A New Multiclass Support Vector Machine

An Approach Using Iterative Majorization and Huber Hinge Errors

Gertjan van den Burg

Supervisor:
prof. dr. P.J.F. Groenen

Co-reader:
dr. D. Fok

Econometric Institute
Erasmus School of Economics
Erasmus University Rotterdam

ERASMUS UNIVERSITEIT ROTTERDAM

July 26, 2012

Thesis submitted for the degree of Master of Science in Econometrics and Management Science.

ABSTRACT

A new multiclass Support Vector Machine (SVM) is presented, which can be used to find the optimal decision boundaries in a multiclass classification problem. In the multiclass classification problem the goal is to construct a decision function based on a set of objects belonging to K different classes, such that the decision function best predicts the class label of a new object.

The binary Support Vector Machine has been proven very successful for the classification of objects belonging to two distinct classes. The present work extends the binary Support Vector Machine to accommodate problems where objects belong to more than two classes. A single optimization problem is constructed, in which all classes are considered simultaneously.

The suggested method is tested on a number of datasets, and compared with a number of other classification methods. Performance comparison is done using a bootstrap scheme which generates a confidence interval for the difference in performance between two classifiers. The different classifiers are tested on four benchmark datasets and it is shown that the proposed multiclass classification method performs at least as good as existing techniques. On one dataset the proposed method performs significantly better than all other methods.

A comparison in computation time is also given, which shows that the proposed method is slower than existing methods on datasets with a large number of objects or a large number of classes. It is nonetheless believed that the proposed method provides a promising new way of looking at multiclass classification problems.

CONTENTS

NOMENCLATURE

α_i	Parameter for the quadratic term in the majorization for object i
χ	Dummy variable in the Huber hinge majorization
ϵ	Stopping criterion in the IM algorithm
κ	Tuning parameter in the Huber hinge error
λ	Regularization parameter in the loss function
ω_i	Coefficient of the majorization of the p-th root of the L_p norm
ϕ	Dummy variable in the Huber hinge majorization
ψ	Dummy variable in the Huber hinge majorization
ρ_i	Individual object weights
ε_i	Case indicator for object i used in the majorization function
$a_{ijk}^{(1)}$	Quadratic term parameter of the majorization of $h(q_i^{(jk)})$
$a_{ijk}^{(p)}$	Quadratic term parameter of the majorization of $h^p(q_i^{(jk)})$
$b_{ijk}^{(1)}$	Linear term parameter of the majorization of $h(q_i^{(jk)})$
$b_{ijk}^{(p)}$	Linear term parameter of the majorization of $h^p(q_i^{(jk)})$
c	Intercept in the binary SVM formulation
$c_{ijk}^{(1)}$	Constant term of the quadratic majorization of $h(q_i^{(jk)})$
$c_{ijk}^{(p)}$	Constant term of the quadratic majorization of $h^p(q_i^{(jk)})$
d	Degree of spline transformations
f_+	General error function for $+1$ objects in the binary SVM
f_-	General error function for -1 objects in the binary SVM
m	Total number of explanatory variables

n	Total number of objects
n_k	Number of objects of class k
p	Parameter of the L_p norm used to weigh errors
q_i	Errors in the binary SVM formulation
$q_i^{(jk)}$	Error of object i onto class j, with $y_i = k$
s	Number of interior knots in the spline transformation
u_{kl}	Element of matrix $\mathbf{U}$
y_i	Class label for object i
G_+	Objects in the binary SVM with $y_i = +1$
G_-	Objects in the binary SVM with $y_i = -1$
G_k	Group of objects for which $y_i = k$
K	Total number of classes
$\Gamma^{(1)}$	Constant terms after majorization of the p-th root of the L_p norm
$\Gamma^{(2)}$	Constant term after majorization of the Huber hinge errors
$\Gamma^{(3)}$	Constant terms after majorization of the quadratic term
$\boldsymbol{\beta}_i'$	Parameters for the linear term in the quadratic majorization for object i
$\boldsymbol{\delta}_{kj}$	Vector along the side of the simplex between vertex k and vertex j
$\mathbf{s}_j^{(k)}$	Vector of spline weights for variable $\mathbf{x}_j$ and class boundary k
$\mathbf{t}$	Translation vector of size $(K-1) \times 1$
$\mathbf{t}^*$	Optimal value of $\mathbf{t}$ for which the loss function is minimized
$\mathbf{u}_k'$	Row of matrix $\mathbf{U}$, coordinates of simplex vertex
$\mathbf{w}$	Weights in the binary SVM formulation
$\mathbf{x}_i'$	Row of matrix $\mathbf{X}$, attribute vector for object i
$\mathbf{x}_j$	Vector of objects for variable j
$\mathbf{y}$	Vector of n category labels
$\mathbf{z}_i'$	Object vector $\mathbf{x}_i'$ augmented with 1
$\mathbf{A}$	Diagonal matrix of quadratic majorization parameters α_i
$\mathbf{B}$	Matrix of quadratic majorization parameter vectors $\boldsymbol{\beta}_i$
$\mathbf{D}$	Collection of spline basis matrices

$\mathbf{D}_j$	Spline basis matrix for variable j
$\mathbf{H}$	Matrix of Huber errors $h(q_i^{(jk)})$ in the MSVM-Maj algorithm
$\mathbf{I}_m$	$m \times m$ identity matrix
$\mathbf{J}$	Identity matrix where the first element is 0
$\mathbf{K}$	Kernel matrix used to include nonlinearity in the attributes
$\mathbf{Q}$	Matrix of $q_i^{(jk)}$ in the MSVM-Maj algorithm
$\mathbf{R}$	Auxiliary matrix used in the MSVM-Maj algorithm
$\mathbf{S}$	Collection of spline weights matrices
$\mathbf{S}_j$	Matrix of spline weights for variable $\mathbf{x}_j$
$\mathbf{U}$	Matrix of simplex coordinates of size $K \times (K-1)$
$\mathbf{V}$	Combined matrix of parameters
$\mathbf{W}$	Matrix of weights of size $m \times (K-1)$
$\mathbf{W}^*$	Optimal value of $\mathbf{W}$ for which the loss function is minimized
$\mathbf{X}$	Input matrix of n objects with m variables
$\mathbf{Z}$	Matrix with rows $\mathbf{z}_i'$

CHAPTER

ONE

INTRODUCTION

Many problems in statistics and economics can be interpreted as classification problems. In a classification problem the variable of interest is a categorical variable with a finite number of outcomes. Generally, the analysis of such a classification problem focuses on creating a classification function which best describes a given dataset. Alternatively, one may be interested in developing a classification rule which accurately predicts the class label of an unclassified sample, based on a dataset of objects where the class labels are known. In this thesis we will focus on the latter problem.

One example of an multiclass classification problem in finance is the credit rating given to companies. For instance, one may be interested in assigning a credit rating label to a company based on financial data of the company. If the ratings and financial data of a number of companies are available, this data can be used to construct a classification rule to find the optimal class label for a new company. Classification problems also exist in the field of marketing research. For instance, customers of a particular store could be classified into different buyer groups based on their purchasing preferences.

Traditional methods which can be used to create such a classification function include, among others, logistic regression and discriminant analysis. A more recent technique which has proven very successful when the outcome is a binary variable, is the Support Vector Machine (Cortes and Vapnik, 1995). The Support Vector Machine (SVM) aims to maximize the minimal separation between the objects in the dataset, based on the explanatory variables (also known as *attributes* in the SVM literature). This technique has been proven very powerful and generalizes well to datasets with a large number of objects and a large number of attributes.

Because of the success of the binary SVM, there has been a large interest in extending the SVM technique to problems where the dependent variable can belong to K different classes, with $K > 2$. However such an extension is not straightforward. Extensions proposed in the literature are often difficult to implement, or require a large number of binary SVM problems to be solved. Moreover, optimization methods for the SVM often depend on a dual formulation of the SVM problem. This dual formulation can appear obscure for practitioners, because of the mathematics involved.

To avoid the dual formulation, Groenen et al. (2008) proposed an iterative method to solve the binary SVM optimization problem. This formulation is based on minimizing a loss function constructed of misclassification errors, and uses an iterative majorization algorithm which allows for arbitrarily close approximations of the SVM solution.

In this thesis a new multiclass classification method will be introduced, as an extension of the work done by Groenen et al. (2008). The proposed method provides a new approach to the multiclass classification problem, by using a new method for calculating the errors in the construction of the loss function. Furthermore, the different weighting functions applied to the errors create a very flexible method, which is believed to be an advantage.

Similar to the approach by Groenen et al. (2008), an iterative algorithm is used to find the optimal decision boundary, without switching to a dual formulation. We believe that the iterative method is easier to understand than the dual space approach. Moreover, because of its simplicity the implementation of the algorithm on a computer is very straightforward.

A very important property of the SVM is that it allows for nonlinearity in the predictor variables. However in this thesis we will focus on the linear method and its properties. Extensions to include nonlinearity will be briefly mentioned in the Discussion.

1.1 Outline

This thesis is structured as follows. First, a short review of the binary support vector machine approach proposed by Groenen et al. (2008) is given in Chapter 2. In Chapter 3 the existing techniques for solving the multiclass SVM problem are examined, and a short motivation for the development of a new method is offered. The details of the proposed approach are given in Chapter 4. An optimization method using the Iterative Majorization approach of De Leeuw (1977) is provided in Chapter 5.

The algorithm will then be tested on a number of benchmark datasets, and the performance will be compared to known methods. Results of these tests are given in Chapter 6. Finally, Chapter 7 discusses the obtained results and concludes. Recommendations for further research and a nomenclature are also included.

CHAPTER

TWO

BINARY SUPPORT VECTOR MACHINES

This chapter contains a review of the theory of binary Support Vector Machines. The material presented here is based on Groenen et al. (2008). Note that the treatment below will be confined to *linear* SVMs, since in the treatment of the multiclass method we also focus on the linear method. Nonlinearity in the binary SVM approach presented here can be found in Groenen et al. (2007).

The SVM classifier is trained on a part of the original dataset known as the *training* dataset. The obtained classification function is then evaluated based on how well it can classify objects in the *test* dataset. The test dataset can be a part of the original dataset that is not in the training dataset or it can consist of previously unknown data.

In the binary SVM, an optimal separating hyperplane[1] is constructed between datapoints of two classes. For each of the n objects in the training dataset a class label y_i is available, as well as a vector of m explanatory variables $\mathbf{x}_i'$. The class labels are coded such that y_i equals -1 if object i belongs to class -1, and $y_i = 1$ if it belongs to class 1. The predicted value q_i of object i is defined as

$$q_i = c + \mathbf{x}_i'\mathbf{w},$$

where $\mathbf{w}$ is a $m \times 1$ vector of weights and c is an intercept. The so-called *margins* of the SVM solution are the hyperplanes defined by $q_i = 1$ and $q_i = -1$. The SVM aims to maximize the distance between the margins of the two classes. In Figure 2.1 a graphical illustration is shown of a perfectly separable classification problem in two dimensions. The solid line shows the optimal decision boundary and the dashed lines show the margins of the SVM solution. The SVM aims to maximize the distance between the margin lines, thereby creating an as large as possible "gap" between the objects of two classes. This property ensures that there is a maximal distinction between objects of two classes.

One of the nice properties of the SVM classifier is that in the perfectly separable case only the objects that lie on the margins contribute to the SVM solution. These objects are called the *support vectors*. In Figure 2.1 these objects are highlighted. If the data is not perfectly separable, objects that lie on the wrong side of the decision boundary also contribute to the solution. Before the classification problem is fully solved it is not known which objects lie on the margins or give errors.

[1] "The optimal hyperplane is the hyperplane that separates the training data with the maximal margin: it determines the direction [...] where the distance between the projections of the training vectors of two different classes is maximal". Quoted from Cortes and Vapnik (1995).

The Support Vector Machine is based on minimizing a loss function formulated by adding the total classification error of all objects and a penalty term to control for the size of $\mathbf{w}$. An object contributes an error when it is projected on the wrong side of the margin corresponding to its class. This means that for an object of class 1, a nonzero error is found when $q_i < 1$ and for an object of class -1 a nonzero error occurs when $q_i > -1$. When the SVM is considered with absolute hinge errors, the error for a wrongly projected object from class 1 equals $1 - q_i$ and the error for a wrongly projected object from class -1 equals $1 + q_i$.

To avoid overfitting of the data, a penalty term for the size of the weight vector $\mathbf{w}$ is added to the loss function. To formalize the loss function let G_+ denote objects from class $+1$, and G_- denote objects from class -1, such that $G_\pm = \{i : y_i = \pm 1\}$. Then the SVM loss function with the absolute hinge error can be expressed as

$$L_{\text{SVM}}(\mathbf{w}, c) = \frac{1}{n} \sum_{i \in G_+} \max(1 - q_i, 0) + \frac{1}{n} \sum_{i \in G_-} \max(1 + q_i, 0) + \lambda \, ||\mathbf{w}||^2, \tag{2.1}$$

where $|| \cdot ||$ denotes the Euclidean norm, and λ a regularization parameter in the penalty term. In the SVM literature this loss function is commonly minimized by switching to a dual formulation (Cortes and Vapnik, 1995) and using a quadratic program solver[2]. Groenen et al. (2008) showed that the above loss function can be minimized in the primal formulation using the Iterative Majorization (IM) algorithm. A review of the IM algorithm is introduced in Chapter 5, where it is applied to the Multicategory SVM loss function.

The loss function above can also be formulated in a more general setting by allowing different specifications for the error functions. If these error functions are denoted by f_+ and f_- for the objects from class $+1$ and -1 respectively, the loss function becomes

$$L_{\text{SVM}}(\mathbf{w}, c) = \frac{1}{n} \sum_{i \in G_+} f_+(q_i) + \frac{1}{n} \sum_{i \in G_-} f_-(q_i) + \lambda \, ||\mathbf{w}||^2. \tag{2.2}$$

In Section 4.2 the choice of different error functions will be discussed more thoroughly.

[2]Other optimization methods can also be used. One of the most commonly used approaches is the Sequential Minimal Optimization method of Platt (1999).

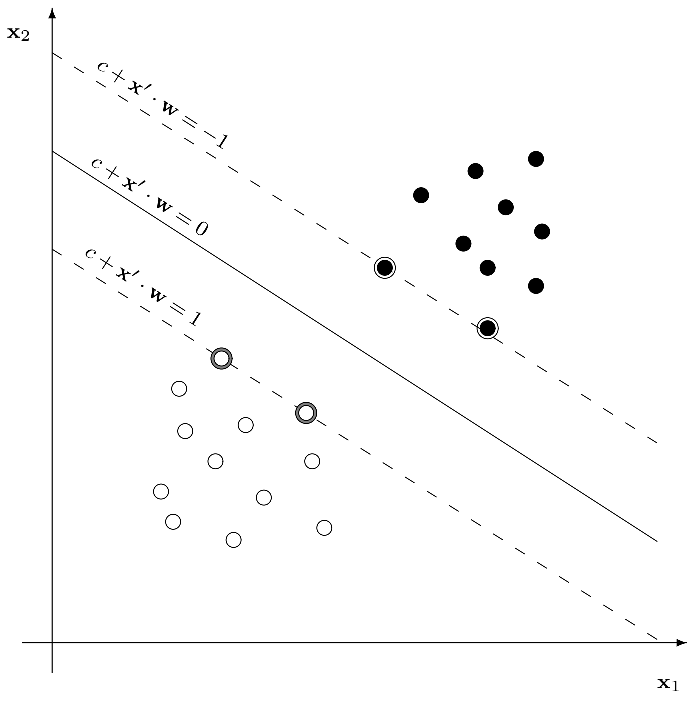

$\mathbf{x}_2$
$c+\boldsymbol{x}'\cdot\mathbf{w}=-1$
$c+\boldsymbol{x}'\cdot\mathbf{w}=0$
$c+\boldsymbol{x}'\cdot\mathbf{w}=1$
$\mathbf{x}_1$

Figure 2.1 – Graphical illustration of the SVM solution of a perfectly separable classification problem with 2 predictor variables. The solid line shows the maximum separating hyperplane. The dotted lines are the margins of the SVM solution. The objects that lie on these margin lines are the support vectors. Objects represented by white circles belong to class +1, and objects represented by black circles belong to class −1. Note that the support vectors have also been highlighted.

CHAPTER

THREE

LITERATURE OVERVIEW

The existing SVM techniques for solving the multiclass classification problem will be reviewed here. The text below is largely based on the excellent review of existing techniques by Rifkin and Klautau (2004). Contrasting the argument presented in that paper, this chapter will close with arguments in favour of the single machine approach proposed in the next chapter.

As noted by Rifkin and Klatau, four distinct SVM approaches to the multiclass classification problem can be identified. First, an optimization problem can be constructed which is based on a single loss function, this approach is known as the *single machine approach*. Second, the *error-correcting coding approaches* solve multiple binary classification problems and combine the solutions of these problems in a specific way to achieve optimal classification. Finally, two approaches exist that use binary classifiers to solve the multiclass classification problem. These methods will be referred to as *naive approaches* and are known as the *all-vs-all* (AVA) approach and the *one-vs-all* (OVA) approach.

Below the existing methods for each approach will be briefly reviewed. The mathematics necessary to solve the optimization problems will be omitted from this review. As noted this review is largely based on the review by Rifkin and Klautau (2004), and an extensive analysis of all different methods and a comparison of the performance of each method can be found therein.

3.1 Naive Approaches

Two main approaches exist to use binary SVM classifiers to solve the multiclass problem. These methods will be discussed here.

3.1.1 All vs. All classification

The AVA classification scheme consists of training a binary classifier between each pair of classes. This means that a total of $K(K-1)/2$ binary SVM classifiers need to be trained. Each of these classifiers yields a classification function $f : \mathbb{R}^m \to \mathbb{R}$, which is in general of the form (Cortes and Vapnik, 1995)

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}, \mathbf{x}_i) + \beta, \tag{3.1}$$

where α_i and β are constants, and $K(\mathbf{x}_i, \mathbf{x}_j)$ is a kernel function[1]. In the binary problem, the sign of $f(\mathbf{x})$ determines the predicted class label of an unclassified object $\mathbf{x}$. Formally this can be expressed as

$$\hat{y}_{\text{SVM}} = \text{sign}(f(\mathbf{x})).$$

Note that the sign function is defined such that $\text{sign}(z) = +1$ if $z > 0$ and $\text{sign}(z) = -1$ if $z < 0$. Moreover, sign(0) can be considered unclassifiable. Now in the AVA approach we let the binary classification function between class i and class j be given by f_{ij}, with $f_{ji} = -f_{ij}$. Here we assume that in the binary SVM classification the objects of class i represent positive samples and objects of class j represent negative samples. If this is done for all $K(K-1)/2$ combinations, the predicted class label in the AVA approach is given by

$$\hat{y}_{\text{AVA}} = \arg\max_i \left(\sum_{j=1}^{K} f_{ij}(\mathbf{x}) \right).$$

3.1.2 One vs. All classification

In the OVA method, a decision boundary is constructed between each class and the collection of all other classes. Thus K binary classifiers are constructed where a single class i is coded as $+1$, and all other classes $j \neq i$ are coded as -1. If the classification function obtained through this binary classification is denoted as f_i, the predicted class of an unclassified object $\mathbf{x}$ can be found as

$$\hat{y}_{\text{OVA}} = \arg\max_i f_i(\mathbf{x}).$$

Rifkin and Klautau (2004) argue that this OVA classification scheme is preferred over other multiclass classification methods, since it is superior in both training time and performance, provided the binary classifiers are well trained.

3.2 Single machine Approaches

As was mentioned in the introduction of this chapter, a single machine approach tries to solve the multiclass classification problem using a single loss function. Several single machine approaches have been proposed, most notably by Vapnik (1998), Weston and Watkins (1998), Bredensteiner and Bennett (1999), Lee et al. (2004) and Crammer and Singer (2002a). As noted by Rifkin and Klatau the methods proposed by Vapnik (1998) and Weston and Watkins (1998) are very similar. In the notation used in this text all these methods aim to find K classification functions f_j, given by

$$f_j(\mathbf{x}) = \sum_{i=1}^{n} \alpha_{ij} K(\mathbf{x}, \mathbf{x}_i) + \beta_j,$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is again a kernel function evaluation, and α_{ij} and β_j are constants to be determined. More generally, this classification function can be denoted as $f_j(\mathbf{x}) = h_j(\mathbf{x}) + \beta_j$ to separate the dependence on $\mathbf{x}$ and the constant term.

[1]In short, a kernel function is a mapping from two input vectors to a real number, which satisfies a number of mathematical constraints. It should be noted that the kernel function and the number of categories are both denoted by the letter K. To avoid confusion, the kernel function will always be written with its arguments.

Of the single machine approaches treated by Rifkin and Klatau, we will only focus on the approach by Lee et al. (2004), since this approach is similar to the one presented in the next chapter. Note that the other single machine approaches are similar. Lee et al. (2004) suggest that the following loss function needs to be minimized

$$L_{\text{LEE}} = \frac{1}{n} \sum_{i=1}^{n} \sum_{\substack{j=1, \\ j \neq y_i}}^{K} \max\left(f_j(\mathbf{x}_i) + \tfrac{1}{K-1}, 0 \right) + \lambda \sum_{j=1}^{K} ||h_j||_{\mathcal{H}}^2 \tag{3.2}$$

$$\text{subject to:} \quad \sum_{j=1}^{K} f_j(\mathbf{x}) = 0.$$

In this expression, n is the number of objects in the training dataset, and λ is a regularization parameter. Furthermore, since the functions f_j are in general defined as functions over a Reproducing Kernel Hilbert Space[2] (RKHS), the square norm in the regularization term is the norm of the function in the RKHS, denoted by $|| \cdot ||_{\mathcal{H}}^2$. Note that the norm is only taken over the h_j, the part of the classification function dependent on $\mathbf{x}$.

Note that in contrast to the binary SVM, the error function is no longer formulated as $\max(z+1, 0)$ but is instead scaled by $1/(K-1)$. As Lee et al. report, this is done to ensure that the minimum of (3.2) yields the optimal decision boundary asymptotically as $n \to \infty$. However, as Rifkin and Klautau (2004) note, the fact that the optimal decision boundary is approached asymptotically, is no indication that this will also be approached with a finite number of datapoints.

Also of interest is the fact that a constraint is added to ensure that the classification functions add up to zero. Moreover, it should be noted that in the summation of the error functions over j, the error where $j = y_i$ is ignored. This has been done to ensure that a correctly classified sample does not contribute an error to the loss function.

The classification rule for an object $\mathbf{x}$ from the test dataset, based on the functions f_j when the loss function is minimized, is given by

$$\hat{y}_{\text{LEE}} = \arg\max_j f_j(\mathbf{x}).$$

As Lee et al. show, their multiclass SVM approach reduces to the standard binary SVM formulation when $K = 2$, with the constraint $f_{-1} = -f_{+1}$.

3.3 Error-correcting Coding Approaches

Error-correcting coding approaches have been put forth by Dietterich and Bakiri (1995), Allwein et al. (2001), and Crammer and Singer (2001, 2002b). As noted by Rifkin and Klautau (2004), the method proposed by Allwein et al. (2001) is an extension of the work done by Dietterich and Bakiri (1995). The method of Allwein et al. will be treated here.

In the Error-correcting Coding (ECC) approach of Allwein et al., a *coding matrix* $\mathbf{M} \in \{-1, 0, 1\}^{K \times \ell}$ is constructed. Here, K is again the number of classes, and ℓ is to be chosen.

[2]For a short review of Reproducing Kernel Hilbert Spaces, see Daumé (2004). It should be noted that an understanding of RKHS is not necessary to understand the current review of the approach by Lee et al. (2004). The regularization term is added to control for the "size" of the parameters in the functions h_j. If the linear kernel function is chosen such that $h_j(\mathbf{x}) = \mathbf{x}'\mathbf{w}_j$, then $||h_j||_{\mathcal{H}}^2 = ||\mathbf{w}_j||^2$. That is, the norm of the regularization parameter reduces to the Euclidean norm.

For each column s of the matrix $\mathbf{M}$, a binary classifier is used to construct a decision function[3] $f_s : \mathcal{X} \to \mathbb{R}$. Allwein et al. do not explicitly specify the classification functions f_s, since the ECC approach can be used with any binary classifier. Here we will restrict our review to the case where the binary classifier is the SVM. The functions f_s will then be of the form (3.1).

The value of an element of $\mathbf{M}$ in a particular row of column s determines the class label in the binary classification problem for that column. For instance, if $K = 3$ and column s of matrix $\mathbf{M}$ equals $[-1, 1, -1]'$, objects in the training dataset for which $y_i \in \{1, 3\}$ constitute the -1 class in the constructed binary classification problem, and objects for which $y_i = 2$ create the $+1$ class. If an element of $\mathbf{M}$ is zero, this class will not be used in the construction of the classifier f_s. It should also be noted that every column of $\mathbf{M}$ should contain at least one $+1$ and one -1 element.

Allwein et al. propose two methods to combine the obtained binary classifiers to predict an unclassified object from the test dataset. First, let $\mathbf{m}'_r$ denote row r of $\mathbf{M}$, and let $\mathbf{f}(\mathbf{x})$ be the vector of predictions of $\mathbf{x}$ for each column of $\mathbf{M}$, such that

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_\ell(\mathbf{x})).$$

Then the class of an unclassified object $\mathbf{x}$ is found by searching for the row $\mathbf{m}'_r$ for which the distance to $\mathbf{f}(\mathbf{x})$ is minimized. Two distance methods are proposed, the first is the *Hamming distance* $d_H(\mathbf{m}'_r, \mathbf{f}(\mathbf{x}))$, defined as

$$d_H(\mathbf{m}'_r, \mathbf{f}(\mathbf{x})) = \sum_{s=1}^{\ell} \left(\frac{1 - \text{sign}(m_{rs} f_s(\mathbf{x}))}{2} \right).$$

In this expression, m_{rs} is the element in row r and column s of the matrix $\mathbf{M}$. Furthermore, the convention is used that $\text{sign}(z) = 0$ if $z = 0$. Formally the predicted class label is then

$$\hat{y} = \arg\min_r d_H(\mathbf{m}'_r, \mathbf{f}(\mathbf{x})).$$

As Allwein et al. note this method has the disadvantage that because the sign function is used, the magnitude of the prediction $f_s(\mathbf{x})$ is not taken into account. This magnitude may however be important as it often is a measure of the *confidence* of a prediction. Therefore, they also propose a *loss-based* distance measure based on the loss function used in the binary classifier. If the binary classifier used is the SVM, the loss function L_{SVM} is defined as[4]

$$L_{SVM}(z) = \max(1 - z, 0).$$

The loss-based distance measure can then be expressed as

$$d_L(\mathbf{m}'_r, \mathbf{f}(\mathbf{x})) = \sum_{s=1}^{\ell} L_{SVM}(m_{rs} f_s(\mathbf{x})),$$

and the predicted class label follows from

$$\hat{y} = \arg\min_r d_L(\mathbf{m}'_r, \mathbf{f}(\mathbf{x})).$$

[3]Allwein et al. use a general space $\mathcal{X}$ for the input vectors $\mathbf{x}$. In our case, the input variables are limited to $\mathcal{X} = \mathbb{R}^m$.

[4]Note that this loss function differs from the binary SVM loss function presented in the previous chapter. Here only the weighting function of the errors is considered, the regularization term is ignored. If we let $z = y_i q_i$ the loss function is similar to the one presented in Chapter 2, with absolute hinge errors.

Finally, there are a number of ways in which the code matrix $\mathbf{M}$ can be constructed. The different coding matrices are described by the minimum distance ρ, measured between pairs of distinct rows. Formally, ρ can be defined as

$$\rho = \min \left\{ \frac{\ell - \mathbf{m}_{r_1}' \mathbf{m}_{r_2}}{2} : r_1 \neq r_2 \right\}.$$

Allwein et al. consider five different coding approaches. First, OVA and AVA methods can be implemented using the coding matrix. Second, a *complete* code can be constructed in which all non-trivial column configurations are used. Finally two types of random codes are considered, known as the *dense* and the *sparse* codes. In the dense code, the number of columns is chosen as $\lceil 10 \log_2(K) \rceil$, and each element is chosen as either -1 or $+1$ with equal probability. The code matrix used is chosen as the matrix with the highest value of ρ out of 10,000 random matrices. The sparse code has $\lceil 15 \log_2(K) \rceil$ columns and each entry is 0 with probability $\frac{1}{2}$, and -1 or $+1$ with probability $\frac{1}{4}$ each. Here the chosen code matrix is again the one with the highest ρ. A constraint on these matrices is that no identical columns or rows consisting of all zeros are allowed.

It is clear that the complete code matrix can become very large when the number of categories K becomes large. If this is the case, the ECC approach may become infeasible. According to Rifkin and Klatau the ECC approach did perform slightly better on some datasets than the OVA approach, but these results were not fully reproducible using *well-trained* binary classifiers in an OVA scheme. Quoting Rifkin and Klatau: *"... it appears that choosing the kernel parameters to maximize the strength of the individual binary classifiers, and then using a one-vs-all multiclass scheme, performs as well as the other coding schemes, and, in the case of the satimage data*[5]*, noticeably better than any of the coding schemes when the underlying classifiers are weak."* In other words, the ECC approach may perform better if the parameters of the underlying binary classifiers are not properly tuned. Otherwise the OVA approach is expected to perform better.

As a final note on ECC, Crammer and Singer develop a method with a continuous code matrix $\mathbf{M}$, which also needs to be optimized in the method. Rifkin and Klautau (2004) note that their method did not outperform a simple OVA scheme.

3.4 Single Machine vs. OVA

Above the alternative SVM methods for multiclass classification have been presented. The OVA approach has a number of properties that make it an attractive method for tackling the multiclass classification problem. The method is simple to understand, has a good performance (as shown by Rifkin et al.), and since only K decision boundaries need to be constructed it is also relatively fast. Therefore, from a practical viewpoint any alternative that is proposed should provide an advantage in either of these domains.

In the next chapter, a new single machine approach will be introduced. The main motivation for this method is to enable a more intuitive understanding of multiclass SVM classification. Furthermore, the method proposed may be superior in regions where there is overlap between objects from different classes, because of the way in which misclassification errors are counted. Moreover, using a binary classification scheme such as OVA may seem unfulfilling from a practitioners view. It may be presumed that a method which is

[5]The satimage dataset is one of the benchmark datasets used often in the Machine Learning literature. It is freely available from the UCI repository (Frank and Asuncion, 2010).

explicitly designed to handle the multiclass problem may be a more suitable solution, than a combination of binary classifiers.

In contrast to the single machine approach proposed by Lee et al. (2004), the method proposed will not use the sum-to-zero constraint, but instead will use projection onto a lower dimensional space. Furthermore, the SVM margins will not be scaled by the number of categories. Instead, this scaling is partially replaced by the aforementioned projection. Finally, the proposed method will move away from the absolute error $\max(1 - z, 0)$, and replace this with a more flexible weighting function. Naturally, we also ensure that for $K = 2$ the proposed method reduces to the binary SVM. We now turn to the details of the proposed multiclass SVM approach.

CHAPTER

FOUR

THE MSVM-MAJ LOSS FUNCTION

In this chapter the Multicategory Support Vector Machine (MSVM) loss function will be presented. First the notation used in the formulation of this loss function will be introduced. Second, a short recap is given of the mathematics of simplexes, which will be used to count the misclassification errors. Then, a review is given of the common error functions used in SVM applications. Finally, the MSVM loss function will be introduced and its properties will be examined.

4.1 Notation & Simplex Theory

Let $\mathbf{X}$ denote the $n \times m$ matrix of n objects with m explanatory variables (attributes). For each object a class label y_i, is reported in the vector $\mathbf{y}$ where $y_i \in \{1, \ldots, K\}$, corresponding to K distinct classes. Based on these class labels, we introduce the groups $G_k = \{i : y_i = k\}$, where each G_k holds the indices of all objects belonging to category k.

Recall that a regular N-simplex is an object in N dimensional space where each of the $N + 1$ vertices are connected with equal distance and the faces connecting the vertices are flat. For example the 2-simplex is an equilateral triangle and the 3-simplex is a tetrahedron. Here a special regular N-simplex is considered which is centered at the origin but where the distance between each pair of vertices is 1[1].

Let $\mathbf{U}$ denote the $K \times (K-1)$ coordinate matrix of a regular $(K-1)$-simplex where the distance between each pair of vertices is 1. Each row of $\mathbf{U}$ denotes the coordinate vector of a single vertex, and is denoted by $\mathbf{u}_k'$. The general expression for an element u_{kl} in the k-th row and l-th column of $\mathbf{U}$ is given by

$$u_{kl} = \begin{cases} -\dfrac{1}{\sqrt{2(l^2+l)}} & \text{if } k \leq l \\ \dfrac{l}{\sqrt{2(l^2+l)}} & \text{if } k = l+1 \\ 0 & \text{if } k > l+1. \end{cases} \tag{4.1}$$

Further on, the eigenvalues of the symmetric $(K-1) \times (K-1)$ matrix $(\mathbf{u}_k - \mathbf{u}_j)(\mathbf{u}_k - \mathbf{u}_j)'$ will be needed. Since the distance between the vertices of the $(K-1)$-simplex equals 1 by de-

[1]Note that this differs from the *unit* regular N-simplex (also known as the *standard* N-simplex) in that in the unit regular simplex the distance from each vertex *to the origin* is fixed at 1. In the regular N-simplex considered here this distance is equal to $1/(N+1)$

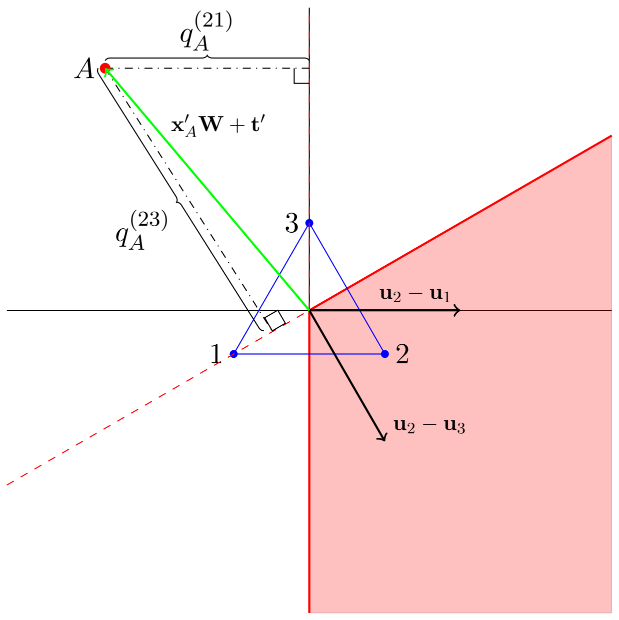

$q_A^{(21)}$
A
$\mathbf{x}'_A\mathbf{W}+\mathbf{t}'$
$q_A^{(23)}$
3
$\mathbf{u}_2-\mathbf{u}_1$
1
2
$\mathbf{u}_2-\mathbf{u}_3$

Figure 4.1 – Graphical illustration of the errors $q_i^{(jy_A)}$ for the object A with $y_A = 2$ and $K = 3$. The figure shows the situation in the transformed space, which is guaranteed to be two-dimensional when $K = 3$. The error $q_A^{(21)}$ is calculated by projecting $\mathbf{x}'_A\mathbf{W}+\mathbf{t}'$ on $\mathbf{u}_2 - \mathbf{u}_1$, and the error $q_A^{(23)}$ is found by projecting $\mathbf{x}'_A\mathbf{W} + \mathbf{t}'$ on $\mathbf{u}_2 - \mathbf{u}_3$. Refer to the main text for further explanation of this figure.

sign, it is clear that $||\mathbf{u}_k - \mathbf{u}_j|| = 1$. Thus this matrix is defined as its own eigendecomposition with a single non-zero eigenvalue equal to 1.

4.2 Errors & Error Functions

Here definitions of the errors and the error functions used in the formulation of the loss function are given.

4.2.1 Errors

For each object in the training dataset an m vector of explanatory variables, $\mathbf{x}'_i$, and a class label, y_i, is reported. The $K - 1$ errors with respect to all other classes $j \neq y_i$ are computed by multiplying $\mathbf{x}'_i$ by a weighting matrix $\mathbf{W}$ of size $m \times (K-1)$. This weighting matrix is independent of the class of object i. The transformed object vector $\mathbf{x}'_i\mathbf{W}$ is then shifted by a $(K-1)$-vector $\mathbf{t}'$, and projected on the edge of a K-simplex corresponding to the class j. Thus for each class j a scalar variable, $q_i^{(jy_i)}$, is computed, defined as

$$q_i^{(jy_i)} = (\mathbf{x}'_i\mathbf{W} + \mathbf{t}')(\mathbf{u}_{y_i} - \mathbf{u}_j). \tag{4.2}$$

Since the simplex is constructed such that the distance between each pair of vertices is 1, projecting the error vector on this simplex gives the distance to the decision boundary.

To illustrate this, we consider Figure 4.1, where the errors of a single object are calculated, and where $K = 3$. The figure shows an object A for which $y_A = 2$ in the transformed space $\mathbf{x}'\mathbf{W} + \mathbf{t}'$. Note that this space is $(K-1)$-dimensional. The 2-simplex is shown in the figure, where the distance between the vertices is still chosen as 1. The chosen class labels are arbitrary. The decision boundaries in this space are the solid lines perpendicular to the edges of the simplex. The decision boundary between class 1 and class 3 has been omitted for legibility. The region of the class 2 objects is simply the region enclosed by the linear class boundaries between vertices 2 and 1 and vertices 2 and 3, and this region has been shaded in the figure.

Since object A does not lie in the shaded region, the errors with respect to the other classes are negative[2]. Following (4.2), these errors are calculated by projecting $\mathbf{x}'_A\mathbf{W} + \mathbf{t}'$ onto the simplex edges. Because the length of the edges is 1, this scalar projection is exactly the magnitude of the distance to the class boundaries. Thus, $(\mathbf{x}'_A\mathbf{W} + \mathbf{t}')(\mathbf{u}_2 - \mathbf{u}_1)$ gives the distance to the boundary between class 1 and class 2, and $(\mathbf{x}'_A\mathbf{W} + \mathbf{t}')(\mathbf{u}_2 - \mathbf{u}_3)$ gives the distance to the boundary between class 3 and class 2.

As shown in the figure, the errors defined in (4.2) measure the distance of the transformed object vector to the class boundaries specified by j and y_i. The margins of the SVM solution can also be defined here, by simply constructing the lines given by $q_i^{(jy_i)} = \pm 1$. These margins have been omitted from Figure 4.1 for legibility. Note further that $q_i^{(jk)} = -q_i^{(kj)}$.

4.2.2 Error Functions

As seen in Chapter 2, different error functions can be chosen to weigh the errors. The most common error function in the SVM literature is the absolute hinge error, defined as[3]

$$f(q) = \max(1 - q, 0),$$

shown in Figure 4.2a. An advantage of this error function is that all the errors are counted by their distance away from the corresponding SVM margin. The disadvantage of this error function is that it is discontinuous at $q = 1$. When minimizing the MSVM loss function, this discontinuity can cause a problem for the minimization method. Another often used error function is the quadratic hinge error, sketched in Figure 4.2b, which is simply the square of the absolute hinge error

$$f(q) = \max(1 - q, 0)^2.$$

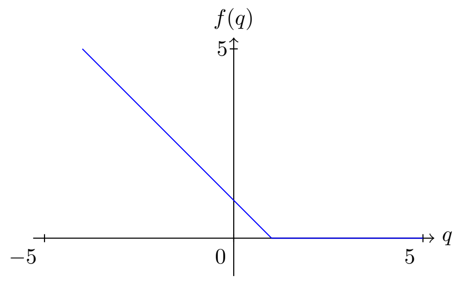
Although this error function is smooth at $q = 1$, errors far away from the SVM boundary are counted very heavy, which may create a large dependency on misclassified objects that lie far from the corresponding class boundary ("outliers"). Finally, the Huber hinge error is defined as

$$h(q) = \begin{cases} 1 - q - \dfrac{\kappa + 1}{2} & \text{if } q \leq -\kappa \\ \dfrac{1}{2(\kappa + 1)}(1 - q)^2 & \text{if } q \in (-\kappa, 1] \\ 0 & \text{if } q > 1 \end{cases} \tag{4.3}$$

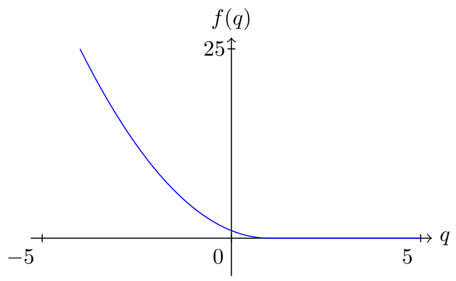
where $\kappa > -1$ (Groenen et al., 2008). This error function is linear on the interval $q \leq -\kappa$ which creates a robustness against "outliers", and quadratic on the interval $(-\kappa, 1]$, such

[2]If the errors are positive with respect to all other classes the object lies in the region corresponding to its class label.

[3]Note that the indices around q are omitted for conciseness.



(a) $f(q) = \max(1 - q, 0)$



(b) $f(x) = \max(1 - q, 0)^2$

Figure 4.2 – Sketch of the absolute hinge and the quadratic hinge error functions.

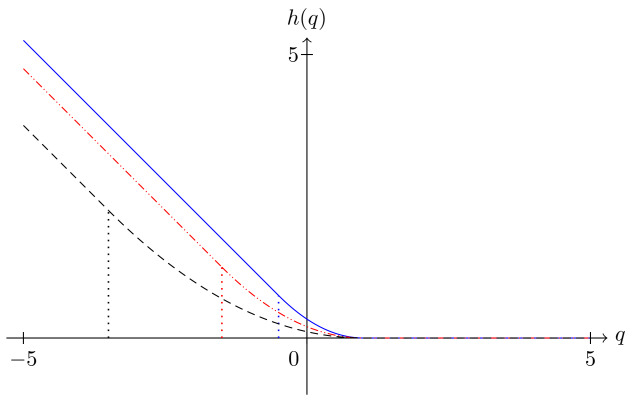


Figure 4.3 – Examples of the possible shapes of the Huber hinge error, for varying values of κ. The values of κ shown are $\kappa = 0.5$ (solid), $\kappa = 1.5$ (dash-dot), $\kappa = 3.5$ (dash). The vertical lines show the transition from the linear part of the function to the quadratic part.

that there is no discontinuity at $q = 1$. Examples of some possible shapes of this function are given in Figure 4.3.

It is important to notice that in the limit $\kappa \to -1$ the Huber hinge error approaches the absolute hinge error. Furthermore, by choosing a large value for κ, the Huber hinge error can mimic the quadratic hinge, albeit with a smaller overall gradient. Thus the Huber hinge error allows for a broader range of error weighting behaviour than the absolute or quadratic hinge errors. Because of these properties, the Huber hinge error is chosen in the formulation of the MSVM loss function, which we now turn to.

4.3 Loss Function

The parameters $\mathbf{W}$ and $\mathbf{t}$ need to be found in order to predict the class label of new samples. This is done by minimizing the total error of all objects in the training dataset. For each object, the errors $q_i^{(jy_i)}$ are calculated with respect to all classes $j \neq y_i$. The errors are then weighted using the Huber hinge function, formulated in the previous section. For each object, these weighted errors are summed using the L_p norm, with $p \in [1, 2]$. This gives control on how the errors of the different categories are combined for each object. These summed category errors are then multiplied by nonnegative individual weights ρ_i, and summed over all objects. This gives the loss function

$$L_{\text{MSVM}}(\mathbf{W},\mathbf{t}) = \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\left(\sum_{j\neq k} h^p\left(q_i^{(jk)}\right)\right)^{1/p} + \lambda\,\text{tr }\mathbf{W}'\mathbf{W} \tag{4.4}$$

where a penalty term $\lambda\,\text{tr }\mathbf{W}'\mathbf{W}$, with $\lambda > 0$, is added to correct for overfitting. In this formulation it should be understood that the summation over $j \neq k$ sums over the elements $j \in \{1,\dots,K\}\backslash\{k\}$. Notice that for $K = 2$, the above loss function reduces to the binary SVM loss function (with Huber hinge errors) specified in (2.2).

The individual weights ρ_i can for instance be used to specify greater importance of specific variable groups, or to correct for unequal group sizes. In the latter case, the weights ρ_i can be chosen as

$$\rho_i = \frac{n}{Kn_k} \qquad \text{if } i \in G_k, \tag{4.5}$$

where n_k is the number of objects in G_k. To predict the class label for an object in the test dataset the following procedure can be used

1. Minimize $L_{\text{MSVM}}(\mathbf{W},\mathbf{t})$ with respect to $\mathbf{W}$ and $\mathbf{t}$ and let

   $$(\mathbf{W}^*,\mathbf{t}^*) = \underset{(\mathbf{W},\mathbf{t})}{\arg\min}\, L_{\text{MSVM}}(\mathbf{W},\mathbf{t}).$$

2. Calculate the distance of the predicted error vector for the unclassified object $\mathbf{x}$ to each of the vertices of the simplex. Then the predicted class for the object is that class for which this distance is minimized. Formally, this can be written as

   $$\hat{y} = \underset{k}{\arg\min}\, ||(\mathbf{x}'\mathbf{W}^* + \mathbf{t}^{*\prime}) - \mathbf{u}_k'||.$$

The reason this last step gives the class prediction comes from the fact that the class boundaries in the *transformed* space (i.e., the space $\mathbf{x}'\mathbf{W}+\mathbf{t}'$) are fixed. The class boundaries in this space therefore always define the same region around a simplex vertex. Hence calculating the distance to the closest vertex is indicative of the region an object lies in, and therefore the predicted class.

An important property for many optimization techniques is that the loss function is convex. In the next section it will be shown that the MSVM loss function defined in (4.4) has this property.

4.4 Convexity of MSVM Loss Function

Before we turn to the proof of the convexity of the proposed loss function, we first introduce some variables to shorten the notation:

$$\mathbf{z}_i' = [1\ \mathbf{x}_i'], \tag{4.6}$$
$$\mathbf{V} = [\mathbf{t}\ \mathbf{W}']', \tag{4.7}$$
$$\boldsymbol{\delta}_{kj} = \mathbf{u}_k - \mathbf{u}_j, \tag{4.8}$$

such that $q_i^{(jk)} = \mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}$. With this notation, $L_{\text{MSVM}}(\mathbf{W},\mathbf{t}) = L_{\text{MSVM}}(\mathbf{V})$ and the convexity proof reduces to showing that $L_{\text{MSVM}}(\mathbf{V})$ is convex in $\mathbf{V}$. Furthermore, we introduce a diagonal matrix $\mathbf{J}$ with $j_{11} = 0$ and $j_{ii} = 1$ if $i > 1$. With this matrix the penalty term tr $\mathbf{W}'\mathbf{W}$ can be written as tr $\mathbf{V}'\mathbf{J}\mathbf{V}$.

Formally, the following theorem must be proven.

Theorem 1. *The MSVM loss function specified in (4.4) is convex in* $\mathbf{V}$*.*

Proof. Since all functions that constitute $L_{\text{MSVM}}(\mathbf{V})$ are bounded from below, the definition of convexity specified in Rockafellar (1997) Theorem 4.1 can be used here[4]. Since $q_i^{(jk)}$ is a linear function in $\mathbf{V}$, it is convex and concave in $\mathbf{V}$.[5] Thus it suffices to show that $L_{\text{MSVM}}(\mathbf{V})$ is convex in $q_i^{(jk)}$ to show that it is convex in $\mathbf{V}$.

The Huber hinge error is a piecewise defined curve of which every separate piece is trivially convex. Since the Huber function is also continuous throughout $\mathbb{R}$, it follows that it is convex everywhere on $\mathbb{R}$. Following Rockafellar Theorem 5.1, a composite function is convex if and only if both functions are convex and the outer function is non-decreasing. The L_p norm is convex by Minkowski's Inequality (see Section A.2) and it is non-decreasing by definition. Thus we conclude that the composition of the L_p norm and the Huber hinge error given by

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

is indeed convex. Moreover, note that both the Huber hinge function and the L_p norm are *proper* convex functions[6]. The composition of these functions is also a proper convex function. Then, following from Rockafellar Theorem 5.2 multiplication with the non-negative weights ρ_i conserves (proper) convexity. By the same theorem it follows that after summation over all objects i and multiplication with $1/n$, the composition remains convex.

It now remains to be shown that the additive penalty term $\text{tr}\ \mathbf{V}'\mathbf{J}\mathbf{V}$ is a proper convex function in $\mathbf{V}$. The proof of this property is given in Section A.2. Multiplying this function with a non-negative constant λ preserves convexity. Hence it is shown that $L_{\text{MSVM}}(\mathbf{V})$ is indeed convex in $\mathbf{V}$. □

We now turn to the optimization algorithm used to minimize $L_{\text{MSVM}}(\mathbf{V})$. This algorithm uses the fact that Theorem 1 holds.

[4]Theorems from Rockafellar (1997) are repeated in Section A.1

[5]Since $\mathbb{R}^{(m+1)\times(K-1)}$ is an affine set, any linear function from $\mathbb{R}^{(m+1)\times(K-1)}$ to $\mathbb{R}$ is convex (Rockafellar, 1997). Notice that all theorems that hold for $\mathbb{R}^n$ also hold for $\mathbb{R}^{(m+1)\times(K-1)}$ since vectorization is always possible. For a review of convex function theory on more general vector spaces refer to Zălinescu (2002).

[6]As noted in the appendix, a convex function f is considered a proper convex function if $f(x) < +\infty$ for at least one x and $f(x) > -\infty$ for all x in the domain.

CHAPTER
FIVE

MAJORIZATION

It is now required to find an optimization scheme to minimize the loss function presented in the previous chapter. Similar to the binary SVM case treated in Groenen et al. (2008), this is done using the Iterative Majorization algorithm of De Leeuw (1977). This chapter starts with a review of this algorithm and a formal treatment of the underlying mathematics.

5.1 Iterative Majorization

Iterative Majorization (IM) is a relatively straightforward method for minimizing a (convex) function. The method is based on introducing a so-called *majorization function* which touches the original function at a *supporting point* and is generally always above the original function. By choosing a simple form for this majorization function, the minimum of this function can easily be found. The value of the original function at this minimum is at most equal to the value at the supporting point, but will generally be smaller. If the minimum of one majorization function is used as the supporting point for a next majorization a sequence of function values is created which converges to the minimum of the original function, provided this is convex. This procedure is illustrated in Figure 5.1 for the case where the majorization function is quadratic.

A formal treatment follows, adapted from Groenen et al. (1995). Note that in this treatment $\mathbf{Y}$ denotes a matrix variable. Let $\phi(\mathbf{Y})$ be a real-valued function of $\mathbf{Y} \in \mathcal{Y}$ which needs to be minimized over the domain $\mathcal{Y}$. In the IM algorithm a majorization function $\psi(\mathbf{Y}, \overline{\mathbf{Y}})$ with $\mathbf{Y}, \overline{\mathbf{Y}} \in \mathcal{Y}$ is introduced, for which

$$\phi(\mathbf{Y}) \leq \psi(\mathbf{Y}, \overline{\mathbf{Y}}) \qquad \text{and} \qquad \phi(\overline{\mathbf{Y}}) = \psi(\overline{\mathbf{Y}}, \overline{\mathbf{Y}}), \tag{5.1}$$

for all $\mathbf{Y}, \overline{\mathbf{Y}} \in \mathcal{Y}$. That is, the original function is always smaller or equal than the majorization function and the majorization function touches the original function at the *supporting point* $\overline{\mathbf{Y}}$. Let $\widehat{\mathbf{Y}}$ denote the minimum of $\psi(\mathbf{Y}, \overline{\mathbf{Y}})$ for fixed $\overline{\mathbf{Y}}$, that is $\widehat{\mathbf{Y}} = \arg\min_{\mathbf{Y} \in \mathcal{Y}} \psi(\mathbf{Y}, \overline{\mathbf{Y}})$. Then the following inequalities hold

$$\phi(\widehat{\mathbf{Y}}) \leq \psi(\widehat{\mathbf{Y}}, \overline{\mathbf{Y}}) \leq \psi(\overline{\mathbf{Y}}, \overline{\mathbf{Y}}) = \phi(\overline{\mathbf{Y}}).$$

The equality $\phi(\widehat{\mathbf{Y}}) = \psi(\widehat{\mathbf{Y}}, \overline{\mathbf{Y}})$ only occurs when $\widehat{\mathbf{Y}}$ is a stationary point of the original function $\phi(\mathbf{Y})$. When the function $\phi(\mathbf{Y})$ is convex, this stationary point corresponds to the global minimum of the function[1]. The IM algorithm can thus be formulated as follows

[1]Note that this global minimum is unique if the function is *strictly* convex.

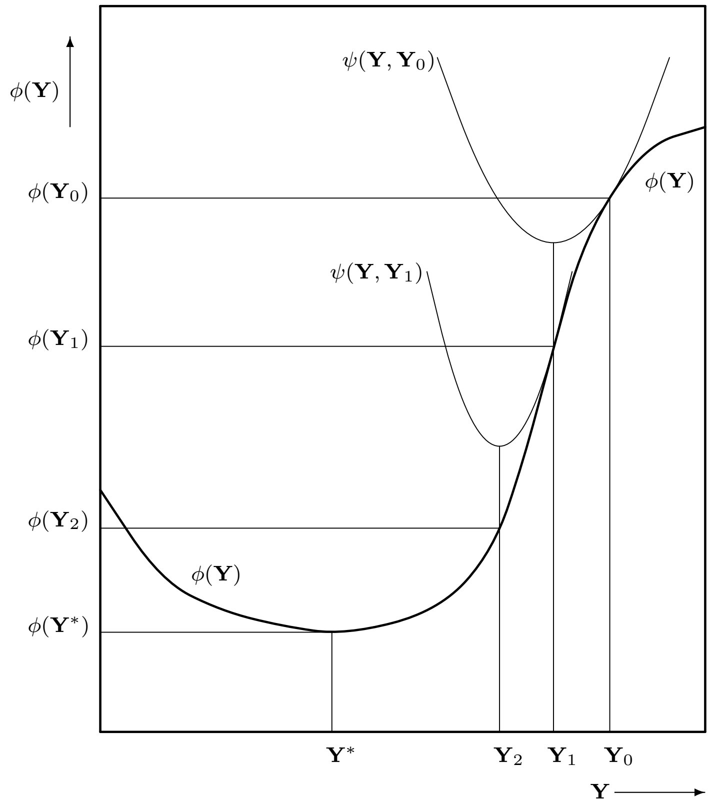


Figure 5.1 – One-dimensional graphical illustration of the Iterative Majorization algorithm, with a quadratic majorization function. Successive configurations $\mathbf{Y}_i$ are found by calculating the minimum of each majorization function ψ. The value of $\phi(\mathbf{Y}_i)$ decreases in every iteration. Adapted from De Leeuw (1988).

```
1. Let $\overline{\mathbf{Y}} = \overline{\mathbf{Y}}_0$, and calculate $\phi(\overline{\mathbf{Y}})$. Here $\overline{\mathbf{Y}}_0$ is a starting value.
2. While $\phi(\overline{\mathbf{Y}}) - \phi(\hat{\mathbf{Y}}) < \epsilon$
    a. Find $\hat{\mathbf{Y}} = \arg\min_{\mathbf{Y} \in \mathcal{Y}} \psi(\mathbf{Y}, \overline{\mathbf{Y}})$.
    b. Let $\overline{\mathbf{Y}} = \hat{\mathbf{Y}}$.
    c. Calculate $\phi(\hat{\mathbf{Y}})$.
3. Return $\hat{\mathbf{Y}}$.
```

In this algorithm, ϵ is a small positive constant. This majorization algorithm yields a non-increasing sequence of function values $\phi(\mathbf{Y})$. Hence the above procedure constitutes a *guaranteed descent* algorithm. Notice that the sequence of function values may never converge if the function is not bounded from below. If a simple functional form is chosen for the majorization function, Step 2a in the algorithm above can be done quickly. A common choice is to use a quadratic majorization function, since the minimum of a quadratic function can be derived analytically and calculated fast.

5.2 Majorization of $L_{\mathbf{MSVM}}$

We now turn to the derivation of a quadratic majorization function for the MSVM loss function introduced in (4.4). The derivation presented below will use a number of different inequalities to arrive at a majorization function for $L_{\text{MSVM}}(\mathbf{W}, \mathbf{t})$. The parameter values obtained in an iteration of the program will be used as a supporting point for the next iteration. The parameters from the previous iteration of the IM algorithm will always be denoted with a bar. Thus, $\overline{\mathbf{W}}$ equals the value of $\mathbf{W}$ from the previous iteration. With this notation, the quadratic majorization function for $L_{\text{MSVM}}(\mathbf{W}, \mathbf{t})$ should be of the form

$$L_{\text{MSVM}}(\mathbf{W}, \mathbf{t}) \leq \text{ tr } \mathbf{W}'\mathbf{X}'\mathbf{A}_{\overline{\mathbf{W}}}\mathbf{X}\mathbf{W} - 2 \,\text{tr } \mathbf{W}'\mathbf{X}'\mathbf{B}_{\overline{\mathbf{W}}} + \mathbf{t}'\mathbf{A}_{\bar{\mathbf{t}}}\mathbf{t} - 2\mathbf{t}'\mathbf{b}_{\bar{\mathbf{t}}} + C_{\overline{\mathbf{W}},\bar{\mathbf{t}}} \tag{5.2}$$

where $\mathbf{A}_{\overline{\mathbf{W}}}$ and $\mathbf{B}_{\overline{\mathbf{W}}}$ are coefficient matrices depending on $\overline{\mathbf{W}}$, and $\mathbf{A}_{\bar{\mathbf{t}}}$ and $\mathbf{b}_{\bar{\mathbf{t}}}$ are respectively a coefficient matrix and coefficient vector depending on $\bar{\mathbf{t}}$. Furthermore, $C_{\overline{\mathbf{W}},\bar{\mathbf{t}}}$ is a constant depending on $\overline{\mathbf{W}}$ and $\bar{\mathbf{t}}$. Naturally the expected majorization function depends on the object matrix $\mathbf{X}$.

5.2.1 Majorization of Case 1 errors

In the derivation we recognize two possible scenarios for each object. If the errors from a previous iteration $h(\overline{q}_i^{(jk)})$ with $j \neq y_i$, are nonzero for at most one $j \neq y_i$, a straightforward majorization of the L_p norm is used. Following equation (2.10.3) from Hardy et al. (1934), the L_p norm follows the inequality

$$\left(\sum_{i=1}^{n} x_i^p \right)^{1/p} \leq \sum_{i=1}^{n} x_i,$$

where equality occurs whenever at most one x_i is nonzero or $p = 1$. If we apply this to the L_p norm of the errors $h(q_i^{(jk)})$, we find

$$\left(\sum_{j \neq k} h^p \left(q_i^{(jk)} \right) \right)^{1/p} \leq \sum_{j \neq k} h \left(q_i^{(jk)} \right). \tag{5.3}$$

To use this inequality as a majorization inequality in the IM algorithm it is required that the inequality reduces to an equality at the supporting point. Thus it is required that

$$\left(\sum_{j \neq k} h^p \left(\overline{q}_i^{(jk)} \right) \right)^{1/p} = \sum_{j \neq k} h \left(\overline{q}_i^{(jk)} \right). \tag{5.4}$$

This equality only holds when there is at most one $h(\overline{q}_i^{(jk)})$ larger than zero, with $j \neq y_i$. Objects for which this requirement is satisfied are called *Case 1 objects.* A case indicator ε_i is defined such that $\varepsilon_i = 1$ if an object i is a Case 1 object. Mathematically this is expressed as

$$\varepsilon_i = \begin{cases} 1 & \text{if } h\left(\overline{q}_i^{(jk)}\right) > 0 \text{ for } j = \ell \neq y_i \text{ and } h\left(\overline{q}_i^{(jk)}\right) = 0 \text{ for } j \neq \ell \neq y_i, \\ 1 & \text{if } h\left(\overline{q}_i^{(jk)}\right) = 0 \text{ for } j \neq y_i, \\ 0 & \text{otherwise.} \end{cases} \tag{5.5}$$

Objects for which $\varepsilon_i = 0$ are called *Case 2 objects*. For these objects more than one $h(\overline{q}_i^{(jk)})$ is larger than zero, with $j \neq y_i$. If the majorization inequality (5.3) is used for the Case 1 objects, the following inequality holds for the MSVM loss function

$$L_{\text{MSVM}}(\mathbf{W}, \mathbf{t}) \leq \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in G_k} \rho_i \left[\varepsilon_i \sum_{j \neq k} h\left(q_i^{(jk)}\right) + (1 - \varepsilon_i) \left(\sum_{j \neq k} h^p \left(q_i^{(jk)}\right) \right)^{1/p} \right] + \lambda \operatorname{tr} \mathbf{W}'\mathbf{W}. \tag{5.6}$$

The majorization of the L_p norm cannot be used for the Case 2 objects, since (5.4) does not hold for these objects. However, in the next section an inequality will be derived which can be used for the Case 2 objects.

5.2.2 Majorization of the p-th root function

We now introduce a linear majorization of the root function $f(x) = x^{1/p}$ with a supporting point at $x = y$. The majorization function is denoted by $g(x, y) = cx + d$. To ensure $g(x, y)$ touches at $x = y$ the following conditions must hold

1. $f'(y) = c$
2. $f(y) = cy + d$

From this it follows that[2]

$$x^{1/p} \leq \frac{1}{p} y^{1/p-1} x + \left(1 - \frac{1}{p}\right) y^{1/p}.$$

If we apply this majorization to the L_p norm of the Case 2 objects, we have the inequality

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

To shorten the notation we introduce

$$\omega_i = \frac{1}{p} \left(\sum_{j \neq k} h^p \left(\overline{q}_i^{(jk)}\right) \right)^{1/p-1}, \tag{5.7}$$

the majorization of L_{MSVM} can then be written as

$$L_{\text{MSVM}}(\mathbf{W}, \mathbf{t}) \leq \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in G_k} \rho_i \left[\varepsilon_i \sum_{j \neq k} h\left(q_i^{(jk)}\right) + (1 - \varepsilon_i)\omega_i \sum_{j \neq k} h^p \left(q_i^{(jk)}\right) \right] + \Gamma^{(1)} + \lambda \operatorname{tr} \mathbf{W}'\mathbf{W}, \tag{5.8}$$

[2]This inequality was also noted by Groenen and Heiser (1996) and follows directly from Theorem 37 of Hardy et al. (1934).

where the constant terms for all objects in Case 2 have been collected in

$$\Gamma^{(1)} = \frac{1}{n}\left(1 - \frac{1}{p}\right)\sum_{k=1}^{K}\sum_{i \in G_k} \rho_i(1-\varepsilon_i)\left(\sum_{j \neq k} h^p\left(\overline{q}_i^{(jk)}\right)\right)^{1/p}. \tag{5.9}$$

To continue the majorization of $L_{\text{MSVM}}(\mathbf{W}, \mathbf{t})$, the quadratic majorization of the Huber functions $h(q)$ and $h^p(q)$ is needed.

5.2.3 Majorization of Huber functions

To keep the focus on the main derivation of the MSVM loss function the derivation of the majorization function for $h^p(q_i^{(jk)})$ has been moved to Appendix B. From this derivation it is found that the parameters of the quadratic majorization function depend explicitly on the value of p. Naturally, these parameters also depend on i, j, and k. Therefore, the quadratic majorization inequality for the Case 2 objects can be expressed as

$$h^p\left(q_i^{(jk)}\right) \leq a_{ijk}^{(p)}\left(q_i^{(jk)}\right)^2 - 2b_{ijk}^{(p)} q_i^{(jk)} + c_{ijk}^{(p)}, \tag{5.10}$$

similarly for the Case 1 objects we have

$$h\left(q_i^{(jk)}\right) \leq a_{ijk}^{(1)}\left(q_i^{(jk)}\right)^2 - 2b_{ijk}^{(1)} q_i^{(jk)} + c_{ijk}^{(1)}. \tag{5.11}$$

Notice that for the Case 1 objects we simply substitute $p = 1$ in the majorization of $h^p(q)$. Plugging these inequalities into (5.8) gives after some rearranging

$$\begin{aligned} L_{\text{MSVM}}(\mathbf{W}, \mathbf{t}) \leq & \frac{1}{n}\sum_{k=1}^{K}\sum_{i \in G_k} \rho_i \varepsilon_i \sum_{j \neq k}\left[a_{ijk}^{(1)}\left(q_i^{(jk)}\right)^2 - 2b_{ijk}^{(1)} q_i^{(jk)}\right] \\ & + \frac{1}{n}\sum_{k=1}^{K}\sum_{i \in G_k} \rho_i(1-\varepsilon_i)\omega_i \sum_{j \neq k}\left[a_{ijk}^{(p)}\left(q_i^{(jk)}\right)^2 - 2b_{ijk}^{(p)} q_i^{(jk)}\right] \\ & + \Gamma^{(2)} + \lambda \operatorname{tr} \mathbf{W}'\mathbf{W}, \end{aligned} \tag{5.12}$$

where $\Gamma^{(2)}$ contains all the constant terms

$$\Gamma^{(2)} = \Gamma^{(1)} + \frac{1}{n}\sum_{k=1}^{K}\sum_{i \in G_k} \rho_i\left[\varepsilon_i \sum_{j \neq k} c_{ijk}^{(1)} + (1-\varepsilon_i)\omega_i \sum_{j \neq k} c_{ijk}^{(p)}\right]. \tag{5.13}$$

To get a form of $L_{\text{MSVM}}(\mathbf{W}, \mathbf{t})$ where the dependence on the parameters is more pronounced, we switch to the notation introduced in Section 4.4. For reference, the notation is repeated here

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

Then it follows that

$$\begin{aligned} q_i^{(jk)} &= \mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}, \\ \left(q_i^{(jk)}\right)^2 &= \mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'\mathbf{V}'\mathbf{z}_i. \end{aligned}$$

Again let $\mathbf{J}$ equal the $(m+1)\times(m+1)$ identity matrix with $j_{11}=0$. The majorization inequality (5.12) can then be written as

$$\begin{aligned} L_{\mathrm{MSVM}}(\mathbf{V}) \leq & \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\varepsilon_i\sum_{j\neq k}\left[a_{ijk}^{(1)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'\mathbf{V}'\mathbf{z}_i - 2b_{ijk}^{(1)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\right] \\ & + \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i(1-\varepsilon_i)\omega_i\sum_{j\neq k}\left[a_{ijk}^{(p)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'\mathbf{V}'\mathbf{z}_i - 2b_{ijk}^{(p)}\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\right] \\ & + \Gamma^{(2)} + \lambda\,\mathrm{tr}\,\mathbf{V}'\mathbf{J}\mathbf{V}. \end{aligned} \tag{5.14}$$

This expression is not yet the desired quadratic form, because of the dependence on $\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'$ (a $(K-1)\times(K-1)$ dimensional matrix). A final majorization step is necessary to remove this dependence.

5.2.4 Majorization of the quadratic term

For the majorization of the quadratic term, the maximum eigenvalue inequality is used, first noted by Bijleveld and De Leeuw (1991). For a description and a proof of this inequality refer to Section A.3. Using this inequality we find that

$$\mathbf{z}_i'\mathbf{V}\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'\mathbf{V}'\mathbf{z}_i \leq \mathbf{z}_i'\mathbf{V}\mathbf{V}'\mathbf{z}_i - 2\mathbf{z}_i'\mathbf{V}(\mathbf{I}-\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}')\overline{\mathbf{V}}'\mathbf{z}_i + \mathbf{z}_i'\overline{\mathbf{V}}(\mathbf{I}-\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}')\overline{\mathbf{V}}'\mathbf{z}_i, \tag{5.15}$$

where we have used that the upper bound on the eigenvalues of $\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}'$ equals 1, as derived in Section 4.1. Plugging this inequality into the majorization inequality above and rearranging yields

$$\begin{aligned} L_{\mathrm{MSVM}}(\mathbf{V}) \leq & \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\left[\varepsilon_i\sum_{j\neq k}a_{ijk}^{(1)} + (1-\varepsilon_i)\omega_i\sum_{j\neq k}a_{ijk}^{(p)}\right]\mathbf{z}_i'\mathbf{V}(\mathbf{V}-2\overline{\mathbf{V}})'\mathbf{z}_i \\ & - \frac{2}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\mathbf{z}_i'\mathbf{V}\left[\varepsilon_i\sum_{j\neq k}\left(b_{ijk}^{(1)} - a_{ijk}^{(1)}\overline{q}_i^{(jk)}\right)\boldsymbol{\delta}_{kj}\right. \\ & \qquad\qquad \left. + (1-\varepsilon_i)\omega_i\sum_{j\neq k}\left(b_{ijk}^{(p)} - a_{ijk}^{(p)}\overline{q}_i^{(jk)}\right)\boldsymbol{\delta}_{kj}\right] \\ & + \Gamma^{(3)} + \lambda\,\mathrm{tr}\,\mathbf{V}'\mathbf{J}\mathbf{V}, \end{aligned} \tag{5.16}$$

where it has been used that $\mathbf{z}_i'\overline{\mathbf{V}}\boldsymbol{\delta}_{kj} = \overline{q}_i^{(jk)}$, and

$$\begin{aligned} \Gamma^{(3)} = & \Gamma^{(2)} \\ & + \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in G_k}\rho_i\mathbf{z}_i'\overline{\mathbf{V}}\left[\varepsilon_i\sum_{j\neq k}a_{ijk}^{(1)}(\mathbf{I}-\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}') + (1-\varepsilon_i)\omega_i\sum_{j\neq k}a_{ijk}^{(p)}(\mathbf{I}-\boldsymbol{\delta}_{kj}\boldsymbol{\delta}_{kj}')\right]\overline{\mathbf{V}}'\mathbf{z}_i. \end{aligned}$$

We now simplify the notation further by introducing

$$\alpha_i = \frac{1}{n}\rho_i\left[\varepsilon_i\sum_{j\neq k}a_{ijk}^{(1)} + (1-\varepsilon_i)\omega_i\sum_{j\neq k}a_{ijk}^{(p)}\right], \tag{5.17}$$

$$\boldsymbol{\beta}_i = \frac{1}{n}\rho_i\left[\varepsilon_i\sum_{j\neq k}\left(b_{ijk}^{(1)} - a_{ijk}^{(1)}\overline{q}_i^{(jk)}\right)\boldsymbol{\delta}_{kj} + (1-\varepsilon_i)\omega_i\sum_{j\neq k}\left(b_{ijk}^{(p)} - a_{ijk}^{(p)}\overline{q}_i^{(jk)}\right)\boldsymbol{\delta}_{kj}\right]. \tag{5.18}$$

The loss function inequality (5.16) can then be written as

$$L_{\text{MSVM}}(\mathbf{V}) \leq \sum_{k=1}^{K} \sum_{i \in G_k} \alpha_i \mathbf{z}_i' \mathbf{V}(\mathbf{V} - 2\overline{\mathbf{V}})' \mathbf{z}_i - \sum_{k=1}^{K} \sum_{i \in G_k} \mathbf{z}_i' \mathbf{V} \boldsymbol{\beta}_i + \Gamma^{(3)} + \lambda \operatorname{tr} \mathbf{V}'\mathbf{J}\mathbf{V}. \quad (5.19)$$

It is now possible to write this expression as a function of matrix operations. 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 where the i-th row of $\mathbf{B}$ equals $\boldsymbol{\beta}_i'$. The summations can be written as trace operations of the matrices using the following identities

$$\sum_{k=1}^{K} \sum_{i \in G_k} \alpha_i \mathbf{z}_i' \mathbf{V} \overline{\mathbf{V}}' \mathbf{z}_i = \operatorname{tr} \overline{\mathbf{V}}' \mathbf{Z}' \mathbf{A} \mathbf{Z} \mathbf{V} \quad (5.20)$$

$$\sum_{k=1}^{K} \sum_{i \in G_k} \mathbf{z}_i' \mathbf{V} \boldsymbol{\beta}_i = \operatorname{tr} \mathbf{B}' \mathbf{Z} \mathbf{V}, \quad (5.21)$$

with $\mathbf{Z}$ the $n \times (m+1)$ matrix with rows $\mathbf{z}_i'$. Then

$$\begin{aligned} L_{\text{MSVM}}(\mathbf{V}) &\leq \operatorname{tr}(\mathbf{V} - 2\overline{\mathbf{V}})' \mathbf{Z}' \mathbf{A} \mathbf{Z} \mathbf{V} - 2 \operatorname{tr} \mathbf{B}' \mathbf{Z} \mathbf{V} + \Gamma^{(3)} + \lambda \operatorname{tr} \mathbf{V}' \mathbf{J} \mathbf{V}, \\ &= \operatorname{tr} \mathbf{V}'(\mathbf{Z}'\mathbf{A}\mathbf{Z} + \lambda \mathbf{J})\mathbf{V} - 2 \operatorname{tr}(\overline{\mathbf{V}}' \mathbf{Z}' \mathbf{A} + \mathbf{B}') \mathbf{Z} \mathbf{V} + \Gamma^{(3)}. \end{aligned} \quad (5.22)$$

This final expression is the desired quadratic majorization inequality, corresponding to the form specified in (5.2). Minimizing the right-hand side of this inequality and solving for $\mathbf{V}$ gives an update of the IM algorithm. We now turn to an explicit derivation of the first-order condition and show that this indeed yields a minimum.

5.2.5 Minimization conditions

By differentiating the right-hand side of (5.22) and setting the derivative to zero, a system of linear equations in $\mathbf{V}$ is found. The derivatives of trace functions are, following Magnus and Neudecker (1988),

$$\begin{aligned} \frac{\partial \operatorname{tr} \mathbf{X}'\mathbf{F}\mathbf{X}}{\partial \mathbf{X}} &= (\mathbf{F} + \mathbf{F}')\, \mathbf{X}, \\ \frac{\partial \operatorname{tr} \mathbf{F}\mathbf{X}}{\partial \mathbf{X}} &= \mathbf{F}', \end{aligned}$$

where $\mathbf{F}$ and $\mathbf{X}$ are arbitrary matrices. Applying this to the majorization inequality (5.22) and setting the derivative to zero yields

$$2(\mathbf{Z}'\mathbf{A}\mathbf{Z} + \lambda \mathbf{J})\mathbf{V} - 2(\mathbf{Z}'\mathbf{A}\mathbf{Z}\overline{\mathbf{V}} + \mathbf{Z}'\mathbf{B}) = \mathbf{0},$$

where it has been used that $\mathbf{A}$ and $\mathbf{J}$ are symmetric matrices. Then the update of $\mathbf{V}$ can be found efficiently by solving the following linear system using Gaussian elimination

$$(\mathbf{Z}'\mathbf{A}\mathbf{Z} + \lambda \mathbf{J})\mathbf{V} = \mathbf{Z}'\mathbf{A}\mathbf{Z}\overline{\mathbf{V}} + \mathbf{Z}'\mathbf{B}. \quad (5.23)$$

It can be shown that the second order condition for a minimum is also satisfied, by applying the second derivative test. To do this, the second derivative of the loss function must be shown to be positive definite. Then it must hold that

$$\mathbf{Z}'\mathbf{A}\mathbf{Z} + \lambda \mathbf{J}$$

is a positive definite matrix. Since the sum of all α_i can never be zero and it is required that $\lambda > 0$, this is indeed the case. To make this argument more clear, note that this matrix can also be written as

$$\mathbf{Z}'\mathbf{A}\mathbf{Z} + \lambda\mathbf{J} = \begin{bmatrix} \sum_{i=1}^{n} \alpha_i & \sum_{i=1}^{n} \alpha_i \mathbf{x}_i' \\ \sum_{i=1}^{n} \alpha_i \mathbf{x}_i & \lambda\mathbf{I}_m + \sum_{i=1}^{n} \alpha_i \mathbf{x}_i \mathbf{x}_i' \end{bmatrix},$$

where $\mathbf{I}_m$ is the $m \times m$ identity matrix. Now it is clear that this matrix is positive definite as long as $\sum_i \alpha_i > 0$ and $\lambda > 0$. The latter condition holds since this is required in the formulation of $L_{\text{MSVM}}(\mathbf{W}, \mathbf{t})$. The requirement that the α_i have a positive sum is satisfied since for all values of $p \in [1, 2]$ we have $a_{ijk}^{(p)} > 0$[3]. Hence, the second order condition holds and solving (5.23) always yields a minimum of the majorization function.

[3]This can be proven from the general expression for a as derived in Appendix B. This proof is omitted from this text, however it is based on reducing the inequality $a > 0$ to $\kappa > -1$, which holds by design. For $p = 1$ this can easily be shown.

CHAPTER
SIX

RESULTS

The details of the implementation of the algorithm will be presented here, along with a number of speed-up techniques that can be used in the algorithm. To emphasize the use of the majorization algorithm, the proposed multiclass classification algorithm will be referred to as *MSVM-Maj*. The methods used to analyse the performance of MSVM-Maj are given, as well as a method to compare the performance of MSVM-Maj with other classifiers. This comparison technique is then used to compare the performance of MSVM-Maj with a number of other classification techniques on four benchmark datasets. The computation time necessary for each of the classification methods is also compared.

6.1 Implementation

The implementation of the IM algorithm in MSVM-Maj is relatively straightforward. A representation in pseudocode is given in Algorithm 1. Note that elements of a matrix are given as $\mathbf{Q}_{i,j}$, where i denotes the row and j the column of matrix $\mathbf{Q}$. The actual implementation has been written in MATLAB code. By making extensive use of matrices and the built-in Hadamard product[1] in MATLAB, it was possible to achieve significant speed-ups over the naive implementation given in Algorithm 1.

As can be seen in the formulation of the algorithm, each iteration consists of recalculating the errors $\overline{q}_i^{(jk)}$ and calculating the Huber weighted errors $h\left(\overline{q}_i^{(jk)}\right)$. Then the case indicator ε_i is determined, and the parameters of the Huber majorization ($\mathbf{A}$ and $\mathbf{B}$) are recalculated. Finally, the new value of $\mathbf{V}$ is found by solving (5.23) using Gaussian elimination.

As mentioned, significant speed-ups can be achieved when the Hadamard product is used. To expand on this, some of these techniques will be given here. First of all, the Case indicator ε_i can be calculated for all i simultaneously and stored in a vector $\mathbf{e}$ using the MATLAB command

```
e = sum((H.*R)>0,2)<=1;
```

Here $\mathbf{H}$ is the $n \times K$ matrix of Huber hinge errors, and `.*` denotes the Hadamard product in MATLAB. Furthermore, $\mathbf{R}$ is an $n \times K$ matrix with elements $r_{ij} = 1$ if $j \neq y_i$, and $r_{iy_i} = 0$. Thus taking the Hadamard product of a matrix with $\mathbf{R}$ sets the elements in the columns $j = y_i$ equal to zero. Consequently summing over the columns of the result gives an efficient

[1]The Hadamard product of two matrices $\mathbf{A}$ and $\mathbf{B}$ of equal dimensions is defined such that if the Hadamard product is denoted by $\circ$, and $\mathbf{C} = \mathbf{A} \circ \mathbf{B}$ then $c_{ij} = a_{ij} \cdot b_{ij}$.

Algorithm 1: The MSVM-Maj algorithm.

```
input : $\mathbf{X}, \mathbf{y}, \boldsymbol{\rho}, p, \kappa, \lambda$
output: $\mathbf{W}, \mathbf{t}$
1  $K \leftarrow \max(\mathbf{y})$
2  $n \leftarrow$ first dimension of $\mathbf{X}$
3  $\epsilon \leftarrow 3 \cdot 10^{-10}$
4  $it \leftarrow 0$
5  $\mathbf{Z} \leftarrow [\mathbf{1}\ \mathbf{X}]$
6  $\overline{L} \leftarrow 0$
7  $\overline{\mathbf{V}} \leftarrow \overline{\mathbf{V}}_0$
8  Generate $\mathbf{J}, \mathbf{U}$
9  Calculate $L \leftarrow L_{\text{MSVM}}(\overline{\mathbf{V}})$ by (4.4)
10 while $it = 0$ or $(\overline{L} - L)/L > \epsilon$ do
11     $\mathbf{Q}_{i,j} \leftarrow \mathbf{z}_i'\overline{\mathbf{V}}(\mathbf{u}_{y_i} - \mathbf{u}_j)$
12     $\mathbf{H}_{i,j} \leftarrow h(\mathbf{Q}_{i,j})$ by (4.3)
13     Calculate $\varepsilon_i$ for all $i$ using $\mathbf{H}_{i,j}$ and (5.5)
14     forall $i \in [1, n]$ do
15         if $\varepsilon_i = 1$ then
16             $\mathbf{A}_{i,i} \leftarrow \frac{1}{n}\rho_i \sum_{j \neq y_i} a_{ijy_i}^{(1)}$
17             $\mathbf{B}_i \leftarrow \frac{1}{n}\rho_i \sum_{j \neq y_i} \left(b_{ijy_i}^{(1)} - a_{ijy_i}^{(1)}\mathbf{Q}_{i,j}\right)(\mathbf{u}_{y_i} - \mathbf{u}_j)'$
18         else
19             $\mathbf{A}_{i,i} \leftarrow \frac{1}{n}\rho_i\,\omega_i \sum_{j \neq y_i} a_{ijy_i}^{(p)}$
20             $\mathbf{B}_i \leftarrow \frac{1}{n}\rho_i\,\omega_i \sum_{j \neq y_i} \left(b_{ijy_i}^{(p)} - a_{ijy_i}^{(p)}\mathbf{Q}_{i,j}\right)(\mathbf{u}_{y_i} - \mathbf{u}_j)'$
21         end
22     end
23     Solve (5.23) for $\mathbf{V}$
24     $\overline{L} \leftarrow L$
25     Calculate $L \leftarrow L_{\text{MSVM}}(\mathbf{V})$ by (4.4)
26     $\overline{\mathbf{V}} \leftarrow \mathbf{V}$
27     $it \leftarrow it + 1$
28 end
29 Extract $\mathbf{W}$ and $\mathbf{t}$ from $\mathbf{V}$
```

method to perform the $\sum_{j \neq k}$ summation used often in the coefficients of the majorization function.

Another method used to speed up computation is to use logical matrices to distinguish between the regions of the Huber hinge error function. With the errors $q_i^{(jk)}$ of Case 1 objects stored in a matrix **Q1**, the following matrices are very useful

```
G1 = (Q1 <= -kappa);
G2 = (Q1 <= 1)&(~G1);
G3 = (~G1)&(~G2);
```

Then calculation of, for example, the quadratic coefficients $a_{ijk}^{(1)}$ is done by simply executing[2]

```
A1 = 1./(4*Phi1) .* (G1 - G3) + 1/(2*(kappa+1)) * G2;
```

[2]Note that it is also required here that `Phi1 = 1 - Q1 - (kappa+1)/2`.

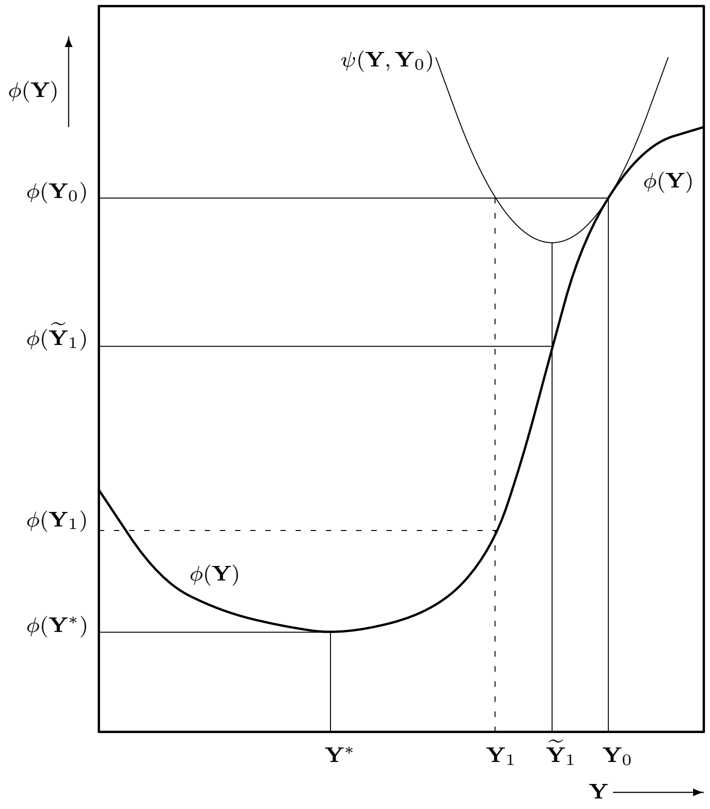


Figure 6.1 – Graphical illustration of step doubling in the IM algorithm. Without step doubling, $\widetilde{\mathbf{Y}}_1$ would be the next configuration of the parameters. However, with step doubling $\mathbf{Y}_1$ is used. It can be seen that in this way a much larger majorization step can be achieved.

Finally, the IM algorithm allows for a speed-up technique which can reduce the number of necessary iterations by half, known as *step doubling* (De Leeuw and Heiser, 1980). Instead of taking the value of $\mathbf{V}$ at the minimum of the quadratic majorization function for the next iteration, it is possible to mirror the supporting point on the opposite side of the quadratic majorization function. This process is illustrated in Figure 6.1. As can be seen in the figure, the use of step doubling creates a larger majorization step and therefore a faster convergence of the algorithm. To use step doubling in Algorithm 1 the following line must be inserted before line 25 of the program

$$\mathbf{V} \leftarrow 2\mathbf{V} - \overline{\mathbf{V}}. \tag{6.1}$$

In the actual Matlab implementation step doubling was applied after 50 iterations. Note that if step doubling is used the convergence of the algorithm is still guaranteed, because the loss function is convex. An explicit proof of this property is omitted from this text.

6.2 Performance Evaluation

To compare the performance of the MSVM algorithm with other classification methods, a number of different evaluation methods are used. First, N-fold cross validation is used. In N-fold cross validation the total sample is split into N disjoint subsamples. One of these subsamples is retained as the *holdout* or *test* dataset and the remaining $N-1$ subsamples are combined to form the *training* dataset. The classification rule is then constructed using the training dataset and tested on the holdout sample. Comparison between the known class labels of the holdout sample and the predicted class labels yields a percentage of the correctly classified samples. This procedure is repeated for all N folds. In the testing procedure used here, $N = 10$ is chosen.

Using N-fold cross validation gives a classification rate for a classifier, which can be used to compare different classification methods. However as noted by Rifkin and Klautau (2004), a comparison of classification rates does not give any information about whether or not two classifiers differ significantly. Therefore, Rifkin et al. propose to use the following bootstrap procedure to compare two classification methods (coded here as "A" and "B")

1. For each object $\mathbf{x}$ in the original dataset, use the following coding scheme
   - CC : Both A and B classify $\mathbf{x}$ correctly,
   - II : Both A and B classify $\mathbf{x}$ incorrectly,
   - CI : A classifies $\mathbf{x}$ correctly, B does not,
   - IC : B classifies $\mathbf{x}$ correctly, A does not.

2. Calculate the empirical probabilities for each of these events as

   $$P(E) = n(E)/n,$$

   where $E \in \{CC, II, CI, IC\}$, $n(E)$ is the number of times this event occurs in the original dataset, and n is the total number of objects in the dataset.

3. Generate a large number (following Rifkin et al., we use 10,000) of bootstrap samples of size n where each object in the sample belongs to one of the four events with probability equal to the empirical probability of that event, $P(E)$. Note that this differs from a regular bootstrap of the predicted class labels because we are interested in the difference of the performance of the two classifiers.

4. For each bootstrap sample, compute the difference between the percentage of CI events and the percentage of IC events in the sample. This gives a distribution of the difference in performance of classifier A and classifier B.

5. Create a confidence interval by reporting the 5% and 95% percentiles of this distribution.

Using this bootstrapping scheme, it follows that there is a significant difference between the classifiers if 0 is not in the confidence interval.

In our experiments, cross validation is first used to find the optimal parameter configuration of each classification method. These parameter configurations are then used in the bootstrap procedure to evaluate whether or not two classifiers differ significantly.

6.3 Experimental Setup

The example datasets used in our experiments are taken from the UCI repository (Frank and Asuncion, 2010). The UCI repository is a public repository of datasets commonly used by researchers in Machine Learning. Because of this, cross validation rates obtained through

Table 6.1 – Properties of the datasets used in the experiments. For each dataset the number of objects (n), the number of explanatory variables (m), and the number of classes (K) are reported.

Dataset	n	m	K
iris	150	4	3
wine	178	14	3
glass	214	9	6
vehicle	846	18	4

SVM classification of a number of datasets are available in the literature (e.g. Hsu and Lin (2002)). Descriptive statistics of each of the four datasets considered is given in Table 6.1.

As can be seen from the table, the *iris*, *wine*, and *glass* datasets have approximately the same number of objects but differ in the number of attributes or the number of classes. The *vehicle* dataset has a comparatively large number of objects and is included to analyse the effects of this on the performance of the MSVM-Maj algorithm. Although it is not explicitly reported here, the number of objects per class is approximately equal for the iris, wine, and vehicle datasets. For the glass dataset the largest class contains 76 objects and the smallest class contains 9 objects. This difference may be influential for some classification methods.

The following classifiers are used to compare the performance of MSVM-Maj with,

- One vs. All classification using SVM-Maj (Groenen et al., 2008)
- One vs. All classification using LibSVM (Chang and Lin, 2011)
- Linear Discriminant Analysis (LDA)
- Multinomial logit model (MLOG)

Note that no other single machine or error-correction coding MSVM approaches are compared with MSVM-Maj. As mentioned in Chapter 3, Rifkin and Klautau (2004) argue that the OVA scheme performs just as well as any single machine or error-correction coding approach. Therefore the comparison with OVA classification approach is considered to be the most important. The LDA and multinomial logit classification methods are included to compare the SVM methods with classic multiclass classification approaches.

For the classifiers where parameters need to be set, a grid search approach is used. For MSVM-Maj this grid search consists of using all combinations of the following parameter sets

- $p \in \{1.0, 1.1, \dots, 2.0\}$
- $\kappa \in \{-0.9, 0.0, 0.5, 1.0, 10\}$
- $\lambda \in \{2^{-15}, 2^{-12}, 2^{-9}, \dots, 2^{12}, 2^{15}\}$

These configurations were tested for the case where $\rho_i = 1, \forall i$ and the case where ρ_i is defined as in (4.5), to correct for group sizes. Hence MSVM-Maj was tested with a total of 1210 configurations[3].

For the SVM-Maj OVA approach a grid search was also used to find the optimal parameter configuration. Three different hinge errors are possible in SVM-Maj (absolute, quadratic and Huber), which are all considered. Similar grids for κ and λ as in MSVM-Maj were used. In both MSVM-Maj and the SVM-Maj OVA approach the stopping criterion of the IM algorithm was set at $\epsilon = 3 \cdot 10^{-10}$. In the SVM-Maj OVA approach an upper bound was set on the number of iterations to limit training time. This bound was set at 2 million iterations. It should be noted that this limit was generally not reached, and using a larger value for ϵ

[3]For the vehicle dataset the $\lambda = 2^{-15}$ case was excluded, due to the large computation time necessary.

Table 6.2 – Maximum cross validation rates for each method and each dataset.

Dataset	MSVM-Maj	SVM-Maj OVA	LibSVM OVA	LDA	MLOG
iris	<u>98.67%</u>	97.33%	97.33%	98.00%	98.00%
wine	97.22%	97.75%	97.75%	<u>100.00%</u>	98.89%
glass	63.64%	<u>67.88%</u>	64.96%	65.36%	66.82%
vehicle	<u>82.03%</u>	80.02%	79.79%	79.20%	80.85%

can avoid the necessity of this upper bound.

The LibSVM OVA approach was also tested, using the linear kernel. Again a grid search was used to find the optimal parameter values of λ. For the LDA an implementation published on the MathWorks File Exchange was used[4]. A built-in MATLAB function was used for the multinomial logit model. Finally, it must be noted that no rescaling of the variables in the datasets was performed prior to using the classification methods.

6.4 Experimental Results

Table 6.2 shows the maximum cross validation rates found in the experiments for each classification method. Overall maximum rates are underlined. It can be seen that no classification method performs consistently better than all others. Moreover, for a number of methods the obtained cross validation rates depend strongly on the initial conditions such as the selected training and test subsamples. Hence, the cross validation rates in Table 6.2 were not found consistently for all methods. However, the maximum obtained rates are still reported in the table.

The dependence on initial conditions is one of the main reasons the cross validation rates are not a reliable indicator of the performance of a method. Hence, it is more insightful to look at the results of the bootstrap procedure introduced in Section 6.2. Since we are mainly interested in how well the performance of MSVM-Maj compares with that of the other classifiers, the bootstrap procedure is only used to compare MSVM-Maj with the other methods, and not to compare the other methods with each other. The parameter configurations that yielded the highest cross validation rates (on average) are used in the bootstrap procedure[5].

For each dataset the bootstrap procedure was applied as follows. Ten training and test subsamples were constructed randomly, and all classifiers were trained on the same training sample. Then the bootstrap samples were generated between MSVM-Maj and each classifier. For each bootstrap sample the 5-th and 95-th percentiles were calculated, to get the confidence interval of the difference in performance between MSVM-Maj and the other classifier. In this way ten confidence intervals were generated, one for each training/test division. This entire procedure was repeated five times, to avoid variations in the confidence intervals as much as possible. The averages of the obtained confidence intervals are reported in Table 6.3. Note that the parameter configurations that yielded the maximum cross validation rates were used in the bootstrap procedure. For MSVM-Maj these parameter configurations are given in Table 6.4.

[4]Published by Will Dwinnell. MathWorks File ID 29673.

[5]Note that no parameters need to be set in both LDA and MLOG. For the other methods the optimal parameters found through the grid search were used.

Table 6.3 – Confidence intervals obtained from the bootstrap procedure for each dataset. MSVM-Maj is compared to each of the given classification method, such that if the confidence interval is positive MSVM-Maj performs better.

Datasets	SVM-Maj OVA	LibSVM OVA	LDA	MLOG
iris	$[-0.0009, 0.0264]$	$[0.0005, 0.0300]$	$[-0.0026, 0.0157]$	$[-0.0020, 0.0148]$
wine	$[-0.0019, 0.0012]$	$[0.0027, 0.0360]$	$[-0.0106, 0.0028]$	$[-0.0072, 0.0040]$
glass	$[-0.0034, 0.0579]$	$[0.0195, 0.0921]$	$[0.0265, 0.1062]$	$[-0.0227, 0.0324]$
vehicle	$[0.0089, 0.0358]$	$[0.0083, 0.0339]$	$[0.0253, 0.0569]$	$[0.0057, 0.0248]$

Table 6.4 – Parameter configurations for the maximum cross validation rates for MSVM-Maj, reported for each dataset. These parameter configurations were used to measure the average training time of the MSVM-Maj algorithm. The format (p, κ, λ) is used.

Dataset	Optimal parameter configuration		
iris	$(1.0, -0.9, 2^{-9})^2$,	$(1.0, 1.0, 2^{-9})^1$,	$(1.1, 0.5, 2^{-9})^1$,
	$(1.1, 1.0, 2^{-9})^1$,	$(1.2, 0.0, 2^{-9})^2$,	$(1.4, 0.5, 2^{-9})^2$,
	$(1.4, 10.0, 2^{-12})^2$,	$(1.5, 10.0, 2^{-12})^2$,	$(1.7, 0.5, 2^{-9})^2$,
	$(1.7, 10.0, 2^{-9})^2$,	$(1.8, 0.0, 2^{-9})^1$,	$(1.8, 0.5, 2^{-9})^1$,
	$(1.8, 1.0, 2^{-9})^1$,	$(1.8, 10.0, 2^{-12})^2$,	$(1.9, 0.0, 2^{-9})^2$
wine	$(1.4, 0.5, 2^{-6})^2$,	$(1.8, 0.0, 2^{-12})^2$,	$(1.9, 10.0, 2^{-15})^1$
glass	$(1.1, -0.9, 2^{-12})^1$,	$(1.3, -0.9, 2^{-15})^1$,	$(1.5, -0.9, 2^{-9})^1$,
	$(1.5, 0.5, 2^{-12})^1$,	$(1.6, 0.0, 2^{-9})^1$,	$(1.6, 0.5, 2^{-15})^1$,
	$(1.6, 1.0, 2^{-15})^1$,	$(1.7, 0.0, 2^{-12})^1$,	$(1.7, 0.5, 2^{-12})^1$,
	$(1.8, -0.9, 2^{-15})^1$,	$(1.8, 0.0, 2^{-12})^1$,	$(1.9, 0.0, 2^{-12})^1$
vehicle		$(1.3, 0.0, 2^{-9})^2$,	$(1.3, -0.9, 2^{-9})^2$

^{1}Unit weighting, $\rho_i = 1$.

^{2}Group size correction weighting, ρ_i defined as in (4.5).

The confidence intervals show the 5-th and 95-th percentile of the difference in performance between two classifiers. Hence if 0 lies outside the interval, we may conclude that there is a significant difference in performance between the two classifiers, at the 5% confidence level, otherwise no such conclusion can be made. With this in mind, it follows from Table 6.3 that MSVM-Maj is significantly better than the LibSVM OVA approach on all datasets. Since this is a rather strong statement, we stress here that the LibSVM classifier has been properly trained to find the optimal parameter value, and this configuration has indeed been used in the bootstrap comparisons.

Table 6.3 also shows that MSVM-Maj performs significantly better than any other technique on the vehicle dataset, and performs better than LDA on the glass dataset. Moreover, the results in Table 6.3 show that no classifier performs significantly better than MSVM-Maj. This is an important result, since this signifies that the proposed method is indeed competitive with existing methods.

For a proper comparison, a number of remarks also need to be made about the training time of each classifier. First of all, the training time of MSVM-Maj depends heavily on the chosen parameters. Most importantly, a small value of λ will give a slower convergence of

Table 6.5 – Average training times for all classification methods at the optimal parameter configurations determined through cross validation. Training time is reported in seconds.

Datasets	MSVM-Maj	SVM-Maj OVA	LibSVM OVA	LDA	MLOG
iris	3.627	5.202	0.015	0.011	0.163
wine	9.176	379.852	10.944	0.012	0.259
glass	131.412	52.471	0.051	0.029	0.822
vehicle	811.866	12.807	12.651	0.058	6.415

the algorithm. This is expected since λ is a regularization parameter which controls for the size of $\mathbf{W}$. A small λ allows for more fine-tuning of $\mathbf{W}$, which makes the algorithm slower. The training time of MSVM-Maj also depends on the value of p. For a small value of p the algorithm is generally faster. This can be explained since a small value of p allows for a larger majorization step in the majorization of the Huber hinge errors (see also Appendix B).

To quantify the difference in the speed of each algorithm, the training time needed for each algorithm has been recorded for all classifiers. The results are presented in Table 6.5. For the MSVM-Maj and SVM-Maj OVA methods the training time was averaged over a number of parameter configurations that yielded optimal results. As mentioned, these parameter configurations are reported for MSVM-Maj in Table 6.4. The training time measurements were done using MATLAB R2012a (64-bit) on an Intel Pentium Dual CPU (1.86 GHz).

As can be seen from Table 6.5, the LDA method is the fastest method on all datasets. Of the SVM methods, the LibSVM OVA approach is generally fastest, although for the wine dataset MSVM-Maj is slightly faster. MSVM-Maj is faster than the SVM-Maj OVA method on the iris and wine datasets, although it must be noted that the SVM-Maj method may have suffered from the fact that the stopping criterion ϵ was chosen very small. If ϵ is chosen larger, the SVM-Maj method will in general achieve convergence faster. A full comparison of the speed of the SVM-Maj algorithm is provided by Groenen et al. (2008). The MSVM-Maj algorithm will also achieve faster convergence if ϵ is decreased.

CHAPTER

SEVEN

DISCUSSION & CONCLUSION

Here some of the results obtained will be discussed. First, some comments will be made on the parameter selection procedure, keeping in mind the practitioner who wants to use the proposed method in their research. Second, a brief overview is given of the ways in which nonlinearity can be included in MSVM-Maj. Some ideas which may be interesting for further research will also be presented. Finally, a conclusion of this thesis is given in the last section of this chapter.

7.1 Parameter Selection

In Chapter 6, a large grid search was used to find the optimal parameter configurations for each of the datasets. In practice, this approach may not be feasible due to the computation time necessary to try all configurations. The computations done on the datasets showed that the parameter configurations that yielded the optimal cross validation rates did exhibit strong variations in the parameters. The parameter configurations that yield the optimal cross validation rates are shown in Table 6.4. As can be seen, none of the parameters are the same for all optimal rates, and therefore none are irrelevant. Thus it is not possible to fix one of the parameters in advance and still generate the best possible classification function.

We can however, create a number of guidelines for reducing the size of the grid search necessary to find the optimal parameters. First of all, it is clear that the optimal parameters are most often found when λ is small. Although a larger value of the regularization parameter generally yields a faster convergence of the algorithm, this reduces the quality of the classifier significantly. This trade-off must be considered by a practitioner interested in using MSVM-Maj for classification. Furthermore, it is advised to start with a coarser grid with respect to the p values. In the grid search presented in the previous chapter a step size of 0.1 was used, but in general steps of 0.2 or 0.3 between $p = 1.0$ and $p = 2.0$ may be sufficient.

Finally, it may be possible to choose a larger stopping criterion ϵ first, and then decrease this once the optimal parameters are roughly estimated. If this is done it should be kept in mind that the MSVM-Maj solution becomes more sensitive to the starting position if ϵ is chosen large. This may be accounted for by choosing a larger number of folds in the cross validation procedure, or by simply repeating the experiments a sufficient number of times.

7.2 Nonlinearity

Two main methods exist to include nonlinearity in an SVM application. The first transforms the variables to piecewise polynomial functions using spline transformations. The second method is more commonly used in SVMs and is based on kernel functions.

7.2.1 Splines

Similar to the approach proposed in Groenen et al. (2007), spline transformations can be used to introduce nonlinearity in the predictor variables. A brief review of this approach is given here.

Following the approach of Groenen et al., the I-Spline transformations of Ramsay (1988) are considered. Splines are piecewise polynomial functions, defined by the degree of the pieces d, and the number of interior knots, s. For each variable vector $\mathbf{x}_j$, a spline basis $\mathbf{D}_j$ is constructed of size $n \times (d+s)$, together with a weights matrix $\mathbf{S}_j$ of size $(d+s) \times (K-1)$. Here $\mathbf{S}_j = [\mathbf{s}_j^{(1)'}, \ldots, \mathbf{s}_j^{(K-1)'}]$, where the weighting vectors $\mathbf{s}_j^{(k)}$ are the weighting vectors for each of the $K-1$ decision boundaries, for variable j. Then the spline transformation of a variable $\mathbf{x}_j$ for class boundary k is given by $\mathbf{D}_j\mathbf{s}_j^{(k)}$, where the weights vectors $\mathbf{s}_j^{(k)}$ are initially unknown. Hence the spline transformation of the attributes is achieved by replacing $\mathbf{X}$ by the matrix $\mathbf{D} = [\mathbf{D}_1\ \mathbf{D}_2\ \ldots\ \mathbf{D}_m]$, and $\mathbf{W}$ by $\mathbf{S} = [\mathbf{S}_1'\ \mathbf{S}_2'\ \ldots\ \mathbf{S}_m']'$.

Note that if the spline weights are all positive, the spline transformation is a monotone transformation. This can aid in the interpretation of the solution.

7.2.2 Kernels

Another more commonly used way to introduce nonlinearity in the SVM method is through the use of kernels, as seen in Chapter 3. Kernels in SVMs are generally implemented using the so-called *kernel trick*. Here each inner product between a pair of object vectors $\mathbf{x}_i'$, is replaced by a kernel function evaluation $K(\mathbf{x}_i, \mathbf{x}_j)$ (Cortes and Vapnik, 1995). The inner products between all pairs of objects are then collected in an $n \times n$ kernel matrix $\mathbf{K}$. The kernel function can be chosen freely, as long as the kernel matrix is positive definite.

In MSVM-Maj, no inner products between pairs of object vectors occur, so the kernel trick cannot be used. However it is still possible to include nonlinearity using kernel functions, similar to the approach proposed by Yip (2012) for binary SVMs. This approach consists of replacing $\mathbf{X}$ with the Cholesky decomposition of $\mathbf{K}$. The QR-decomposition of this Cholesky decomposition can then be used to predict the class labels of objects from the training sample.

7.3 Recommendations for Further Research

As was noted earlier, the large grid search necessary to find the optimal parameter configuration for the MSVM-Maj algorithm is inefficient, and in general not practical. Therefore, it would certainly be desirable to either decrease the training time of the algorithm significantly (in which case the time needed for the grid search is reduced), or to create a method with which the optimal parameter configuration can be found. The training time of the algorithm can for instance be reduced by rescaling the attributes in the dataset before the algorithm is run.

Methods for automated detection of the optimal parameter configuration for the MSVM-Maj algorithm may be difficult to formulate. Although this has not been formally proven, there is no additional convexity in the parameters p, κ, or λ. Hence any approach to automate the parameter search must be robust against possible local maxima.

Applications using the binary SVM have repeatedly shown that one of the main reasons for its success comes from the ability to include nonlinearity in the attributes in an efficient way. Therefore it is considered essential to ensure that nonlinearity can easily be incorporated in MSVM-Maj, to make the method competitive with existing techniques.

Finally, it may be interesting to see whether a GPU[1] implementation of MSVM-Maj can be made. The grid search for the optimal parameters constantly uses the same original dataset, which makes parallelization of the grid search relatively simple. Moreover, the GPU platform is explicitly designed to allow for parallel computing, and is especially good in performing matrix operations (Owens et al., 2008). Since the MSVM-Maj algorithm uses matrix operations at every iteration, it may be possible to achieve a significant reduction in computation time by using the GPU.

7.4 Conclusion

A new multiclass support vector machine has been introduced which is based on minimizing the total misclassification error of each object. By using a number of different weightings of the errors, it is possible to get a very flexible method which can be tuned to find the optimal classification boundary between classes of a given dataset.

In Chapter 3 it was noted that any new multiclass SVM method must be an improvement over existing methods in either interpretability, simplicity, performance or training time. We believe that with regards to the interpretability and simplicity the proposed method is easier to understand than existing multiclass methods. The introduced weighting mechanism is built up in such a way that the effect of each weighting can be easily interpreted.

It was demonstrated that the MSVM-Maj algorithm performs at least as good as existing methods, and performs significantly better than some methods at the 5% confidence level. A comparison of the training time of the MSVM-Maj algorithm showed that it is in general slower than existing methods for datasets with a large number of objects or a large number of classes. Ideas for decreasing this training time have been suggested.

We have focussed here only on the linear multiclass SVM classification. Possible strategies to include nonlinearity in the predictor variables have been suggested, and it is believed that if this feature is included, MSVM-Maj can become a proper alternative to existing multiclass classification methods. In general, we believe that the proposed multiclass Support Vector Machine approach provides a promising new way of analysing multiclass classification problems.

[1]Graphics Processing Unit.

BIBLIOGRAPHY

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

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

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

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

Crammer, K. and Singer, Y. (2001). Improved output coding for classification using continuous relaxation. *Advances in Neural Information Processing Systems*, pages 437–443.

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

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

Daumé, H. (2004). From zero to reproducing kernel hilbert spaces in twelve pages or less.

De Leeuw, J. (1977). Applications of convex analysis to multidimensional scaling. *Recent Developments in Statistics*, pages 133–146.

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

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

Dietterich, T. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Arxiv preprint cs/9501101*.

Frank, A. and Asuncion, A. (2010). UCI machine learning repository.

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

Groenen, P., Mathar, R., and Heiser, W. (1995). The majorization approach to multidimensional scaling for minkowski distances. *Journal of Classification*, 12(1):3–19.

Groenen, P., Nalbantov, G., and Bioch, J. (2007). Nonlinear support vector machines through iterative majorization and i-splines. *Advances in data analysis*, pages 149–161.

Groenen, P., Nalbantov, G., and Bioch, J. (2008). Svm-maj: A majorization approach to linear support vector machines with different hinge errors. *Advances in Data Analysis and Classification*, 2:17–43.

Hardy, G., Littlewood, J., and Polya, G. (1934). *Inequalities*. Cambridge University Press.

Horn, R. and Johnson, C. (1990). *Matrix analysis*. Cambridge Univ Pr.

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

Lee, Y., Lin, Y., and Wahba, G. (2004). Multicategory support vector machines. *Journal of the American Statistical Association*, 99(465):67–81.

Magnus, J. and Neudecker, H. (1988). *Matrix differential calculus with applications in statistics and econometrics*. Wiley.

Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., and Phillips, J. (2008). Gpu computing. *Proceedings of the IEEE*, 96(5):879–899.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods*, pages 185–208. MIT Press.

Ramsay, J. (1988). Monotone regression splines in action. *Statistical Science*, pages 425–441.

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

Rockafellar, R. (1997). *Convex analysis*, volume 28. Princeton Univ Pr.

Vapnik, V. (1998). Statistical learning theory. 1998.

Weston, J. and Watkins, C. (1998). Multi-class support vector machines. Technical report, Citeseer.

Yip, H. S. (2012). Predicting the political spectrum with support vector machine.

Zălinescu, C. (2002). *Convex analysis in general vector spaces*. World Scientific Pub Co Inc.

APPENDIX

A

THEOREMS & PROOFS

Some additional theorems and proofs that are used in the main text are omitted there for conciseness. They are given below.

A.1 Theorems from Rockafellar (1997)

In the proof of the convexity of $L_{\text{MSVM}}(\mathbf{V})$ a number of theorems from Rockafellar (1997) are used. These theorems are given here, however the proofs are omitted. Theorem numbers in this section correspond to those in Rockafellar (1997).

Theorem 4.1. *Let f be a function from $\mathbb{R}^n$ to $(-\infty, +\infty]$. Then f is convex on $\mathbb{R}^n$ if and only if*

$$f((1-\gamma)\boldsymbol{x} + \gamma\boldsymbol{y}) \leq (1-\gamma)f(\boldsymbol{x}) + \gamma f(\boldsymbol{y}), \quad 0 < \gamma < 1,$$

for every $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{R}^n$.

Note that a function is called *strictly convex* if the inequality sign in the above expression can be changed to a $<$ sign. We further remark from Rockafellar (1997) page 24, that a convex function f is called *proper* if $f(x) < +\infty$ for at least one x and $f(x) > -\infty$ for every x.

Theorem 5.1. *Let f be a convex function from $\mathbb{R}^n$ to $(-\infty, +\infty]$, and let ϕ be a convex function from $\mathbb{R}$ to $(-\infty, +\infty]$ which is non-decreasing. Then $h(x) = \phi(f(x))$ is convex on $\mathbb{R}^n$ (where one sets $\phi(+\infty) = +\infty$).*

One important example of this theorem is $h(x) = f(x)^p$ which is convex for $p > 1$ if f is convex and non-negative.

Theorem 5.2. *If f_1 and f_2 are proper convex functions on $\mathbb{R}^n$, then $f_1 + f_2$ is convex.*

As a consequence of this theorem any linear combination $\lambda_1 f_1 + \cdots + \lambda_m f_m$ of proper convex functions with non-negative weights is convex.

A.2 Convexity Proofs

The following theorems are used in the proof of Theorem 1.

Theorem 2. *The L_p norm is a convex function.*

Proof. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ with elements x_i and y_i for $i = 1, \dots, n$, respectively, and let $\gamma \in (0, 1)$. Then

$$\begin{aligned}
||\gamma\mathbf{x} + (1-\gamma)\mathbf{y}||_p &\leq ||\gamma\mathbf{x}||_p + ||(1-\gamma)\mathbf{y}||_p \\
&= \left(\sum_{i=1}^{n} |\gamma x_i|^p\right)^{1/p} + \left(\sum_{i=1}^{n} |(1-\gamma)y_i|^p\right)^{1/p} \\
&= \left(\gamma^p \sum_{i=1}^{n} |x_i|^p\right)^{1/p} + \left((1-\gamma)^p \sum_{i=1}^{n} |y_i|^p\right)^{1/p} \\
&= \gamma||\mathbf{x}||_p + (1-\gamma)||\mathbf{y}||_p.
\end{aligned}$$

Here, the inequality follows from Minkowski's Inequality and the third line follows because γ is non-negative. □

Theorem 3. *Let $\boldsymbol{A} \in \mathbb{R}^{(m+1)\times(K-1)}$ and $\gamma \in (0, 1)$. Also, let $\boldsymbol{P} \in \mathbb{R}^{(m+1)\times(m+1)}$ be a symmetric positive semi-definite matrix. The function $f : \mathbb{R}^{(m+1)\times(K-1)} \to \mathbb{R}$ defined as*

$$f(\boldsymbol{A}) = \text{tr } \boldsymbol{A}'\boldsymbol{P}\boldsymbol{A}$$

is a convex function.

Proof. First let $\mathbf{B} \in \mathbb{R}^{(m+1)\times(K-1)}$. Then by Rockafellar Theorem 4.1 it needs to be shown that

$$f(\gamma\mathbf{A} + (1-\gamma)\mathbf{B}) \leq \gamma f(\mathbf{A}) + (1-\gamma) f(\mathbf{B}).$$

The proof starts by rewriting the left hand side of this inequality as follows

$$\begin{aligned}
f(\gamma\mathbf{A} + (1-\gamma)\mathbf{B}) &= \text{ tr } (\gamma\mathbf{A} + (1-\gamma)\mathbf{B})'\mathbf{P}(\gamma\mathbf{A} + (1-\gamma)\mathbf{B}) \\
&= \text{ tr } \Big(\gamma^2\mathbf{A}'\mathbf{P}\mathbf{A} + \gamma(1-\gamma)\mathbf{B}'\mathbf{P}\mathbf{A} + \gamma(1-\gamma)\mathbf{A}'\mathbf{P}\mathbf{B} \\
&\quad + (1-\gamma)^2\mathbf{B}'\mathbf{P}\mathbf{B}\Big) \\
&= \gamma^2 \text{ tr } \mathbf{A}'\mathbf{P}\mathbf{A} + \gamma(1-\gamma) \text{ tr } \mathbf{B}'\mathbf{P}\mathbf{A} + \gamma(1-\gamma) \text{ tr } \mathbf{A}'\mathbf{P}\mathbf{B} \\
&\quad + (1-\gamma)^2 \text{ tr } \mathbf{B}'\mathbf{P}\mathbf{B}.
\end{aligned}$$

Using the fact that tr $\mathbf{B}'\mathbf{P}\mathbf{A} =$ tr $\mathbf{A}'\mathbf{P}\mathbf{B}$ and rewriting gives

$$f(\gamma\mathbf{A} + (1-\gamma)\mathbf{B}) = \gamma^2 \text{ tr } (\mathbf{A}-\mathbf{B})'\mathbf{P}(\mathbf{A}-\mathbf{B}) + (1-2\gamma) \text{ tr } \mathbf{B}'\mathbf{P}\mathbf{B} + 2\gamma \text{ tr } \mathbf{A}'\mathbf{P}\mathbf{B}.$$

Since $\gamma^2 < \gamma$, $\forall\gamma \in (0, 1)$, and since $\mathbf{P}$ is a positive *semi-definite* matrix it follows that

$$\begin{aligned}
f(\gamma\mathbf{A} + (1-\gamma)\mathbf{B}) &\leq \gamma \text{ tr } (\mathbf{A}-\mathbf{B})'\mathbf{P}(\mathbf{A}-\mathbf{B}) + (1-2\gamma) \text{ tr } \mathbf{B}'\mathbf{P}\mathbf{B} + 2\gamma \text{ tr } \mathbf{A}'\mathbf{P}\mathbf{B} \\
&= \gamma \text{ tr } \mathbf{A}'\mathbf{P}\mathbf{A} + \gamma \text{ tr } \mathbf{B}'\mathbf{P}\mathbf{B} + \text{ tr } \mathbf{B}'\mathbf{P}\mathbf{B} - 2\gamma \text{ tr } \mathbf{B}'\mathbf{P}\mathbf{B} \\
&= \gamma f(\mathbf{A}) + (1-\gamma) f(\mathbf{B}).
\end{aligned}$$

□

Notice that if $\mathbf{P}$ is a positive *definite* matrix, the function in Theorem 3 is strictly convex.

A.3 Maximum Eigenvalue Inequality

Theorem 4. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a symmetric matrix and let λ_i be the eigenvalues of $\boldsymbol{A}$ with corresponding eigenvectors $\boldsymbol{v}_i$, for $i = 1, \ldots, n$. Let $\ell \geq \lambda_i$, $\forall i$ and let $\boldsymbol{I}$ be the $n \times n$ identity matrix, then*

$$\boldsymbol{x}'(\boldsymbol{A} - \ell\boldsymbol{I})\boldsymbol{x} \leq 0,$$

for all $\boldsymbol{x} \in \mathbb{R}^n$.

Proof. The characteristic polynomial for $\mathbf{A}$ is given by

$$\det(\mathbf{A} - x\mathbf{I}) = (x - \lambda_1)(x - \lambda_2)\cdots(x - \lambda_n) = 0$$

Now let $y = x - \ell$ then

$$\det(\mathbf{A} - (y+\ell)\mathbf{I}) = \det((\mathbf{A} - \ell\mathbf{I}) - y\mathbf{I}) = (y - (\lambda_1 - \ell))(y - (\lambda_2 - \ell))\cdots(y - (\lambda_n - \ell)) = 0$$

Hence the eigenvalues of $\mathbf{A} - \ell\mathbf{I}$ are $\lambda_i - \ell \leq 0, \forall i$. Then this matrix is negative semidefinite[1], and $\mathbf{x}'(\mathbf{A} - \ell\mathbf{I})\mathbf{x} \leq 0$. □

An example of this theorem which is used in the main text follows.

Example: Let $\mathbf{A}$ be a symmetric $n \times n$ square matrix with eigenvalues λ_i and let $\ell \geq \lambda_i, \forall i$. Then let $\mathbf{x} = \mathbf{f} - \mathbf{g}$ be a $n \times 1$ vector. Applying the above theorem yields

$$(\mathbf{f} - \mathbf{g})' (\mathbf{A} - \ell\mathbf{I}) (\mathbf{f} - \mathbf{g}) \leq 0.$$

Which can be rewritten to

$$\begin{aligned} \mathbf{f}'\mathbf{A}\mathbf{f} &\leq \ell\mathbf{f}'\mathbf{f} + \mathbf{f}' (\mathbf{A} - \ell\mathbf{I}) \mathbf{g} + \mathbf{g}' (\mathbf{A} - \ell\mathbf{I}) \mathbf{f} - \mathbf{g}' (\mathbf{A} - \ell\mathbf{I}) \mathbf{g}, \\ &\leq \ell\mathbf{f}'\mathbf{f} + 2\mathbf{f}' (\mathbf{A} - \ell\mathbf{I}) \mathbf{g} - \mathbf{g}' (\mathbf{A} - \ell\mathbf{I}) \mathbf{g}, \end{aligned}$$

where in the last line it has been used that $\mathbf{A}$ is symmetric.

If in the above example we substitute $\mathbf{f} = \mathbf{V}'\mathbf{z}_i$, $\mathbf{A} = \boldsymbol{\delta}_{kj}\boldsymbol{\delta}'_{kj}$ and $\mathbf{g} = \overline{\mathbf{V}}'\mathbf{z}_i$ with $\ell = 1$, the inequality in (5.15) is found.

[1]See e.g. Horn and Johnson (1990), Corollary 7.2.1

APPENDIX

B

HUBER HINGE MAJORIZATION

To keep the main derivation of the majorization function concise, the majorization of $h^p(q)$ will be presented here. For reference, the definition of $h^p(q)$ is repeated first

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

Let the majorization function be given by $g(q, \overline{q}) = aq^2 - 2bq + c$. The majorization conditions for the IM algorithm can then be stated as

1. $h^p(\overline{q}) = g(\overline{q}, \overline{q})$,
2. $[h^p(\overline{q})]' = g'(\overline{q}, \overline{q})$,
3. $h^p(q) \leq g(q, \overline{q})$.

Since the Huber function is a piecewise defined function, the derivation can be separated into three parts. Notice that the value of $\overline{q}$ determines in which segment the majorization function touches, this value therefore determines the coefficients of the quadratic majorization.

B.1 Huber majorization for $\overline{q} < -\kappa$

If $\overline{q} < -\kappa$ the following conditions must hold

$$\begin{aligned} \left(1 - \overline{q} - \frac{\kappa+1}{2}\right)^p &= a\overline{q}^2 - 2b\overline{q} + c \\ -p\left(1 - \overline{q} - \frac{\kappa+1}{2}\right)^{p-1} &= 2a\overline{q} - 2b \end{aligned}$$

To shorten the notation we define a dummy variable

$$\phi = 1 - \overline{q} - \frac{\kappa+1}{2}. \tag{B.1}$$

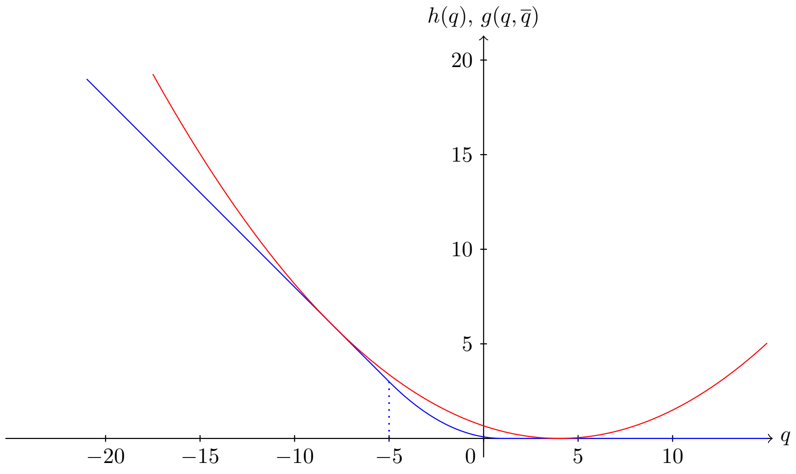


Figure B.1 – Example of the majorization function derived in Section B.1. The Huber function is plotted in blue with $p = 1$ and $\kappa = 5$. The majorization function is constructed with the supporting point at $\overline{q} = -8$, and is plotted in red.

Solving the above equations for b and c gives

$$b = a\overline{q} + \tfrac{1}{2}p\phi^{p-1},$$
$$c = a\overline{q}^2 + p\overline{q}\phi^{p-1} + \phi^p.$$

It is desirable that the majorization function provides an as large as possible majorization step. This would imply that the majorization function $g(q, \overline{q})$ has the minimum value 0. This condition provides us with the equation needed to find a. Let q_{min} denote the value of q where $g(q, \overline{q})$ has its minimum, then $g'(q_{min}, \overline{q}) = 0$ implies $q_{min} = b/a$ and $g(q_{min}, \overline{q}) = 0$ yields

$$c = \frac{b^2}{a}.$$

Solving this for a gives

$$a = \tfrac{1}{4}p^2\phi^{p-2}.$$

Figure B.1 shows an example of a majorization which touches at $q = \overline{q}$ and has its minimum at $q > 1$. This majorization is only possible whenever $q_{min} \geq 1$, due to the value of $h^p(q)$ and the third majorization condition. Solving $q_{min} \geq 1$ for $\overline{q}$ reveals that the above value for a holds if and only if

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

Thus whenever $\overline{q} \in ((p + \kappa - 1)/(p - 2), -\kappa]$ a different value for a is needed. The expression for a derived for $\overline{q} \in (-\kappa, 1]$ in the next section, can then be used. Notice that the above expression for a is valid only when $p \neq 2$. A separate derivation for $p = 2$ is necessary and will follow.

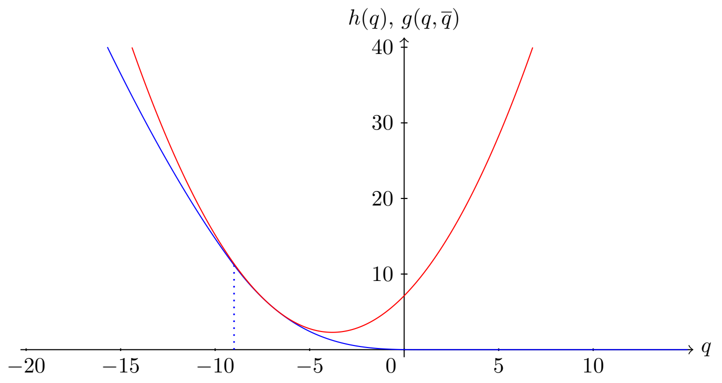


Figure B.2 – Example of the majorization function derived in Section B.2. The Huber function is plotted in blue with $p = 1.5$ and $\kappa = 9$. The majorization function is constructed with the supporting point at $\overline{q} = -7$, and is plotted in red.

B.2 Huber majorization for $\overline{q} \in (-\kappa, 1]$

For $\overline{q} \in (-\kappa, 1]$ the majorization conditions stated above imply

$$\begin{aligned} (2(\kappa+1))^{-p}(1-\overline{q})^{2p} &= a\overline{q}^2 - 2b\overline{q} + c, \\ -2p(2(\kappa+1))^{-p}(1-\overline{q})^{2p-1} &= 2a\overline{q} - 2b. \end{aligned}$$

Again, we introduce a dummy variable

$$\psi = \frac{1-\overline{q}}{\sqrt{2(\kappa+1))}}, \tag{B.2}$$

and we solve the system of equations for b and c to arrive at

$$\begin{aligned} b &= a\overline{q} + p\frac{\psi^{2p}}{1-\overline{q}}, \\ c &= a\overline{q}^2 + \psi^{2p}\left(1 + \frac{2p\overline{q}}{1-\overline{q}}\right). \end{aligned}$$

A value of a can be found by realizing that the second derivative of $h^p(q)$ is bounded on the interval $q \in (-\kappa, 1]$. The maximum value on this interval is

$$[h^p(-\kappa)]'' = \tfrac{1}{2}p(2p-1)\left(\frac{\kappa+1}{2}\right)^{p-2}.$$

By equating this to $g''(q,\overline{q}) = 2a$ we find a value for a which ensures that $h^p(q) \leq g(q,\overline{q})$ on the interval $q \in (-\kappa, 1]$.

$$a = \tfrac{1}{4}p(2p-1)\left(\frac{\kappa+1}{2}\right)^{p-2} \tag{B.3}$$

Notice that we use an upper bound for a, which means that for some values of p and κ a more efficient majorization function exists. However a general expression for which $g(q,\overline{q})$ does not intersect the Huber function, has not been found. Figure B.2 shows a graphical illustration of this majorization.

B.3 Huber majorization for $\overline{q} > 1$

When $\overline{q} > 1$ the majorization derived above for $\overline{q} \leq -\kappa$ is useful. Since we've shown that for all values of $\overline{q} \leq (p+\kappa-1)/(p-2)$ a majorization can be found for which $g(q_{min}, \overline{q}) = 0$, it is also possible that whenever $\overline{q} > 1$ a majorization function can be constructed that touches $h^p(q)$ at a second supporting point, $\chi < -\kappa$. Then as a function of χ the coefficients a, b and c are

$$\begin{aligned}
a &= \tfrac{1}{4}p^2\left(1-\chi-\frac{\kappa+1}{2}\right)^{p-2} \\
b &= a\chi + \tfrac{1}{2}p\left(1-\chi-\frac{\kappa+1}{2}\right)^{p-1} \\
c &= a\chi^2 + \left(1-\chi-\frac{\kappa+1}{2}\right)^{p} + p\chi\left(1-\chi-\frac{\kappa+1}{2}\right)^{p-1}
\end{aligned}$$

Since $g'(\overline{q}, \overline{q}) = 0$ we know that $b = a\overline{q}$, which allows us to find the relation between χ and $\overline{q}$

$$\chi = \frac{p\overline{q}+\kappa-1}{p-2}$$

Notice that χ is a dummy variable similar to ϕ and ψ. Plugging the above into the expressions for a, b and c gives

$$\begin{aligned}
a &= \tfrac{1}{4}p^2\left(\frac{p\phi}{p-2}\right)^{p-2} \\
b &= a\chi + \tfrac{1}{2}p\left(\frac{p\phi}{p-2}\right)^{p-1} \\
c &= a\chi^2 + p\chi\left(\frac{p\phi}{p-2}\right)^{p-1} + \left(\frac{p\phi}{p-2}\right)^{p}
\end{aligned}$$

Notice that these expressions also do not hold when $p = 2$. A second derivation is needed for this special case.

B.4 Huber majorization for $p = 2$

Some of the above expressions for the coefficients break down when $p = 2$. However the upper bound of a derived for $\overline{q} \in (-\kappa, 1]$, is still valid. Plugging $p = 2$ into (B.3) gives $a = 3/2$, which can be used throughout the entire range of $\overline{q}$. With this value of a the expressions derived for b and c for $\overline{q} < -\kappa$ remain equal to

$$\begin{aligned}
b &= \tfrac{3}{2}\overline{q} + \tfrac{1}{2}p\phi^{p-1}, \\
c &= \tfrac{3}{2}\overline{q}^2 + p\overline{q}\phi^{p-1} + \phi^p.
\end{aligned}$$

For $\overline{q} \in (-\kappa, 1]$ the parameters derived above are also still valid, hence

$$\begin{aligned}
b &= \tfrac{3}{2}\overline{q} + \frac{p\psi^{2p}}{1-\overline{q}}, \\
c &= \tfrac{3}{2}\overline{q}^2 + \psi^{2p}\left(1+\frac{2p\overline{q}}{1-\overline{q}}\right).
\end{aligned}$$

For $\overline{q} > 1$ we simply use the majorization conditions and $a = 3/2$ to arrive at

$$\begin{aligned} b &= \tfrac{3}{2}\overline{q}, \\ c &= \tfrac{3}{2}\overline{q}^2. \end{aligned}$$

To summarize, the values for a, b and c are

$$a = \begin{cases} \frac{1}{4}p^2\phi^{p-2} & \text{if } \overline{q} \leq \dfrac{p+\kappa-1}{p-2} \text{ and } p \neq 2 \\ \frac{1}{4}p(2p-1)\left(\dfrac{\kappa+1}{2}\right)^{p-2} & \text{if } \overline{q} \in \left(\dfrac{p+\kappa-1}{p-2}, 1\right] \text{ and } p \neq 2 \\ \frac{1}{4}p^2\left(\dfrac{p\phi}{p-2}\right)^{p-2} & \text{if } \overline{q} > 1 \text{ and } p \neq 2 \\ \frac{3}{2} & \text{if } p = 2 \end{cases}$$

$$b = \begin{cases} a\overline{q} + \frac{1}{2}p\phi^{p-1} & \text{if } \overline{q} \leq -\kappa \\ a\overline{q} + \dfrac{p\psi^{2p}}{1-\overline{q}} & \text{if } \overline{q} \in (-\kappa, 1] \\ a\chi + \frac{1}{2}p\left(\dfrac{p\phi}{p-2}\right)^{p-1} & \text{if } \overline{q} > 1 \text{ and } p \neq 2 \\ \frac{3}{2}\overline{q} & \text{if } \overline{q} > 1 \text{ and } p = 2 \end{cases}$$

$$c = \begin{cases} a\overline{q}^2 + \phi^p + p\overline{q}\phi^{p-1} & \text{if } \overline{q} \leq -\kappa \\ a\overline{q}^2 + \psi^{2p}\left(1 + \dfrac{2p\overline{q}}{1-\overline{q}}\right) & \text{if } \overline{q} \in (-\kappa, 1] \\ a\chi^2 + p\chi\left(\dfrac{p\phi}{p-2}\right)^{p-1} + \left(\dfrac{p\phi}{p-2}\right)^{p} & \text{if } \overline{q} > 1 \text{ and } p \neq 2 \\ \frac{3}{2}\overline{q}^2 & \text{if } \overline{q} > 1 \text{ and } p = 2 \end{cases}$$

where the dummy variables are defined as

$$\begin{aligned} \phi &= 1 - \overline{q} - \frac{\kappa+1}{2}, \\ \psi &= \frac{1-\overline{q}}{\sqrt{2(\kappa+1)}}, \\ \chi &= \frac{p\overline{q}+\kappa-1}{p-2}. \end{aligned}$$