Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

The Majorization Approach to SVM: The SVMMaj Package in R

Hoksan Yip
Erasmus University Rotterdam

Patrick J.F. Groenen
Erasmus University Rotterdam

Georgi Nalbantov
Maastricht University

Abstract

Support Vector Machines (SVMs) have gained considerable popularity over the last two decades for binary classification. This paper concentrates on a recent optimization approach to SVMs, the SVM majorization approach, or **SVM-Maj** for short. This method is aimed at small and medium sized Support Vector Machine (SVM) problems, in which **SVM-Maj** performs well relative to other solvers. To obtain an SVM solution, most other solvers need to solve the dual problem. In contrast, **SVM-Maj** solves the primal SVM optimization iteratively thereby converging to the SVM solution. Furthermore, the simplicity of **SVM-Maj** makes it intuitively more accessible to the researcher than the state-of-art decomposition methods. Moreover, **SVM-Maj** can easily handle any well-behaved error function, while the traditional SVM solvers focus particularly on the absolute-hinge error. In this paper, the **SVM-Maj** approach is enhanced to include the use of different kernels, the standard way in the SVM literature for handling nonlinearities in the predictor space. In addition, we introduce the R package **SVMMaj** that implements this methodology. Amongst its features are the weighting of the error for individual objects in the training dataset, handling nonlinear prediction through monotone spline transformations and through kernels, and functions to do cross validation.

Keywords: Support Vector Machine, **SVM-Maj**, R, Majorization.

1. Introduction

For understanding what a support vector machine (SVM) is, consider the following data analysis problem: there are two groups of objects, say products of type A and type B, having

some common attributes such as color, price, weight, etc. The task of separating the two types of objects from each other is formally referred to as the binary classification task. Given the values for the attributes of a new object, we would like to assign this new object to a given group, or class. Such a binary classification task is dealt with routinely in medical, technical, economic, humanitarian, and other fields.

Numerous learning methods have been designed to solve the binary classification task, including Linear Discriminant Analysis, Binary Logistic Regression, Neural Networks, Decision Trees, Naive Bayes classifier, and others. In this paper, we focus on a method that has gained considerable popularity over the last two decades, called the Support Vector Machines (SVMs) classifier (Vapnik 1995). SVMs have emerged as one of the most popular and high-performing learning methods for classification. They have been successfully applied to areas ranging from bioinformatics (see, e.g., Furey *et al.* 2000; Guyon *et al.* 2002) to image recognition (see, e.g., Chapelle *et al.* 1999; Pontil and Verri 1998) and marketing (see, e.g., Cui and Curry 2005). In essence, SVMs divide two groups of objects from each other by building a hyperplane in the space of the predictor variables that separates them in an adequate way. The rapid success of SVMs can be attributed to a number of key qualities: robustness of results, avoiding overfitting, and the possibility to handle nonlinearities of the predictor variables easily through so-called kernel functions. In addition, the evaluation of an SVM model on test objects is relatively fast and simple. The SVM is formulated as a well-defined quadratic optimization problem that has a unique solution. Overfitting, that is, fitting an available training data-set too well, is avoided via a *penalty* term that suppresses too complex potential fits. Nonlinearities can be handled in two ways: (1) by a preprocessing step of the predictor variables, for example, by polynomial expansion or the use of splines (Groenen *et al.* 2007) and (2) by using the *kernel* trick that allows nonlinear solutions to be computed implicitly. Note that the use of kernels is very prominent in the SVM literature.

A variety of solvers for the SVM optimization task have been proposed in the literature. One of the initial ideas has been to apply general-purpose quadratic optimization solvers such as **LOQO** (Vanderbei 1994) and **MINOS** (Murtagh and Saunders 1998). One of the problems of such solvers is that they require the whole *kernel* matrix to be stored in memory, which is quadratic in the number of observations n. For small scale problems, this is not a problem, but for medium and large scale problems other methods are needed. One attempt to overcome the complete storage of the kernel matrix is by chunking (Boser *et al.* 1992; Osuna *et al.* 1997a). This method concentrates on a (working) subset of all training objects at a given iteration, effectively splitting the learning task into smaller subproblems that easily fit into the memory of a computer. Alternatively, direction search was proposed (Boser *et al.* 1992) that updates all unknown coefficients at each iteration in a certain feasible direction.

More recently, decomposition methods have established themselves as the mainstream technique for solving SVMs (Osuna *et al.* 1997b; Saunders *et al.* 1998; Joachims 1999). At each iteration, the decomposition method optimizes a subset of coefficients, and leaves the remaining coefficients unchanged. Solving a series of very simple optimization subproblems, this approach has proven to be one of the fastest for large scale SVM problems. The most popular decomposition method is the so-called Sequential Minimal Optimization (SMO) (Platt 1999), where only two coefficients are updated at each iteration, which can actually be done analytically. SMO is the basis of popular SVM solvers such as **LibSVM** (Chang and Lin 2001) and **SVMlight** (Joachims 1999), which has been implemented in the R packages **e1071** (Meyer *et al.* 2014) respectively **klaR** (Weihs *et al.* 2005). For the linear SVM case, alternative tech-

niques to the decomposition methods have recently been put forward, such as the cutting plane algorithm (Joachims 2006).

This paper concentrates on a recent optimization approach to solving SVMs (Groenen *et al.* 2007, 2008), referred to as the majorization approach to SVMs, or **SVM-Maj** for short. This method is aimed at small and medium sized SVM problems. An overview of some popular SVM solvers and some of their properties is given in Table 1. The other solvers are **LibSVM**, **SVMlight**, **SVMTorch** (Collobert and Bengio 2001), **mySVM** (Rüping 2000), **SVM-Perf** (Joachims 2006), and **LibLINEAR** (Fan *et al.* 2008).

Properties	**SVM-Maj**	**LIBSVM**	**SVMlight**	**SVMTorch**	**mySVM**	**SVM-Perf**	**LIBLINEAR**
Nonlinear kernels	yes	yes	yes	yes	yes	no	no
Splines	yes	no	no	no	no	no	no
Suitable for large n	yes	yes	yes	yes	yes	no	yes
Suitable for large k	yes	yes	yes	no	yes	no	yes
Suitable for large n and large k	no	yes	yes	no	yes	no	
Dual approach	no	yes	yes	yes	yes	yes	no
Allows quadratic hinge	yes	yes*	no	no	yes	no	yes
Allows Huber hinge	yes	no	no	no	no	no	no
Allows k-fold cross val.	yes	yes	yes	no	yes	no	yes
Language	MATLAB, R	C++,Java	C, C++	C++	C	C++	C, C++
Gui interface or command line interface (**cli**)	MATLAB, R	*cli*	*cli*	*cli*	*cli*	*cli*	*cli*
Availability in R	**SVMMaj**	**e1071****	**klaR****	no	no	no	**LiblineaR**
Multi-class problems	no	yes	yes	yes	no	yes	yes

* after modification
** for non-linear kernels, the package **kernlab** can be used

Table 1: Comparison table of different SVM solvers available in 2015.

In this paper, the method of Groenen *et al.* (2008) is enhanced to the use of different kernels, the standard way in the SVM literature for handling nonlinearities in the predictor space. Moreover, we offer here a new special treatment for the case where the number of objects is less than the number of attributes. Three cases are distinguished for which **SVM-Maj** computes the solution efficiently: (1) the case with many more observations n than variables k, (2) the case with many variables k but medium sized n, and (3) the case that n is medium sized and the kernel space or the space of the variables is of lower rank than k or n. The case of large n is relatively more difficult for all SVM solvers, including **SVM-Maj**. This particularly holds when kernels are used. For this case, the alternative of introducing nonlinearity through splines in **SVM-Maj** can be used so that still for medium sized n, nonlinearity can be handled efficiently.

Among the advantages of the **SVM-Maj** approach are its intuitive optimization algorithm, its versatility, and the competitively fast estimation speed for medium sized problems. The majorization solver is an iterative algorithm with two easily tractable alternating steps that reveal the nature of solving the SVM optimization problem in an appealing way. The relative simplicity of **SVM-Maj** arguably makes it intuitively more accessible to the researcher than the state-of-art decomposition methods. Traditional SVM solvers focus particularly on the absolute-hinge error (the standard SVM error function), whereas the majorization algorithm has been designed to incorporate any well-behaved error function. This property can be viewed as a attractive feature of the majorization approach. The **SVM-Maj** package comes with the following in-built error functions: the classic absolute-hinge, Huber-hinge, and quadratic hinge. Furthermore, **SVM-Maj** solves the *primal* SVM optimization problem

even in the nonlinear case, in contrast to other solvers, which solve the *dual* problem. The advantage of solving the primal is that **SVM-Maj** converges to the optimal solution in each iterative step. In contrast, other methods solving the dual optimization problem need the dual problem to be fully converged to attain a solution.

The main aim of this paper is to introduce the **SVMMaj** package for R (R Development Core Team 2011). Its main features are: implementation of the **SVM-Maj** majorization algorithm for SVMs, handling of nonlinearity through splines and kernels, the ability to handle several error functions (among other the classic hinge, quadratic hinge and Huber hinge error).

In addition, **SVMMaj** is able to assign a fixed weight to each individual objects in a training dataset to receive different individual weights. In this way, the user can set importance of misclassifying the object by varying the individual weight. These weights can also be set per class. The **SVMMaj** package comes with a cross validation function to asses out-of-sample performance or evaluate meta parameters that come with certain SVM models.

This paper is organized as follows. First, Section 2 gives a brief introduction to SVMs. In Section 3, we describe the **SVM-Maj** algorithm and its update for each iteration is derived. Furthermore, Section 3 also discusses some computational efficiencies, and Section 4 presents the way nonlinearities are handled. Section 5 gives several detailed examples of how the **SVM-Maj** package can be used, and Section 6 concludes. The appendix gives technical derivations underlying the **SVM-Maj** algorithm.

2. Definitions

First of all, let us introduce some notations: n denotes the number of object; k denotes the number of attributes, $\mathbf{X}$ is an $n \times k$ matrix containing the k attributes, without a column of ones to specify the intercept, $\mathbf{y}$ is an $n \times 1$ vector with 1 if object i belongs to class 1 and -1 if it belongs to class -1, r denotes the rank of matrix $\mathbf{X}$, and $\lambda > 0$ is the penalty parameter for the penalty term. The purpose of SVM is to produce a linear combination of predictor variables $\mathbf{X}$ such that a positive prediction is classified to class $+1$ and a negative prediction to class -1. Let $\boldsymbol{\beta}$ and α be parameters of the linear combination $\alpha + \mathbf{x}_i^\top \boldsymbol{\beta}$. Then, for a given intercept α and vector with attribute weights $\boldsymbol{\beta}$, the predicted class is given by

$$\hat{y}_i = \text{sign}(\mathbf{x}_i^\top \boldsymbol{\beta} + \alpha) = \text{sign}(q_i + \alpha) = \text{sign}(\tilde{q}_i),$$

where $q_i = \mathbf{x}_i \boldsymbol{\beta}$. Here, the term $\tilde{q}_i = q_i + \alpha$ is used to indicate the predicted score for object i, which also accounts for the intercept α.

To find the optimal of the linear combination, we use the SVM loss function as a function of α and $\boldsymbol{\beta}$ to be minimized. The SVM loss function can be represented in several ways. Here, we follow the notation of Groenen *et al.* (2008) that allows for general error functions $f_1(\tilde{q}_i)$ and $f_{-1}(\tilde{q}_i)$. The goal of SVMs is to minimize the SVM loss function

$$\begin{aligned} & L_{\text{SVM}}(\alpha, \boldsymbol{\beta}) \\ & \quad = \sum_{i \in G_1} w_i f_1(\tilde{q}_i) + \sum_{i \in G_{-1}} w_i f_{-1}(\tilde{q}_i) + \lambda \boldsymbol{\beta}^\top \boldsymbol{\beta} \\ & \quad = \text{Class 1 errors} + \text{Class -1 errors} + \text{Penalty for nonzero } \boldsymbol{\beta}, \end{aligned}$$

over α and $\boldsymbol{\beta}$, where G_1 and G_{-1} respectively denote the sets of class 1 and -1 objects, and w_i is a given nonnegative importance weight of observation i. Note that Groenen *et al.* (2008)

proved that minimizing $L_{\text{SVM}}(\alpha, \boldsymbol{\beta})$ is equivalent to minimizing the SVM error function in Vapnik (1995) and Burges (1998). Figure 1 and Table 2 contain different error functions that can be used, such as the classical hinge error standard in SVMs, the quadratic hinge, and the Huber hinge. Figure 1 plots the error functions as function value of $\tilde{q}_i$. All three hinge functions have the property that their error is only larger than zero if the predicted value $\tilde{q}_i < 1$ for class $+1$ objects and $\tilde{q}_i > -1$ for class -1 objects. The classic absolute hinge error is linear in $\tilde{q}_i$ and is thus robust for outliers. However, it is also non-smooth at $\tilde{q}_i = 1$. The quadratic hinge is smooth but may be more sensitive to outliers. The Huber loss is smooth as well as robust. Those observations $\mathbf{x}_i$ yielding zero errors do not contribute to the SVM solution and could therefore be removed without changing the SVM solution. The remaining observations $\mathbf{x}_i$ that do have an error or have $|\tilde{q}_i| = 1$ determine the solution and are called support vectors, hence the name Support Vector Machine. Unfortunately, before the SVM solution is available it is unknown which of the observations are support vectors and which are not. Therefore, the SVM always needs all available data to obtain a solution.

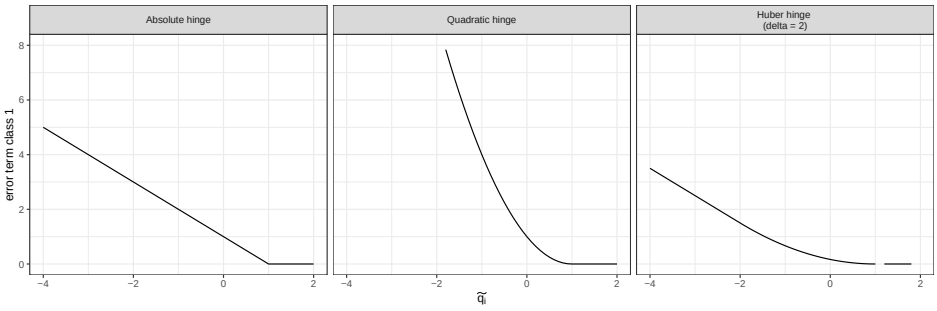

Figure 1: This figure shows the hinge error functions. Note that the error function is only nonzero if $\tilde{q} < 1$.

Error function	$f_1(\tilde{q}_i)$
Absolute hinge	$\max(0, 1 - \tilde{q}_i)$
Quadratic hinge	$\max(0, 1 - \tilde{q}_i)^2$
Huber hinge	$\begin{cases} \frac{1}{2(\delta+1)} \max(0, 1 - \tilde{q}_i)^2 & \tilde{q}_i > -\delta \\ \frac{(\delta - 1)}{2} - \tilde{q}_i & \tilde{q}_i \leq -\delta \end{cases}$

Table 2: Common error function used. The further $\tilde{q}_i$ is from wrongly predicted, the higher its error. Note that $f_{-1}(\tilde{q}_i) = f_1(-\tilde{q}_i)$, therefore, only $f_1(\tilde{q}_i)$ is described below.

The weight w_i of observation i can be interpreted as the relative importance of the error of the observation. One can also assign the same weights for all observations in G_1 and different

weight for those in G_{-1}. Assigning weights per class is especially useful when one class is substantially larger than the other. By assigning a larger weight for the smaller subset, one can correct the dominance of the errors of the larger subset. For example, if there are n_1 objects in G_1 and n_{-1} objects in G_{-1}, then choosing

$$w_i = \begin{cases} (n_1 + n_{-1})/(2n_1) & i \in G_1 \\ (n_1 + n_{-1})/(2n_{-1}) & i \in G_{-1} \end{cases}$$

is one way to obtain equal weighting of the classes.

A more compact expression of L_{SVM} is obtained by exploiting the symmetric relation $f_{-1}(\tilde{q}_i) = f_1(-\tilde{q}_i) = f_1(y_i\tilde{q}_i)$. Then, the SVM loss function can be simplified into

$$\begin{array}{rcl} L_{\text{SVM}}(\alpha, \boldsymbol{\beta}) & = & \sum_{i=1}^{n} w_i f_1(y_i\tilde{q}_i) \quad + \quad \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} \\ & = & \text{Error} \qquad\qquad\quad + \quad \text{Penalty for nonzero } \boldsymbol{\beta}. \end{array} \tag{1}$$

To minimize this function, we will use the **SVM-Maj** algorithm, discussed in the next section.

3. The SVM-Maj algorithm

The **SVMMaj** package minimizes the SVM loss function by using the **SVM-Maj** algorithm, that is based on the ideas of majorization (see, e.g., De Leeuw and Heiser 1980; De Leeuw 1994; Heiser 1995; Lange *et al.* 2000; Kiers 2002; Hunter and Lange 2004; Borg and Groenen 2005). Groenen *et al.* (2007, 2008) developed the algorithm for linear SVMs. Here, the **SVM-Maj** algorithm is extended to nonlinear situations that use a kernel matrix.

The **SVM-Maj** algorithm uses iterative majorization to minimize the loss function (1). The main point of this algorithm is to iteratively replace the original function $f(x)$ by an auxiliary function $g(x, \overline{x})$ at supporting point $\overline{x}$, for which the minimum can be easily computed. The auxiliary function, called the majorization function, has the properties that:

- the minimum x^* of $g(x, \overline{x})$ can be found easily, and
- $g(x, \overline{x})$ is always larger than or equal to the $f(x)$, and
- at the supporting point $\overline{x}$, $g(\overline{x}, \overline{x})$ is equal to $f(\overline{x})$.

Combining these properties gives the so-called sandwich inequality $f(x^*) \leq g(x^*, \overline{x}) \leq g(\overline{x}, \overline{x}) = f(\overline{x})$. That is, for each support point $\overline{x}$, we can find another point x^* of whose function value $f(x^*)$ is lower or equal to the former $f(\overline{x})$. Using this point x^* as the next iteration's support point and repeating the procedure, a next update can be obtained with a lower $f(x)$ value. This iterative process produces a series of function values that is non-increasing and generally decreasing. Note that this principle of majorization also works if the argument of f is a vector. Moreover, if f is strictly convex, as is the case with the SVM loss function, the updates converge to the minimum of the original function as the number of iterations increases.

The first step is to find the majorizing functions for the different hinge errors. The majorization function of the **SVM-Maj** algorithm is a quadratic function so that its minimum is obtained by setting the derivative to zero. Given the support point $\overline{x}$, a quadratic majorization function $g(x, \overline{x})$ can be written as

$$g(x, \overline{x}) = a(\overline{x})x^2 - 2b(\overline{x})x + o(\overline{x}). \tag{2}$$

As the parameter $o(\overline{x})$ is irrelevant for determining $\hat{x}$ as it is constant for a given $\overline{x}$, we shall write o instead of $o(\overline{x})$ to indicate all terms that are not dependent on x in the majorizing function.

Conform Groenen *et al.* (2007, 2008), $f_1(y_i\tilde{q}_i)$ in (1) can be majorized by $g(\tilde{q}_i, \overline{\tilde{q}}_i, y_i) = a_i\tilde{q}_i^2 - 2b_i\tilde{q}_i + o_i$ with the parameters a_i, b_i, and o_i given in Table 3. As $f_1(y_i\tilde{q}_i) \leq g(\tilde{q}_i, \overline{\tilde{q}}_i, y_i)$, we can obtain the majorization function for the SVM loss function (1), using $\tilde{\mathbf{q}} = \mathbf{X}\boldsymbol{\beta} + \alpha\mathbf{1} = \mathbf{q} + \alpha\mathbf{1}$ and the support point $\overline{\tilde{\mathbf{q}}} = \overline{\alpha}\mathbf{1} + \overline{\mathbf{q}}$

$$\begin{aligned} \mathrm{L}_{\mathrm{SVM}}(\alpha, \boldsymbol{\beta}) &= \sum_i w_i f_1(y_i\tilde{q}_i) + \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} \\ &\leq \sum_i w_i g(\tilde{q}_i, \overline{\tilde{q}}_i, y_i) + \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} + o \\ &= \tilde{\mathbf{q}}^\top\mathbf{A}\tilde{\mathbf{q}} - 2\tilde{\mathbf{q}}^\top\mathbf{b} + \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} + o \\ &= (\alpha\mathbf{1} + \mathbf{q})^\top\mathbf{A}(\alpha\mathbf{1} + \mathbf{q}) - 2(\alpha\mathbf{1} + \mathbf{q})^\top\mathbf{b} + \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} + o \\ &= \mathrm{Maj}(\alpha, \boldsymbol{\beta}), \end{aligned} \tag{3}$$

where $\mathbf{A}$ is an $n \times n$ diagonal matrix with elements $w_i a_i$ and $\mathbf{b}$ an $n \times 1$ vector with elements $w_i b_i$ (with a_i and b_i from Table 3). As $w_i > 0$ for all i, we can conclude that the diagonal matrix $\mathbf{A}$ is positive definite, which implies that the majorization function is strictly quadratically convex, thereby guaranteeing a unique minimum of the majorizing function at the right hand side of (3). Groenen *et al.* (2008) showed in their article that the minimum of (3), that is, an update for the iteration process, can be computed as followed:

$$\left(\tilde{\mathbf{X}}^\top\mathbf{A}\tilde{\mathbf{X}} + \lambda\mathbf{J}\right)\begin{bmatrix}\alpha^+ \\ \boldsymbol{\beta}^+\end{bmatrix} = \tilde{\mathbf{X}}^\top\mathbf{b}, \tag{4}$$

where $\tilde{\mathbf{X}} = \begin{bmatrix}\mathbf{1} & \mathbf{X}\end{bmatrix}$ and $\mathbf{J} = \begin{bmatrix}0 & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{I}\end{bmatrix}$. Appendix A shows the derivation of this update, while Algorithm 1 shows the steps taken by the **SVM-Maj** algorithm.

3.1. Computational efficiencies

Before we continue with the discussion of the extended options for the **SVM-Maj** algorithm, we will introduce a few modifications of the **SVM-Maj** algorithm to obtain computational efficiencies. One efficiency can be achieved by using the QR-decomposition on matrix $\mathbf{X}$ to find a more efficient update. Another efficiency improvement can be obtained in case of using the quadratic or Huber hinge and the number of updates generally decreases when using a *relaxed update*. The algorithm is summarized in Algorithm 1 and the most important steps are discussed below and in the appendix.

Efficient updates by using QR-decomposition

Usually, the loss function will be optimized by optimizing $\boldsymbol{\beta}$. However, when $k > n$, it is more efficient to optimize $\mathbf{q}$ instead, as it has a lower dimensionality. In this situation, the dimensional space in which the optimal parameter values lies will be smaller and therefore it is more efficient and generally faster to compute an update. In case that $r = \mathrm{rank}(\mathbf{X}) < \min(n, k)$, an even more efficient update exists. Moreover, in a higher dimensional space, one

Error term	Parameters
Absolute hinge	$a_i = \frac{1}{4}\max(\lvert 1 - y_i\overline{\tilde{q}}_i\rvert, \epsilon)^{-1}$ $b_i = y_i a_i(1 + \lvert y_i\overline{\tilde{q}}_i - 1\rvert)$ $o_i = a_i(1 + \lvert y_i\overline{\tilde{q}}_i - 1\rvert)^2$
Quadratic hinge	$a_i = 1$ $b_i = y_i a_i(1 + \max(y_i\overline{\tilde{q}}_i - 1, 0))$ $o_i = a_i(1 + \max(y_i\overline{\tilde{q}}_i - 1, 0))^2$
Huber hinge	$a_i = \frac{1}{2}(k+1)^{-1}$ $b_i = y_i a_i(1 + \max(y_i\overline{\tilde{q}}_i - 1, 0) + \min(y_i\overline{\tilde{q}}_i + k, 0))$ $o_i = a_i(1 + \max(y_i\overline{\tilde{q}}_i - 1, 0) + \min(y_i\overline{\tilde{q}}_i + k, 0))^2 + \min(y_i\overline{\tilde{q}}_i + k, 0)$

Table 3: Parameter values of a_i, b_i and o_i of the majorization function.

only needs a part of the space of $\boldsymbol{\beta}$ to find the optimal $\mathbf{q}$. The appendix discusses these issues more in detail and introduces efficient and consistent updates for **SVM-Maj** for all possible situations. An overview of all efficient updates in each occasion can be found in Table 4.

Equation 4 optimizes $\boldsymbol{\beta}$ to optimize the loss function. However, in some cases, there exist an even more efficient update. Appendix A derives an efficient update for each of these cases by making use of the singular value decomposition (SVD)

$$\underset{(n\times k)}{\mathbf{X}} = \underset{(n\times r)}{\mathbf{P}}\ \underset{(r\times r)}{\mathbf{\Lambda}}\ \underset{(r\times k)}{\mathbf{Q}^\top}, \tag{5}$$

where $\mathbf{P}^\top\mathbf{P} = \mathbf{I}$ and $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$ and $\mathbf{\Lambda}$ is a diagonal matrix. However, **SVMMaj** package uses the QR-decomposition to determine the rank of $\mathbf{X}$, as it is a computationally more efficient way to determine the rank of $\mathbf{X}$ than doing so through the SVD-decomposition. Let the QR-decomposition of $\mathbf{X}^\top$ be given by $\mathbf{X}^\top = \mathbf{V}\mathbf{Z}^\top$ with $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$ and $\mathbf{Z}$ a lower triangular matrix. An additional QR-decomposition of $\mathbf{Z}$ gives $\mathbf{Z} = \mathbf{U}\mathbf{R}$ with $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$ and $\mathbf{R}$ an upper triangular matrix. Then we have

$$\underset{(n\times k)}{\mathbf{X}} = \underset{(n\times r)}{\mathbf{U}}\ \underset{(r\times r)}{\mathbf{R}}\ \underset{(r\times k)}{\mathbf{V}^\top}. \tag{6}$$

Note that matrices $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices and and therefore have the same properties as $\mathbf{P}$ and $\mathbf{Q}$. Thus, we can replace matrices $\mathbf{P}$, $\mathbf{\Lambda}$ and $\mathbf{Q}$ by respectively $\mathbf{U}$, $\mathbf{R}$ and $\mathbf{V}$. The update (4) can then be computed through

$$(\tilde{\mathbf{Z}}^\top\mathbf{A}\tilde{\mathbf{Z}} + \lambda\mathbf{J})\begin{bmatrix}\alpha^+\\ \boldsymbol{\rho}^+\end{bmatrix} = \tilde{\mathbf{Z}}^\top\mathbf{b}, \tag{7}$$

where $\tilde{\mathbf{Z}} = \begin{bmatrix}\mathbf{1} & \mathbf{Z}\end{bmatrix} = \begin{bmatrix}\mathbf{1} & \mathbf{U}\mathbf{R}\end{bmatrix}$ and where $\boldsymbol{\rho} = \mathbf{V}^\top\boldsymbol{\beta}$. Furthermore, $\boldsymbol{\beta}$ and $\mathbf{q}$ can be derived from $\boldsymbol{\beta} = \mathbf{V}\boldsymbol{\rho}$ and $\mathbf{q} = \mathbf{Z}\boldsymbol{\rho} = \mathbf{U}\mathbf{R}\boldsymbol{\rho}$. Note that in most cases, the decomposition of $\mathbf{Z}$ is not necessary to perform, so that only a single QR-decomposition performance is needed.

Algorithm: SVM-Maj

input : $\mathbf{y} = n \times 1$ vector with class labels $+1$ and -1 ,
$\mathbf{X} = n \times k$ matrix of explanatory variables ,
$\lambda > 0$ the penalty parameter,
Hinge= {absolute,quadratic,huber} the hinge error function,
$\delta > 0$ the Huber hinge parameter,
relax determining from which step to use the relaxed update (11),
method specifies whether $\boldsymbol{\rho}$ (using matrix decomposition) will be used or $\boldsymbol{\beta}$.

output: α_t, ($\boldsymbol{\rho}_t$ or $\boldsymbol{\beta}_t$)

$t = 0$;
Set ϵ to a small positive value;
Set ($\boldsymbol{\rho}_0$ or $\boldsymbol{\beta}_0$) and α_0 to random initial values;
Compute $L_0 = L_{\text{SVM}}(\alpha_0, \boldsymbol{\rho}_0)$ or $L_{\text{SVM}}(\alpha_0, \boldsymbol{\beta}_0)$ according to (1);
if *hinge* $\neq$ *absolute & method* $= \boldsymbol{\beta}$ **then**
  Find $\mathbf{S}$ that solves (8);
else if *hinge* $\neq$ *absolute & method* $= \boldsymbol{\rho}$ **then**
  Find $\mathbf{S}$ that solves (9);
end
while $t = 0$ *or* $(L_{t-1} - L_t)/L_t > \epsilon$ **do**
  $t = t + 1$;
  Compute a_i and b_i by Table 3;
  Set diagonal elements of $\mathbf{A}$ to $w_i a_i$ and $\mathbf{b}$ to $w_i b_i$;
  if *hinge* $=$ *absolute & method* $= \boldsymbol{\rho}$ **then**
    Find α_t and $\boldsymbol{\rho}_t$ that solves (7): $\left(\widetilde{\mathbf{Z}}^\top \mathbf{A} \widetilde{\mathbf{Z}} + \lambda \mathbf{J}\right) \begin{bmatrix} \alpha_t \\ \boldsymbol{\rho}_t \end{bmatrix} = \widetilde{\mathbf{Z}}^\top \mathbf{b}$;
  else if *hinge* $=$ *absolute & method* $= \boldsymbol{\beta}$ **then**
    Find α_t and $\boldsymbol{\rho}_t$ that solves (4): $\left(\widetilde{\mathbf{X}}^\top \mathbf{A} \widetilde{\mathbf{X}} + \lambda \mathbf{J}\right) \begin{bmatrix} \alpha_t \\ \boldsymbol{\beta}_t \end{bmatrix} = \widetilde{\mathbf{X}}^\top \mathbf{b}$;
  else if *hinge* $\neq$ *absolute & method* $= \boldsymbol{\rho}$ **then**
    Find α_t and ($\boldsymbol{\rho}_t$ or $\boldsymbol{\beta}_t$) that solves: $\begin{bmatrix} \alpha^+ \\ \boldsymbol{\rho}^+ \end{bmatrix} = \mathbf{Sb}$
  else if *hinge* $\neq$ *absolute & method* $= \boldsymbol{\beta}$ **then**
    Find α_t and ($\boldsymbol{\rho}_t$ or $\boldsymbol{\beta}_t$) that solves: $\begin{bmatrix} \alpha^+ \\ \boldsymbol{\beta}^+ \end{bmatrix} = \mathbf{Sb}$;
  end
  if $t \geq$ *relax* **then**
    Replace $\alpha_t = 2\alpha_t - \alpha_{t-1}$;
    Replace $\boldsymbol{\rho}_t = 2\boldsymbol{\rho}_t - \boldsymbol{\rho}_{t-1}$ or $\boldsymbol{\beta}_t = 2\boldsymbol{\beta}_t - \boldsymbol{\beta}_{t-1}$;
  end
  Compute $L_t = L_{\text{SVM}}(\alpha_t, \boldsymbol{\rho}_t)$ or $L_{\text{SVM}}(\alpha_t, \boldsymbol{\beta}_t)$ according to (1);
end

Algorithm 1: The SVM majorization algorithm

Method	Situation
$\boldsymbol{\beta}$	$n \geq k, r = k$
$\mathbf{q}$	$n \leq k, r = n$
$\boldsymbol{\theta}$	$r < \min(n, k)$

Table 4: An overview of the most efficient updates for each situation.

As this decomposition is already performed to determine the rank of $\mathbf{X}$, it is efficient to use update (7) in all three cases distinguished in Table 4. Nevertheless, in case n and k are both very large, it may be more efficient not to perform a matrix decomposition at all to avoid unnecessary computations. Instead, one can use update (4).

Quadratic and Huber hinge

For the quadratic and Huber hinge, the parameters a_i does not depend on the support point $\overline{\widetilde{\mathbf{q}}}$, which means that the matrix $\mathbf{A}$ remain fixed during the algorithm steps. In fact, the parameters a_i are the same for all i, that is, $a_i = a$ for all i. Therefore, extra computational efficiency can be obtained by solving the linear system

$$(a\widetilde{\mathbf{X}}^\top\widetilde{\mathbf{X}} + \lambda\mathbf{J})\mathbf{S} = \widetilde{\mathbf{X}}^\top, \tag{8}$$

or, when using QR-decomposition,

$$(a\widetilde{\mathbf{Z}}^\top\widetilde{\mathbf{Z}} + \lambda\mathbf{J})\mathbf{S} = \widetilde{\mathbf{Z}}^\top, \tag{9}$$

so that the original update (7) can be simplified into

$$\begin{bmatrix}\alpha^+\\ \boldsymbol{\rho}^+\end{bmatrix} = \mathbf{S}\mathbf{b}, \tag{10}$$

so that in each iteration the update can be obtained by a single matrix multiplication instead of solving a linear system.

Computational efficiency by the relaxed updates

The parameter updates obtained by (7) find the minimum of the majorization function in (3). This guarantees that the next update will always be better than the previous point. However, it turns out that often the updates converge faster when making the update twice as long by using a relaxed update (De Leeuw and Heiser 1980):

$$\boldsymbol{\theta}^*_{t+1} = \boldsymbol{\theta}_{t+1} + (\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) = 2\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t. \tag{11}$$

This relaxation is most effective when **SVM-Maj** has already performed several iterations. Preliminary experimentation revealed that this is the case when at least 20 iterations without relaxed updates have been performed. Therefore, we will implement (11) after 20 iterations to increase model estimation efficiency.

4. Nonlinearities

In the previous sections, we have only discussed the **SVM-Maj** algorithm for the linear case. However, often a better prediction can be obtained by allowing for nonlinearity of the predictor variables. Therefore, one might consider to use nonlinearity in the model estimation. In the **SVMMaj** package, two different implementation of nonlinearity can be used: I-splines and kernels. One can choose one of these implementation, or both of them.

Splines are piecewise polynomial functions of a specified variable. The **SVMMaj** package can transform each explanatory variable into I-splines (Ramsay 1988). This transformation will split the original predictor variable $\mathbf{x}_j$ into a number of spline bases vectors gathered in the matrix $\mathbf{B}_j$. After specifying the interior knots k_s that define the boundaries of the pieces and the degree d_s of the polynomials, one can transform the variable $\mathbf{x}_j$ into the basis $\mathbf{B}_j$ of a size of $n \times (d_s + k_s)$. This spline basis $\mathbf{B}_j$ will then be used as a set of $(d_s + k_s)$ variables to find a linear separating plane of a higher dimension. The piecewise polynomial transformation of variable $\mathbf{x}_j$ can then be obtained through computing the linear combination of the spline bases $\mathbf{B}_j\boldsymbol{\gamma}_j$ with the initially unknown weights $\boldsymbol{\gamma}_j$. Then, all spline bases $\mathbf{B}_j$ and weights $\boldsymbol{\gamma}_j$ are gathered in $\mathbf{B} = [\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_k]$ and $\boldsymbol{\beta}^\top = [\boldsymbol{\gamma}_1^\top, \boldsymbol{\gamma}_2^\top, \ldots, \boldsymbol{\gamma}_k^\top]$. Note that here we use I-splines as they have the property that if multiplied by positive weights, there is a guaranteed monotone relation with the original variable $\mathbf{x}_j$. This property can aid the interpretation of the weights as $\boldsymbol{\beta}$ can be split into a vector of positive and one of negative weights.

It is also possible to map the matrix $\mathbf{X}$ differently into a higher dimensional space through so-called kernels. Let us map the explanatory variables of observation i, that is, $\mathbf{x}_i$ into $\phi(\mathbf{x}_i)$ with mapping function $\phi : \Re^k \to \Re^m$. Furthermore, let us denote $k_{ij} = \boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\phi}(\mathbf{x}_j)$ as the inner product of the transformed vectors of $\mathbf{x}_i$ and $\mathbf{x}_j$. Then, the *kernel matrix* $\mathbf{K}$ is denoted as the inner product matrix with value k_{ij} in row i and column j. Note that the kernel matrix always is of size $n \times n$ and $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$ with row i equal to $\phi(\mathbf{x}_i)^\top$. Using this property, we can summarize the mapping of $\mathbf{X}$ even when $m \to \infty$ into a matrix of finite size, even if $m \to \infty$. This method is also known as the 'kernel trick'.

We will now show that this kernel matrix $\mathbf{K}$ can be used to find an efficient majorization update. Then the matrix $\boldsymbol{\Phi}$ is used instead of $\mathbf{X}$ to derive the majorization update. Let us perform the QR-decomposition on $\boldsymbol{\Phi}$ analogous to (6), that is

$$\underset{(n \times m)}{\boldsymbol{\Phi}} = \underset{(n \times r)}{\mathbf{U}} \underset{(r \times r)}{\mathbf{R}} \underset{(r \times m)}{\mathbf{V}^\top} = \underset{(n \times r)}{\mathbf{Z}} \underset{(r \times m)}{\mathbf{V}^\top}, \tag{12}$$

where r denotes the rank of $\boldsymbol{\Phi}$ satisfying $r \leq \min(n, m) = n \ll \infty$. Note that the update (7) does not require $\mathbf{V}$. Moreover, the relation between $\boldsymbol{\Phi}$ and $\mathbf{K}$ can be given by $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$. Using decomposition (12) yields

$$\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top = (\mathbf{Z}\mathbf{V}^\top)(\mathbf{V}\mathbf{Z}^\top) = \mathbf{Z}\mathbf{Z}^\top. \tag{13}$$

As $\mathbf{Z}$ is a lower triangular matrix, it can be obtained by performing a Cholesky decomposition on $\mathbf{K}$. Therefore, without actually computing the mapped space $\boldsymbol{\Phi}$, it is still possible to perform **SVM-Maj** by using the kernel matrix $\mathbf{K}$.

There is a wide variety of available kernels to obtain nonlinearity of the predictors. Table 5 shows some important examples of often used kernel functions.

Type of kernel	Kernel function $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$
linear	$\mathbf{x}_i^\top \mathbf{x}_j$
homogeneous polynomial	$(\text{scale}\,\mathbf{x}_i^\top \mathbf{x}_j)^{\text{degree}}$
nonhomogeneous polynomial	$(\text{scale}\,\mathbf{x}_i^\top \mathbf{x}_j + 1)^{\text{degree}}$
radial basis function	$\exp(-\text{sigma}\lvert\mathbf{x}_i - \mathbf{x}_j\rvert^2)$
Laplace	$\exp(-\text{sigma}\lvert\mathbf{x}_i - \mathbf{x}_j\rvert)$

Table 5: Table of some examples of kernel functions, which can be used in the **SVMMaj** package.

As $\mathbf{V}$ is usually unknown, $\boldsymbol{\beta}$ cannot be calculated. Nevertheless, it is still possible to predict the class labels of an unseen sample $\mathbf{X}_2$. Using $\mathbf{q} = \boldsymbol{\Phi}\boldsymbol{\beta}$ and (12), we can derive

$$\begin{aligned}\boldsymbol{\beta} &= \mathbf{V}\boldsymbol{\rho} \\ &= \mathbf{V}(\mathbf{R}^\top\mathbf{U}^\top\mathbf{U}(\mathbf{R}^\top)^{-1})\boldsymbol{\rho} \\ &= (\mathbf{V}\mathbf{R}\mathbf{U}^\top)\mathbf{U}(\mathbf{R}^\top)^{-1}\boldsymbol{\rho} \\ &= \boldsymbol{\Phi}^\top\mathbf{U}(\mathbf{R}^\top)^{-1}\boldsymbol{\rho}.\end{aligned}$$

The predicted values of an unseen test sample $\mathbf{X}_2$ are

$$\mathbf{q}_2 = \boldsymbol{\Phi}_2\boldsymbol{\beta} = \boldsymbol{\Phi}_2\boldsymbol{\Phi}^\top\mathbf{U}(\mathbf{R}^\top)^{-1}\boldsymbol{\rho} = \mathbf{K}_2\mathbf{U}(\mathbf{R}^\top)^{-1}\boldsymbol{\rho}, \tag{14}$$

where $\boldsymbol{\Phi}_2$ and $\boldsymbol{\Phi}$ are denoted as the transformed matrix of respectively $\mathbf{X}_2$ and $\mathbf{X}$ into the high dimensional space and $\mathbf{K}$ denotes the kernel matrix $\boldsymbol{\Phi}_2\boldsymbol{\Phi}^\top$.

5. The SVMMaj package in R

The **SVM-Maj** algorithm for the Support Vector Machine (SVM) is implemented in the **SVM-Maj** package in R. Its main functions are `svmmaj`, which estimates the SVM, and

`svmmajcrossval`, which performs a grid search of k-fold cross validations using **SVM-Maj** to find the combination of input values, (such as λ and degree in the case of a polynomial kernel) giving the best prediction performance.

The `svmmaj` function requires the $n \times k$ attribute matrix `X` and the $n \times 1$ vector `y` with class labels. Apart from the data objects, other parameter input values can be given as input to tune the model: `lambda`, `hinge`, `weights.obs`, `scale` and parameters for nonlinearities and settings of the algorithm itself. Table 6 shows the arguments of function `svmmaj` and its default values. For example,

```
R> svmmaj(X, y, lambda = 2, hinge = "quadratic", scale = "interval")
```

runs the SVM model with $\lambda = 2$, using a *quadratic* hinge and for each attribute, the values are scaled to the interval [0,1].

The function `svmmajcrossval` uses the same parameter input values and additionally the parameters to be used as grid points of the k-fold cross validation. These parameters should be given in the list object `search.grid`, e.g.,

Parameter	Values	Description
`lambda`	scalar	Nonnegative penalty parameter of the penalty term. Default value is 1.
`hinge`		Specifies the hinge error term:
	`'absolute'`*	= Use the absolute hinge error.
	`'quadratic'`	= Use the quadratic hinge error.
	`'huber'`	= Use the Huber hinge error.
`weights.obs`	vector	Vector with the nonnegative weight for the residual of each object or class, which can either be a vector of length 2 to specify the weights for each class, or a vector of length `length(y)` giving the weights for each object i. Default value is `c(1,1)`
`scale`	`'zscore'`	=Rescales all variables in $\mathbf{X}$ to have zero mean and standard deviation equal to 1.
	`'interval'`	=Rescales all variables in $\mathbf{X}$ to be in the range $[0, 1]$.
	`'none'`*	=No scaling is done.
`decomposition`	logical	In case of a linear kernel, `decomposition=FALSE` will not perform a decomposition at all and uses update 4 instead.
`spline.knots`	integer	Number of interior knots of the spline which is equal to one less than the number of adjacent intervals. Default value is 0, which corresponds with a single interval.
`spline.degree`	integer	Degree of the spline basis. Default value is 1, representing the linear case.
`kernel`		The kernel function of package `kernlab`, which specifies which kernel type to be used. Possible kernels and its parameter values are among others:
	`vanilladot`	= The linear kernel: `k(x,x') = <x,x'>`.
	`polydot`	= The polynomial kernel: `k(x,x') = (scale <x,x'> + offset) ^ degree`.
	`rbfdot`	= The Gaussian Radial Basis Function (RBF) kernel: `k(x,x') = exp(-sigma |x - x'|) ^ 2`.
	`laplacedot`	= The Laplacian kernel: `k(x,x') = exp(-sigma |x - x'|)` .
`kernel.sigma`	1	Kernel parameter `sigma` used in the RBF and Laplacian kernel.
`kernel.degree`	1	Kernel parameter `degree` used in the polynomial kernel.
`kernel.scale`	1	Kernel parameter `scale` used in the polynomial kernel.
`kernel.offset`	0	Kernel parameter `offset` used in the polynomial kernel.
`convergence`	double	Specifies the convergence criterion of the algorithm. Default value is `3e-7`
`print.step`	logical	`print.step=TRUE` shows the progress of the iteration. Default value is `FALSE`.
`initial.point`	vector	Initial values $[\alpha_0 \ \boldsymbol{\rho}_0]$ (see Algorithm 1) to be used in the **SVM-Maj** algorithm, the length of the vector equals the rank of explanatory matrix `X` plus one.
`increase.step`	integer	The iteration level from where the relaxed update (11) will be used. Default value is 20.
`na.action`	`na.action`	Specifies which `na.action` object will be used to handle missing values. Default is omit observations with missing values.

Table 6: Parameter input values of `svmmaj`. The default settings are marked with *

```
R> svmmajcrossval(X, y, search.grid = list(lambda = c(1, 2, 4)))
```

performs a cross validation of `X` and `y` with as grid points $\lambda = 1, 2, 4$.

As an example, we use the `AusCredit` data set of the `libsvm` data sets (Chang and Lin 2001), which is included in the **SVMMaj** package. This data set consists of in total 690 credit requests, 307 of which are classified as positive and 383 as negative. These classifications are stored in `AusCredit$y` with class label `Rejected` to represent the negative responses, and label `Accepted` for the positive responses. In total, 14 attributes of each applicant has been stored as explanatory variables in `AusCredit$X`. Due to confidentiality, the labels of all explanatory variables are not available. Moreover, the observations in the data set `AusCredit` is split into `AusCredit.tr`, consisting the first 400 observations, and `AusCredit.te`, with the remaining 290 observations. `AusCredit.tr` will be used to estimate the model, while the `AusCredit.te` is used to analyze the prediction performances.

Example 1 *Train the SVM-model using the data set of Australian credit requests to classify the creditibility of the applicant.*

The command

```
R> library("SVMMaj")
```

loads the **SVMMaj** package into R. In this example, we will use the components `X` and `y` in the training set `AusCredit.tr`, which consists of explanatory variables respectively class labels of 400 credit requests, to train the model. Afterwards, the characteristics of the remaining 290 requests, which has been stored into component `X` of test set `AusCredit.te`, are used to predict the classification of the applicant. Using the actual class labels, `AusCredit.te$y`, the out of sample prediction performance is analyzed by comparing the ones with the predicted using the SVM model as estimated with `AusCredit.tr`. Running the SVM on the training data is done as follows.

```
R> model <- svmmaj(AusCredit.tr$X, AusCredit.tr$y, lambda = 100)
R> model

Model:
    update method                 svd
    attribute dimension           400 14
    degrees of freedom            14
    number of iterations          40
    loss value                    300.6324
    number of support vectors     359
```

As a result, the trained model will be returned as an `svmmaj`-object. The `print` method of this object shows which update method is used, the number of iterations before convergence, the found minimum loss value and the number of support vectors. In case no kernel is used, the matrix $\mathbf{Z}$ from update (7) is obtained through the QR-decomposition shown in (6) by default. In case a nonlinear kernel is used, $\mathbf{Z}$ is being calculated through the Cholesky decomposition (13). One can choose not to perform a decomposition when using a nonlinear kernel by

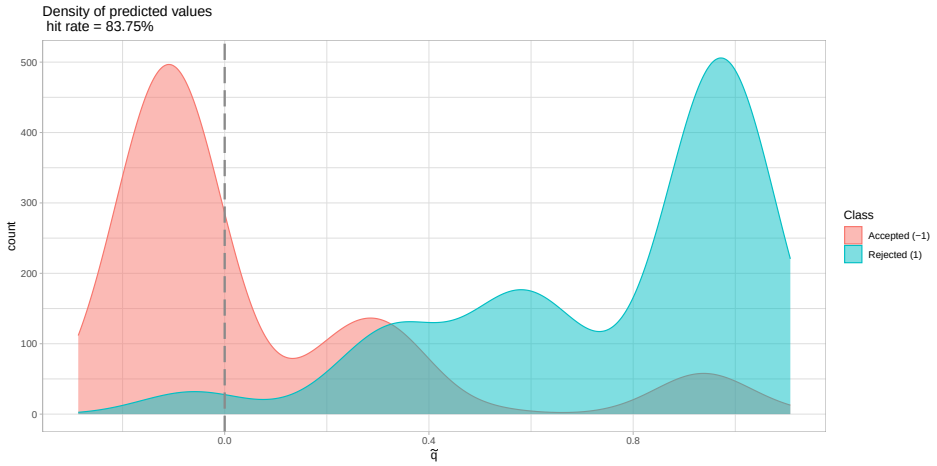

Figure 2: This figure shows the distribution of predicted values $\overline{\tilde{q}}$ of the two classes of Example 1, which can be obtained through the `plot()` method. These densities are created by the `density` function. This function specifies beforehand the bandwidth value to plot the density, which is shown on top of the graph.

specifying `decomposition = FALSE`. Then the original update (4) will be used. A graph of the distribution of the predicted values $\tilde{\mathbf{q}}$ for each class can be plotted via the `plot()` method using the `density` function of R, see Figure 2. The distribution shows that the majority of the `Accepted` class (-1) respondents received predicted $\tilde{q}_i$ close to -1 and only a few close to $+1$. The same holds for the `Rejected` class $(+1)$ respondents showing that the majority of respondents are correctly classified. A more detailed description of the model can be requested by using the `summary()` method.

```
R> summary(model)

Call:
   svmmaj.default(X = AusCredit.tr$X, y = AusCredit.tr$y, lambda = 100)

Settings:
    lambda                          100
    hinge error                     absolute
    spline basis                    no
    type of kernel                  linear

Data:
    class labels                    Accepted Rejected
    rank of X                       14
    number of predictor variables   14
    number of objects               400
    omitted objects                 0

Model:
```

```
    update method                       svd
    number of iterations                40
    loss value                          300.6324
    number of support vectors           359

Confusion matrix:
            Predicted(yhat)
Observed (y) Accepted Rejected Total
    Accepted      122       57   179
    Rejected        8      213   221
    Total         130      270   400

Classification Measures:

   hit rate                             0.837
   weighted hit rate                    0.838
   misclassification rate               0.163
   weighted missclassification rate     0.162

                   TP        FP Precision
    Accepted    0.682    0.3184     0.938
    Rejected    0.964    0.0362     0.789
```

The `Settings` segment describes the parameter settings used to estimate the model. In this example, the scales of the explanatory variables have not been changed and a linear model is specified because no spline or nonlinear kernel is used. Furthermore, the penalty term of the loss function consists of an absolute hinge, with a penalty parameter λ of 1. In the `Data` segment, the properties of the input data are shown: the labels of each class, the rank of the explanatory variable matrix $\mathbf{X}$ (in case of using I-splines, this will be the rank of the resulting spline bases) and the size of the data (the number of objects and number of predictor variables). **SVMMaj** has the possibility to handle missing values through a specified `na.action`-object. In case observations with missing values are omitted, the number of omitted observations will also be printed in this segment. The `Model` segment summarizes the trained model as a result of the **SVM-Maj** algorithm: it specifies which update has been used, the number of iterations needed to obtain this model, the optimal loss value and the number of support vectors. The classification performance of the model on the data used to estimate can be found in the last segment, `Classification table`, where the confusion matrix is followed by measured *true positive (TP)*, *false positive (FP)* and *precision* below. True positive of a class denotes the proportion of objects of that class that are being predicted correctly, whereas false positive denotes proportion of the incorrectly predicted objects of a class. The precision of a class is the proportion of correctly predicted objects of the class among all objects predicted to be classified as that class.

Next, we would like to test how well the estimated SVM model predicts an unseen sample: the 290 objects in `AusCredit.te$X` is used as hold-out sample. This is done through the `predict()` method.

```
R> predict(model, AusCredit.te$X)
```

```
Prediction frequencies:
          Accepted Rejected
frequency       85      205

Confusion matrix:
            Predicted(yhat)
Observed (y) Accepted Rejected Total
    Accepted       85        0    85
    Rejected        0      205   205
    Total          85      205   290

Classification Measures:

   hit rate                          1
   misclassification rate            0

                   TP        FP Precision
    Accepted        1         0         1
    Rejected        1         0         1
```

If the actual class labels are known, one can include this object in the method to show the prediction performance:

```
R> predict(model, AusCredit.te$X, AusCredit.te$y)

Prediction frequencies:
          Accepted Rejected
frequency       85      205

Confusion matrix:
            Predicted(yhat)
Observed (y) Accepted Rejected Total
    Accepted       76       52   128
    Rejected        9      153   162
    Total          85      205   290

Classification Measures:

   hit rate                          0.79
   weighted hit rate                 0.79
   misclassification rate            0.21
   weighted missclassification rate  0.21

                   TP        FP Precision
    Accepted     0.594    0.4062     0.894
    Rejected     0.944    0.0556     0.746
```

The classification measures of this unseen sample prediction are similar to these of the in-sample predictions, which mean that this model has no problem of overfitting. Moreover, average hit rate of 85% indicates that this model predicts the objects quite well. Note the difference in true positive value between the classes; it appears that this model predicts classes in class `Accepted` slightly better than the other class.

To show a the distribution of $\hat{\mathbf{q}}$ for all applicants or, if the actual class labels are given, for each class, the argument `show.plot=TRUE` can be included, which result into a figure as in Figure 3.

```
Prediction frequencies:
          Accepted Rejected
frequency       85      205

Confusion matrix:
           Predicted(yhat)
Observed (y) Accepted Rejected Total
    Accepted       76       52   128
    Rejected        9      153   162
    Total          85      205   290

Classification Measures:

   hit rate                           0.79
   weighted hit rate                  0.79
   misclassification rate             0.21
   weighted missclassification rate   0.21

                  TP        FP Precision
    Accepted   0.594    0.4062     0.894
    Rejected   0.944    0.0556     0.746
```

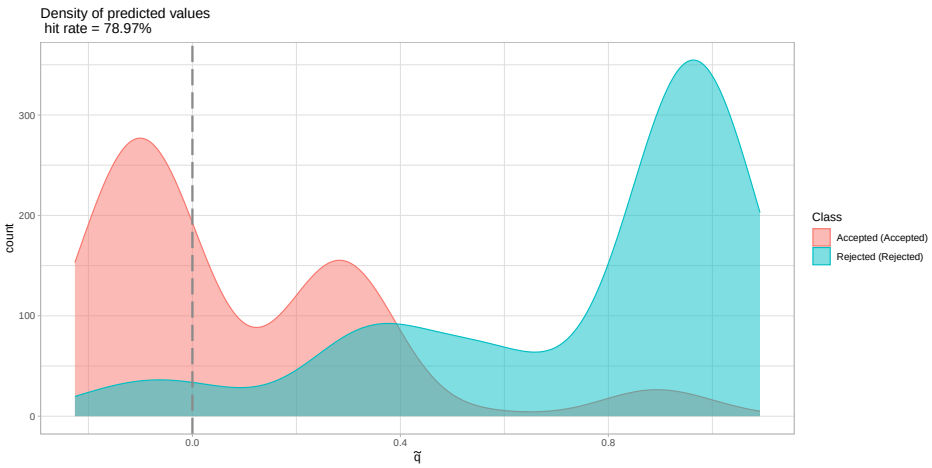

Figure 3: Densities of the predictions in $\tilde{\mathbf{q}}$ split by class.

As this model does not contain any nonlinearity and $\mathbf{X}$ is nonstandardized, $\tilde{q}_i$ for the holdout sample can also be directly computed using the weights `model$beta` and constant `model$theta[1]` found in the model and multiply this with the explanatory variables of the unseen sample `AusCredit.te$X`, after being coerced into a `matrix` object.

```
R> model <- svmmaj(AusCredit.tr$X, AusCredit.tr$y, scale = "none", lambda = 100)
R> alpha <- model$theta[1]
R> beta  <- model$beta
R> qu  <-  drop(alpha + data.matrix(AusCredit.te$X) %*% beta)
```

The predicted classes are then easily obtained by computing.

```
R> y <- factor(qu < 0, levels = c(TRUE, FALSE), labels = model$classes)
```

5.1. Cross validation

Until now, the default settings have been used. However, it is recommended to experiment with different parameters to obtain an optimal prediction model, in particular by varying the penalty parameter λ. To determine the optimal parameter values to be used in further analysis, one can use the `svmmajcrossval` function to perform cross validation with different parameter values. To show an example of using cross validation to determine λ, consider the `voting` data set the `libsvm` data sets (Chang and Lin 2001). This data set corresponds with 434 members of the U.S. House of Representatives Congressmen consisting of 167 republicans and 267 democrats. As the explanatory variables, for each of the 16 different political propositions, the votes of the politicians are registered. Using the SVM, we are trying to predict the political wing of the last 134 members using their 16 votes on the propositions as predictor variables. The first 300 members are used as a training sample. A five-fold cross validation is performed on these 300 members in the training sample, with a fine grid of lambda values $\lambda = 10^{-6}, 10^{-5.5}, \ldots, 10^{5}, 10^{5.5}, 10^{6}$. Among these lambda values, the optimal λ value is the one which results in the lowest misclassification rate.

Example 2 *Performing cross validation using the voting data sets to find an optimal value for* `lambda`*.*

```
R> library("SVMMaj")
```

In this example, we will use the data in `voting.tr` to perform five-fold cross validation to determine the optimal `lambda`. Then, this `lambda` is used in an SVM analysis on the entire data set `voting.tr`. Subsequently, this model is used to predict the classification of the unseen sample `voting.te`. Then, `voting.te$X` can be used to compare the prediction with the actual classification `voting.te$y`. This procedure can be executed in R using the following commands:

```
R> results.absolute <- svmmajcrossval(
+                                     voting.tr$X, voting.tr$y,
```

```
+                                    search.grid = list(lambda = 10^seq(4, -4)),
+                                    hinge = "absolute", convergence = 1e-4)
R> model <- svmmaj(
+                 voting.tr$X, voting.tr$y, hinge = "absolute",
+                 lambda = results.absolute$param.opt$lambda)
R> q.absolute <- predict(model, voting.te$X, voting.te$y)
```

5.2. Hinge functions

SVMMaj allows using the quadratic hinge or the Huber hinge instead of the standard absolute hinge. An important advantage of the quadratic hinge is that it has the potential to be computationally much more efficient than the absolute hinge. Let us perform a similar cross validation on the `voting` data set using the quadratic hinge by

```
R> results.quadratic <- svmmajcrossval(
+                                     voting.tr$X, voting.tr$y,
+                                     search.grid = list(lambda = 10^seq(4, -2, length.out
+                                     hinge = "quadratic", convergence = 1e-4)
R> model <- svmmaj(
+                 voting.tr$X, voting.tr$y, hinge = "quadratic",
+                 lambda = results.quadratic$param.opt$lambda)
R> q.quadratic <- predict(model, voting.te$X, voting.te$y)
```

A summary of the results of these examples can be found in Table 7. Figure 4 shows the hit rate of the cross validation using different lambda values. It can be seen that both hinges have similar out-of-sample predictive power. It is also clear that per iteration, the quadratic hinge is much faster than the absolute hinge. The effect of the increased computational efficiency of the quadratic hinge is in this example canceled by a large increase of the number of iterations as it requires more iterations to converges.

	Absolute	Quadratic
CPU-time (sec)	1.84	3.50
Mean no. of iter	211.64	839.73
CPU-time (per iter)	0.01	0.00
Optimal p	0.40	1.00
Hit rate (average TP)	0.56	0.54

Table 7: Results of the fivefold cross validation estimation using the `svmcrossval` function. CPU-time is the time in CPU seconds needed to perform fivefold cross validation and Optimal p the value of which 10^p returns the best hit rate in the cross validation. Mean no. of iterations denotes the average value of the sum of the number of iterations per `lambda`-value. CPU-time per iter is the mean computation time needed to perform one iteration. Hit rate is the average TP value by predicting the class labels of the holdout sample of 134 congress men using the first 300 congress men as sample of estimation.

5.3. Nonlinearities

The package **SVMMaj** can also implement nonlinearity in the model. One can choose to

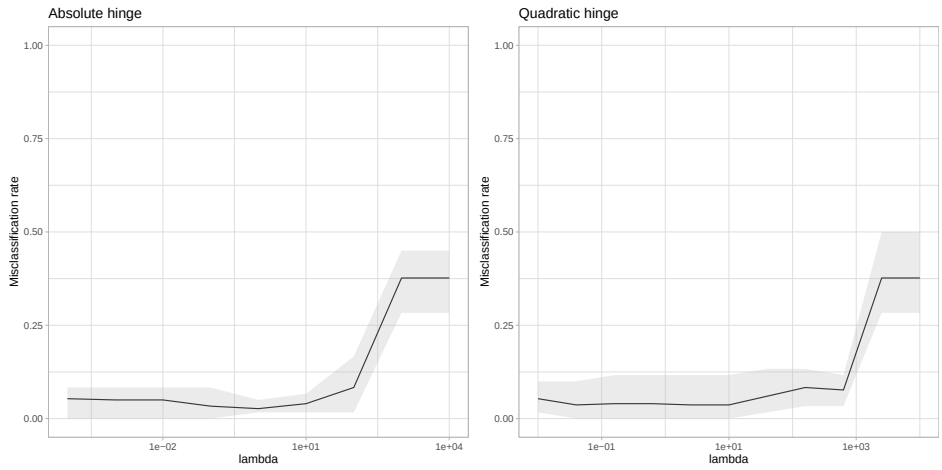

Figure 4: The function `svmmajcrossval` performs a five-fold cross validation of different `lambda` values of both absolute hinge (left panel) and the quadratic hinge (right panel). This figure shows the misclassification rate of different `lambda` values.

use I-splines, kernel matrices, or both methods to specify the nonlinearity. In the latter case, **SVMMaj** will first convert the explanatory matrix `X` into spline-basis and subsequently generate the kernel matrix of the spline-basis. An advantage of using I-splines over kernels is that on can easily plot the effect of one variable on the predicted value $\hat{q}_i$. In the following example, we will use the `diabetes` dataset of `libsvm` data sets (Chang and Lin 2001).

Example 3 *Train a model with nonlinearities on a data set of 786 females of Pima Indian heritage, predicting the presence of diabetes using several demographic and medial variables.*

In this example, we will use the nonlinear options of `svmmaj` to train a model consisting of 768 female at least 21 years old of Pima Indian heritage, of which 268 are being tested positive for diabetes. In this model, we use 8 variables of state-of-health measures to classify these patients as positive for diabetes or negative. As the number of persons with a positive test result is smaller than the ones with negative results, the loss term of the patients belonging to the former group will be weighted twice as heavy by the extra argument `weights.obs=list(positive=2,negative=1)` to indicate the double weight on the second group. As the rank of the explanatory matrix $\mathbf{X}$ is expected to be large, we will use quadratic hinge to make use of the computational efficiency discussed before.

I-Spline

One way of applying nonlinearities in the model is using splines. In this example, we will transform each variable into spline basis of 5 knots and a degree 2, yielding a rank of $8 \times (5 + 2) = 56$, that is, $k = 8$ times the number of columns per spline basis (5 interior knots plus the degree of 2). Five-fold cross validation is used with a grid of $\lambda = 10^{-6}, 10^{-5}, ..., 10^{4}, 10^{5}, 10^{6}$ to determine the optimal λ value.

```
R> results.spline <- svmmajcrossval(
+                                   diabetes.tr$X, diabetes.tr$y,
```

```
+                                    scale = "interval", search.grid = list(lambda = 10^seq(
+                                    hinge = "quadratic", spline.knots = 5, spline.degree =
+                                    weights.obs = list(positive = 2, negative = 1))
R> model.spline <- svmmaj(
+                         diabetes.tr$X, diabetes.tr$y,
+                         scale = "interval", lambda = results.spline$param.opt$lambda,
+                         spline.knots = 5, spline.degree = 2, hinge = "quadratic",
+                         weights.obs = list(positive = 2, negative = 1))
R> predict(
+         model.spline, diabetes.te$X, diabetes.te$y,
+         weights = list(positive = 2, negative = 1))

Prediction frequencies:
          negative positive
frequency       91       77

Confusion matrix:
            Predicted(yhat)
Observed (y) negative positive Total
    negative       80       28   108
    positive       11       49    60
    Total          91       77   168

Classification Measures:

   hit rate                           0.768
   weighted hit rate                  0.781
   misclassification rate             0.232
   weighted missclassification rate   0.219

                  TP        FP Precision
    negative   0.741     0.259     0.879
    positive   0.817     0.183     0.636
```

The optimal `lambda` for the model using I-splines can be found in `results.spline$param.opt`, which is 10^1. This `lambda` value equals the penalty value which will give the lowest misclassification rate in the cross-validation. One of the advantages of using splines to handle nonlinear prediction is the possibility to show the effect of a variable by plotting its estimated transformation. Figure 5 shows these plots of the splines per variable, which can be used for interpretation of the effects of each individual variable. In this figure, one can clearly see nonlinear effects in most variables. For example, the *diabetes pedigree* (`x7`) shows a positive relation with respect to female patients with `positive` results, but with a diminishing returns to scale. On the other hand, *Age* (`x8`) has a reverse v-shape: respondents who are around 40 are more likely to have diabetes. Overall, *glucose concentration* (`x2`) and *BMI* (`x6`) have the largest effect on the class prediction when its values are large.

Kernel

Another way of implementing nonlinearity in the model by using a kernel. In this example, we will use the Radial Basis Function in our model training. To find the optimal λ and σ values we performed a five fold cross validation with the grids: $\lambda = 10^{-6}, 10^{-5}...10^{4}, 10^{5}, 10^{6}$ and $\sigma = 2^{-5}, 2^{-4}...2^{4}, 2^{5}$ by the following commands.

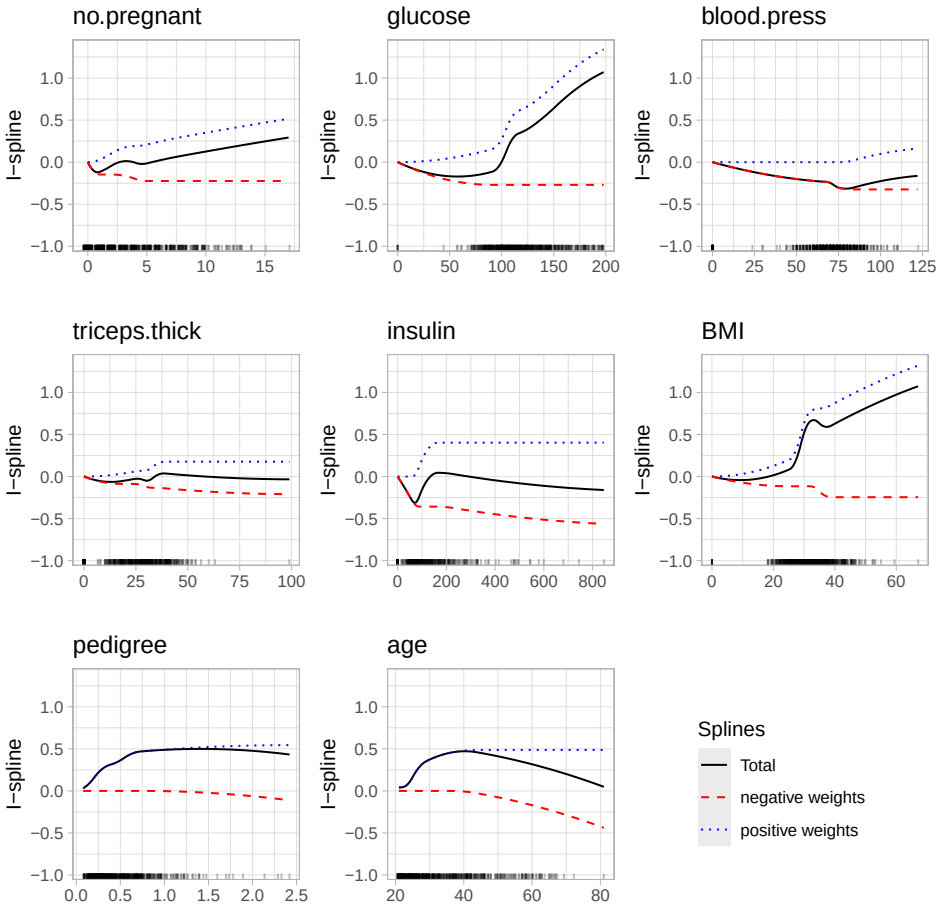

Figure 5: The spline plots of each of 8 variables used to predict the result of a test for diabetes among females of Pima Indian heritages. This model has been performed with `lambda = 10` and the spline basis with 5 inner knots and a degree of 2. Each graph denotes the loss term of the corresponding variable with different values. The higher the predicted value $\tilde{q}$, the higher the probability of positive test.

```
R> library(kernlab)
R> results.kernel <- svmmajcrossval(
+                                   diabetes.tr$X, diabetes.tr$y,
+                                   scale = "interval",
+                                   search.grid = list(
+                                     kernel.sigma = 2^seq(-4, 4, by = 2),
```

```
+                                    lambda = 10^seq(-4, 4, by = 2)
+                                  ), hinge = "quadratic", kernel = rbfdot,
+                                  weights.obs = list(positive = 2, negative = 1),
+                                  options = list(convergence = 1e-4))
R> model.kernel <- svmmaj(
+                        diabetes.tr$X, diabetes.tr$y,
+                        scale = "interval", lambda = results.kernel$param.opt$lambda,
+                        kernel.sigma = results.kernel$param.opt$kernel.sigma,
+                        kernel = rbfdot, hinge = "quadratic",
+                        weights.obs = list(positive = 2, negative = 1))
R> predict(
+         model.kernel, diabetes.te$X, diabetes.te$y,
+         weights = list(positive = 2, negative = 1))

Prediction frequencies:
          negative positive
frequency       82       86

Confusion matrix:
            Predicted(yhat)
Observed (y) negative positive Total
    negative       73       35   108
    positive        9       51    60
    Total          82       86   168

Classification Measures:

   hit rate                           0.738
   weighted hit rate                  0.768
   misclassification rate             0.262
   weighted missclassification rate   0.232

                  TP        FP Precision
    negative   0.676     0.324     0.890
    positive   0.850     0.150     0.593
```

Observing the prediction results, we can see that the model using I-splines has a higher TP-value of female persons having `positive` result in diabetes as well as the hit rate (average TP-value) suggesting that for these data the I-spline transformation is better able to pick up the nonlinearities in the predictor variables than the radial basis kernel.

6. Discussion

This paper introduces the R-package **SVMMaj**, This package implements the **SVM-Maj** algorithm of (Groenen *et al.* 2007, 2008) with the addition of nonlinear models with kernels. One of the advantages of the **SVM-Maj** approach is the competitively fast training speed for

medium sized problems. Furthermore, it allows individual objects in a training dataset to receive different individual weight.

Another advantage of **SVM-Maj** is the possibility to use different loss functions, besides the commonly used absolute hinge. In this package, the absolute hinge, quadratic hinge and Huber hinge has been implemented. Nevertheless, this can be expanded to any other error function $f(q)$ that satisfies the following condition: the second derivative of its function has a bounded maximum, so that a quadratic majorization function can be found. If in addition $f(q)$ is convex, then the overall loss function (3) is strictly convex, so that the the **SVM-Maj** algorithm is guaranteed to stop at the global minimum.

A. Efficient updates for SVM-Maj

Recall that the relationship between $\mathbf{q}$ and $\boldsymbol{\beta}$ can be written as

$$\mathbf{q} = \mathbf{X}\boldsymbol{\beta}. \tag{15}$$

In some situations, different values of $\boldsymbol{\beta}$ may lead to the same $\mathbf{q}$, when deriving $\mathbf{q}$ from $\boldsymbol{\beta}$. In other situations, the opposite could happen, that is, several values of $\mathbf{q}$ may result into the same $\boldsymbol{\beta}$ value. Thus, there is not always a one-to-one mapping of $\mathbf{q}$ to $\boldsymbol{\beta}$ and the reverse. Therefore, one should take these possible situations into account when performing iterative majorization. To illustrate this, we will use the singular value decomposition (SVD) of $\mathbf{X}$:

$$\underset{(n \times k)}{\mathbf{X}} = \underset{(n \times r)}{\mathbf{P}} \quad \underset{(r \times r)}{\mathbf{\Lambda}} \quad \underset{(r \times k)}{\mathbf{Q}^\top}, \tag{16}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are orthonormal matrices which satisfy $\mathbf{P}^\top\mathbf{P} = \mathbf{I}$ and $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$ and $\mathbf{\Lambda}$ is a diagonal matrix. The relationship between $\mathbf{q}$ and $\boldsymbol{\beta}$ can then be written as:

$$\mathbf{q} = \mathbf{X}\boldsymbol{\beta} = \mathbf{P}\mathbf{\Lambda}\mathbf{Q}^\top\boldsymbol{\beta}, \tag{17}$$

$$\underset{(r \times 1)}{\mathbf{P}^\top\mathbf{q}} = \underset{(r \times r)}{\mathbf{\Lambda}} \quad \underset{(r \times 1)}{\mathbf{Q}^\top\boldsymbol{\beta}}. \tag{18}$$

Let us first examine the left part of the equation. $\mathbf{q}$ can be written as a combination of two orthogonal vectors, a projection of $\mathbf{q}$ on $\mathbf{P}$ and a projection on its complement $(\mathbf{I} - \mathbf{P}\mathbf{P}^\top)$, that is,

$$\mathbf{q} = \mathbf{P}\mathbf{P}^\top\mathbf{q} + (\mathbf{I} - \mathbf{P}\mathbf{P}^\top)\mathbf{q} = \mathbf{q}_B + \mathbf{q}_N. \tag{19}$$

Multiplying both sides with $\mathbf{P}^\top$ gives

$$\begin{aligned}
\mathbf{P}^\top\mathbf{q} &= \mathbf{P}^\top(\mathbf{q}_B + \mathbf{q}_N) \\
&= \mathbf{P}^\top\mathbf{q}_B + \mathbf{P}^\top(\mathbf{I} - \mathbf{P}\mathbf{P}^\top)\mathbf{q} \\
&= \mathbf{P}^\top\mathbf{q}_B + (\mathbf{P}^\top - \mathbf{P}^\top)\mathbf{q} \\
&= \mathbf{P}^\top\mathbf{q}_B + \mathbf{0} \\
&= \mathbf{P}^\top\mathbf{q}_B.
\end{aligned}$$

In other words, the left part of (18) is only dependent of $\mathbf{q}_B$. When $r = n$, $\mathbf{P}\mathbf{P}^\top$ equals $\mathbf{I}$ and thus, $\mathbf{q} = \mathbf{q}_B + \mathbf{q}_N = \mathbf{q}_B + (\mathbf{I} - \mathbf{I})\mathbf{q} = \mathbf{q}_B$. In this case there is always an unique solution of

$$\mathbf{P}^\top\mathbf{q} = \mathbf{\Lambda}\boldsymbol{\theta}, \text{for any } \boldsymbol{\theta} \in \Re^r. \tag{20}$$

However, when $r < n$, $\mathbf{P}\mathbf{P}^\top = \mathbf{I}$ does not hold and there are infinitely many solutions to (20) so that an one-to-one relationship of $\mathbf{q}$ and $\boldsymbol{\beta}$ in (15) is lost indeed.

Similarly, $\boldsymbol{\beta}$ can be written as $\boldsymbol{\beta} = \mathbf{Q}\mathbf{Q}^\top\boldsymbol{\beta} + (\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top)\boldsymbol{\beta} = \boldsymbol{\beta}_B + \boldsymbol{\beta}_N$, and $\mathbf{Q}^\top\boldsymbol{\beta} = \mathbf{Q}^\top(\boldsymbol{\beta}_B + \boldsymbol{\beta}_N) = \mathbf{Q}^\top\boldsymbol{\beta}_B$. If $r < k$ then there is no unique solution to $\mathbf{Q}^\top\boldsymbol{\beta} = \boldsymbol{\theta}$ with $\boldsymbol{\theta} \in \Re^r$. Note that $\boldsymbol{\beta}_B$ and $\boldsymbol{\beta}_N$ are independent to each other and that

$$\boldsymbol{\beta}_N^\top\boldsymbol{\beta}_B = \boldsymbol{\beta}_B^\top\boldsymbol{\beta}_N = \boldsymbol{\beta}^\top(\mathbf{Q}\mathbf{Q}^\top)(\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top)\boldsymbol{\beta} = 0, \tag{21}$$

as $\mathbf{Q}\mathbf{Q}^\top(\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top) = \mathbf{0}$.

As a result, we have to take care that a proper relation between $\boldsymbol{\beta}$ and $\mathbf{q}$ is retained, when more efficient updates are derived. In the next section, we will examine three different situations and introduce the optimization method by optimizing the parameters which has the lowest dimension, that is, $\min(r, n, k)$. In this way, it can be assured that $\mathbf{q}_N$ and $\boldsymbol{\beta}_N$ are both zero vectors. Subsequently, we will discuss the use of each optimization method in each situation.

A.1. $\boldsymbol{\beta}$-method: Full rank and more objects than variables ($n > k$ and $r = k$)

When the number of variables is smaller than the number of objects, and if $\mathbf{X}$ is of full rank, then $\mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$ and each $\boldsymbol{\beta}$ will give an unique $\mathbf{q}$. As $\dim(\boldsymbol{\beta}) < \dim(\mathbf{q})$, it is most efficient to optimize the loss function through $\boldsymbol{\beta}$. The majorization function can as follows be written as a function of $\boldsymbol{\beta}$.

$$\text{Maj}(\boldsymbol{\beta}, \alpha) = (\alpha\mathbf{1} + \mathbf{X}\boldsymbol{\beta})^\top\mathbf{A}(\alpha\mathbf{1} + \mathbf{X}\boldsymbol{\beta}) - 2(\alpha\mathbf{1} + \mathbf{X}\boldsymbol{\beta})^\top\mathbf{b} + \lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} + o. \tag{22}$$

To derive an update, we set the first derivatives of (22) with respect to $\boldsymbol{\beta}$ and α to zero, which yields

$$\begin{aligned}\mathbf{1}^\top\mathbf{A}(\mathbf{1}\alpha + \mathbf{X}\boldsymbol{\beta}) &= \mathbf{1}^\top\mathbf{b}\\ \mathbf{X}^\top\mathbf{A}(\mathbf{1}\alpha + \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta} &= \mathbf{X}^\top\mathbf{b},\end{aligned}$$

or in matrix form,

$$\begin{bmatrix}\mathbf{1}^\top\mathbf{A}\mathbf{1} & \mathbf{1}^\top\mathbf{A}\mathbf{X}\\ \mathbf{X}^\top\mathbf{A}\mathbf{1} & \mathbf{X}^\top\mathbf{A}\mathbf{X} + \lambda\mathbf{I}\end{bmatrix}\begin{bmatrix}\alpha\\ \boldsymbol{\beta}\end{bmatrix} = \begin{bmatrix}\mathbf{1}^\top\mathbf{b}\\ \mathbf{X}^\top\mathbf{b}\end{bmatrix}.$$

Using the fact that $\alpha_0\mathbf{1} + \mathbf{X}\boldsymbol{\beta} = \begin{bmatrix}\mathbf{1} & \mathbf{X}\end{bmatrix}\begin{bmatrix}\alpha\\ \boldsymbol{\beta}\end{bmatrix}$, an update of both $\boldsymbol{\beta}$ and α_0 can be derived by solving the linear system

$$\left(\begin{bmatrix}\mathbf{1}^\top\\ \mathbf{X}^\top\end{bmatrix}\mathbf{A}\begin{bmatrix}\mathbf{1} & \mathbf{X}\end{bmatrix} + \lambda\begin{bmatrix}0 & \mathbf{0}^\top\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\right)\begin{bmatrix}\alpha^+\\ \boldsymbol{\beta}^+\end{bmatrix} = \begin{bmatrix}\mathbf{1}^\top\mathbf{b}\\ \mathbf{X}^\top\mathbf{b}\end{bmatrix},$$

or, in compact form

$$\left(\widetilde{\mathbf{X}}^\top\mathbf{A}\widetilde{\mathbf{X}} + \lambda\mathbf{J}\right)\begin{bmatrix}\alpha^+\\ \boldsymbol{\beta}^+\end{bmatrix} = \widetilde{\mathbf{X}}^\top\mathbf{b}, \tag{23}$$

where $\widetilde{\mathbf{X}} = \begin{bmatrix}\mathbf{1} & \mathbf{X}\end{bmatrix}$ and $\mathbf{J} = \begin{bmatrix}0 & \mathbf{0}^\top\\ \mathbf{0} & \mathbf{I}\end{bmatrix}$.

To prove that the the linear system (23) has a unique solution under the assumption that $\mathbf{X} \neq \mathbf{0}$, we will show that the matrix $\widetilde{\mathbf{X}}^\top \mathbf{A} \widetilde{\mathbf{X}} + \lambda \mathbf{J}$ is positive definite. Let us examine the following equation

$$\begin{aligned} \begin{bmatrix} \alpha & \boldsymbol{\beta}^\top \end{bmatrix} \left(\widetilde{\mathbf{X}}^\top \mathbf{A} \widetilde{\mathbf{X}} + \lambda \mathbf{J} \right) \begin{bmatrix} \alpha \\ \boldsymbol{\beta}^\top \end{bmatrix} &= \begin{bmatrix} \alpha & \boldsymbol{\beta} \end{bmatrix} \widetilde{\mathbf{X}}^\top \mathbf{A} \widetilde{\mathbf{X}} \begin{bmatrix} \alpha \\ \boldsymbol{\beta} \end{bmatrix} + \lambda \begin{bmatrix} \alpha & \boldsymbol{\beta}^\top \end{bmatrix} \mathbf{J} \begin{bmatrix} \alpha \\ \boldsymbol{\beta} \end{bmatrix} \\ &= (\alpha \mathbf{1} + \mathbf{X}\boldsymbol{\beta})^\top \mathbf{A} (\alpha \mathbf{1} + \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^\top \boldsymbol{\beta}. \end{aligned} \tag{24}$$

From this equation, we can see that (24) equals the sum of two positive values. Furthermore, the right part $\lambda \boldsymbol{\beta}^\top \boldsymbol{\beta}$ of the equation equals zero only if $\boldsymbol{\beta} = \mathbf{0}$, whereas the left part will be zero only when $(\alpha \mathbf{1} + \mathbf{X}\boldsymbol{\beta}) = \mathbf{0}$. As both equations only hold when $\alpha = 0$ and $\boldsymbol{\beta} = \mathbf{0}$, we know that the matrix $\widetilde{\mathbf{X}}^\top \mathbf{A} \widetilde{\mathbf{X}} + \lambda \mathbf{J}$ is positive definite.

An update of $\mathbf{q}$ can be calculated by $\mathbf{q}^+ = \mathbf{X}\boldsymbol{\beta}^+$. Note that $\mathbf{q}_N$ necessarily equals zero, as

$$\mathbf{q}_N = (\mathbf{I} - \mathbf{P}\mathbf{P}^\top)\mathbf{q} = (\mathbf{I} - \mathbf{P}\mathbf{P}^\top)\mathbf{P}\boldsymbol{\Lambda}\mathbf{Q}^\top \boldsymbol{\beta} = (\mathbf{P} - \mathbf{P}\mathbf{P}^\top\mathbf{P})\boldsymbol{\Lambda}\mathbf{Q}^\top \boldsymbol{\beta} = \mathbf{0}.$$

A.2. q-method: Full rank and less objects than variables ($n < k$ and $r = n$)

When $k > n$, or more general $k > r$, we know that there are no unique solutions to $\mathbf{q} = \mathbf{X}\boldsymbol{\beta}$ and that $\mathbf{X}\boldsymbol{\beta} = \mathbf{X}(\boldsymbol{\beta}_B + \boldsymbol{\beta}_N) = \mathbf{X}\boldsymbol{\beta}_B$, that is, $\mathbf{q}$ is not dependent of $\boldsymbol{\beta}_N$. Consider the penalty term $\lambda \boldsymbol{\beta}^\top \boldsymbol{\beta}$. As $\mathbf{Q}\mathbf{Q}^\top(\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top) = \mathbf{0}$, the penalty term can be simplified as

$$\lambda \boldsymbol{\beta}^\top \boldsymbol{\beta} = \lambda(\boldsymbol{\beta}_B + \boldsymbol{\beta}_N)^\top(\boldsymbol{\beta}_B + \boldsymbol{\beta}_N) = \lambda \boldsymbol{\beta}_B^\top \boldsymbol{\beta}_B + \lambda \boldsymbol{\beta}_N^\top \boldsymbol{\beta}_N$$

Moreover, as $\lambda \boldsymbol{\beta}_N^\top \boldsymbol{\beta}_N \geq 0$ and $\mathbf{q}$ does not depend on $\boldsymbol{\beta}_N$, $\boldsymbol{\beta}_N$ can be set to zero, with the result that $\boldsymbol{\beta} = \boldsymbol{\beta}_B$. Nevertheless, when the number of variables k is larger than the number of objects n, and when $\mathbf{X}$ is of full rank, that is $r = k$, then $\mathbf{P}\mathbf{P}^\top = \mathbf{I}$ and each $\mathbf{q}$ will give an unique $\boldsymbol{\beta}$. As $\dim(\mathbf{q}) < \dim(\boldsymbol{\beta})$, it is most efficient to optimize the loss function through $\mathbf{q}$. $\boldsymbol{\beta}$ can then be derived using (18), that is,

$$\boldsymbol{\beta} = \mathbf{Q}\boldsymbol{\Lambda}^{-1}\mathbf{P}^\top \mathbf{q} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{P}^\top \mathbf{P}\boldsymbol{\Lambda}^{-2}\mathbf{P}^\top \mathbf{q} = \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{q},$$

using the fact that $(\mathbf{X}\mathbf{X}^\top)(\mathbf{P}\boldsymbol{\Lambda}^{-2}\mathbf{P}^\top)(\mathbf{X}\mathbf{X}^\top) = (\mathbf{P}\boldsymbol{\Lambda}^2\mathbf{P})(\mathbf{P}\boldsymbol{\Lambda}^{-2}\mathbf{P}^\top)(\mathbf{P}\boldsymbol{\Lambda}^2\mathbf{P}) = (\mathbf{P}\boldsymbol{\Lambda}^2\mathbf{P})$. Note that $\boldsymbol{\beta}$ does not depend on $\mathbf{q}_N$, as $\boldsymbol{\beta} = \mathbf{Q}\boldsymbol{\Lambda}^{-1}\mathbf{P}^\top \mathbf{q} = \mathbf{Q}\boldsymbol{\Lambda}^{-1}\mathbf{P}^\top \mathbf{q}_B$. The penalty term $\lambda \boldsymbol{\beta}^\top \boldsymbol{\beta}$ can then be written as

$$\begin{aligned} \lambda \boldsymbol{\beta}^\top \boldsymbol{\beta} &= \lambda(\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{q})^\top(\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{q}) \\ &= \lambda \mathbf{q}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{q} = \lambda \mathbf{q}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{q} = \lambda \mathbf{q}^\top \mathbf{K}^{-1}\mathbf{q}, \end{aligned}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}^\top = \mathbf{P}\boldsymbol{\Lambda}^2\mathbf{P}^\top$.

Therefore, the majorization function can as follows be written as a function of $\mathbf{q}$.

$$\text{Maj}(\boldsymbol{\beta}, \alpha) = (\alpha \mathbf{1} + \mathbf{q})^\top \mathbf{A}(\alpha \mathbf{1} + \mathbf{q}) - 2(\alpha \mathbf{1} + \mathbf{q})^\top \mathbf{b} + \lambda \mathbf{q}^\top \mathbf{K}^{-1}\mathbf{q}. \tag{25}$$

The first-order conditions of (25) are

$$\begin{aligned} \mathbf{1}^\top \mathbf{A}(\mathbf{1}\alpha + \mathbf{q}) &= \mathbf{1}^\top \mathbf{b}, \\ \mathbf{A}(\mathbf{1}\alpha + \mathbf{q}) + \lambda \mathbf{K}^{-1}\mathbf{q} &= \mathbf{b}. \end{aligned}$$

Using $\begin{bmatrix}\mathbf{1} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\alpha \\ \mathbf{q}\end{bmatrix} = \mathbf{1}\alpha + \mathbf{q} = \tilde{\mathbf{q}}$, the parameters $\mathbf{q}$ and α can be updated by deriving

$$\left(\begin{bmatrix}\mathbf{1}^\top \\ \mathbf{I}\end{bmatrix}\mathbf{A}\begin{bmatrix}\mathbf{1} & \mathbf{I}\end{bmatrix} + \lambda\begin{bmatrix}0 & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{K}^{-1}\end{bmatrix}\right)\begin{bmatrix}\alpha^+ \\ \mathbf{q}^+\end{bmatrix} = \begin{bmatrix}\mathbf{1}^\top\mathbf{b} \\ \mathbf{b}\end{bmatrix},$$

or, in compact form

$$(\tilde{\mathbf{I}}^\top\mathbf{A}\tilde{\mathbf{I}} + \lambda\mathbf{L})\begin{bmatrix}\alpha^+ \\ \mathbf{q}^+\end{bmatrix} = \tilde{\mathbf{I}}^\top\mathbf{b}, \tag{26}$$

where $\tilde{\mathbf{I}} = \begin{bmatrix}\mathbf{1} & \mathbf{I}\end{bmatrix}$ and $\mathbf{L} = \begin{bmatrix}0 & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{K}^{-1}\end{bmatrix}$.

Similar to (23), we can show that $(\tilde{\mathbf{I}}^\top\mathbf{A}\tilde{\mathbf{I}} + \lambda\mathbf{L})$ is positive definite and thus (26) can always be solved.

Also, the corresponding update for $\boldsymbol{\beta}$, which is

$$\boldsymbol{\beta}^+ = \mathbf{Q}\boldsymbol{\Lambda}^{-1}\mathbf{P}^\top\mathbf{q} = \mathbf{X}(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{q}, \tag{27}$$

equals $\boldsymbol{\beta}_B$, as

$$\boldsymbol{\beta}_N = (\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top)\boldsymbol{\beta} = (\mathbf{I} - \mathbf{Q}\mathbf{Q}^\top)\mathbf{Q}\boldsymbol{\Lambda}^{-1}\mathbf{P}^\top\mathbf{q} = (\mathbf{Q} - \mathbf{Q}\mathbf{Q}^\top\mathbf{Q})\boldsymbol{\Lambda}^{-1}\mathbf{P}^\top\mathbf{q} = \mathbf{0}.$$

A.3. $\boldsymbol{\theta}$-method: Rank is smaller than either n or k ($r <$ min(n, k))

When $r < \min(n, k)$, the interdependence of $\mathbf{q}$ and $\mathbf{X}$ can be summarized in an $r \times 1$ vector $\boldsymbol{\theta} = \mathbf{Q}^\top\boldsymbol{\beta} = \boldsymbol{\Lambda}^{-1}\mathbf{P}^\top\mathbf{q}$ from (18). As $\mathbf{q}_N$ and $\boldsymbol{\beta}_N$ are not of interest, it is efficient to optimize the loss function by $\boldsymbol{\theta}$. $\boldsymbol{\beta}$ will then be calculated through $\boldsymbol{\beta} = \mathbf{Q}\boldsymbol{\theta}$, which will assure that $\boldsymbol{\beta} = \mathbf{Q}\boldsymbol{\theta} = \mathbf{Q}\mathbf{Q}^\top\boldsymbol{\beta} = \boldsymbol{\beta}_B$, in a similar way, it can be shown that $\mathbf{q} = \mathbf{q}_B$. Using the fact that $\mathbf{q} = \mathbf{X}\boldsymbol{\beta} = \mathbf{P}\boldsymbol{\Lambda}\mathbf{Q}^\top\boldsymbol{\beta} = \mathbf{P}\boldsymbol{\Lambda}\boldsymbol{\theta}$ and $\boldsymbol{\theta}^\top\boldsymbol{\theta} = \boldsymbol{\beta}^\top\mathbf{Q}\mathbf{Q}^\top\boldsymbol{\beta} = \boldsymbol{\beta}_B^\top\boldsymbol{\beta}_B = \boldsymbol{\beta}^\top\boldsymbol{\beta}$, the majorization function can be written as

$$\text{Maj}(\boldsymbol{\beta}, \alpha) = (\alpha\mathbf{1} + \mathbf{P}\boldsymbol{\Lambda}\boldsymbol{\theta})^\top\mathbf{A}(\alpha\mathbf{1} + \mathbf{P}\boldsymbol{\Lambda}\boldsymbol{\theta}) - 2(\alpha\mathbf{1} + \mathbf{P}\boldsymbol{\Lambda}\boldsymbol{\theta})^\top\mathbf{b} + \lambda\boldsymbol{\theta}^\top\boldsymbol{\theta}, \tag{28}$$

with the first-order condition

$$\begin{aligned}\mathbf{1}^\top\mathbf{A}(\mathbf{1}\alpha + \mathbf{P}\boldsymbol{\Lambda}\boldsymbol{\theta}) &= \mathbf{1}^\top\mathbf{b} \\ \boldsymbol{\Lambda}\mathbf{P}^\top\mathbf{A}(\mathbf{1}\alpha + \mathbf{P}\boldsymbol{\Lambda}\boldsymbol{\theta}) + \lambda\mathbf{I}\boldsymbol{\theta} &= \boldsymbol{\Lambda}\mathbf{P}^\top\mathbf{b}.\end{aligned}$$

Using $\begin{bmatrix}\mathbf{1} & \mathbf{P}\boldsymbol{\Lambda}\end{bmatrix}\begin{bmatrix}\alpha \\ \boldsymbol{\theta}\end{bmatrix} = \mathbf{1}\alpha_0 + \mathbf{P}\boldsymbol{\Lambda}\boldsymbol{\theta} = \tilde{\mathbf{q}}$, the parameters $\boldsymbol{\theta}$ and α_0 can be updated by deriving

$$\left(\begin{bmatrix}\mathbf{1}^\top \\ \boldsymbol{\Lambda}\mathbf{P}^\top\end{bmatrix}\mathbf{A}\begin{bmatrix}\mathbf{1} & \mathbf{P}\boldsymbol{\Lambda}\end{bmatrix} + \lambda\begin{bmatrix}0 & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{I}\end{bmatrix}\right)\begin{bmatrix}\alpha^+ \\ \boldsymbol{\theta}^+\end{bmatrix} = \begin{bmatrix}\mathbf{1}^\top\mathbf{b} \\ \boldsymbol{\Lambda}\mathbf{P}^\top\mathbf{b}\end{bmatrix},$$

or, in compact form

$$(\widetilde{\mathbf{P}}^\top\mathbf{A}\widetilde{\mathbf{P}} + \lambda\mathbf{J})\begin{bmatrix}\alpha^+ \\ \boldsymbol{\theta}^+\end{bmatrix} = \widetilde{\mathbf{P}}^\top\mathbf{b}, \tag{29}$$

where $\widetilde{\mathbf{P}} = \begin{bmatrix}\mathbf{1} & \mathbf{P}\boldsymbol{\Lambda}\end{bmatrix}$.

The advantage of this method is that $r \leq \min(n, k)$, which means that it restricts to the space of which the relationship between $\mathbf{q}$ and $\boldsymbol{\beta}$ is described. Moreover, it is assured that the dimension of $\boldsymbol{\theta}$ is the lowest of three (that is, $r \leq \min(n, k)$), and thus it is most efficient and consistent algorithm. However, this method requires the SVD or the QR decomposition to be computed, which may need much computational time in case n and k are large. Therefore, one should consider the alternatives when the matrix $\mathbf{X}$ is of full rank, that is when $r = \min(n, k)$.

References

Borg I, Groenen P (2005). *Modern multidimensional scaling*. Springer-Verlag New York, Inc.

Boser BE, Guyon IM, Vapnik VN (1992). "A training algorithm for optimal margin classifiers." In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pp. 144–152. ACM, New York, NY, USA.

Burges CJC (1998). "A Tutorial on Support Vector Machines for Pattern Recognition." *Data Min. Knowl. Discov.*, **2**, 121–167.

Chang CC, Lin CJ (2001). *LIBSVM: a library for support vector machines*. Software available at https://www.csie.ntu.edu.tw/~cjlin/libsvm/.

Chapelle O, Haffner P, Vapnik V (1999). "Support vector machines for histogram-based image classification." *IEEE Transactions on Neural Networks*, **10**(5), 1055–1064.

Collobert R, Bengio S (2001). "SVMTorch: support vector machines for large-scale regression problems." *Journal of Machine Learning Research*, **1**, 143–160.

Cui D, Curry D (2005). "Prediction in Marketing Using the Support Vector Machine." *Marketing Science*, **24**(4), 595–615.

De Leeuw J (1994). "Block relaxation algorithms in statistics." In H-H Bock, W Lenski, MM Richter (eds.), *Information systems and data analysis*, pp. 308–324. Springer-Verlag, Berlin.

De Leeuw J, Heiser W (1980). *Multidimensional scaling with restrictions on the configuration*, volume 5, pp. 285–317. In P. R. Krishnaiah (Ed.), Amsterdam, North-Holland.

Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008). "LIBLINEAR: A Library for Large Linear Classification." *Journal of Machine Learning Research*, **9**, 1871–1874.

Furey TS, Cristianini N, Duffy N, Bednarski DW, Schummer M, Haussler D (2000). "Support vector machine classification and validation of cancer tissue samples using microarray expression data." *Bioinformatics*, **16**(10), 906–914.

Groenen PJF, Nalbantov GI, Bioch JC (2007). "Nonlinear Support Vector Machines Through Iterative Majorization and I-Splines." In *Advances in Data Analysis*, Studies in Classification, Data Analysis, and Knowledge Organization, pp. 149–161. Springer-Verlag Berlin Heidelberg.

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

Guyon I, Weston J, Barnhill S, Vapnik V (2002). "Gene Selection for Cancer Classification using Support Vector Machines." *Machine Learning*, **46**, 389–422.

Heiser WJ (1995). *Convergent computation by iterative majorization: Theory and applications in multidimensional data analysis*, pp. 157–189. Oxford University Press, Oxford.

Hunter DR, Lange K (2004). "A tutorial on MM algorithms." *The American Statistician*, **39**, 30–37.

Joachims T (1999). *Making large-scale support vector machine learning practical*, pp. 169–184. MIT Press, Cambridge, MA, USA.

Joachims T (2006). "Training linear SVMs in linear time." In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pp. 217–226. ACM, New York, NY, USA.

Kiers HAL (2002). "Setting up alternating least squares and iterative majorization algorithms for solving various matrix optimization problems." *Computational Statistics and Data Analysis*, **41**, 157–170.

Lange K, Hunter DR, Yang I (2000). "Optimization transfer using surrogate objective functions." *Journal of Computational and Graphical Statistics*, **9**, 1–20.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2014). *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-4, URL https://CRAN.R-project.org/package=e1071.

Murtagh B, Saunders M (1998). "**MINOS** 5.5: User's Guide." *Technical Report SOL 83-20R*, Dept. of Operations Research, Stanford University.

Osuna E, Freund R, Girosi F (1997a). "An Improved Training Algorithm for Support Vector Machines." In *IEEE Workshop on Neural Networks and Signal Processing*. IEEE Press.

Osuna E, Freund R, Girosi F (1997b). "Support vector machines: Training and applications." *Technical Report AIM-1602*, MIT Artificial Intelligence Laboratory.

Platt J (1999). *Fast training of support vector machines using sequential minimal optimization*, chapter 12, pp. 185–208. MIT Press, Cambridge, MA, USA.

Pontil M, Verri A (1998). "Support Vector Machines for 3D Object Recognition." In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 20–6, pp. 637–646. IEEE Computer Society, Los Alamitos, CA, USA.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL https://www.R-project.org.

Ramsay J (1988). "Monotone regression splines in action." *Statistical Science*, **3**(4), 425–461.

Rüping S (2000). *mySVM-Manual*. Software available at https://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/.

Saunders C, Stitson M, Weston J, Léon B, Schölkopf B, Smola A (1998). "Support vector machine reference manual." *Technical Report CSD-TR-98-03*, Department of Computer Science, Royal Holloway, University of London.

Vanderbei RJ (1994). "Interior-point methods: Algorithms and formulations." *Informs Journal on Computing*, **6**(1), 32–34.

Vapnik VN (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc. 2nd edition, 2000.

Weihs C, Ligges U, Luebke K, Raabe N (2005). "klaR Analyzing German Business Cycles." In D Baier, R Decker, L Schmidt-Thieme (eds.), *Data Analysis and Decision Support*, pp. 335–343. Springer-Verlag, Berlin.

Affiliation:

Patrick J.F. Groenen
Econometric Institute
Erasmus University Rotterdam
3062 PA Rotterdam, The Netherlands
E-mail: groenen@ese.eur.nl
URL: https://people.few.eur.nl/groenen

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd