al. (2005):

1. Partition the data into k disjoint subsets of preferably equal size.
2. Independently train an SVM on each of the data subsets.
3. Combine the SVs of, e.g., pairs or triples of SVMs to create new subsets.
4. Repeat steps 2 and 3 for some time.
5. Train an SVM on all SVs that were finally obtained in step 4.

This algorithm (depicted in the right-hand side of Fig. 1) will be called *Cascade SVM* or simply *cascade*. In the original paper, Graf et al. also considered the possibility of multiple runs through the cascade for each data set. After finishing a run through the cascade the subsets for the first step of the next run are created by combining the remaining SVs of the final model with each subset from the first step of the first run. For speed reasons we always only perform one run through the cascade.

4 Bagging-Like Support Vector Machines

Another generic and well-known concept in machine learning is bagging. Its main advantage is that derived methods are usually accurate and very easy to parallelize. Chawla et al. (2003) introduced and analyzed a simple variant, which proved to perform well on large data sets for decision trees and neural networks. Unlike in traditional bagging algorithms, the original data set is randomly split into n disjoint (and not overlapping) subsamples, which all contain $\frac{1}{n}$-th of the data. Then a classification model is trained on each of these subsets. Classification of new data is done by majority voting with ties being broken randomly. Hence, using SVMs means that the training of this bagging-like method is equivalent to the first step of the Cascade SVM. By comparing these two methods we can analyze if the additional steps of the cascade (and the invested runtime) improves the accuracy of the procedure.

Figure 1 shows the structures of a 4-2 Cascade SVM (C-4-2)—with 4 being the number of subsets in the first step and 2 representing the number of models being combined after every single step—and a bagged SVM using three bags.

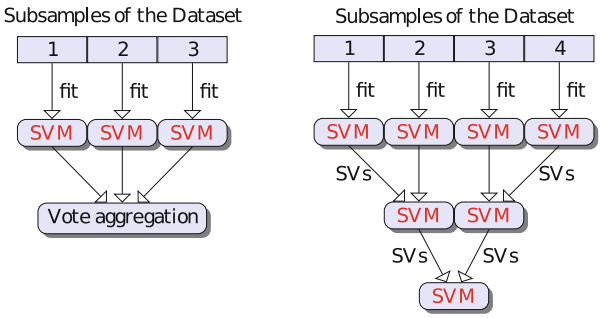

Fig. 1 Schemes for bagged SVM (*left*) and cascade (*right*)

5 Stepwise Bagging

It can easily be seen that the possibility to parallelize the Cascade SVM decreases in every step. This leads to the problem that an increasing number of cores will stay idle during the later stages, and in the last stage only one core can be used. We will also observe in the following experimental results that both algorithms—cascade and bagging—will perform suboptimally in some cases either with regard to runtime or accuracy. We therefore made the following changes to the described cascade algorithm in order to maximally use the number of available cores and to generally improve the algorithm by combining the advantages of both methods:

1. In the first stage, the data is partitioned in k subsets as usual.
2. At beginning of each subsequent stage in the cascade, all remaining vectors are combined into one set and then randomly divided into k overlapping subsets. The size of the subsets is fixed to the size of the subsets of the first stage, but not larger than 2/3 of the current data, if the former cannot be done. Overlapping occurs as vectors are drawn with replacement.
3. In the final stage, a bagged SVM is created instead of a single model.
4. As it is problematic to determine the number of stages of this approach we try to infer the optimal stopping time: At the beginning of the training process we hold out 5 % of the training data as an internal validation set. After each stage we measure the error of the bagged model of k SVMs from the current stage on this validation data. If the accuracy compared to the previous stage decreases, we stop the process and return the bagged model of the previous stage.
5. We have noticed that in some cases of a preliminary version of this stepwise bagging algorithm the performance degraded when the support vectors contained many wrongly classified examples. This happens in situations with high Bayes error/label noise, because all misclassified examples automatically become support vectors and will therefore always be contained in the training set for the next stage. As this seems somewhat counterintuitive, we have opted not to

select the support vectors in each stage, but instead only the SVs on and within the margin. This has the additional advantage that the set of relevant observations is reduced even further.

6 Experimental Setup

We evaluate the mentioned parallelization schemes on seven large data sets.[1] Their respective names and characteristics are listed in Table 1. Some of the data sets are multi-class, but as we want to focus on the analysis of the basic SVM algorithm, which at least in its usual from can only handle two-class problems, we have transformed the multi-class problems into binary ones. The transformation is stated in Table 1 as well. We have also eliminated every feature from every data set which was either constant or for which more than $n-1000$ samples shared the same feature value.

As we are mainly interested in analyzing the possible speedup of the training algorithm we have taken a practical approach w.r.t. the hyperparameter tuning in this article: For all data sets we have randomly sampled 10 % of all observations for tuning and then performed a usual grid search for $C \in 2^{-5}, 2^{-3}, \ldots, 2^{15}$ and $\sigma \in 2^{-15}, 2^{-13}, \ldots, 2^{3}$, estimating the misclassification error by fivefold cross-validation. The whole procedure was repeated five times, for every point (C, σ) the average misclassification rate was calculated, and the optimal configuration was selected. In case of ties, a random one was sampled from the optimal candidates. Table 1 displays the thereby obtained parameterizations and these have been used in all subsequent experiments.

Table 1 Data sets, data characteristics, used hyperparameters, proportional size of smallest class and multi-class binarization

Data set	Size	Features	C	σ	Smaller class	Binarization
covertype	581,012	46	8	0.500	0.488	Class 2 vs. rest
cod	429,030	9	32	0.125	0.333	
ijcnn1	191,681	22	32	3.13e−2	0.096	
miniboone	130,064	50	32768	7.81e−3	0.281	
acoustic	98,528	50	8	3.13e−2	0.500	Class 3 vs. rest
mnist	70,000	478	32	1.95e−3	0.492	Even vs. odd
connect	67,557	94	512	4.88e−4	0.342	Class 1 vs. rest

To compare the speedups of the different parallelization methods, we have run the following algorithms: The basic SVM, the Cascade SVM with C-27-3

[1] Can be obtained either from the LIBSVM web page or the UCI repository.

(27-9-3-1 SVMs), C-27-27 (27-1 SVMs), C-9-3 (9-3-1 SVMs) and C-9-9 (9-1 SVMs), bagging with nine bags (B-9) and the stepwise bagging approach also with nine subsets (SWB-9). For the basic SVM algorithm we have used the implementation provided in the `kernlab` R package by Karatzoglou et al. (2004). For all parallel methods we have used nine cores. For all algorithms we have repeatedly (ten times) split the data into a 3/4 part for training (with the already mentioned hyperparameters) and 1/4 for testing. In all cases we have measured the misclassification rate on the test set and the time in seconds that the whole training process lasted.

7 Results

The results of our experiments are displayed in Fig. 2 and can loosely be separated into three groups. For the data sets *acoustic*, *cod* and *miniboone*, the bagged SVMs lead to a better or at least equally good accuracy as the Cascade SVMs in only a fraction of its training time. Since the bagged SVM is nothing else but the first step of a cascade, this means that the subsequent steps of the cascade do not increase or even decrease the quality of the prediction. This does not mean that the Cascade SVM leads to bad results on all of these sets. In the case of *miniboone* for example it performs nearly as good as the classic SVM in just about 4,000 s compared to 65,000. But bagging does the same in only 350 s. On these data sets the stepwise bagging procedure usually leads to an accuracy that is equal to those of the standard bagging SVM and needs at worst as much time as the cascade.

The second group consists of *connect*, *covertype* and *mnist*. On these datasets the Cascade SVM leads to results that are as accurate as those from the classic SVM but only needs about half of the training time. The bagged SVM on the other hand again performs several times faster but cannot achieve the same accuracy as the other methods. So in these cases, at the cost of an at least ten times higher training time, the further steps of the cascade actually do increase the accuracy. The stepwise bagging SVM produces results that lie between the cascade and the standard bagging SVMs in both accuracy and training time. The actual boost in accuracy varies from data set to data set.

For the last dataset *ijcnn1*, all three methods perform very fast (cascade 15, bagging 25, stepwise bagging 14 times faster) but none of them achieves an accuracy that is as good as the classic SVM. Again the cascade outperforms bagging w.r.t accuracy while SWB lies between the other two methods.

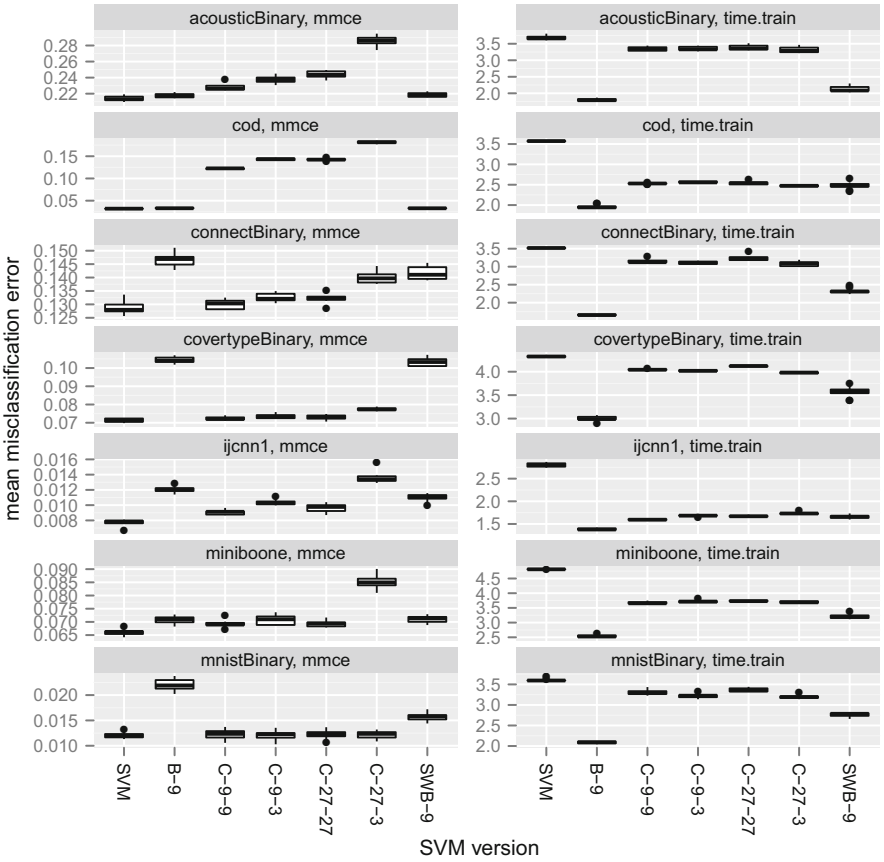

Fig. 2 Misclassification rates and training times (the latter on log10 scale) for normal and parallel SVMs

8 Conclusion and Outlook

We have analyzed simple parallelization schemes for parallel SVMs, namely a bagging-like approach, the Cascade SVM and a new combination of the two. On the considered data sets we could often observe a drastic reduction in training time through the parallelization with only minor losses in accuracy. Especially our new combined approach showed promising results. But still none of the considered algorithms shows optimal results across all considered data sets, and more work has to be done in this regard. One of the major missing features of our method is an efficient procedure for hyperparameter tuning that does not require many evaluations on large subsets of the training data. We have already begun preliminary experiments for this and will continue our research in this direction.

References

Border, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research, 6*, 1579–1619.

Chawla, N. V., Moore, T. E., Jr., Hall, L. O., Bowyer, K. W., Kegelmeyer, P., & Springer, C. (2003). Distributed learning with bagging-like performance. *Pattern Recognition Letter, 24*, 455–471.

Graf, H. P., Cosatto, E., Bottou, L., Durdanovic, I., & Vapnik, V. (2005). Parallel support vector machines: The cascade SVM. *Advances in Neural Information Processing Systems, 17*, 521–528.

Karatzoglou, A., Smola, A., Hornik, K., & Zeileis, A. (2004). kernlab - An S4 package for kernel methods in R. *Journal of Statistical Software, 11*(9), 1–20.

Koch, P., Bischl, B., Flasch, O., Bartz-Beilstein, T., & Konen, W. (2012). On the tuning and evolution of support vector kernels. *Evolutionary Intelligence, 5*, 153–170.

Schoelkopf, B., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. Cambridge: MIT.