Journal of Machine Learning Research 17 (2016) 1-42 Submitted 12/14; Revised 11/16; Published 12/16

GenSVM: A Generalized Multiclass Support Vector Machine

Gerrit J.J. van den Burg burg@ese.eur.nl
Patrick J.F. Groenen groenen@ese.eur.nl
Econometric Institute
Erasmus University Rotterdam
P.O. Box 1738
3000 DR Rotterdam
The Netherlands

Editor: Sathiya Keerthi

Abstract

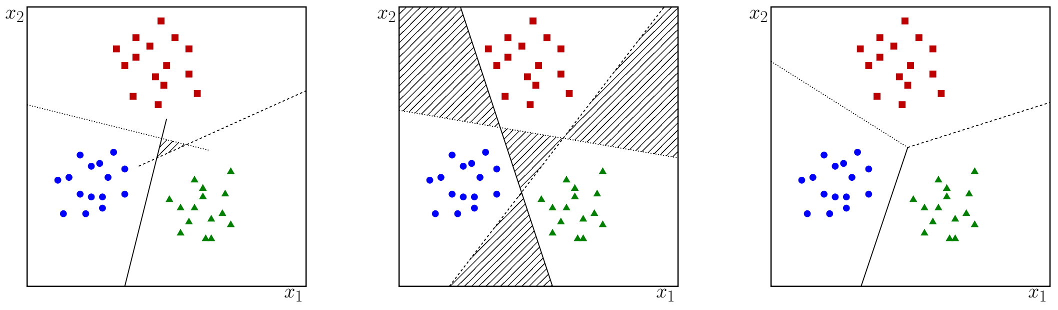
Traditional extensions of the binary support vector machine (SVM) to multiclass problems are either heuristics or require solving a large dual optimization problem. Here, a generalized multiclass SVM is proposed called GenSVM. In this method classification boundaries for a K-class problem are constructed in a $(K-1)$-dimensional space using a simplex encoding. Additionally, several different weightings of the misclassification errors are incorporated in the loss function, such that it generalizes three existing multiclass SVMs through a single optimization problem. An iterative majorization algorithm is derived that solves the optimization problem without the need of a dual formulation. This algorithm has the advantage that it can use warm starts during cross validation and during a grid search, which significantly speeds up the training phase. Rigorous numerical experiments compare linear GenSVM with seven existing multiclass SVMs on both small and large data sets. These comparisons show that the proposed method is competitive with existing methods in both predictive accuracy and training time, and that it significantly outperforms several existing methods on these criteria.

Keywords: support vector machines, SVM, multiclass classification, iterative majorization, MM algorithm, classifier comparison

1. Introduction

For binary classification, the support vector machine has shown to be very successful (Cortes and Vapnik, 1995). The SVM efficiently constructs linear or nonlinear classification boundaries and is able to yield a sparse solution through the so-called support vectors, that is, through those observations that are either not perfectly classified or are on the classification boundary. In addition, by regularizing the loss function the overfitting of the training data set is curbed. Due to its desirable characteristics several attempts have been made to extend the SVM to classification problems where the number of classes K is larger than two. Overall, these extensions differ considerably in the approach taken to include multiple classes. Three types of approaches for multiclass SVMs (MSVMs) can be distinguished.

First, there are heuristic approaches that use the binary SVM as an underlying classifier and decompose the K-class problem into multiple binary problems. The most commonly used heuristic is the one-vs-one (OvO) method where decision boundaries are constructed



(a) One vs. One (b) One vs. All (c) Non-heuristic

Figure 1: Illustration of ambiguity regions for common heuristic multiclass SVMs. In the shaded regions *ties* occur for which no classification rule has been explicitly trained. Figure (*c*) corresponds to an SVM where all classes are considered simultaneously, which eliminates any possible ties. Figures inspired by Statnikov et al. (2011).

between each pair of classes (Kreßel, 1999). OvO requires solving $K(K-1)$ binary SVM problems, which can be substantial if the number of classes is large. An advantage of OvO is that the problems to be solved are smaller in size. On the other hand, the one-vs-all (OvA) heuristic constructs K classification boundaries, one separating each class from all the other classes (Vapnik, 1998). Although OvA requires fewer binary SVMs to be estimated, the complete data set is used for each classifier, which can create a high computational burden. Another heuristic approach is the directed acyclic graph (DAG) SVM proposed by Platt et al. (2000). DAGSVM is similar to the OvO approach except that the class prediction is done by successively voting away unlikely classes until only one remains. One problem with the OvO and OvA methods is that there are regions of the space for which class predictions are ambiguous, as illustrated in Figures 1a and 1b.

In practice, heuristic methods such as the OvO and OvA approaches are used more often than other multiclass SVM implementations. One of the reasons for this is that there are several software packages that efficiently solve the binary SVM, such as LibSVM (Chang and Lin, 2011). This package implements a variation of the sequential minimal optimization algorithm of Platt (1999). Implementations of other multiclass SVMs in high-level (statistical) programming languages are lacking, which reduces their use in practice.[1]

The second type of extension of the binary SVM use error correcting codes. In these methods the problem is decomposed into multiple binary classification problems based on a constructed coding matrix that determines the grouping of the classes in a specific binary subproblem (Dietterich and Bakiri, 1995; Allwein et al., 2001; Crammer and Singer, 2002b). Error correcting code SVMs can thus be seen as a generalization of OvO and OvA. In Dietterich and Bakiri (1995) and Allwein et al. (2001), a coding matrix is constructed that determines which class instances are paired against each other for each binary SVM. Both approaches require that the coding matrix is determined beforehand. However, it is a priori

1. An exception to this is the method of Lee et al. (2004), for which an `R` implementation exists. See `http://www.stat.osu.edu/~yklee/software.html`.

unclear how such a coding matrix should be chosen. In fact, as Crammer and Singer (2002b) show, finding the optimal coding matrix is an NP-complete problem.

The third type of approaches are those that optimize one loss function to estimate all class boundaries simultaneously, the so-called single machine approaches (Rifkin and Klautau, 2004). In the literature, such methods have been proposed by, among others, Weston and Watkins (1998), Bredensteiner and Bennett (1999), Crammer and Singer (2002a), Lee et al. (2004), and Guermeur and Monfrini (2011). The method of Weston and Watkins (1998) yields a fairly large quadratic problem with a large number of slack variables, that is, $K - 1$ slack variables for each observation. The method of Crammer and Singer (2002a) reduces this number of slack variables by only penalizing the largest misclassification error. In addition, their method does not include a bias term in the decision boundaries, which is advantageous for solving the dual problem. Interestingly, this approach does not reduce parsimoniously to the binary SVM for $K = 2$. The method of Lee et al. (2004) uses a sum-to-zero constraint on the decision functions to reduce the dimensionality of the problem. This constraint effectively means that the solution of the multiclass SVM lies in a $(K - 1)$-dimensional subspace of the full K dimensions considered. The size of the margins is reduced according to the number of classes, such that asymptotic convergence is obtained to the Bayes optimal decision boundary when the regularization term is ignored (Rifkin and Klautau, 2004). Finally, the method of Guermeur and Monfrini (2011) is a quadratic extension of the method developed by Lee et al. (2004). This extension keeps the sum-to-zero constraint on the decision functions, drops the nonnegativity constraint on the slack variables, and adds a quadratic function of the slack variables to the loss function. This means that at the optimum the slack variables are only positive on average, which differs from common SVM formulations.

The existing approaches to multiclass SVMs suffer from several problems. All current single machine multiclass extensions of the binary SVM rely on solving a potentially large dual optimization problem. This can be disadvantageous when a solution has to be found in a small amount of time, since iteratively improving the dual solution does not guarantee that the primal solution is improved as well. Thus, stopping early can lead to poor predictive performance. In addition, the dual of such single machine approaches should be solvable quickly in order to compete with existing heuristic approaches.

Almost all single machine approaches rely on misclassifications of the observed class with each of the other classes. By simply summing these misclassification errors (as in Lee et al., 2004) observations with multiple errors contribute more than those with a single misclassification do. Consequently, observations with multiple misclassifications have a stronger influence on the solution than those with a single misclassification, which is not a desirable property for a multiclass SVM, as it overemphasizes objects that are misclassified with respect to multiple classes. Here, it is argued that there is no reason to penalize certain misclassification regions more than others.

Single machine approaches are preferred for their ability to capture the multiclass classification problem in a single model. A parallel can be drawn here with multinomial regression and logistic regression. In this case, multinomial regression reduces exactly to the binary logistic regression method when $K = 2$, both techniques are single machine approaches, and many of the properties of logistic regression extend to multinomial regression. Therefore, it can be considered natural to use a single machine approach for the multiclass SVM that reduces parsimoniously to the binary SVM when $K = 2$.

The idea of casting the multiclass SVM problem to $K - 1$ dimensions is appealing, since it reduces the dimensionality of the problem and is also present in other multiclass classification methods such as multinomial regression and linear discriminant analysis. However, the sum-to-zero constraint employed by Lee et al. (2004) creates an additional burden on the dual optimization problem (Dogan et al., 2011). Therefore, it would be desirable to cast the problem to $K - 1$ dimensions in another manner. Below a simplex encoding will be introduced to achieve this goal. The simplex encoding for multiclass SVMs has been proposed earlier by Hill and Doucet (2007) and Mroueh et al. (2012), although the method outlined below differs from these two approaches. Note that the simplex coding approach by Mroueh et al. (2012) was shown to be equivalent to that of Lee et al. (2004) by Ávila Pires et al. (2013). An advantage of the simplex encoding is that in contrast to methods such as OvO and OvA, there are no regions of ambiguity in the prediction space (see Figure 1c). In addition, the low dimensional projection also has advantages for understanding the method, since it allows for a geometric interpretation. The geometric interpretation of existing single machine multiclass SVMs is often difficult since most are based on a dual optimization approach with little attention for a primal problem based on hinge errors.

A new flexible and general multiclass SVM is proposed, called GenSVM. This method uses the simplex encoding to formulate the multiclass SVM problem as a single optimization problem that reduces to the binary SVM when $K = 2$. By using a flexible hinge function and an ℓ_p norm of the errors the GenSVM loss function incorporates three existing multiclass SVMs that use the sum of the hinge errors, and extends these methods. In the linear version of GenSVM, $K - 1$ linear combinations of the features are estimated next to the bias terms. In the nonlinear version, kernels can be used in a similar manner as can be done for binary SVMs. The resulting GenSVM loss function is convex in the parameters to be estimated. For this loss function an iterative majorization (IM) algorithm will be derived with guaranteed descent to the global minimum. By solving the optimization problem in the primal it is possible to use warm starts during a hyperparameter grid search or during cross validation, which makes the resulting algorithm very competitive in total training time, even for large data sets.

To evaluate its performance, GenSVM is compared to seven of the multiclass SVMs described above on several small data sets and one large data set. The smaller data sets are used to assess the classification accuracy of GenSVM, whereas the large data set is used to verify feasibility of GenSVM for large data sets. Due to the computational cost of these rigorous experiments only comparisons of linear multiclass SVMs are performed, and experiments on nonlinear MSVMs are considered outside the scope of this paper. Existing comparisons of multiclass SVMs in the literature do not determine any statistically significant differences in performance between classifiers, and resort to tables of accuracy rates for the comparisons (for instance Hsu and Lin, 2002). Using suggestions from the benchmarking literature predictive performance and training time of all classifiers is compared using performance profiles and rank tests. The rank tests are used to uncover statistically significant differences between classifiers.

This paper is organized as follows. Section 2 introduces the novel generalized multiclass SVM. In Section 3, features of the iterative majorization theory are reviewed and a number of useful properties are highlighted. Section 4 derives the IM algorithm for GenSVM, and presents pseudocode for the algorithm. Extensions of GenSVM to nonlinear classification boundaries are discussed in Section 5. A numerical comparison of GenSVM with existing multiclass SVMs on empirical data sets is done in Section 6. Section 7 concludes the paper.

2. GenSVM

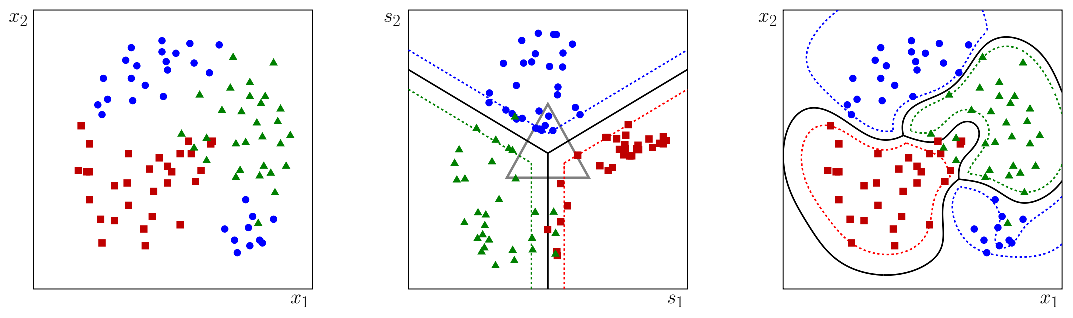
Before introducing GenSVM formally, consider a small illustrative example of a hypothetical data set of $n = 90$ objects with $K = 3$ classes and $m = 2$ attributes. Figure 2a shows the data set in the space of these two attributes x_1 and x_2, with different classes denoted by different symbols. Figure 2b shows the $(K-1)$-dimensional simplex encoding of the data after an additional RBF kernel transformation has been applied and the mapping has been optimized to minimize misclassification errors. In this figure, the triangle shown in the center corresponds to a regular K-simplex in $K-1$ dimensions, and the solid lines perpendicular to the faces of this simplex are the decision boundaries. This $(K-1)$-dimensional space will be referred to as the *simplex space* throughout this paper. The mapping from the input space to this simplex space is optimized by minimizing the misclassification errors, which are calculated by measuring the distance of an object to the decision boundaries in the simplex space. Prediction of a class label is also done in this simplex space, by finding the nearest simplex vertex for the object. Figure 2c illustrates the decision boundaries in the original space of the input attributes x_1 and x_2. In Figures 2b and 2c, the support vectors can be identified as the objects that lie on or beyond the dashed margin lines of their associated class. Note that the use of the simplex encoding ensures that for every point in the predictor space a class is predicted, hence no ambiguity regions can exist in the GenSVM solution.

The misclassification errors are formally defined as follows. Let $\mathbf{x}_i \in \mathbb{R}^m$ be an object vector corresponding to m attributes, and let y_i denote the class label of object i with $y_i \in \{1, \ldots, K\}$, for $i \in \{1, \ldots, n\}$. Furthermore, let $\mathbf{W} \in \mathbb{R}^{m \times (K-1)}$ be a weight matrix, and define a translation vector $\mathbf{t} \in \mathbb{R}^{K-1}$ for the bias terms. Then, object i is represented in the $(K-1)$-dimensional simplex space by $\mathbf{s}_i' = \mathbf{x}_i'\mathbf{W} + \mathbf{t}'$. Note that here the *linear* version of GenSVM is described, the nonlinear version is described in Section 5.

To obtain the misclassification error of an object, the corresponding simplex space vector $\mathbf{s}_i'$ is projected on each of the decision boundaries that separate the true class of an object from another class. For the errors to be proportional with the distance to the decision boundaries, a regular K-simplex in $\mathbb{R}^{K-1}$ is used with distance 1 between each pair of vertices. Let $\mathbf{U}_K$ be the $K \times (K-1)$ coordinate matrix of this simplex, where a row $\mathbf{u}_k'$ of $\mathbf{U}_K$ gives the coordinates of a single vertex k. Then, it follows that with $k \in \{1, \ldots, K\}$ and $l \in \{1, \ldots, K-1\}$ the elements of $\mathbf{U}_K$ are given by

$$u_{kl} = \begin{cases} -\frac{1}{\sqrt{2(l^2+l)}} & \text{if } k \leq l \\ \frac{l}{\sqrt{2(l^2+l)}} & \text{if } k = l+1 \\ 0 & \text{if } k > l+1. \end{cases} \tag{1}$$

See Appendix A for a derivation of this expression. Figure 3 shows an illustration of how the misclassification errors are computed for a single object. Consider object A with true class



(a) Input space (b) Simplex space (c) Input space with boundaries

Figure 2: Illustration of GenSVM for a 2D data set with $K = 3$ classes. In (a) the original data is shown, with different symbols denoting different classes. Figure (b) shows the mapping of the data to the $(K-1)$-dimensional simplex space, after an additional RBF kernel mapping has been applied and the optimal solution has been determined. The decision boundaries in this space are fixed as the perpendicular bisectors of the faces of the simplex, which is shown as the gray triangle. Figure (c) shows the resulting boundaries mapped back to the original input space, as can be seen by comparing with (a). In Figures (b) and (c) the dashed lines show the margins of the SVM solution.

$y_A = 2$. It is clear that object A is misclassified as it is not located in the shaded area that has Vertex $\mathbf{u}_2$ as the nearest vertex. The boundaries of the shaded area are given by the perpendicular bisectors of the edges of the simplex between Vertices $\mathbf{u}_2$ and $\mathbf{u}_1$ and between Vertices $\mathbf{u}_2$ and $\mathbf{u}_3$, and form the decision boundaries for class 2. The error for object A is computed by determining the distance from the object to each of these decision boundaries. Let $q_A^{(21)}$ and $q_A^{(23)}$ denote these distances to the class boundaries, which are obtained by projecting $\mathbf{s}'_A = \mathbf{x}'_A\mathbf{W} + \mathbf{t}'$ on $\mathbf{u}_2 - \mathbf{u}_1$ and $\mathbf{u}_2 - \mathbf{u}_3$ respectively, as illustrated in the figure. Generalizing this reasoning, scalars $q_i^{(kj)}$ can be defined to measure the projection distance of object i onto the boundary between class k and j in the simplex space, as

$$q_i^{(kj)} = (\mathbf{x}'_i\mathbf{W} + \mathbf{t}')(\mathbf{u}_k - \mathbf{u}_j). \tag{2}$$

It is required that the GenSVM loss function is both general and flexible, such that it can easily be tuned for the specific data set at hand. To achieve this, a loss function is constructed with a number of different weightings, each with a specific effect on the object distances $q_i^{(kj)}$. In the proposed loss function, flexibility is added through the use of the Huber hinge function instead of the absolute hinge function, and by using the ℓ_p norm of the hinge errors instead of the sum. The motivation for these choices follows.

As is customary for SVMs a hinge loss is used to ensure that instances that do not cross their class margin will yield zero error. Here, the flexible and continuous Huber hinge loss

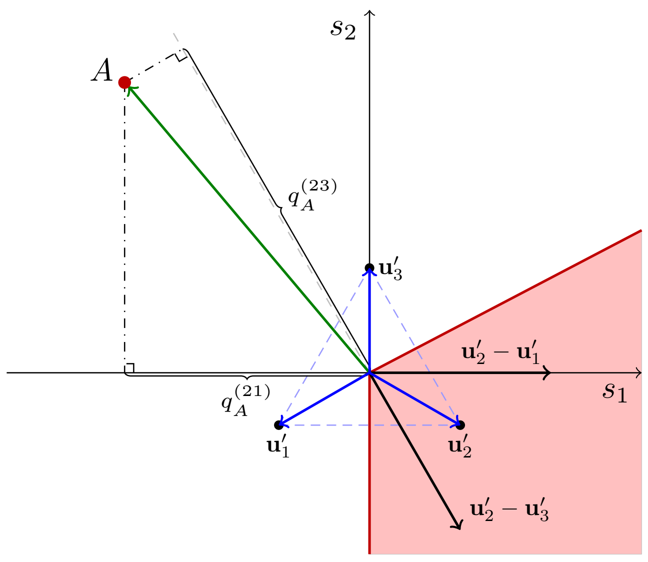


Figure 3: Graphical illustration of the calculation of distances $q_i^{(y_A j)}$ for an object A with $y_A = 2$ and $K = 3$. The figure shows the situation in the $(K-1)$-dimensional space. The distance $q_A^{(21)}$ is calculated by projecting $\mathbf{s}_A' = \mathbf{x}_A'\mathbf{W} + \mathbf{t}'$ on $\mathbf{u}_2 - \mathbf{u}_1$, and the distance $q_A^{(23)}$ is found by projecting $\mathbf{s}_A'$ on $\mathbf{u}_2 - \mathbf{u}_3$. The boundary between the class 1 and class 3 regions has been omitted for clarity, but lies along $\mathbf{u}_2$.

is used (after the Huber error in robust statistics, see Huber, 1964), which is defined as

$$h(q) = \begin{cases} 1 - q - \frac{\kappa+1}{2} & \text{if } q \leq -\kappa \\ \frac{1}{2(\kappa+1)}(1-q)^2 & \text{if } q \in (-\kappa, 1] \\ 0 & \text{if } q > 1, \end{cases} \tag{3}$$

with $\kappa > -1$. The Huber hinge loss has been independently introduced in Chapelle (2007), Rosset and Zhu (2007), and Groenen et al. (2008). This hinge error is zero when an instance is classified correctly with respect to its class margin. However, in contrast to the absolute hinge error, it is continuous due to a quadratic region in the interval $(-\kappa, 1]$. This quadratic region allows for a softer weighting of objects close to the decision boundary. Additionally, the smoothness of the Huber hinge error is a desirable property for the iterative majorization algorithm derived in Section 4.1. Note that the Huber hinge error approaches the absolute hinge for $\kappa \downarrow -1$, and the quadratic hinge for $\kappa \to \infty$.

The Huber hinge error is applied to each of the distances $q_i^{(y_i j)}$, for $j \neq y_i$. Thus, no error is counted when the object is correctly classified. For each of the objects, errors with respect to the other classes are summed using an ℓ_p norm to obtain the total object error

$$\left(\sum_{\substack{j=1 \\ j \neq y_i}}^{K} h^p\left(q_i^{(y_i j)}\right) \right)^{1/p}.$$

The ℓ_p norm is added to provide a form of regularization on Huber weighted errors for instances that are misclassified with respect to multiple classes. As argued in the Introduction, simply summing misclassification errors can lead to overemphasizing of instances with multiple misclassification errors. By adding an ℓ_p norm of the hinge errors the influence of such instances on the loss function can be tuned. With the addition of the ℓ_p norm on the hinge errors it is possible to illustrate how GenSVM generalizes existing methods. For instance, with $p = 1$ and $\kappa \downarrow -1$, the loss function solves the same problem as the method of Lee et al. (2004). Next, for $p = 2$ and $\kappa \downarrow -1$ it resembles that of Guermeur and Monfrini (2011). Finally, for $p = \infty$ and $\kappa \downarrow -1$ the ℓ_p norm reduces to the max norm of the hinge errors, which corresponds to the method of Crammer and Singer (2002a). Note that in each case the value of κ can additionally be varied to include an even broader family of loss functions.

To illustrate the effects of p and κ on the total object error, refer to Figure 4. In Figures 4a and 4b, the value of p is set to $p = 1$ and $p = 2$ respectively, while maintaining the absolute hinge error using $\kappa = -0.95$. A reference point is plotted at a fixed position in the area of the simplex space where there is a nonzero error with respect to two classes. It can be seen from this reference point that the value of the combined error is higher when $p = 1$. With $p = 2$ the combined error at the reference point approximates the Euclidean distance to the margin, when $\kappa \downarrow -1$. Figures 4a, 4c, and 4d show the effect of varying κ. It can be seen that the error near the margin becomes more quadratic with increasing κ. In fact, as κ increases the error approaches the squared Euclidean distance to the margin, which can be used to obtain a quadratic hinge multiclass SVM. Both of these effects will become stronger when the number of classes increases, as increasingly more objects will have errors with respect to more than one class.

Next, let $\rho_i \geq 0$ denote optional object weights, which are introduced to allow flexibility in the way individual objects contribute to the total loss function. With these individual weights it is possible to correct for different group sizes, or to give additional weights to misclassifications of certain classes. When correcting for group sizes, the weights can be chosen as

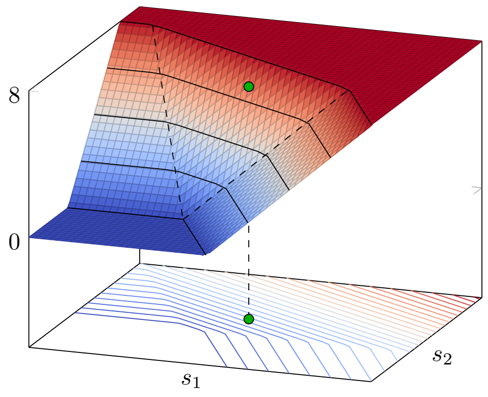
$$\rho_i = \frac{n}{n_k K}, \quad i \in G_k, \tag{4}$$

where $G_k = \{i : y_i = k\}$ is the set of objects belonging to class k, and $n_k = |G_k|$. The complete GenSVM loss function combining all n objects can now be formulated as

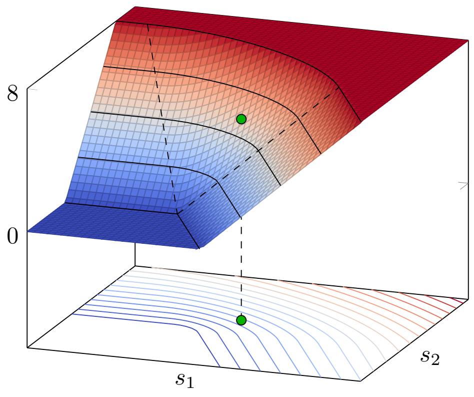
$$L_{\text{MSVM}}(\mathbf{W}, \mathbf{t}) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in G_k} \rho_i \left(\sum_{j \neq k} h^p \left(q_i^{(kj)} \right) \right)^{1/p} + \lambda \operatorname{tr} \mathbf{W}'\mathbf{W}, \tag{5}$$

where $\lambda \operatorname{tr} \mathbf{W}'\mathbf{W}$ is the penalty term to avoid overfitting, and $\lambda > 0$ is the regularization parameter. Note that for the case where $K = 2$, the above loss function reduces to the loss function for binary SVM given in Groenen et al. (2008), with Huber hinge errors.

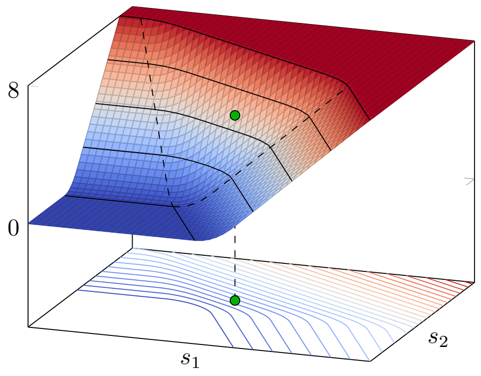
The outline of a proof for the convexity of the loss function in (5) is given. First, note that the distances $q_i^{(kj)}$ in the loss function are affine in $\mathbf{W}$ and $\mathbf{t}$. Hence, if the loss function is convex in $q_i^{(kj)}$ it is convex in $\mathbf{W}$ and $\mathbf{t}$ as well. Second, the Huber hinge function is trivially convex in $q_i^{(kj)}$, since each separate piece of the function is convex, and the Huber



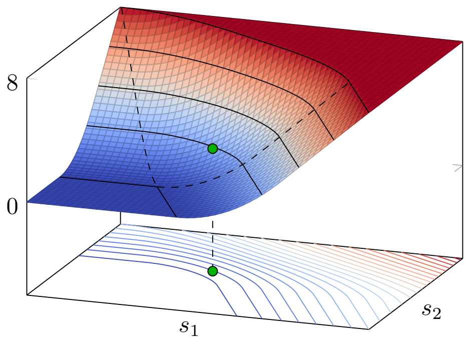
(a) $p = 1$ and $\kappa = -0.95$



(b) $p = 2$ and $\kappa = -0.95$



(c) $p = 1$ and $\kappa = 1.0$



(d) $p = 1$ and $\kappa = 5.0$

Figure 4: Illustration of the ℓ_p norm of the Huber weighted errors. Comparing figures (a) and (b) shows the effect of the ℓ_p norm. With $p = 1$ objects that have errors w.r.t. both classes are penalized more strongly than those with only one error, whereas with $p = 2$ this is not the case. Figures (a), (c), and (d) compare the effect of the κ parameter, with $p = 1$. This shows that with a large value of κ, the errors close to the boundary are weighted quadratically. Note that s_1 and s_2 indicate the dimensions of the simplex space.

hinge is continuous. Third, the ℓ_p norm is a convex function by the Minkowski inequality, and it is monotonically increasing by definition. Thus, it follows that the ℓ_p norm of the Huber weighted instance errors is convex (see for instance Rockafellar, 1997). Next, since it is required that the weights ρ_i are non-negative, the sum in the first term of (5) is a convex combination. Finally, the penalty term can also be shown to be convex, since tr $\mathbf{W}'\mathbf{W}$ is the square of the Frobenius norm of $\mathbf{W}$, and it is required that $\lambda > 0$. Thus, it holds that the loss function in (5) is convex in $\mathbf{W}$ and $\mathbf{t}$.

Predicting class labels in GenSVM can be done as follows. Let $(\mathbf{W}^*, \mathbf{t}^*)$ denote the parameters that minimize the loss function. Predicting the class label of an unseen sample $\mathbf{x}'_{n+1}$ can then be done by first mapping it to the simplex space, using the optimal projection: $\mathbf{s}'_{n+1} = \mathbf{x}'_{n+1}\mathbf{W}^* + \mathbf{t}'^*$. The predicted class label is then simply the label corresponding to the nearest simplex vertex as measured by the squared Euclidean norm, or

$$\hat{y}_{n+1} = \arg\min_k \|\mathbf{s}'_{n+1} - \mathbf{u}'_k\|^2, \qquad \text{for } k = 1, \ldots, K. \tag{6}$$

3. Iterative Majorization

To minimize the loss function given in (5), an iterative majorization (IM) algorithm will be derived. Iterative majorization was first described by Weiszfeld (1937), however the first application of the algorithm in the context of a line search comes from Ortega and Rheinboldt (1970, p. 253—255). During the late 1970s, the method was independently developed by De Leeuw (1977) as part of the SMACOF algorithm for multidimensional scaling, and by Voss and Eckhardt (1980) as a general minimization method. For the reader unfamiliar with the iterative majorization algorithm a more detailed description has been included in Appendix B and further examples can be found in for instance Hunter and Lange (2004).

The asymptotic convergence rate of the IM algorithm is linear, which is less than that of the Newton-Raphson algorithm (De Leeuw, 1994). However, the largest improvements in the loss function will occur in the first few steps of the iterative majorization algorithm, where the asymptotic linear rate does not apply (Havel, 1991). This property will become very useful for GenSVM as it allows for a quick approximation to the exact SVM solution in few iterations.

There is no straightforward technique for deriving the majorization function for any given function. However, in the next section the derivation of the majorization function for the GenSVM loss function is presented using an "outside-in" approach. In this approach, each function that constitutes the loss function is majorized separately and the majorization functions are combined. Two properties of majorization functions that are useful for this derivation are now formally defined. In these expressions, $\overline{x}$ is a supporting point, as defined in Appendix B.

P1. Let $f_1 : \mathcal{Y} \to \mathcal{Z}$, $f_2 : \mathcal{X} \to \mathcal{Y}$, and define $f = f_1 \circ f_2 : \mathcal{X} \to \mathcal{Z}$, such that for $x \in \mathcal{X}$, $f(x) = f_1(f_2(x))$. If $g_1 : \mathcal{Y} \times \mathcal{Y} \to \mathcal{Z}$ is a majorization function of f_1, then $g : \mathcal{X} \times \mathcal{X} \to \mathcal{Z}$ defined as $g = g_1 \circ f_2$ is a majorization function of f. Thus for $x, \overline{x} \in \mathcal{X}$ it holds that $g(x, \overline{x}) = g_1(f_2(x), f_2(\overline{x}))$ is a majorization function of $f(x)$ at $\overline{x}$.

P2. Let $f_i : \mathcal{X} \to \mathcal{Z}$ and define $f : \mathcal{X} \to \mathcal{Z}$ such that $f(x) = \sum_i a_i f_i(x)$ for $x \in \mathcal{X}$, with $a_i \geq 0$ for all i. If $g_i : \mathcal{X} \times \mathcal{X} \to \mathcal{Z}$ is a majorization function for f_i at a point $\overline{x} \in \mathcal{X}$, then $g : \mathcal{X} \times \mathcal{X} \to \mathcal{Z}$ given by $g(x, \overline{x}) = \sum_i a_i g_i(x, \overline{x})$ is a majorization function of f.

Proofs of these properties are omitted, as they follow directly from the requirements for a majorization function given in Appendix B. The first property allows for the use of the "outside-in" approach to majorization, as will be illustrated in the next section.

4. GenSVM Optimization and Implementation

In this section, a quadratic majorization function for GenSVM will be derived. Although it is possible to derive a majorization algorithm for general values of the ℓ_p norm parameter,[2]

2. For a majorization algorithm of the ℓ_p norm with $p \geq 2$, see Groenen et al. (1999).

the following derivation will restrict this value to the interval $p \in [1, 2]$ since this simplifies the derivation and avoids the issue that quadratic majorization can become slow for $p > 2$. Pseudocode for the derived algorithm will be presented, as well as an analysis of the computational complexity of the algorithm. Finally, an important remark on the use of warm starts in the algorithm is given.

4.1 Majorization Derivation

To shorten the notation, define

$$\begin{aligned} \mathbf{V} &= [\mathbf{t}\ \mathbf{W}']', \\ \mathbf{z}_i' &= [1\ \mathbf{x}_i'], \\ \boldsymbol{\delta}_{kj} &= \mathbf{u}_k - \mathbf{u}_j, \end{aligned}$$

such that $q_i^{(kj)} = \mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}$. With this notation it becomes sufficient to optimize the loss function with respect to $\mathbf{V}$. Formulated in this manner (5) becomes

$$L_{\text{MSVM}}(\mathbf{V}) = \frac{1}{n}\sum_{k=1}^{K}\sum_{i \in G_k} \rho_i \left(\sum_{j \neq k} h^p\left(q_i^{(kj)}\right)\right)^{1/p} + \lambda \operatorname{tr} \mathbf{V}'\mathbf{J}\mathbf{V}, \tag{7}$$

where $\mathbf{J}$ is an $m+1$ diagonal matrix with $\mathbf{J}_{i,i} = 1$ for $i > 1$ and zero elsewhere. To derive a majorization function for this expression the "outside-in" approach will be used, together with the properties of majorization functions. In what follows, variables with a bar denote supporting points for the IM algorithm. The goal of the derivation is to find a quadratic majorization function in $\mathbf{V}$ such that

$$L_{\text{MSVM}}(\mathbf{V}) \leq \operatorname{tr} \mathbf{V}'\mathbf{Z}'\mathbf{A}\mathbf{Z}'\mathbf{V} - 2\operatorname{tr} \mathbf{V}'\mathbf{Z}'\mathbf{B} + C, \tag{8}$$

where $\mathbf{A}$, $\mathbf{B}$, and C are coefficients of the majorization depending on $\overline{\mathbf{V}}$. The matrix $\mathbf{Z}$ is simply the $n \times (m+1)$ matrix with rows $\mathbf{z}_i'$.

Property P2 above means that the summation over instances in the loss function can be ignored for now. Moreover, the regularization term is quadratic in $\mathbf{V}$, and thus requires no majorization. The outermost function for which a majorization function has to be found is thus the ℓ_p norm of the Huber hinge errors. Hence it is possible to consider the function $f(\mathbf{x}) = \|\mathbf{x}\|_p$ for majorization. A majorization function for $f(\mathbf{x})$ can be constructed, but a discontinuity in the derivative at $\mathbf{x} = \mathbf{0}$ will remain (Tsutsu and Morikawa, 2012).

To avoid the discontinuity in the derivative of the ℓ_p norm, the following inequality is needed (Hardy et al., 1934, eq. 2.10.3)

$$\left(\sum_{j \neq k} h^p\left(q_i^{(kj)}\right)\right)^{1/p} \leq \sum_{j \neq k} h\left(q_i^{(kj)}\right).$$

This inequality can be used as a majorization function only if equality holds at the supporting point

$$\left(\sum_{j \neq k} h^p\left(\overline{q}_i^{(kj)}\right)\right)^{1/p} = \sum_{j \neq k} h\left(\overline{q}_i^{(kj)}\right).$$

It is not difficult to see that this only holds if at most one of the $h\left(\overline{q}_i^{(kj)}\right)$ errors is nonzero for $j \neq k$. Thus an indicator variable ε_i is introduced which is 1 if at most one of these errors is nonzero, and 0 otherwise. Then it follows that

$$\begin{aligned} L_{\text{MSVM}}(\mathbf{V}) \leq & \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in G_k} \rho_i \left[\varepsilon_i \sum_{j \neq k} h\left(q_i^{(kj)}\right) + (1-\varepsilon_i) \left(\sum_{j \neq k} h^p\left(q_i^{(kj)}\right) \right)^{1/p} \right] \\ & + \lambda \operatorname{tr} \mathbf{V}'\mathbf{J}\mathbf{V}. \end{aligned} \tag{9}$$

Now, the next function for which a majorization needs to be found is $f_1(x) = x^{1/p}$. From the inequality $a^\alpha b^\beta < \alpha a + \beta b$, with $\alpha + \beta = 1$ (Hardy et al., 1934, Theorem 37), a linear majorization inequality can be constructed for this function by substituting $a = x$, $b = \overline{x}$, $\alpha = 1/p$ and $\beta = 1 - 1/p$ (Groenen and Heiser, 1996). This yields

$$f_1(x) = x^{1/p} \leq \frac{1}{p} \overline{x}^{1/p-1} x + \left(1 - \frac{1}{p}\right) \overline{x}^{1/p} = g_1(x, \overline{x}).$$

Applying this majorization and using property P1 gives

$$\begin{aligned} \left(\sum_{j \neq k} h^p\left(q_i^{(kj)}\right) \right)^{1/p} \leq & \frac{1}{p} \left(\sum_{j \neq k} h^p\left(\overline{q}_i^{(kj)}\right) \right)^{1/p-1} \left(\sum_{j \neq k} h^p\left(q_i^{(kj)}\right) \right) \\ & + \left(1 - \frac{1}{p}\right) \left(\sum_{j \neq k} h^p\left(\overline{q}_i^{(kj)}\right) \right)^{1/p}. \end{aligned}$$

Plugging this into (9) and collecting terms yields

$$\begin{aligned} L_{\text{MSVM}}(\mathbf{V}) \leq & \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in G_k} \rho_i \left[\varepsilon_i \sum_{j \neq k} h\left(q_i^{(kj)}\right) + (1-\varepsilon_i)\omega_i \sum_{j \neq k} h^p\left(q_i^{(kj)}\right) \right] \\ & + \Gamma^{(1)} + \lambda \operatorname{tr} \mathbf{V}'\mathbf{J}\mathbf{V}, \end{aligned}$$

with

$$\omega_i = \frac{1}{p} \left(\sum_{j \neq k} h^p\left(\overline{q}_i^{(kj)}\right) \right)^{1/p-1}. \tag{10}$$

The constant $\Gamma^{(1)}$ contains all terms that only depend on previous errors $\overline{q}_i^{(kj)}$. The next majorization step by the "outside-in" approach is to find a quadratic majorization function for $f_2(x) = h^p(x)$, of the form

$$f_2(x) = h^p(x) \leq a(\overline{x}, p)x^2 - 2b(\overline{x}, p)x + c(\overline{x}, p) = g_2(x, \overline{x}).$$

Since this derivation is mostly an algebraic exercise it has been moved to Appendix C. In the remainder of this derivation, $a_{ijk}^{(p)}$ will be used to abbreviate $a(\overline{q}_i^{(kj)}, p)$, with similar abbreviations for b and c. Using these majorizations and making the dependence on $\mathbf{V}$ explicit by substituting $q_i^{(kj)} = \mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}$ gives

$$\begin{aligned} L_{\text{MSVM}}(\mathbf{V}) \leq & \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\varepsilon_i\sum_{j\neq k}\left[a_{ijk}^{(1)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'\mathbf{V}'\mathbf{z}_i - 2b_{ijk}^{(1)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\right] \\ & + \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i(1-\varepsilon_i)\omega_i\sum_{j\neq k}\left[a_{ijk}^{(p)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'\mathbf{V}'\mathbf{z}_i - 2b_{ijk}^{(p)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\right] \\ & + \Gamma^{(2)} + \lambda\,\text{tr}\;\mathbf{V}'\mathbf{J}\mathbf{V}, \end{aligned}$$

where $\Gamma^{(2)}$ again contains all constant terms. Due to dependence on the matrix $\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'$, the above majorization function is not yet in the desired quadratic form of (8). However, since the maximum eigenvalue of $\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'$ is 1 by definition of the simplex coordinates, it follows that the matrix $\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}' - \mathbf{I}$ is negative semidefinite. Hence, it can be shown that the inequality $\mathbf{z}_i'(\mathbf{V}-\overline{\mathbf{V}})(\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'-\mathbf{I})(\mathbf{V}-\overline{\mathbf{V}})'\mathbf{z}_i \leq 0$ holds (Bijleveld and De Leeuw, 1991, Theorem 4). Rewriting this gives the majorization inequality

$$\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'\mathbf{V}'\mathbf{z}_i \leq \mathbf{z}_i'\mathbf{V}\mathbf{V}'\mathbf{z}_i - 2\mathbf{z}_i'\mathbf{V}(\mathbf{I}-\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}')\overline{\mathbf{V}}\mathbf{z}_i + \mathbf{z}_i'\overline{\mathbf{V}}(\mathbf{I}-\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}')\overline{\mathbf{V}}'\mathbf{z}_i.$$

With this inequality the majorization inequality becomes

$$\begin{aligned} L_{\text{MSVM}}(\mathbf{V}) \leq & \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\mathbf{z}_i'\mathbf{V}(\mathbf{V}'-2\overline{\mathbf{V}}')\mathbf{z}_i\sum_{j\neq k}\left[\varepsilon_i a_{ijk}^{(1)} + (1-\varepsilon_i)\omega_i a_{ijk}^{(p)}\right] \qquad (11) \\ & - \frac{2}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\mathbf{z}_i'\mathbf{V}\sum_{j\neq k}\left[\varepsilon_i\left(b_{ijk}^{(1)} - a_{ijk}^{(1)}\overline{q}_i^{(kj)}\right)\right. \\ & \qquad\qquad \left. + (1-\varepsilon_i)\omega_i\left(b_{ijk}^{(p)} - a_{ijk}^{(p)}\overline{q}_i^{(kj)}\right)\right]\boldsymbol{\delta}_{kj} \\ & + \Gamma^{(3)} + \lambda\,\text{tr}\;\mathbf{V}'\mathbf{J}\mathbf{V}, \end{aligned}$$

where $\overline{q}_i^{(kj)} = \mathbf{z}_i'\overline{\mathbf{V}}\boldsymbol{\delta}_{kj}$. This majorization function is quadratic in $\mathbf{V}$ and can thus be used in the IM algorithm. To derive the first-order condition used in the update step of the IM algorithm (step 2 in Appendix B), matrix notation for the above expression is introduced. Let $\mathbf{A}$ be an $n \times n$ diagonal matrix with elements α_i, and let $\mathbf{B}$ be an $n \times (K-1)$ matrix with rows $\boldsymbol{\beta}_i'$, where

$$\alpha_i = \frac{1}{n}\rho_i\sum_{j\neq k}\left[\varepsilon_i a_{ijk}^{(1)} + (1-\varepsilon_i)\omega_i a_{ijk}^{(p)}\right], \qquad (12)$$

$$\boldsymbol{\beta}_i' = \frac{1}{n}\rho_i\sum_{j\neq k}\left[\varepsilon_i\left(b_{ijk}^{(1)} - a_{ijk}^{(1)}\overline{q}_i^{(kj)}\right) + (1-\varepsilon_i)\omega_i\left(b_{ijk}^{(p)} - a_{ijk}^{(p)}\overline{q}_i^{(kj)}\right)\right]\boldsymbol{\delta}_{kj}'. \qquad (13)$$

Then the majorization function of $L_{\text{MSVM}}(\mathbf{V})$ given in (11) can be written as

$$\begin{aligned} L_{\text{MSVM}}(\mathbf{V}) \leq &\; \text{tr}\;(\mathbf{V}-2\overline{\mathbf{V}})'\mathbf{Z}'\mathbf{A}\mathbf{Z}\mathbf{V} - 2\,\text{tr}\;\mathbf{B}'\mathbf{Z}\mathbf{V} + \Gamma^{(3)} + \lambda\,\text{tr}\;\mathbf{V}'\mathbf{J}\mathbf{V} \\ = &\; \text{tr}\;\mathbf{V}'(\mathbf{Z}'\mathbf{A}\mathbf{Z} + \lambda\mathbf{J})\mathbf{V} - 2\,\text{tr}\;(\overline{\mathbf{V}}'\mathbf{Z}'\mathbf{A} + \mathbf{B}')\mathbf{Z}\mathbf{V} + \Gamma^{(3)}. \end{aligned}$$

This majorization function has the desired functional form described in (8). Differentiation with respect to $\mathbf{V}$ and equating to zero yields the linear system

$$(\mathbf{Z}'\mathbf{A}\mathbf{Z} + \lambda\mathbf{J})\mathbf{V} = \mathbf{Z}'\mathbf{A}\mathbf{Z}\overline{\mathbf{V}} + \mathbf{Z}'\mathbf{B}. \tag{14}$$

The update $\mathbf{V}^+$ that solves this system can then be calculated efficiently by Gaussian elimination.

4.2 Algorithm Implementation and Complexity

Pseudocode for GenSVM is given in Algorithm 1. As can be seen, the algorithm simply updates all instance weights at each iteration, starting by determining the indicator variable ε_i. In practice, some calculations can be done efficiently for all instances by using matrix algebra. When step doubling (see Appendix B) is applied in the majorization algorithm, line `25` is replaced by $\mathbf{V} \leftarrow 2\mathbf{V}^+ - \overline{\mathbf{V}}$. In the implementation step doubling is applied after a burn-in of 50 iterations. The implementation used in the experiments described in Section 6 is written in `C`, using the ATLAS (Whaley and Dongarra, 1998) and LAPACK (Anderson et al., 1999) libraries. The source code for this `C` library is available under the open source GNU GPL license, through an online repository. A thorough description of the implementation is available in the package documentation.

The complexity of a single iteration of the IM algorithm is $O(n(m+1)^2)$ assuming that $n > m > K$. As noted earlier, the convergence rate of the general IM algorithm is linear. Computational complexity of standard SVM solvers that solve the dual problem through decomposition methods lies between $O(n^2)$ and $O(n^3)$ depending on the value of λ (Bottou and Lin, 2007). An efficient algorithm for the method of Crammer and Singer (2002a) developed by Keerthi et al. (2008) has a complexity of $O(n\overline{m}K)$ per iteration, where $\overline{m} \leq m$ is the average number of nonzero features per training instance. In the methods of Lee et al. (2004) and Weston and Watkins (1998), a quadratic programming problem with $n(K-1)$ dual variables needs to be solved, which is typically done using a standard solver. An analysis of the exact convergence of GenSVM, including the expected number of iterations needed to achieve convergence at a factor ϵ, is outside the scope of the current work and a subject for further research.

4.3 Smart Initialization

When training machine learning algorithms to determine the optimal hyperparameters, it is common to use cross validation (CV). With GenSVM it is possible to initialize the matrix $\overline{\mathbf{V}}$ such that the final result of a fold is used as the initial value for $\mathbf{V}_0$ for the next fold. This same technique can be used when searching for the optimal hyperparameter configuration in a grid search, by initializing the weight matrix with the outcome of the previous configuration. Such warm-start initialization greatly reduces the time needed to perform cross validation with GenSVM. It is important to note here that using warm starts is not easily possible with dual optimization approaches. Therefore, the ability to use warm starts can be seen as an advantage of solving the GenSVM optimization problem in the primal.

Algorithm 1: GenSVM Algorithm

```
Input: $\mathbf{X}, \mathbf{y}, \boldsymbol{\rho}, p, \kappa, \lambda, \epsilon$
Output: $\mathbf{V}$
$K \leftarrow \max(\mathbf{y})$
$t \leftarrow 1$
$\mathbf{Z} \leftarrow [\mathbf{1}\ \mathbf{X}]$
Let $\overline{\mathbf{V}} \leftarrow \mathbf{V}_0$
Generate $\mathbf{J}$ and $\mathbf{U}_K$
$L_t = L_{\text{MSVM}}(\overline{\mathbf{V}})$
$L_{t-1} = (1 + 2\epsilon)L_t$
while $(L_{t-1} - L_t)/L_t > \epsilon$ do
    for $i \leftarrow 1$ to $n$ do
        Compute $\overline{q}_i^{(y_i j)} = \mathbf{z}_i'\overline{\mathbf{V}}\boldsymbol{\delta}_{y_i j}$ for all $j \neq y_i$
        Compute $h\left(\overline{q}_i^{(y_i j)}\right)$ for all $j \neq y_i$ by (3)
        if $\varepsilon_i = 1$ then
            Compute $a_{ijy_i}^{(1)}$ and $b_{ijy_i}^{(1)}$ for all $j \neq y_i$ according to Table 4 in Appendix C
        else
            Compute $\omega_i$ following (10)
            Compute $a_{ijy_i}^{(p)}$ and $b_{ijy_i}^{(p)}$ for all $j \neq y_i$ according to Table 4 in Appendix C
        end
        Compute $\alpha_i$ by (12)
        Compute $\boldsymbol{\beta}_i$ by (13)
    end
    Construct $\mathbf{A}$ from $\alpha_i$
    Construct $\mathbf{B}$ from $\boldsymbol{\beta}_i$
    Find $\mathbf{V}^+$ that solves (14)
    $\overline{\mathbf{V}} \leftarrow \mathbf{V}$
    $\mathbf{V} \leftarrow \mathbf{V}^+$
    $L_{t-1} \leftarrow L_t$
    $L_t \leftarrow L_{\text{MSVM}}(\mathbf{V})$
    $t \leftarrow t + 1$
end
```

5. Nonlinearity

One possible method to include nonlinearity in a classifier is through the use of spline transformations (see for instance Hastie et al., 2009). With spline transformations each attribute vector $\mathbf{x}_j$ is transformed to a spline basis $\mathbf{N}_j$, for $j = 1, \ldots, m$. The transformed input matrix $\mathbf{N} = [\mathbf{N}_1, \ldots, \mathbf{N}_m]$ is then of size $n \times l$, where l depends on the degree of the spline transformation and the number of interior knots chosen. An application of spline transformations to the binary SVM can be found in Groenen et al. (2007).

A more common way to include nonlinearity in machine learning methods is through the use of the kernel trick, attributed to Aizerman et al. (1964). With the kernel trick, the dot product of two instance vectors in the dual optimization problem is replaced by the dot product of the same vectors in a high dimensional feature space. Since no dot products appear in the primal formulation of GenSVM, a different method is used here.

By applying a preprocessing step on the kernel matrix, nonlinearity can be included using the same algorithm as the one presented for the linear case. Furthermore, predicting class labels requires a postprocessing step on the obtained matrix $\mathbf{V}^*$. A full derivation is given in Appendix D.

6. Experiments

To assess the performance of the proposed GenSVM classifier, a simulation study was done comparing GenSVM with seven existing multiclass SVMs on 13 small data sets. These experiments are used to precisely measure predictive accuracy and total training time using performance profiles and rank plots. To verify the feasibility of GenSVM for large data sets an additional simulation study is done. The results of this study are presented separately in Section 6.4. Due to the large number of data sets and methods involved, experiments were only done for the linear kernel. Experiments on nonlinear multiclass SVMs would require even more training time than for linear MSVMs and is considered outside the scope of this paper.

6.1 Setup

Implementations of the heuristic multiclass SVMs (OvO, OvA, and DAG) were included through LibSVM (v. 3.16, Chang and Lin, 2011). LibSVM is a popular library for binary SVMs with packages for many programming languages, it is written in `C++` and implements a variation of the SMO algorithm of Platt (1999). The OvO and DAG methods are implemented in this package, and a `C` implementation of OvA using LibSVM was created for these experiments.[3] For the single-machine approaches the MSVMpack package was used (v. 1.3, Lauer and Guermeur, 2011), which is written in `C`. This package implements the methods of Weston and Watkins (W&W, 1998), Crammer and Singer (C&S, 2002a), Lee et al. (LLW, 2004), and Guermeur and Monfrini (MSVM^2, 2011). Finally, to verify if implementation differences are relevant for algorithm performance the LibLinear (Fan et al., 2008) implementation of the method by Crammer and Singer (2002a) is also included (denoted LL C&S). This implementation uses the optimization algorithm by Keerthi et al. (2008).

To compare the classification methods properly, it is desirable to remove any bias that could occur when using cross validation (Cawley and Talbot, 2010). Therefore, *nested* cross validation is used (Stone, 1974), as illustrated in Figure 5. In nested CV, a data set is randomly split in a number of *chunks*. Each of these chunks is kept apart from the remaining chunks once, while the remaining chunks are combined to form a single data set. A grid search is then applied to this combined data set to find the optimal hyperparameters with which to predict the test chunk. This process is then repeated for each of the chunks. The predictions of the test chunk will be unbiased since it was not included in the grid search. For this reason, it is argued that this approach is preferred over approaches that simply report maximum accuracy rates obtained during the grid search.

3. The LibSVM code used for DAGSVM is the same code as was used in Hsu and Lin (2002) and is available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools`.

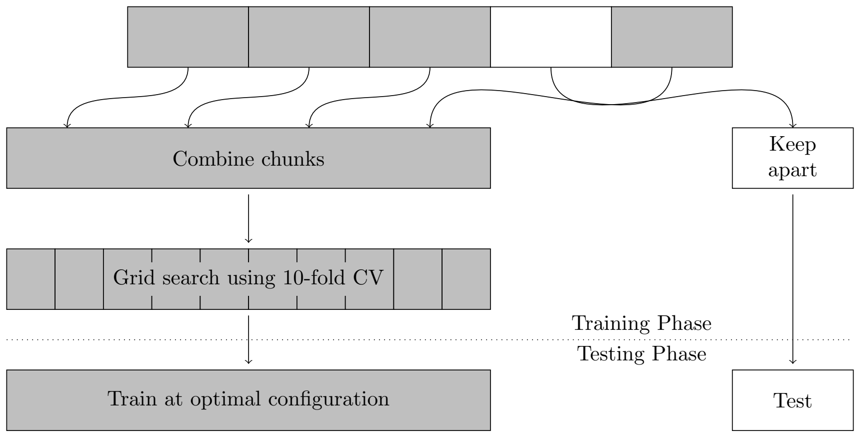


Figure 5: An illustration of nested cross validation. A data set is initially split in five *chunks*. Each chunk is kept apart once, while a grid search using 10-fold CV is applied to the combined data from the remaining 4 chunks. The optimal parameters obtained there are then used to train the model one last time, and predict the chunk that was kept apart.

For the experiments 13 data sets were selected from the UCI repository (Bache and Lichman, 2013). The selected data sets and their relevant statistics are shown in Table 1. All attributes were rescaled to the interval $[-1, 1]$. The `image segmentation` and `vowel` data sets have a predetermined train and test set, and were therefore not used in the nested CV procedure. Instead, a grid search was done on the provided training set for each classifier, and the provided test set was predicted at the optimal hyperparameters obtained. For the data sets without a predetermined train/test split, nested CV was used with 5 initial chunks. Hence, $5 \cdot 11 + 2 = 57$ pairs of independent train and test data sets are obtained.

While running the grid search, it is desirable to remove any fluctuations that may result in an unfair comparison. Therefore, it was ensured that all methods had the same CV split of the training data for the same hyperparameter configuration (specifically, the value of the regularization parameter). In practice, it can occur that a specific CV split is advantageous for one classifier but not for others (either in time or performance). Thus, ideally the grid search would be repeated a number of times with different CV splits, to remove this variation. However, due to the size of the grid search this is considered to be infeasible. Finally, it should be noted here that during the grid search 10-fold cross validation was applied in a non-stratified manner, that is, without resampling of small classes.

The following settings were used in the numerical experiments. The regularization parameter was varied on a grid with $\lambda \in \{2^{-18}, 2^{-16}, \dots, 2^{18}\}$. For GenSVM the grid search was extended with the parameters $\kappa \in \{-0.9, 0.5, 5.0\}$ and $p \in \{1.0, 1.5, 2.0\}$. The stopping parameter for the GenSVM majorization algorithm was set at $\epsilon = 10^{-6}$ during the grid search in the training phase and at $\epsilon = 10^{-8}$ for the final model in the testing phase. In addition, two different weight specifications were used for GenSVM: the unit weights with $\rho_i = 1, \forall i$, as well as the group-size correction weights introduced in (4). Thus, the grid search consists of 342 configurations for GenSVM, and 19 configurations

Data set	Instances (n)	Features (m)	Classes (K)	$\min n_k$	$\max n_k$
breast tissue	106	9	6	14	22
iris	150	4	3	50	50
wine	178	13	3	48	71
image segmentation*	210/2100	18	7	30	30
glass	214	9	6	9	76
vertebral	310	6	3	60	150
ecoli	336	8	8	2	143
vowel*	528/462	10	11	48	48
balancescale	625	4	3	49	288
vehicle	846	18	4	199	218
contraception	1473	9	3	333	629
yeast	1484	8	10	5	463
car	1728	6	4	65	1210

Table 1: Data set summary statistics. Data sets with an asterisk have a predetermined test data set. For these data sets, the number of training instances is denoted for the train and test data sets respectively. The final two columns denote the size of the smallest and the largest class, respectively.

for the other methods. Since nested CV is used for most data sets, it is required to run 10-fold cross validation on a total of 28158 hyperparameter configurations. To enhance the reproducibility of these experiments, the exact predictions made by each classifier for each configuration were stored in a text file.

To run all computations in a reasonable amount of time, the computations were performed on the Dutch National LISA Compute Cluster. A master-worker program was developed using the message passing interface in Python (Dalcín et al., 2005). This allows for efficient use of multiple nodes by successively sending out tasks to worker threads from a single master thread. Since the total training time of a classifier is also of interest, it was ensured that all computations were done on the exact same core type.[4] Furthermore, training time was measured from within the `C` programs, to ensure that only the time needed for the cross validation routine was measured. The total computation time needed to obtain the presented results was about 152 days, using the LISA Cluster this was done in five and a half days wall-clock time.

During the training phase it showed that several of the single machine methods implemented through MSVMpack did not converge to an optimal solution within reasonable amount of time.[5] Instead of limiting the maximum number of iterations of the method, MSVMpack was modified to stop after a maximum of 2 hours of training time per configuration. This results in 12 minutes of training time per cross validation fold. The solution found after this amount of training time was used for prediction during cross validation.

4. The specific type of core used is the Intel Xeon E5-2650 v2, with 16 threads at a clock speed of 2.6 GHz. At most 14 threads were used simultaneously, reserving one for the master thread and one for system processes.
5. The default MSVMpack settings were used with a chunk size of 4 for all methods.

Whenever training was stopped prematurely, this was recorded.[6] Of the 57 training sets, 24 configurations had prematurely stopped training in one or more CV splits for the LLW method, versus 19 for W&W, 9 for MSVM^2, and 2 for C&S (MSVMpack). For the LibSVM methods, 13 optimal configurations for OvA reached the default maximum number of iterations in one or more CV folds, versus 9 for DAGSVM, and 3 for OvO. No early stopping was needed for GenSVM or for LL C&S.

Determining the optimal hyperparameters requires a performance measure on the obtained predictions. For binary classifiers it is common to use either the hitrate or the area under the ROC curve as a measure of classifier performance. The hitrate only measures the percentage of correct predictions of a classifier and has the well known problem that no correction is made for group sizes. For instance, if 90% of the observations of a test set belong to one class, a classifier that always predicts this class has a high hitrate, regardless of its discriminatory power. Therefore, the adjusted Rand index (ARI) is used here as a performance measure (Hubert and Arabie, 1985). The ARI corrects for chance and can therefore more accurately measure discriminatory power of a classifier than the hitrate can. Using the ARI for evaluating supervised learning algorithms has previously been proposed by Santos and Embrechts (2009).

The optimal parameter configurations for each method on each data set were chosen such that the maximum predictive performance was obtained as measured with the ARI. If multiple configurations obtained the highest performance during the grid search, the configuration with the smallest training time was chosen. The results on the training data show that during cross validation GenSVM achieved the highest classification accuracy on 41 out of 57 data sets, compared to 15 and 12 for DAG and OvO, respectively. However, these are results on the training data sets and therefore can contain considerable bias. To accurately assess the out-of-sample prediction accuracy the optimal hyperparameter configurations were determined for each of the 57 training sets, and the test sets were predicted with these parameters. To remove any variations due to random starts, building the classifier and predicting the test set was repeated 5 times for each classifier.

Below the simulation results on the small data sets will be evaluated using performance profiles and rank tests. Performance profiles offer a visual representation of classifier performance, while rank tests allow for identification of statistically significant differences between classifiers. For the sake of completeness tables of performance scores and computation times for each method on each data set are provided in Appendix E. To promote reproducibility of the empirical results, all the code used for the classifier comparisons and all the obtained results will be released through an online repository.

6.2 Performance Profiles

One way to get insight in the performance of different classification methods is through *performance profiles* (Dolan and Moré, 2002). A performance profile shows the empirical cumulative distribution function of a classifier on a performance metric.

6. For the classifiers implemented through LibSVM very long training times were only observed for the OvA method, however due to the nature of this method it is not trivial to stop the calculations after a certain amount of time. This behavior was observed in about 1% of all configurations tested on all data sets, and is therefore considered negligible. Also, for the LibSVM methods it was recorded whenever the maximum number of iterations was reached.

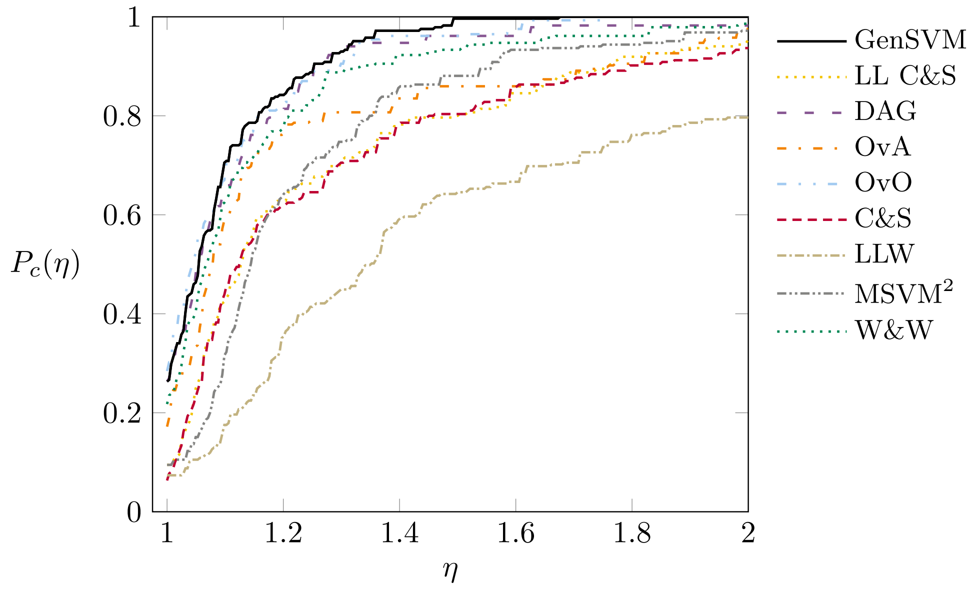


Figure 6: Performance profiles for classification accuracy created from all repetitions of the test set predictions. The methods OvA, C&S, LL C&S, MSVM^2, W&W, and LLW will always have a smaller probability of being within a factor η of the maximum performance than the GenSVM, OvO, or DAG methods.

Let $\mathcal{D}$ denote the set of data sets, and $\mathcal{C}$ denote the set of classifiers. Further, let $p_{d,c}$ denote the performance of classifier $c \in \mathcal{C}$ on data set $d \in \mathcal{D}$ as measured by the ARI. Now define the performance ratio $v_{d,c}$ as the ratio between the best performance on data set d and the performance of classifier c on data set d, that is

$$v_{d,c} = \frac{\max\{p_{d,c} : c \in \mathcal{C}\}}{p_{d,c}}.$$

Thus the performance ratio is 1 for the best performing classifier on a data set and increases for classifiers with a lower performance. Then, the performance profile for classifier c is given by the function

$$P_c(\eta) = \frac{1}{N_D} \left|\{d \in \mathcal{D} : v_{d,c} \leq \eta\}\right|,$$

where $N_D = |\mathcal{D}|$ denotes the number of data sets. Thus, the performance profile estimates the probability that classifier c has a performance ratio below η. Note that $P_c(1)$ denotes the empirical probability that a classifier achieves the highest performance on a given data set.

Figure 6 shows the performance profile for classification accuracy. Estimates of $P_c(1)$ from Figure 6 show that there is a 28.42% probability that OvO achieves the optimal performance, versus 26.32% for both GenSVM and DAGSVM. Note that this includes cases where each of these methods achieves the best performance. Figure 6 also shows that although there is a small difference in the probabilities of GenSVM, OvO, and DAG within

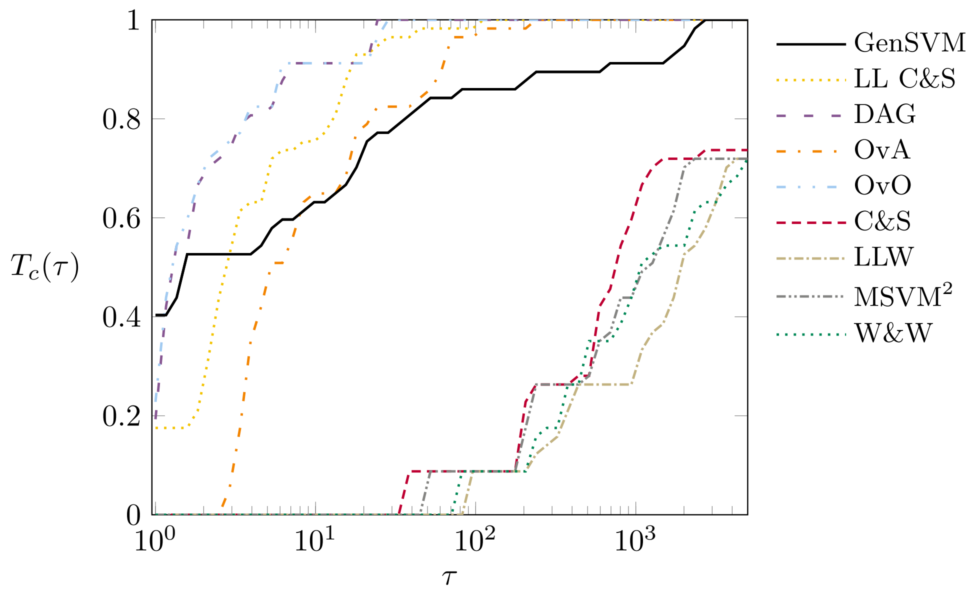


Figure 7: Performance profiles for training time. GenSVM has a priori about 40% chance of requiring the smallest time to perform the grid search on a given method. The methods implemented through MSVMpack always have a lower chance of being within a factor τ of the smallest training time than any of the other methods.

a factor of 1.08 of the best predictive performance, for $\eta \geq 1.08$ GenSVM almost always has the highest probability. It can also be concluded that since the performance profiles of the MSVMpack implementation and the LibLinear implementation of the method of Crammer and Singer (2002a) nearly always overlap, implementation differences have a negligible effect on the classification performance of this method. Finally, the figure shows that OvA and the methods of Lee et al. (2004), Crammer and Singer (2002a), Weston and Watkins (1998), and Guermeur and Monfrini (2011) always have a smaller probability of being within a given factor of the optimal performance than GenSVM, OvO, or DAG do.

Similarly, a performance profile can be constructed for the training time necessary to do the grid search. Let $t_{d,c}$ denote the total training time for classifier c on data set d. Next, define the performance ratio for time as

$$w_{d,c} = \frac{t_{d,c}}{\min\{t_{d,c} : c \in \mathcal{C}\}}.$$

Note that here the classifier with the smallest training time has preference. Therefore, comparison of classifier computation time is done with the lowest computation time achieved on a given data set d. Again, the ratio is 1 when the lowest training time is reached, and it increases for higher computation time. Hence, the performance profile for time is defined as

$$T_c(\tau) = \frac{1}{N_D} |\{d \in \mathcal{D} : w_{d,c} \leq \tau\}|.$$

The performance profile for time estimates the probability that a classifier c has a time ratio below τ. Again, $T_c(1)$ denotes the fraction of data sets where classifier c achieved the smallest training time among all classifiers.

Figure 7 shows the performance profile for the time needed to do the grid search. Since large differences in training time were observed, a logarithmic scale is used for the horizontal axis. This performance profile clearly shows that all MSVMpack methods suffer from long computation times. The fastest methods are GenSVM, OvO, and DAG, followed by the LibLinear implementation of C&S. From the value of $T_c(1)$ it is found that GenSVM has the highest probability of being the fastest method for the total grid search, with a probability of 40.35%, versus 22.81% for OvO, 19.30% for DAG, and 17.54% for LibLinear C&S. The other methods never achieve the smallest grid search time. It is important to note here that the grid search for GenSVM is 18 times larger than that of the other methods. These results illustrate the incredible advantage GenSVM has over other methods by using warm starts in the grid search.

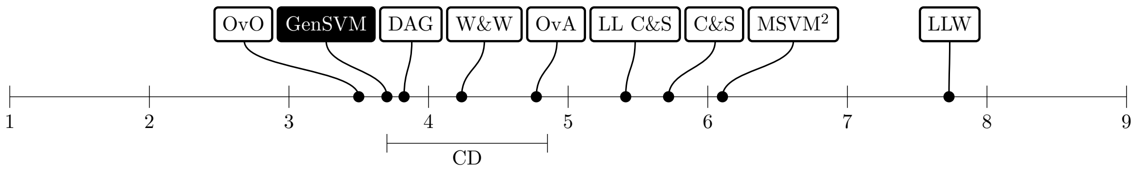
In addition to the performance profile, the average computation time per hyperparameter configuration was also examined. Here, GenSVM has an average training time of 0.97 seconds per configuration, versus 20.56 seconds for LibLinear C&S, 24.84 seconds for OvO, and 25.03 seconds for DAGSVM. This is a considerable difference, which can be explained again by the use of warm starts in GenSVM (see Section 4.3). When the total computation time per data set is averaged, it is found that GenSVM takes on average 331 seconds per data set, LibLinear C&S 391 seconds, OvO 472 seconds, and DAG 476 seconds. The difference between DAGSVM and OvO can be attributed to the prediction strategy used by DAGSVM. Thus it can be concluded that on average GenSVM is the fastest method during the grid search, despite the fact it has 18 times more hyperparameters to consider than the other methods.

6.3 Rank Tests

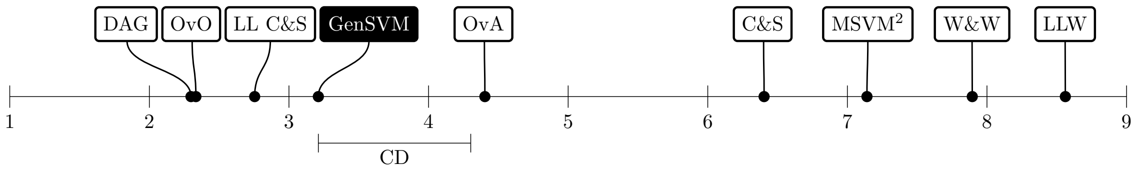
Following suggestions from Demšar (2006), ranks are used to investigate significant differences between classifiers. The benefit of using ranks instead of actual performance metrics is that ranks have meaning when averaged across different data sets, whereas average performance metrics do not. Ranks are calculated for the performance as measured by the ARI, the total training time needed to do the grid search, and the average time per hyperparameter configuration. When ties occur fractional ranks are used.

Figure 8 shows the average ranks for both classification performance and total and average training time for all classifiers. From Figure 8a it can be seen that GenSVM is in second place in terms of overall classification performance measured by the ARI. Only OvO has higher performance than GenSVM *on average*. Similarly, Figure 8b shows the average ranks for the total training time. Here, GenSVM is on average the fourth fastest method for the complete grid search. When looking at the rank plot for the average training time per hyperparameter configuration, it is clear that the warm starts used during training in GenSVM are very useful as it ranks as the fastest method on this metric, as shown in Figure 8c.

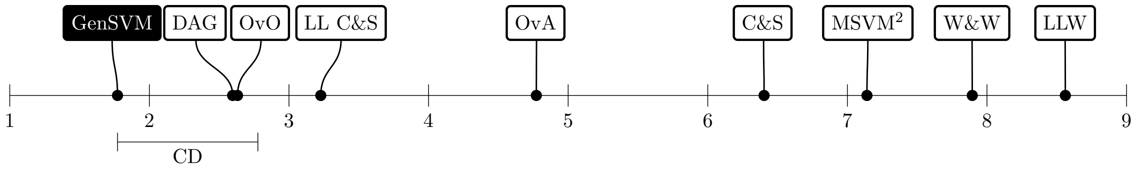
As Demšar (2006) suggests, the Friedman rank test can be used to find significant differences between classifiers (Friedman, 1937, 1940). If r_{cd} denotes the fractional rank of



(a) Classification Performance



(b) Total training time



(c) Average training time

Figure 8: Figure (a) shows the average ranks for performance, (b) shows the average ranks for the total computation time needed for the grid search, and (c) shows the ranks for the average time per hyperparameter configuration. It can be seen that GenSVM obtains the second overall rank in predictive performance, fourth overall rank in total training time, and first overall rank in average training time. In all figures, CD shows the critical difference of Holm's procedure. Classifiers beyond this CD differ significantly from GenSVM at the 5% significance level.

classifier c on data set d, then with N_C classifiers and N_D data sets the Friedman statistic is given by

$$\chi_F^2 = \frac{12N_D}{N_C(N_C+1)}\left[\sum_c R_c^2 - \frac{N_C(N_C+1)^2}{4}\right]. \tag{15}$$

Here, $R_c = 1/N_D \sum_d r_{cd}$ denotes the average rank of classifier c. This test statistic is distributed following the χ^2 distribution with $N_C - 1$ degrees of freedom. As Demšar (2006) notes, Iman and Davenport (1980) showed that the Friedman statistic is undesirably conservative and the F-statistic is to be used instead, which is given by

$$F_F = \frac{(N_D-1)\chi_F^2}{N_D(N_C-1)-\chi_F^2},$$

and is distributed following the F-distribution with $N_C - 1$ and $(N_C - 1)(N_D - 1)$ degrees of freedom. Under the null hypothesis of either test there is no significant difference in the performance of any of the algorithms.

When performing the Friedman test, it is found that with the ranks for classifier performance $\chi_F^2 = 116.3$ ($p < 10^{-16}$), and $F_F = 19.2$ ($p = 10^{-16}$). Hence, with both tests the null hypothesis of equal classification accuracy can be rejected. Similarly, for training time the test statistics are $\chi_F^2 = 384.8$ ($p < 10^{-16}$) and $F_F = 302.4$ ($p \approx 10^{-16}$). Therefore, the null hypothesis of equal training time can also be rejected. When significant differences are found through the Friedman test, Demšar (2006) suggests to use Holm's step-down procedure as a post-hoc test, to find which classifiers differ significantly from a chosen reference classifier (Holm, 1979). Here, GenSVM is used as a reference classifier, since comparing GenSVM with existing methods is the main focus of these experiments.

Holm's procedure is based on testing whether the z-statistic comparing classifier i with classifier j is significant, while adjusting for the familywise error rate. Following Demšar (2006), this z-statistic is given by

$$z = (R_{\text{GenSVM}} - R_i)\sqrt{\frac{6N_D}{N_C(N_C+1)}}, \tag{16}$$

where R_{GenSVM} is the average rank of GenSVM and R_i the average rank of another classifier, for $i = 1, \ldots, N_C - 1$. Subsequently, the p-values computed from this statistic are sorted in increasing order, as $p_1 < p_2 < \ldots < p_{N_C-1}$. Then, the null hypothesis of equal classification accuracy can be rejected if $p_i < \alpha/(N_C - i)$. If for some i the null hypothesis cannot be rejected, all subsequent tests will also fail. By inverting this procedure, a critical difference (CD) can be computed that indicates the minimal difference between the reference classifier and the next best classifier.[7] These critical differences are also illustrated in the rank plots in Figure 8.

Using Holm's procedure, it is found that for predictive performance GenSVM significantly outperforms the method of Lee et al. (2004) ($p < 10^{-14}$), the method of Guermeur and Monfrini (2011) ($p = 10^{-6}$), the MSVMpack implementation of Crammer and Singer (2002a) ($p = 4 \cdot 10^{-5}$), and the LibLinear implementation of the same method ($p = 0.0004$) at the 5% significance level. Note that since this last method is included twice these test results are conservative. In terms of total training time, GenSVM is significantly faster than all methods implemented through MSVMpack (C&S, W&W, MSVM^2, and LLW) and OvA at the 5% significance level. Recall that the hyperparameter grid for GenSVM is 18 times larger than that of the other methods. When looking at average training time per hyperparameter configuration, GenSVM is significantly faster than all methods except OvO and DAG, at the 1% significance level.

6.4 Large Data sets

The above results focus on the predictive performance of GenSVM as compared to other multiclass SVM methods. To assess the practicality of GenSVM for large data sets additional simulations were done on three more data sets. The `covtype` data set ($n = 581016$, $m = 54$, $K = 7$) and the `kddcup99` data set ($n = 494021$, $m = 116$, $K = 23$) were selected from the UCI repository (Bache and Lichman, 2013).[8] Additionally, the `fars` data set ($n = 100968$, $m = 338$, $K = 8$) was retrieved from the Keel repository (Alcalá et al.,

7. This is done by taking the smallest value of $\alpha/(N_C - i)$ for which the null hypothesis is rejected, looking up the corresponding z-statistic, and inverting (16).
8. For kddcup99 the 10% training data set and the corrected test data set are used here, both available through the UCI repository.

Package	Method	Covtype	Fars	KDDCup-99
GenSVM	GenSVM	0.3571**	0.8102***	0.9758
LibLinear	L1R-L2L	0.3372	0.8080	0.9762
LibLinear	L2R-L1L (D)	0.3405	0.7995	0.9789
LibLinear	L2R-L2L	0.3383	0.8090**	0.9781
LibLinear	L2R-L2L (D)	0.3393	0.8085*	0.9744
LibLinear	C&S	0.3582***	0.8081	0.9758
LibSVM	DAG		0.8056	0.9809***
LibSVM	OvA		0.7872	0.9800*
LibSVM	OvO		0.8055	0.9804**
MSVMpack	C&S	0.3432*	0.7996	0.9741
MSVMpack	LLW	0.3117	0.7846	0.9660
MSVMpack	MSVM^2	0.3165	0.6567	0.9658
MSVMpack	W&W	0.2848	0.7719	0.6446

Table 2: Overview of predictive performance on large data sets, as measured by the ARI. Asterisks are used to mark the three best performing methods for each data set, with three stars denoting the best performing method.

2010). For large data sets the LibLinear package (Fan et al., 2008) is often used, so the SVM methods from this package were added to the list of alternative methods.[9]

LibLinear includes five different SVM implementations: a coordinate descent algorithm for the ℓ_2-regularized ℓ_1-loss and ℓ_2-loss dual problems (Hsieh et al., 2008), a coordinate descent algorithm for the ℓ_1-regularized ℓ_2-loss SVM (Yuan et al., 2010; Fan et al., 2008), a Newton method for the primal ℓ_2-regularized ℓ_2-loss SVM problem (Lin et al., 2008), and finally a sequential dual method for the multiclass SVM by Crammer and Singer (2002a) introduced by Keerthi et al. (2008). This last method was again included to facilitate a comparison between the implementations of LibLinear and MSVMpack. Note that with the exception of this last method all methods in LibLinear are binary SVMs that implement the one-vs-all strategy.

With the different variants of the linear multiclass SVMs included in LibLinear, a total of 13 methods were considered for these large data sets. Since training of the hyperparameters for each method leads to a high computational burden the nested CV procedure was replaced by a grid search using ten-fold CV on a training set of 80% of the data, followed by out-of-sample prediction on the remaining 20% using the final model. The `kddcup99` data set comes with a separate test data set of 292302 instances, so this was used for the out-of-sample predictions. The grid search on the training set used the same hyperparameter configurations as for the small data sets above, with 342 configurations for GenSVM and 19

9. Yet another interesting SVM approach to multiclass classification is the Pegasos method by Shalev-Shwartz et al. (2011). However, the LibLinear package includes five different approaches to SVM, including a fast solver for the method by Crammer and Singer (2002a), which makes it more convenient to include in the list of methods. Moreover, according to the LibLinear documentation (Fan et al., 2008): "LibLinear is competitive or even faster than state of the art linear classifiers such as Pegasos (Shalev-Shwartz et al., 2011) and SVM$^{\text{perf}}$ (Joachims, 2006)".

		Covtype		Fars		KDDCup-99	
Package	Method	Total	Mean	Total	Mean	Total	Mean
GenSVM	GenSVM	166949	488	131174	384	1768303	5170
LibLinear	L1R-L2L	69469	3656	4199	221	34517	1817
LibLinear	L2R-L1L (D)	134908	7100	6995	368	16347	860
LibLinear	L2R-L2L	4168	219	746	39	3084	162
LibLinear	L2R-L2L (D)	159781	8410	7897	416	16974	893
LibLinear	C&S	166719	8775	124764	6567	5425	286
LibSVM	DAG	80410	40205	81557	8156	61111	3595
LibSVM	OvA	77335	77335	54965	18322	73871	12312
LibSVM	OvO	140826	46942	84580	8458	81023	4501
MSVMpack	C&S	350397	18442	351664	18509	365733	19249
MSVMpack	LLW	370790	19515	380943	20050	361329	19017
MSVMpack	MSVM2	370736	19512	346140	18218	353479	18604
MSVMpack	W&W	367245	19329	344880	18152	367685	19352

Table 3: Overview of training time for each of the large data sets. The average training time per hyperparameter configuration is also shown. All values are reported in seconds. For LibSVM the full grid search could never be completed, and results are averaged only over the finished configurations.

configurations for the other methods. The only difference was that for GenSVM $\epsilon = 10^{-9}$ was used when training the final model. To accelerate the GenSVM computations, support for sparse matrices was added.

Due to the large data set sizes, many methods had trouble converging within a reasonable amount of time. Therefore, total computation time was limited to five hours per hyperparameter configuration per method, both during CV and when training the final model. Where possible this limitation was included in the main optimization routine of each method, such that training was stopped when convergence was reached or when more than five hours had passed. Additionally, for all methods the CV procedure was stopped prematurely if more than five hours had passed after completion of a fold. In this case, cross validation performance is only measured for the folds that were completed. These computations were again performed on the Dutch National LISA Compute Cluster.

Table 2 shows the out-of-sample predictive performance of the different MSVMs on the large data sets. It can be seen that GenSVM is the best performing method on the `fars` data set and the second best method on the `covtype` data set, just after LL C&S. The LibSVM methods outperform the other methods on the `kddcup99` data set, with DAGSVM having the highest performance. No results are available for LibSVM for the `covtype` data set because convergence could not be reached within the five hour time limit during the test phase.

Results on the computation time are reported in Table 3. The ℓ_2-regularized ℓ_2-loss method by Lin et al. (2008) is clearly the fastest method. However, for the `covtype` data set GenSVM total training time is competitive with some of the other LibLinear methods, and outperforms these methods in terms of average training time. For the `fars` data set the average training time of GenSVM is also competitive with some of the LibLinear methods, most notably the method by Crammer and Singer (2002a). The MSVMpack methods seem to be infeasible for such large data sets, as computations were stopped by the five hour time limit for almost all hyperparameter configurations. Early stopping was also needed for the LibLinear implementation of C&S on the `covtype` and `fars` data sets, and for the LibSVM methods on all data sets. For GenSVM, early stopping was only needed for the `kddcup99` data set, which explains the high total computation time there. Especially on these large data sets the advantage of using warm starts in GenSVM is visible: training time was less than 30 seconds in 30% of hyperparameters on `fars`, 23% on `covtype`, and 11% on `kddcup99`.

7. Discussion

A generalized multiclass support vector machine has been introduced, called GenSVM. The method is general in the sense that it subsumes three multiclass SVMs proposed in the literature and it is flexible due to several different weighting options. The simplex encoding of the multiclass classification problem used in GenSVM is intuitive and has an elegant geometrical interpretation. An iterative majorization algorithm has been derived to minimize the convex GenSVM loss function in the primal. This primal optimization approach has computational advantages due to the possibility to use warm starts, and because it can be easily understood. The ability to use warm starts contributes to small training time during cross validation in a grid search, and allows GenSVM to perform competitively on large data sets.

Rigorous computational tests of linear multiclass SVMs on small data sets show that GenSVM significantly outperforms three existing multiclass SVMs (four implementations) on predictive performance at the 5% significance level. On this metric, GenSVM is the second-best performing method overall and the best method among single-machine multiclass SVMs, although the difference with the method of Weston and Watkins (1998) could not be shown to be statistically significant. GenSVM outperforms five other methods on total training time and has the smallest total training time when averaged over all data sets, despite the fact that its grid of hyperparameters is 18 times larger than that of other methods. Due to the possibility of warm starts it also has the smallest average training time per hyperparameter and significantly outperforms all but two alternative methods in this regard at the 1% significance level. For the large data sets, it was found that GenSVM still achieves high classification accuracy and that total training time remains manageable due to the warm starts. In practice, the number of hyperparameters could be reduced if smaller training time is desired. Since GenSVM outperforms existing methods on a number of data sets and achieves fast training time it is a worthwhile addition to the collection of methods available to the practitioner.

In the comparison tests MSVMpack (Lauer and Guermeur, 2011) was used to access four single machine multiclass SVMs proposed in the literature. A big advantage of using this library is that it allows for a single straightforward `C` implementation, which greatly reduces the programming effort needed for the comparisons. However, as is noted in the MSVMpack documentation, slight differences exist between MSVMpack and method-specific implementations. For instance, on small data sets MSVMpack can be slower, due to working set selection and shrinking procedures in other implementations. However, classification performance is comparable between MSVMpack and method-specific implementations, as was verified by adding the LibLinear implementation of the method of Crammer and Singer (2002a) to the list of alternative methods. Thus, we argue that the results for predictive accuracy presented above are accurate regardless of implementation, but small differences can exist for training time when other implementations for single machine MSVMs are used.

Another interesting conclusion that can be drawn from the experimental results is that the one-vs-all method never performs as good as one-vs-one, DAGSVM, or GenSVM. In fact, the profile plot in Figure 6 shows that OvA always has a smaller probability of obtaining the best classification performance as either of these three methods. These results are also reflected in the classification accuracy of the LibLinear methods on the large data set. In the literature, the paper by Rifkin and Klautau (2004) is often cited as evidence that OvA performs well (see for instance Keerthi et al., 2008). However, the simulation results in this paper suggest that OvA is in fact inferior to OvO, DAG, and GenSVM.

This paper was focused on linear multiclass SVMs. An obvious extension is to incorporate nonlinear multiclass SVMs through kernels. Due to the large number of data sets and the long training time the numerical experiments were limited to linear multiclass SVM. Nonlinear classification through kernels can be achieved by linear methods through a preprocessing step of an eigendecomposition on the kernel matrix, which is a process of the order $O(n^3)$. In this case, GenSVM will benefit from precomputing kernels before starting the grid search, or using a larger stopping criterion in the IM algorithm by increasing ϵ in Algorithm 1. In addition, approximations can be done by using rank approximated kernel matrices, such as the Nyström method proposed by Williams and Seeger (2001). Such enhancements are considered topics for further research.

Finally, the potential of using GenSVM in an online setting is recognized. Since the solution can be found quickly when a warm-start is used, GenSVM may be useful in situations where new instances have to be predicted at a certain moment, and the true class label arrives later. Then, re-estimating the GenSVM solution can be done as soon as the true class label of an object arrives, and a previously known solution can be used as a warm start. It is expected that in this scenario only a few iterations of the IM algorithm are needed to arrive at a new optimal solution. This, too, is considered a subject for further research.

Acknowledgments

The computational experiments of this work were performed on the Dutch National LISA Compute Cluster, and supported by the Dutch National Science Foundation (NWO). The authors thank SURFsara (`www.surfsara.nl`) for the support in using the LISA cluster.

Appendix A. Simplex Coordinates

The simplex used in the formulation of the GenSVM loss function is a regular K-simplex in $\mathbb{R}^{K-1}$ with distance 1 between each pair of vertices, which is centered at the origin. Since these requirements alone do not uniquely define the simplex coordinates in general, it will be chosen such that at least one of the vertices lies on an axis. The 2-simplex in $\mathbb{R}^1$ is uniquely defined with the coordinates $-\frac{1}{2}$ and $+\frac{1}{2}$. Using these requirements, it is possible to define a recursive formula for $\mathbf{U}_K$, the simplex coordinate matrix of the K-simplex in $\mathbb{R}^{K-1}$ as

$$\mathbf{U}_K = \begin{bmatrix} \mathbf{U}_{K-1} & \mathbf{1}t \\ \mathbf{0}' & s \end{bmatrix}, \qquad \text{with} \quad \mathbf{U}_2 = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \end{bmatrix}.$$

Note that the matrix $\mathbf{U}_K$ has K rows and $K-1$ columns. Since the simplex is centered at zero it holds that the elements in each column sum to 0, implying that $s = -(K-1)t$. Denote by $\mathbf{u}_i'$ the i-th row of $\mathbf{U}_K$ and by $\tilde{\mathbf{u}}_i'$ the i-th row of $\mathbf{U}_{K-1}$, then it follows from the edge length requirement that

$$\|\mathbf{u}_i' - \mathbf{u}_K'\|^2 = \|\tilde{\mathbf{u}}_i' - \mathbf{0}' + t - s\|^2 = \|\tilde{\mathbf{u}}_i'\|^2 + (t-s)^2 = 1, \quad \forall i \neq K.$$

From the requirement of equal distance from each vertex to the origin it follows that

$$\begin{aligned} \|\mathbf{u}_i'\|^2 &= \|\mathbf{u}_K'\|^2, \\ \|\tilde{\mathbf{u}}_i'\|^2 + t^2 &= s^2, \quad \forall i \neq K. \end{aligned}$$

Combining these two expressions yields the equation $2s^2 - 2st - 1 = 0$. Substituting $s = -(K-1)t$ and choosing $s > 0$ and $t < 0$ gives

$$t = \frac{-1}{\sqrt{2K(K-1)}}, \quad s = \frac{K-1}{\sqrt{2K(K-1)}}.$$

Note that using $K = 2$ in these expressions gives $t = -\frac{1}{2}$ and $s = \frac{1}{2}$, as expected. The recursive relationship defined above then reveals that the first $K-1$ elements in column $K-1$ of the matrix are equal to t, and the K-th element in column $K-1$ is equal to s. This can then be generalized for an element u_{kl} in row k and column l of $\mathbf{U}_K$, yielding the expression given in (1).

Appendix B. Details of Iterative Majorization

In this section a brief introduction to iterative majorization is given, following the description of Voss and Eckhardt (1980). The section concludes with a note on step doubling, a common technique to speed up quadratic majorization algorithms.

Given a continuous function $f : \mathcal{X} \to \mathbb{R}$ with $\mathcal{X} \subseteq \mathbb{R}^d$, construct a *majorization function* $g(x, \overline{x})$ such that

$$\begin{aligned} &f(\overline{x}) = g(\overline{x}, \overline{x}), \\ &f(x) \leq g(x, \overline{x}) \text{ for all } x \in \mathcal{X}, \end{aligned}$$

with $\overline{x} \in \mathcal{X}$ a so-called *supporting point*. In general, the majorization function is constructed such that its minimum can easily be found, for instance by choosing it to be quadratic in x. If $f(x)$ is differentiable at the supporting point, the above conditions imply $\nabla f(\overline{x}) = \nabla g(\overline{x}, \overline{x})$. The following procedure can now be used to find a stationary point of $f(x)$,

1. Let $\overline{x} = x_0$, with x_0 a random starting point.

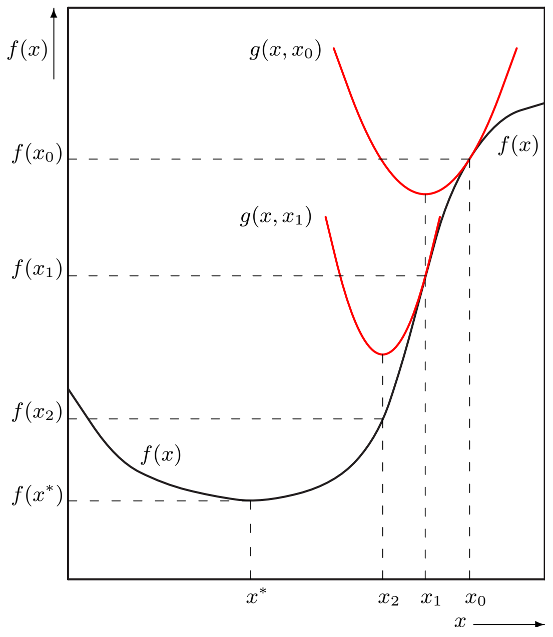


Figure 9: One-dimensional graphical illustration of the iterative majorization algorithm, adapted from De Leeuw (1988). The minimum of a majorization function $g(x, x_r)$ provides the supporting point for the next majorization function $g(x, x_{r+1})$. The sequence of supporting points $\{x_r\}$ converges towards the stationary point x^* if $f(x)$ is bounded from below, as is the case here.

2. Minimize $g(x, \overline{x})$ with respect to x, such that $x^+ = \arg\min g(x, \overline{x})$.

3. If $f(\overline{x}) - f(x^+) < \epsilon f(x^+)$ stop, otherwise let $\overline{x} = x^+$ and go to step 2.

In this algorithm ϵ is a small constant. Note that $f(x)$ must be bounded from below on $\mathcal{X}$ for the algorithm to converge. In fact, the following *sandwich inequality* can be derived (De Leeuw, 1993)

$$f(x^+) \leq g(x^+, \overline{x}) \leq g(\overline{x}, \overline{x}) = f(\overline{x}).$$

This inequality shows that if $f(x)$ is bounded from below the iterative majorization algorithm achieves global convergence to a stationary point of the function (Voss and Eckhardt, 1980). The iterative majorization algorithm is illustrated in Figure 9, where the majorization functions are shown as a quadratic function. As can be seen from the illustration, the sequence of supporting points $\{x_r\}$ converges to the stationary point x^* of the function $f(x)$. In practical situations, this convergence is to a local minimum of $f(x)$.

For quadratic majorization the number of iterations can often be reduced by using a technique known as *step doubling* (De Leeuw and Heiser, 1980). Step doubling reduces the number of iterations by using $\overline{x} = x_{r+1} = 2x^+ - x_r$ as the next supporting point in Step 3 of the algorithm, instead of $\overline{x} = x_{r+1} = x^+$. Intuitively, step doubling can be understood as stepping over the minimum of the majorization function to the point lying directly "opposite" the supporting point $\overline{x}$ (see also Figure 9). Note that the guaranteed descent of the IM algorithm still holds when using step doubling, since $f(2x^+ - \overline{x}) \leq g(2x^+ - \overline{x}, \overline{x}) = g(\overline{x}, \overline{x}) = f(\overline{x})$. In practice, step doubling reduces the number of iterations by half. A caveat of using step doubling is that the distance to the stationary point can be increased if the initial point is far from this point. Therefore, in practical applications, a burn-in should be used before step doubling is applied.

Appendix C. Huber Hinge Majorization

In this appendix, the majorization function will be derived of the Huber hinge error raised to the power p. Thus, a quadratic function $g(x, \overline{x}) = ax^2 - 2bx + c$ is required, which is a majorization function of

$$f(x) = h^p(x) = \begin{cases} \left(1 - x - \frac{\kappa+1}{2}\right)^p & \text{if } x \leq -\kappa \\ \frac{1}{(2(\kappa+1))^p}(1-x)^{2p} & \text{if } x \in (-\kappa, 1] \\ 0 & \text{if } x > 1, \end{cases}$$

with $p \in [1, 2]$. Each piece of $f(x)$ provides a possible region for the supporting point $\overline{x}$. These regions will be treated separately, starting with $\overline{x} \in (-\kappa, 1]$.

Since the majorization function must touch $f(x)$ at the supporting point, we can solve $f(\overline{x}) = g(\overline{x}, \overline{x})$ and $f'(\overline{x}) = g'(\overline{x}, \overline{x})$ for b and c to find

$$b = a\overline{x} + \frac{p}{1-\overline{x}} \left(\frac{1-\overline{x}}{\sqrt{2(\kappa+1)}} \right)^{2p}, \tag{17}$$

$$c = a\overline{x}^2 + \left(1 + \frac{2p\overline{x}}{1-\overline{x}}\right) \left(\frac{1-\overline{x}}{\sqrt{2(\kappa+1)}} \right)^{2p}, \tag{18}$$

whenever $\overline{x} \in (-\kappa, 1]$. Note that since $p \in [1, 2]$ the function $f(x)$ can become proportional to a fourth power on the interval $x \in (-\kappa, 1]$. The upper bound of the second derivative of $f(x)$ on this interval is reached at $x = -\kappa$. Equating $f''(-\kappa)$ to $g''(-\kappa, \overline{x}) = 2a$ and solving for a yields

$$a = \tfrac{1}{4}p(2p-1)\left(\frac{\kappa+1}{2}\right)^{p-2}. \tag{19}$$

Figure 10a shows an illustration of the majorization function when $\overline{x} \in (-\kappa, 1]$.

For the interval $\overline{x} \leq -\kappa$ the following expressions are found for b and c using similar reasoning as above

$$b = a\overline{x} + \tfrac{1}{2}p\left(1 - \overline{x} - \frac{\kappa+1}{2}\right)^{p-1}, \tag{20}$$

$$c = a\overline{x}^2 + p\overline{x}\left(1 - \overline{x} - \frac{\kappa+1}{2}\right)^{p-1} + \left(1 - \overline{x} - \frac{\kappa+1}{2}\right)^p. \tag{21}$$

To obtain the largest possible majorization step it is desired that the minimum of the majorization function is located at $x \geq 1$, such that $g(x_{min}, \overline{x}) = 0$. This requirement yields

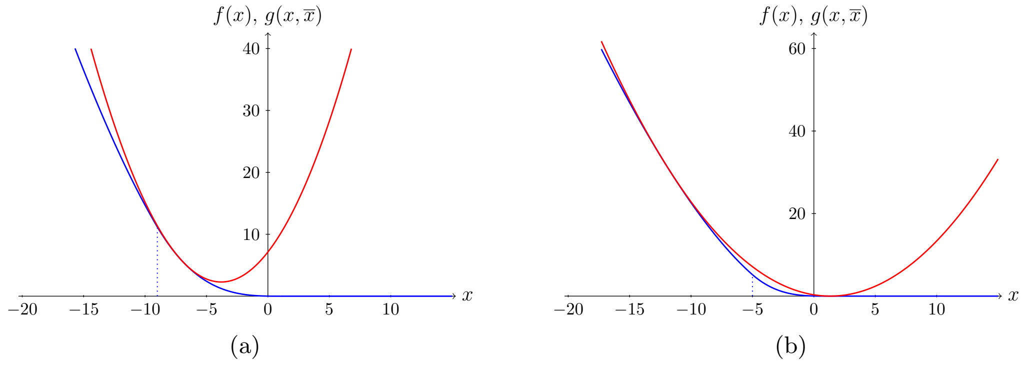


Figure 10: Graphical illustration of the majorization of the function $f(x) = h^p(x)$. Figure (a) shows the case where $\overline{x} \in (-\kappa, 1]$, whereas (b) shows the case where $\overline{x} \leq (p + \kappa - 1)/(p - 2)$. In both cases $p = 1.5$. It can be seen that in (b) the minimum of the majorization function lies at $x > 1$, such that the largest possible majorization step is obtained.

$c = b^2/a$, which gives

$$a = \tfrac{1}{4}p^2 \left(1 - \overline{x} - \frac{\kappa + 1}{2}\right)^{p-2}. \tag{22}$$

Note however that due to the requirement that $f(x) \leq g(x, \overline{x})$ for all $x \in \mathbb{R}$, this majorization is not valid for all values of $\overline{x}$. Solving the requirement for the minimum of the majorization function, $g(x_{min}, \overline{x}) = 0$ for $\overline{x}$ yields

$$\overline{x} \leq \frac{p + \kappa - 1}{p - 2}.$$

Thus, if $\overline{x}$ satisfies this condition, (22) can be used for a, whereas for cases where $\overline{x} \in ((p + \kappa - 1)/(p - 2), -\kappa]$, the value of a given in (19) can be used. Figure 10b shows an illustration of the case where $\overline{x} \leq (p + \kappa - 1)/(p - 2)$.

Next, a majorization function for the interval $\overline{x} > 1$ is needed. Since it has been derived that for the interval $\overline{x} \leq (p + \kappa - 1)/(p - 2)$ the minimum of the majorization function lies at $x \geq 1$, symmetry arguments can be used to derive the majorization function for $\overline{x} > 1$, and ensure that it is also tangent at $x = (p\overline{x} + \kappa - 1)/(p - 2)$. This yields the coefficients

$$a = \tfrac{1}{4}p^2 \left(\frac{p}{p-2}\left(1 - \overline{x} - \frac{\kappa + 1}{2}\right)\right)^{p-2}, \tag{23}$$

$$b = a \left(\frac{p\overline{x} + \kappa - 1}{p - 2}\right) + \tfrac{1}{2}p \left(\frac{p}{p-2}\left(1 - \overline{x} - \frac{\kappa + 1}{2}\right)\right)^{p-1}, \tag{24}$$

$$c = a \left(\frac{p\overline{x} + \kappa - 1}{p - 2}\right)^2 + p \left(\frac{p\overline{x} + \kappa - 1}{p - 2}\right) \left(\frac{p}{p-2}\left(1 - \overline{x} - \frac{\kappa + 1}{2}\right)\right)^{p-1} + \left(\frac{p}{p-2}\left(1 - \overline{x} - \frac{\kappa + 1}{2}\right)\right)^{p}. \tag{25}$$

Region	a	b	c
$\overline{x} \leq \dfrac{p+\kappa-1}{p-2}$	(22)	(20)	(21)
$\overline{x} \in \left(\dfrac{p+\kappa-1}{p-2}, -\kappa\right]$	(19)	(20)	(21)
$\overline{x} \in (-\kappa, 1]$	(19)	(17)	(18)
$\overline{x} > 1,\, p \neq 2$	(23)	(24)	(25)
$\overline{x} > 1,\, p = 2$	(19)	$a\overline{x}$	$a\overline{x}^2$

Table 4: Overview of quadratic majorization coefficients for different pieces of $h^p(x)$, depending on $\overline{x}$.

Finally, observe that some of the above coefficients are invalid if $p = 2$. However, since the upper bound on the interval $\overline{x} \in (-\kappa, 1]$ given in (19) is still valid if $p = 2$, it is possible to do a separate derivation with this value for a to find for $\overline{x} > 1$, $b = a\overline{x}$ and $c = a\overline{x}^2$. For the other regions the previously derived coefficients still hold. Table 4 gives an overview of the various coefficients depending on the location of $\overline{x}$.

Appendix D. Kernels in GenSVM

To include kernels in GenSVM a preprocessing step is needed on the kernel matrix, and a postprocessing step is needed on the obtained parameters before doing class prediction. Let $k : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^+$ denote a positive definite kernel satisfying Mercer's theorem, and let $\mathcal{H}_k$ denote the corresponding reproducing kernel Hilbert space. Furthermore, define a feature mapping $\phi : \mathbb{R}^m \to \mathcal{H}_k$ as $\phi(\mathbf{x}) = k(\mathbf{x}, \cdot)$, such that by the reproducing property of k it holds that $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle_{\mathcal{H}_k}$.

Using this, the kernel matrix $\mathbf{K}$ is defined as the $n \times n$ matrix with elements $k(\mathbf{x}_i, \mathbf{x}_j)$ on the i-th row and j-th column. Thus, if $\mathbf{\Phi}$ denotes the $n \times l$ matrix with rows $\phi(\mathbf{x}_i)$ for $i = 1, \ldots, n$ and $l \in [1, \infty]$, then $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}'$. Note that it depends on the chosen kernel whether $\mathbf{\Phi}$ is finite dimensional. However, the rank of $\mathbf{\Phi}$ can still be determined through $\mathbf{K}$, since $r = \text{rank}(\mathbf{\Phi}) = \text{rank}(\mathbf{K}) \leq \min(n, l)$.

Now, let the reduced singular value decomposition of $\mathbf{\Phi}$ be given by

$$\mathbf{\Phi} = \mathbf{P}\mathbf{\Sigma}\mathbf{Q}',$$

where $\mathbf{P}$ is $n \times r$, $\mathbf{\Sigma}$ is $r \times r$, and $\mathbf{Q}$ is $l \times r$. Note that here, $\mathbf{P}'\mathbf{P} = \mathbf{I}_r$, $\mathbf{Q}'\mathbf{Q} = \mathbf{I}_r$, and $\mathbf{\Sigma}$ is diagonal. Under the mapping $\mathbf{X} \to \mathbf{\Phi}$ it follows that the simplex space vectors become

$$\begin{aligned} \mathbf{S} &= \mathbf{\Phi}\mathbf{W} + \mathbf{1}\mathbf{t}' \\ &= \mathbf{P}\mathbf{\Sigma}\mathbf{Q}'\mathbf{W} + \mathbf{1}\mathbf{t}' \\ &= \mathbf{M}\mathbf{Q}'\mathbf{W} + \mathbf{1}\mathbf{t}'. \end{aligned}$$

Here $\mathbf{W}$ is $l \times (K-1)$ to correspond to the dimensions of $\mathbf{\Phi}$, and the $n \times r$ matrix $\mathbf{M} = \mathbf{P\Sigma}$ has been introduced. In general $\mathbf{W}$ cannot be determined, since l might be infinite. This problem can be solved as follows. Decompose $\mathbf{W}$ in two parts, $\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2$, where $\mathbf{W}_1$ is in the linear space of $\mathbf{Q}$ and $\mathbf{W}_2$ is orthogonal to that space, thus

$$\begin{aligned}\mathbf{W}_1 &= \mathbf{QQ}'\mathbf{W},\\ \mathbf{W}_2 &= (\mathbf{I}_l - \mathbf{QQ}')\mathbf{W}.\end{aligned}$$

Then it follows that

$$\begin{aligned}\mathbf{S} &= \mathbf{MQ}'\mathbf{W} + \mathbf{1t}'\\ &= \mathbf{MQ}'(\mathbf{W}_1 + \mathbf{W}_2) + \mathbf{1t}'\\ &= \mathbf{MQ}'(\mathbf{W}_1 + (\mathbf{I}_l - \mathbf{QQ}')\mathbf{W}) + \mathbf{1t}'\\ &= \mathbf{MQ}'\mathbf{W}_1 + \mathbf{M}(\mathbf{Q}' - \mathbf{Q}'\mathbf{QQ}')\mathbf{W} + \mathbf{1t}'\\ &= \mathbf{MQ}'\mathbf{W}_1 + \mathbf{M}(\mathbf{Q}' - \mathbf{Q}')\mathbf{W} + \mathbf{1t}'\\ &= \mathbf{MQ}'\mathbf{W}_1 + \mathbf{1t}'\\ &= \mathbf{MQ}'\mathbf{W}_1 + \mathbf{1t}',\end{aligned}$$

where it has been used that $\mathbf{Q}'\mathbf{Q} = \mathbf{I}_r$. If the penalty term of the GenSVM loss function is considered, it is found that

$$P_\lambda(\mathbf{W}) = \lambda \operatorname{tr} \mathbf{W}'\mathbf{W} = \lambda \operatorname{tr} \mathbf{W}_1'\mathbf{W}_1 + \lambda \operatorname{tr} \mathbf{W}_2'\mathbf{W}_2,$$

since

$$\begin{aligned}\mathbf{W}_1'\mathbf{W}_2 &= \mathbf{W}'\mathbf{QQ}'(\mathbf{I}_l - \mathbf{QQ}')\mathbf{W}\\ &= \mathbf{W}'\mathbf{QQ}'\mathbf{W} - \mathbf{W}'\mathbf{QQ}'\mathbf{W}\\ &= \mathbf{O}.\end{aligned}$$

Here again it has been used that $\mathbf{Q}'\mathbf{Q} = \mathbf{I}_r$, and $\mathbf{O}$ is defined as a $(K-1) \times (K-1)$ dimensional matrix of zeroes. Note that the penalty term depends on $\mathbf{W}_2$ whereas the simplex vectors $\mathbf{S}$ do not. Therefore, at the optimal solution it is required that $\mathbf{W}_2$ is zero, to minimize the loss function.

Since $\mathbf{W}_1$ is still $l \times (K-1)$ dimensional with l possibly infinite, consider the substitution $\mathbf{W}_1 = \mathbf{Q\Omega}$, with $\mathbf{\Omega}$ an $r \times (K-1)$ matrix. The penalty term in terms of $\mathbf{\Omega}$ then becomes

$$P_\lambda(\mathbf{W}_1) = \lambda \operatorname{tr} \mathbf{W}_1'\mathbf{W}_1 = \lambda \operatorname{tr} \mathbf{\Omega}'\mathbf{Q}'\mathbf{Q\Omega} = \lambda \operatorname{tr} \mathbf{\Omega}'\mathbf{\Omega} = P_\lambda(\mathbf{\Omega}).$$

Note also that

$$\begin{aligned}\mathbf{S} &= \mathbf{MQ}'\mathbf{W}_1 + \mathbf{1t}'\\ &= \mathbf{MQ}'\mathbf{Q\Omega} + \mathbf{1t}'\\ &= \mathbf{M\Omega} + \mathbf{1t}'.\end{aligned}$$

The question remains on how to determine the matrices $\mathbf{P}$ and $\mathbf{\Sigma}$, given that the matrix $\mathbf{\Phi}$ cannot be determined explicitly. These matrices can be determined by the eigendecomposition of $\mathbf{K}$, where $\mathbf{K} = \mathbf{P\Sigma}^2\mathbf{P}'$. In the case where $r < n$, $\mathbf{\Sigma}^2$ contains only the first r

eigenvalues of $\mathbf{K}$, and $\mathbf{P}$ the corresponding r columns. Hence, if $\mathbf{K}$ is not of full rank, a dimensionality reduction is achieved in $\mathbf{\Omega}$. The complexity of finding the eigendecomposition of the kernel matrix is $O(n^3)$.

Since the distances $q_i^{(kj)}$ in the GenSVM loss function can be written as $q_i^{(kj)} = \mathbf{s}_i'\boldsymbol{\delta}_{kj}$ it follows that the errors can again be calculated in this formulation. Finally, to predict the simplex space vectors of a test set $\mathbf{X}_2$ the following is used. Let $\mathbf{\Phi}_2$ denote the feature space mapping of $\mathbf{X}_2$, then

$$\begin{aligned}\mathbf{S}_2 &= \mathbf{\Phi}_2\mathbf{W}_1 + \mathbf{1}\mathbf{t}' \\ &= \mathbf{\Phi}_2\mathbf{Q}\mathbf{\Omega} + \mathbf{1}\mathbf{t}' \\ &= \mathbf{\Phi}_2\mathbf{Q}\mathbf{\Sigma}\mathbf{P}'\mathbf{P}\mathbf{\Sigma}^{-1}\mathbf{\Omega} + \mathbf{1}\mathbf{t}' \\ &= \mathbf{\Phi}_2\mathbf{\Phi}'\mathbf{P}\mathbf{\Sigma}^{-1}\mathbf{\Omega} + \mathbf{1}\mathbf{t}' \\ &= \mathbf{K}_2\mathbf{P}\mathbf{\Sigma}^{-1}\mathbf{\Omega} + \mathbf{1}\mathbf{t}' \\ &= \mathbf{K}_2\mathbf{M}\mathbf{\Sigma}^{-2}\mathbf{\Omega} + \mathbf{1}\mathbf{t}',\end{aligned}$$

where $\mathbf{K}_2 = \mathbf{\Phi}_2\mathbf{\Phi}'$ is the kernel matrix between the test set and the training set, and it was used that $\mathbf{\Sigma}\mathbf{P}'\mathbf{P}\mathbf{\Sigma}^{-1} = \mathbf{I}_r$, and $\mathbf{\Phi}' = \mathbf{Q}\mathbf{\Sigma}\mathbf{P}'$ by definition.

With the above expressions for $\mathbf{S}$ and $P_\lambda(\mathbf{\Omega})$, it is possible to derive the majorization function of the loss function for the nonlinear case. The first order conditions can then again be determined, which yields the following system

$$\left(\begin{bmatrix}\mathbf{1}' \\ \mathbf{M}'\end{bmatrix}\mathbf{A}\begin{bmatrix}\mathbf{1} & \mathbf{M}\end{bmatrix} + \lambda\begin{bmatrix}0 & \mathbf{0}' \\ \mathbf{0} & \mathbf{I}_r\end{bmatrix}\right)\begin{bmatrix}\mathbf{t}' \\ \mathbf{\Omega}\end{bmatrix} = \begin{bmatrix}\mathbf{1}' \\ \mathbf{M}'\end{bmatrix}\mathbf{A}\begin{bmatrix}\mathbf{1} & \mathbf{M}\end{bmatrix}\begin{bmatrix}\bar{\mathbf{t}}' \\ \bar{\mathbf{\Omega}}\end{bmatrix} + \begin{bmatrix}\mathbf{1}' \\ \mathbf{M}'\end{bmatrix}\mathbf{B}. \tag{26}$$

This system is analogous to the system solved in linear GenSVM. In fact, it can be shown that by writing $\mathbf{Z} = [\mathbf{1}\ \mathbf{M}]$ and $\mathbf{V} = [\mathbf{t}'\ \mathbf{\Omega}]'$, this system is equivalent to (14). This property is very useful for the implementation of GenSVM, since nonlinearity can be included by simply adding a preprocessing and postprocessing step to the existing GenSVM algorithm.

Appendix E. Additional Simulation Results

Tables 5 and 6 respectively show the predictive accuracy rates and ARI scores on each data set averaged over each of the 5 test folds. For readability all scores are rounded to four decimal digits, however identifying the classifier with the highest score was done on the full precision scores. As can be seen, the choice of performance metric has an effect on which classification method has the highest classification performance. Regardless of the performance metric the tables show that MSVM^2 and W&W never achieve the maximum classification performance on a data set. Note that conclusions drawn from tables of performance scores are quite limited and the results presented in Section 6 provide more insight into the performance of the various classifiers.

Table 7 shows the computation time averaged over the five nested CV folds for each data set and each method. In the grid search GenSVM considered 342 hyperparameter configurations versus 19 configurations for the other methods. Despite this difference GenSVM outperformed the other methods on five data sets, DAG outperformed other methods on four data sets, OvO on two, and LibLinear C&S was fastest on the remaining two data sets. To illustrate the effect of the larger grid search in GenSVM on the computation time, Table 8 shows the average computation time per hyperparameter configuration. This table shows that GenSVM is faster than other methods on nine out of thirteen data sets, which illustrates the influence of warm starts in the GenSVM grid search.

Data set	GenSVM	LL C&S	DAG	OvA	OvO	C&S	LLW	MSVM^2	W&W
balancescale	0.9168	0.8883	0.9168	0.8928	0.9168	0.8922	0.8701	0.8714	0.9008
breasttissue	0.7113	0.7005	0.6515	0.7455	0.6515	0.6944	0.5391	0.6663	0.5711
car	0.8279	0.6185	0.8449	0.8131	0.8524	0.6489	0.7898	0.7855	0.8273
contraception	0.5027	0.4773	0.5017	0.4739	0.5010	0.4699	0.4751	0.4964	0.4972
ecoli	0.8630	0.8547	0.8629	0.8510	0.8659	0.8576	0.7450	0.8456	0.8098
glass	0.6448	0.5813	0.6542	0.5746	0.6450	0.6342	0.4504	0.5988	0.6215
imageseg	0.9169	0.9103	0.9162	0.9210	0.9219	0.9088	0.7741	0.8157	0.8962
iris	0.9600	0.8893	0.9533	0.9467	0.9533	0.8813	0.7640	0.8320	0.9253
vehicle	0.8016	0.7978	0.7955	0.7872	0.7990	0.7941	0.6870	0.7550	0.7993
vertebral	0.8323	0.8458	0.8355	0.8484	0.8419	0.8439	0.7890	0.8432	0.8426
vowel	0.4762	0.4242	0.4957	0.3398	0.5065	0.4221	0.2273	0.3277	0.5017
wine	0.9776	0.9841	0.9608	0.9775	0.9719	0.9775	0.9843	0.9775	0.9717
yeast	0.5343	0.5841	0.5748	0.5175	0.5802	0.5818	0.4217	0.5590	0.5811

Table 5: Predictive accuracy rates for each of the classification methods on all data sets. All numbers are out-of-sample prediction accuracies averaged over the 5 independent test folds. Maximum scores per data set are determined on the full precision scores and are underlined.

Data set	GenSVM	LL C&S	DAG	OvA	OvO	C&S	LLW	MSVM^2	W&W
balancescale	0.8042	0.7355	0.8042	0.7238	0.8042	0.7466	0.6634	0.6653	0.7698
breasttissue	0.5222	0.4964	0.4591	0.5723	0.4787	0.4755	0.4043	0.5585	0.4655
car	0.5381	0.3290	0.5545	0.5238	0.5491	0.3131	0.4874	0.4808	0.5337
contraception	0.0762	0.0532	0.0757	0.0535	0.0747	0.0525	0.0393	0.0658	0.0699
ecoli	0.7668	0.7606	0.7755	0.7652	0.7776	0.7578	0.6236	0.7499	0.7385
glass	0.2853	0.2478	0.2970	0.2346	0.2910	0.2792	0.1776	0.2494	0.2861
imageseg	0.8318	0.8221	0.8280	0.8402	0.8390	0.8193	0.6422	0.6810	0.7991
iris	0.8783	0.7057	0.8609	0.8384	0.8609	0.6879	0.5549	0.6057	0.7918
vehicle	0.6162	0.6009	0.6057	0.5979	0.6057	0.5925	0.4933	0.5397	0.6121
vertebral	0.6649	0.6797	0.6606	0.6862	0.6778	0.6742	0.6480	0.6859	0.6836
vowel	0.2472	0.2474	0.2895	0.1624	0.3218	0.2257	0.1425	0.2043	0.2767
wine	0.9320	0.9585	0.8848	0.9378	0.9200	0.9362	0.9498	0.9332	0.9250
yeast	0.2519	0.2501	0.2415	0.2419	0.2477	0.2534	0.1301	0.2235	0.2501

Table 6: Predictive ARI scores for each of the classification methods on all data sets. All numbers are out-of-sample ARI scores averaged over the 5 independent test folds. Maximum scores per data set are determined on the full precision scores and are underlined.

Data set	GenSVM	LL C&S	DAG	OvA	OvO	C&S	LLW	MSVM^2	W&W
balancescale	44.3	88.0	86.4	155.8	84.9	34549	73671	79092	35663
breasttissue	136.0	52.9	3.8	65.2	3.8	28782	74188	38625	81961
car	251.2	1239.1	1513.0	4165.2	1517.4	47408	95197	46978	85050
contraception	82.5	1128.5	1948.3	5079.1	1913.4	45163	88844	43402	40335
ecoli	603.0	88.9	34.7	183.2	34.8	28907	95989	39590	131571
glass	254.6	110.8	48.6	198.2	47.7	27938	89073	37194	108499
imageseg	558.2	67.1	2.4	151	3.2	32691	73300	48576	97218
iris	55.7	13.8	1.9	32.9	1.5	12822	47196	38060	77409
vehicle	186.4	376.8	307.9	1373.3	309.9	37605	49988	40665	43511
vertebral	23.5	66.6	24.4	63.1	24.3	24716	70798	36168	23888
vowel	1282.4	463.9	83.7	3900.2	86.1	36270	95036	49924	82990
wine	129.6	0.1	0.2	0.2	0.2	12854	70439	18389	41018
yeast	1643.3	1181.7	1434.6	4251.1	1423.6	44112	103240	56603	86802

Table 7: Computation time in seconds for each of the methods on all data sets. Values are averaged over the five nested CV splits. Minimum values per data set are underlined. Note that the size of the grid search is 18 times larger in GenSVM than in other methods.

Data set	GenSVM	LL C&S	DAG	OvA	OvO	C&S	LLW	MSVM^2	W&W
balancescale	0.130	4.632	4.546	8.199	4.468	1818	3877	4163	1877
breasttissue	0.398	2.785	0.201	3.434	0.201	1515	3905	2033	4314
car	0.734	65.217	79.629	219.221	79.863	2495	5010	2473	4476
contraception	0.241	59.396	102.544	267.319	100.704	2377	4676	2284	2123
ecoli	1.763	4.680	1.828	9.643	1.831	1521	5052	2084	6925
glass	0.744	5.832	2.559	10.432	2.511	1470	4688	1958	5710
imageseg	1.632	3.530	0.128	7.947	0.167	1721	3858	2557	5117
iris	0.163	0.725	0.101	1.729	0.081	675	2484	2003	4074
vehicle	0.545	19.830	16.206	72.282	16.308	1979	2631	2140	2290
vertebral	0.069	3.504	1.286	3.32	1.279	1301	3726	1904	1257
vowel	3.750	24.415	4.406	205.274	4.533	1909	5002	2628	4368
wine	0.379	0.003	0.010	0.011	0.010	677	3707	968	2159
yeast	4.805	62.197	75.506	223.74	74.925	2322	5434	2979	4569

Table 8: Average computation time in seconds per hyperparameter configuration for each of the methods on all data sets. Values are averaged over the five nested CV splits. Minimum values per data set are determined on the full precision values and are underlined.

References

A. Aizerman, E.M. Braverman, and L.I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25: 821–837, 1964.

J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2010.

E.L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *The Journal of Machine Learning Research*, 1:113–141, 2001.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' guide*. SIAM, third edition, 1999.

B. Ávila Pires, C. Szepesvari, and M. Ghavamzadeh. Cost-sensitive multiclass classification risk bounds. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1391–1399, 2013.

K. Bache and M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

C.C.J.H. Bijleveld and J. De Leeuw. Fitting longitudinal reduced-rank regression models by alternating least squares. *Psychometrika*, 56(3):433–447, 1991.

L. Bottou and C.-J. Lin. Support vector machine solvers. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007.

E.J. Bredensteiner and K.P. Bennett. Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12(1):53–79, 1999.

G.C. Cawley and N.L.C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107, 2010.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011.

O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5): 1155–1178, 2007.

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

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002a.

K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233, 2002b.

L. Dalcín, R. Paz, and M. Storti. MPI for Python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115, 2005.

J. De Leeuw. Applications of convex analysis to multidimensional scaling. In J.R. Barra, F. Brodeau, G. Romier, and B. Van Cutsem, editors, *Recent Developments in Statistics*, pages 133–146. North Holland Publishing Company, Amsterdam, 1977.

J. De Leeuw. Convergence of the majorization method for multidimensional scaling. *Journal of Classification*, 5(2):163–180, 1988.

J. De Leeuw. Fitting distances by least squares. Technical Report 130, Los Angeles: Interdivisional Program in Statistics, UCLA, 1993.

J. De Leeuw. Block-relaxation algorithms in statistics. In H.-H. Bock, W. Lenski, and M.M. Richter, editors, *Information Systems and Data Analysis*, pages 308–324. Springer Berlin Heidelberg, 1994.

J. De Leeuw and W.J. Heiser. Multidimensional scaling with restrictions on the configuration. *Multivariate Analysis*, 5:501–522, 1980.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

U. Dogan, T. Glasmachers, and C. Igel. Fast training of multi-class support vector machines. Technical Report 03/2011, University of Copenhagen, Faculty of Science, 2011.

E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

P.J.F. Groenen and W.J. Heiser. The tunneling method for global optimization in multidimensional scaling. *Psychometrika*, 61(3):529–550, 1996.

P.J.F. Groenen, W.J. Heiser, and J.J. Meulman. Global optimization in least-squares multidimensional scaling by distance smoothing. *Journal of Classification*, 16(2):225–254, 1999.

P.J.F. Groenen, G. Nalbantov, and J.C. Bioch. Nonlinear support vector machines through iterative majorization and I-splines. In R. Decker and H.-J. Lenz, editors, *Advances in Data Analysis*, pages 149–161. Springer Berlin Heidelberg, 2007.

P.J.F. Groenen, G. Nalbantov, and J.C. Bioch. SVM-Maj: a majorization approach to linear support vector machines with different hinge errors. *Advances in Data Analysis and Classification*, 2(1):17–43, 2008.

Y. Guermeur and E. Monfrini. A quadratic loss multi-class SVM for which a radius–margin bound applies. *Informatica*, 22(1):73–96, 2011.

G.H. Hardy, J.E. Littlewood, and G. Pólya. *Inequalities.* Cambridge University Press, 1934.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, New York, 2nd edition, 2009.

T. F. Havel. An evaluation of computational strategies for use in the determination of protein structure from distance constraints obtained by nuclear magnetic resonance. *Progress in Biophysics and Molecular Biology*, 56(1):43–78, 1991.

S.I. Hill and A. Doucet. A framework for kernel-based multi-category classification. *Journal of Artificial Intelligence Research*, 30:525–564, 2007.

S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S.S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, pages 408–415, 2008.

C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

P.J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

D.R. Hunter and K. Lange. A tutorial on MM algorithms. *The American Statistician*, 58 (1):30–37, 2004.

R.L. Iman and J.M. Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics – Theory and Methods*, 9(6):571–595, 1980.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226, 2006.

S.S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A sequential dual method for large scale multi-class linear SVMs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 408–416, 2008.

U.H.G. Kreßel. Pairwise classification and support vector machines. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods*, pages 255–268. MIT Press, 1999.

F. Lauer and Y. Guermeur. MSVMpack: A multi-class support vector machine package. *The Journal of Machine Learning Research*, 12:2269–2272, 2011.

Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines: theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99(465):67–81, 2004.

C.-J. Lin, R.C. Weng, and S.S. Keerthi. Trust region Newton method for logistic regression. *The Journal of Machine Learning Research*, 9:627–650, 2008.

Y. Mroueh, T. Poggio, L. Rosasco, and J. Slotine. Multiclass learning with simplex coding. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2789–2797. Curran Associates, Inc., 2012.

J.M. Ortega and W.C. Rheinboldt. *Iterative Solutions of Nonlinear Equations in Several Variables.* New York: Academic Press, 1970.

J.C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods*, pages 185–208. MIT press, 1999.

J.C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. In S.A. Solla, T.K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 547–553. MIT Press, 2000.

R. Rifkin and A. Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

R.T. Rockafellar. *Convex Analysis.* Princeton University Press, 1997.

S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *The Annals of Statistics*, 35(3):1012–1030, 2007.

J.M. Santos and M. Embrechts. On the use of the adjusted Rand index as a metric for evaluating supervised classification. In *Proceedings of the 19th International Conference on Artificial Neural Networks: Part II*, pages 175–184. Springer-Verlag, 2009.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

A. Statnikov, C.F. Aliferis, D.P. Hardin, and I. Guyon. *A Gentle Introduction to Support Vector Machines in Biomedicine: Theory and Methods.* World Scientific, 2011.

M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

H. Tsutsu and Y. Morikawa. An l_p norm minimization using auxiliary function for compressed sensing. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, 2012.

V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

H. Voss and U. Eckhardt. Linear convergence of generalized Weiszfeld's method. *Computing*, 25(3):243–251, 1980.

E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Mathematical Journal*, 43:355–386, 1937.

J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, University of London, Royal Holloway, Department of Computer Science, 1998.

R.C. Whaley and J.J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–27. IEEE Computer Society, 1998.

C.K.I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T.K. Leen, T.G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale L1-regularized linear classification. *The Journal of Machine Learning Research*, 11:3183–3234, 2010.