Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

SVM imbalance correction by conformal kernel transformations

Master's thesis presented to apply for the Master
in Computer Engineering and Telecommunications degree

By
Álvaro Barbero Jiménez
under the direction of
José Ramón Dorronsoro Ibero

Madrid, June 6, 2008

Contents

Abstract

Class imbalance is a common problem when dealing with real life classification problems. Support Vector Machines, a well-known family of learning algorithms is prone to produce a skewed classification function when the training patterns of one class are outnumbered by the quantity of samples of the other classes. In this work we study methods available in the literature to alleviate this skewness, concretely those relating the technique of using conformal transformations of the kernel function to adapt the feature space in a data-dependent way. After that, we propose two new methods (MDT and TSVT) as an addition to the previous ones, and compare the obtained results between both some of the reviewed and proposed methods.

Chapter 1

Introduction

Quite often in the area of machine learning we find problems difficult to solve due to the great differences in the frequency of appearance of the classes of objects we would like to classify. This phenomenon is known as class imbalance, and is rather natural in a wide range of classification problems, e.g. in medical diagnosis problems the number of patients suffering the disease is smaller than the number of healthy ones. Or, in credit card fraud detection, legal transactions are much more common than those from stolen cards.

Unfortunately, usually the class with small probability of appearance (the minority class) is the one we are more eager to classify correctly, as a good accuracy can be obtained for the majority class by just predicting every object as belonging to that class. Even when using more complex machine learning methods it is not uncommon to only obtain slight improvements in the classification accuracy. For example, if a neural network is used in order to minimize the squared mean error in the predictions, it is highly probable – again – for the network to classify almost every pattern as belonging to the majority class, hence providing little improvement over the simple classifier stated before. This is due to the fact that the low number of available patterns of the minority class has a small influence in the error criterion used. Other criterions to take into account this imbalance may be used, but they are not always applicable over any learning algorithm.

Among the well-known machine learning algorithms the Support Vector Machines (SVMs) [1, 2] have shown to have good generalization abilities in classification problems. However, they are not free from the difficulties created by class imbalance, although a wide number of proposals have appeared in the literature to try to overcome this problem. Some of the most promising are those known as conformal transformation methods. These methods pursue the finding of an adapted high dimensional "feature space" in which the SVM is trained. The adaption is made through one or more transformations of a basic kernel function, aimed at obtaining a better classification accuracy. Despite the method was originally proposed to obtain a data-dependent kernel function, some authors have modified it to deal with imbalanced datasets.

In this work we shall present the results of our studies in the use of conformal transformation methods to improve classification accuracy with SVMs over imbalanced datasets. The work is divided as follows. In chapter 2 we will present a review of the theory of SVMs and CH-SVMs to provide a basis for the following chapters (readers with knowledge in this area may skip some sections). In chapter 3 we show the effect of imbalance on SVMs and some basic methods to overcome it, as well as an introduction to conformal transformations. In chapter 4 we review the proposed conformal transformations methods available in the literature. In chapter 5 we explain our proposals of conformal transformation methods. We will show the experimental results of these methods in chapter 6. Finally we will conclude in chapter 7 that although the theory under these methods seems to provide a powerful way to obtain a data-dependent kernel function, more work would be required to obtain practical improvements. Additionally we include at the end an appendix presenting other related works done during the Master course, and another one detailing the notation used in this work.

Chapter 2

Review of SVMs

2.1 Support Vector Machines for classification

2.1.1 SVMs as an improvement over the perceptron algorithm

Support Vector Machines (SVMs) are a well-known algorithm in the machine learning theory [1, 2], which can be thought as an improvement of the classic perceptron algorithm. The original perceptron algorithm proposed by Rosenblatt [3, 4] pursues finding a hyperplane such that, given data belonging to two different classes, it splits the input space in two regions that only contain data points of one of the classes. In other words, the perceptron looks for any separating hyperplane. An illustrative example is shown in figure 2.1.1.

In a formal way, a valid perceptron solution must satisfy the restrictions

$$(w \cdot x^i + b)y_i \geq 0 \, \forall i, \tag{2.1.1}$$

where w stands for the normal vector of the hyperplane, b for its bias, x^i for each of the training data points belonging to a set X, y_i for their respective class labels (coded as $\{-1, +1\}$), and $\cdot$ denotes the dot product between two vectors. It can be easily seen that when the classification made by the hyperplane $(w \cdot x^0 + b)$ for a data point x^0 gives the same sign as the point's label y^0, the obtained value for $(w \cdot x^0 + b)y^0$ must be ≥ 0. So, the restrictions in (2.1.1) ensure that all the training points are correctly classified by the perceptron.

On the other hand, a SVM tries to find the "best" separating hyperplane. By "best" we mean that it looks for an hyperplane that is placed as far as possible from the data points. Intuitively, one can see that this kind of hyperplane may have some better generalization abilities because the training data could be noisy. If that is the case we will make wrong classifications with unseen data if the hyperplane is too close to the training points, as points of the same class would easily appear on both sides of the hyperplane. An illustrative example is shown in figure 2.1.2.

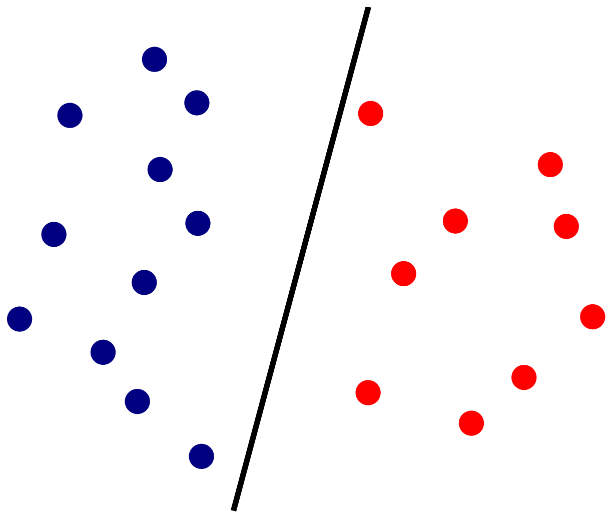

Figure 2.1.1: An example of a separating hyperplane obtained by the Rosenblatt's perceptron algorithm.

This intuitive idea can be formally expressed in the following way: we define a margin m as the distance from the hyperplane to the nearest data point

$$m = \min_i \frac{(w \cdot x^i + b)y_i}{\|w\|}. \tag{2.1.2}$$

We would like to maximize this margin while, at the same time, we obtain a separating hyperplane. This aim can be formally expressed as a constrained optimization problem in the form

$$\begin{aligned} \max_{w,b} \quad & m \\ \text{s.t.} \quad & \tfrac{(wx^i+b)y_i}{\|w\|} \geq m \quad \forall i, \end{aligned} \tag{2.1.3}$$

which is similar to the perceptron constraints in (2.1.1) but with the aim of maximizing m. A strong statistical theory [2] confirms that this approach is able to reduce an upper bound on the generalization error of the classifier. But usually an alternative formulation of the problem is used, which can be shown to be equivalent [1]:

$$\begin{aligned} \min_{w,b} \quad & \|w\|^2 \\ \text{s.t.} \quad & (w \cdot x^i + b)y_i \geq 1 \quad \forall i. \end{aligned} \tag{2.1.4}$$

Here we attempt to minimize the norm of the weight vector ($\|w\|^2$) while maintaining a margin of at least 1. Note that the norm has been dropped out of the margin's expression in this alternative formulation, so it no longer stands for a "strict" geometrical distance.

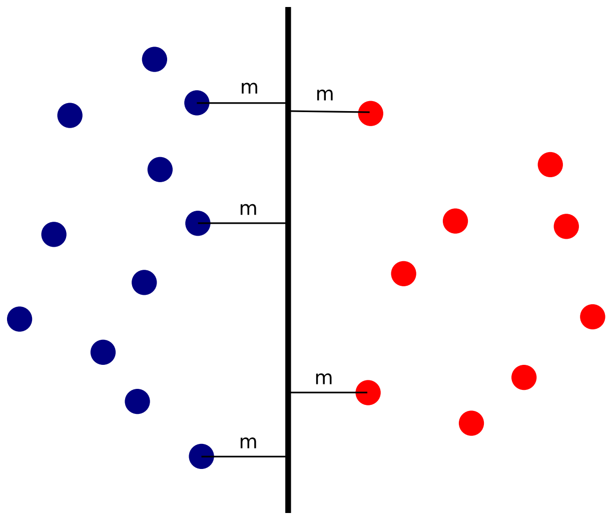

Figure 2.1.2: An example of a separating hyperplane obtained by a SVM. m denotes the margin of the SVM.

The minimization problem proposed in (2.1.4) belongs to the family of the quadratic programming problems, and therefore can be solved using any of the available quadratic programming packages. Alas, solving this problem directly is costly and computational problems may arise, so generally another point of view is used.

2.1.2 The Karush-Kuhn-Tucker conditions

When dealing with constrained optimization problems, if only equality constraints appear, the Lagrange multipliers can be used to easily find the optimum. But when inequality constraints are present (like in (2.1.4)), a generalization of this method must be used instead: the Karush-Kuhn-Tucker conditions (KKT) [5]. We will first explain the general scenario of this technique, and after that we will show how they can be useful for solving the SVM problem.

A general optimization problem can be described as

$$\begin{array}{ll} \min\limits_{W} & F(W) \\ \text{s.t.} & \left\{ \begin{array}{ll} g_i(W) \leq 0 & (i = 1, ..., m) \\ h_j(W) = 0 & (j = 1, ..., l) \end{array} \right\} \end{array}, \tag{2.1.5}$$

where F is the function to optimize, W is a vector containing its arguments, g_i are the inequality restrictions imposed over the values of W, and h_j are the equality restrictions. F and every g_i and h_i are all functions $\Re^N \longrightarrow \Re$. We will focus on the particular case where the optimization problem is convex: that is, when F is a convex function and the restrictions g_i and h_j are affine functions. If this is the case and if certain conditions are met, then the optimum of the function is obtained. But before detailing such conditions, we first define the Lagrangian of an optimization problem of this kind, which will prove to be a useful expression. The Lagrangian $L(W)$ has the form

$$L(W) = F(W) + \sum_{i=1}^{m} \alpha_i g_i(W) + \sum_{j=1}^{l} \beta_j h_j(W), \tag{2.1.6}$$

where α_i, β_j stand for the Lagrange multipliers. Now, a parameter vector W^* would be the optimum of F under the given restrictions if

- F is a convex function.
- All the restrictions g_i, h_j $\forall$ i,j are affine functions.
- W^* is a feasible choice (satisfies the restrictions g_i, h_j $\forall$ i,j).
- The gradient of the Lagrangian (2.1.6) in W^* equals the zero vector $\vec{0}$, that is: $\nabla_W L(W^*) = \nabla_W F(W^*) + \sum_{i=1}^{m} \alpha_i \nabla_W g_i(W^*) + \sum_{j=1}^{l} \beta_j \nabla_W h_j(W^*) = \vec{0}$.
- Lagrange multipliers meet $\alpha_i \geq 0$, β_j free, $\forall i = 1, ..., m$ and $\forall j = 1, ..., l$.
- $\alpha_i g_i(W^*) = 0 \; \forall \; i$ (KKT conditions).

In this way, these conditions provide a sufficient condition for a parameter vector W^* to be optimal. By applying this method to the SVM optimization problem stated before in (2.1.4), we could find the optimal parameters $W^* = (w^*, b^*)$. In that problem the function F to optimize is clearly convex, as well as the inequality restrictions g_i are affine (there are no equality restrictions). So, the Lagrangian of the SVM problem would be

$$L_p = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \alpha_i (1 - y_i(x^i \cdot w + b)). \tag{2.1.7}$$

We shall call this expression L_p because, as we will see later, is the Lagrangian of the primal formulation of the SVM. Observe that the derivatives of this Lagrangian respect to the parameters w and b take the form

$$\nabla_w L_p = w - \textstyle\sum_{i=1}^{N} \alpha_i y_i x^i, \tag{2.1.8}$$

$$\frac{\partial L_p}{\partial b} = - \textstyle\sum_{i=1}^{N} \alpha_i y_i. \tag{2.1.9}$$

In this way, an optimal (w^*, b^*) choice for the SVM will be obtained if

- (w^*, b^*) is a feasible choice, that is, $(w \cdot x^i + b)y_i \geq 1 \, \forall i$,
- $\exists$ Lagrange multipliers $\alpha_i \geq 0$, $\forall i = 1, ..., m$, in which the remaining restrictions are based,
- $w = \sum_{i=1}^{N} \alpha_i y_i x^i$,
- $0 = \sum_{i=1}^{N} \alpha_i y_i$,
- $\alpha_i(1 - y_i(w \cdot x^i + b)) = 0 \, \forall i$ (KKT conditions).

Hence, we have cast the original SVM problem (2.1.4), which was a quadratic optimization problem, into a constraint satisfaction problem. However, this problem is also difficult to solve since these restrictions do not give a closed expression for the optimum. Nevertheless, they provide a good procedure to check if the optimum has been reached, which is used as a stopping criterion in some well known SVM training algorithms, like SVMLight [6]. Furthermore they will provide a way to cast once again the problem into its called "dual formulation", which will provide an efficient method for solving the problem.

2.1.3 Dual formulation of the SVM

We can define another problem based in an original quadratic programming problem (which we shall call "primal problem" from now on) in the form

$$\begin{array}{ll} \max\limits_{\alpha,\beta} & \Theta(\alpha, \beta) = \inf_W L(W, \alpha, \beta) \\ \text{s.t.} & \qquad \alpha_i \geq 0 \quad \forall i. \end{array} \tag{2.1.10}$$

In this new problem, we try to find a maximum of the function Θ, which only depends on the Lagrange multipliers α and β. Θ is, in fact, the infimum of the Lagrangian with respect to the parameter vector W. This particular problem, which we shall call "dual problem", meets some special qualities that are useful in the task of solving the primal problem. The first one, named the "Weak duality theorem", is as follows.

Theorem 1 *(Weak duality theorem) Given a primal optimization problem in the form* (2.1.5) *and a dual defined as in* (2.1.10)*, if $\hat{W}$ is a feasible solution of the primal problem and ($\hat{\alpha}$,$\hat{\beta}$) are a feasible solution of the dual problem, then the following relationship holds:*

$$\begin{aligned} \Theta(\hat{\alpha}, \hat{\beta}) &= \inf_W L(W, \hat{\alpha}, \hat{\beta}) \leq L(\hat{W}, \hat{\alpha}, \hat{\beta}) \\ &= F(\hat{W}) + \hat{\alpha} g(\hat{W}) + \hat{\beta} h(\hat{W}) \leq F(\hat{W}) \end{aligned} \tag{2.1.11}$$

Proof Since $\Theta(\hat{\alpha},\hat{\beta})$ is the infimum of $L(W,\hat{\alpha},\hat{\beta})$, inequality $\Theta(\hat{\alpha},\hat{\beta}) \leq L(W,\hat{\alpha},\hat{\beta})$ must hold for any W. On the other hand, since $\hat{W}$ is a feasible solution of the primal problem, it must satisfy its constrains, so $g_i(\hat{W}) \leq 0$ and $h_j(\hat{W}) = 0\,\forall i,j$. Then, as ($\hat{\alpha}$,$\hat{\beta}$) are a feasible solution of the dual problem, the restrictions $\hat{\alpha} \geq 0\,\forall i$ are meet. In this way, the term $\hat{\alpha}g(\hat{W})$ can not be positive, and the term $\hat{\beta}h(\hat{W}) = 0$, and so the theorem is proved. □

This result is useful in the sense that it gives a relationship between the objective function of the primal and dual problems. Observe the following corollary:

Corollary 1 *The value of the dual is upper bounded by the value of the primal:*

$$\sup_{\hat{\alpha},\hat{\beta}} \Theta(\hat{\alpha},\hat{\beta}) \leq \inf_{\hat{W}} F(\hat{W}). \tag{2.1.12}$$

So, by maximizing the dual problem one cannot surpass the value obtained by minimizing the primal one. From this fact we can state another corollary:

Corollary 2 *If some values (W^*,α^*,β^*) are such that W^* is a feasible solution of the primal problem and (α^*,β^*) are a feasible solution of the dual problem, and $F(W^*) = \Theta(\alpha^*,\beta^*)$, then (W^*,α^*,β^*) is the optimum solution for both problems.*

Proof As by corollary 1 the value of the dual is bounded by the primal, if we have obtained the same value in both problems then it is impossible to increment the value of the dual, nor to decrease the value of the primal. And so both the dual maximization problem and the primal minimization one are solved. □

However, this is not enough to be able to solve one of the problems and obtain the solution in the other. Observe that, for any feasible solution, the values of the objective functions are related via the inequality 2.1.11. In this way, both problems may arrive at different objective function values at their optimums and so the conditions of Corollary 2 might not be met. This difference in these values at the optimum is called the dual gap.

Nevertheless, it can be shown [7] that if the constraints of the primal problem are affine functions (as is the case in SVM) another theorem can be stated:

Theorem 2 *(Strong duality theorem) Given a primal optimization problem in the form* (2.1.5) *and a dual defined as in* (2.1.10)*, if the constraints of the primal are affine functions, then the dual gap is* 0*. That is,*

$$F(W^*) = \Theta(\alpha^*,\beta^*), \tag{2.1.13}$$

where W^* stands for the optimal solution of the primal problem, and (α^*,β^*) for the optimal solution of the dual problem.

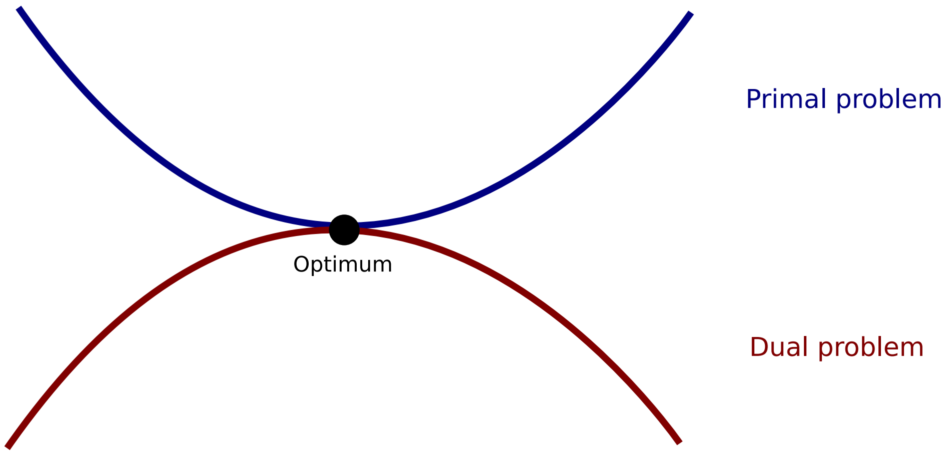

Figure 2.1.3: A depiction of the primal and dual optimization problems. While the former minimizes a convex function, the latter maximizes a concave one, arriving at the same value at their optimums due to the nonexistence of dual gap.

We can depict the general idea of these results in figure 2.1.3. By corollary 1 the possible values of the primal are always bigger than those of the dual, and by the strong duality theorem both problems converge at the same value.

Summing up, we could just forget about the primal problem and solve the dual one, knowing that we will obtain the same value at the optimum. The caveat of this procedure is that we originally wanted to obtain the optimal values for the parameters W of the primal problem, not the value of $f(W^*)$ at the optimum W^*. If we solve the dual we will instead obtain the optimal values of the parameters of the dual, (α^*, β^*), so it may seem that there is no point in doing this. But in fact there is a way in the SVM case to obtain the optimal primal problem values from the dual ones, as we will see in the next subsection.

2.1.4 Computing the dual of the SVM

As explained in the previous subsection, a dual problem can be formulated and solved instead of the primal optimization one, arriving at the same objective value. But if we want to obtain the optimal values of the parameters of the primal problem, we must transform the obtained solution of the dual into a solution of the primal. In order to do this, first consider the way the dual problem is constructed (2.1.10). The infimum of L with respect to the parameters W can be obtained by computing $\nabla_W L = 0$ and substituting the obtained expression back into L. But the interesting fact is that the expression obtained by computing that derivative will also give us a relationship between the parameters of the primal and dual problems. This is clearly seen when applying this theory to the SVM optimization problem (2.1.4). Since in the SVM the parameters of the problem are

$W = (w, b)$ we compute the derivatives of the SVM Lagrangian (2.1.7) with respect to them:

$$\nabla_w L_p = 0 \longrightarrow w = \textstyle\sum_{i=1}^N \alpha^i y_i x^i, \tag{2.1.14}$$
$$\frac{\partial L_p}{\partial b} = 0 \longrightarrow \textstyle\sum_{i=1}^N \alpha_i y_i = 0. \tag{2.1.15}$$

Now substituting back the obtained expression (2.1.14) for w and using the one in (2.1.15) for b to perform some simplifications, the obtained dual of the SVM takes the form

$$\begin{array}{ll} max_\alpha & L_D = -\frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j x^i \cdot x^j + \sum_{i=1}^N \alpha_i \\ \text{s.t.} & \left\{ \begin{array}{c} \alpha_i \geq 0 \quad \forall i \\ \sum_{i=1}^N \alpha_i y_i = 0 \end{array} \right\}. \end{array} \tag{2.1.16}$$

Observe that an additional constraint has appeared, due to (2.1.15). The derivative with respect to w does not appear as a constraint because it has been substituted in the main expression, and w no longer exists as a parameter in the dual problem.

The obtained dual problem is far easier to solve, since the restrictions are simpler. Many of the most popular methods in SVM training solve this dual to obtain the optimum, such as SMO [8] or SVMLight [6]. However we have not yet explained how to recover the primal problem solution (w^*, b^*) from the dual one α^*. The method for w^* is simple: just using the expression (2.1.14) we can compute w^* as a weighted sum of the training patterns, with α^* as the weights vector. To obtain b^* we should recall one of the KKT conditions that had to be met at the optimum, in particular

$$\alpha_i^*(1 - y_i(w^* \cdot x^i + b^*)) = 0 \quad \forall i. \tag{2.1.17}$$

By using some pattern x^i with $\alpha_i^* > 0$ and (2.1.17), b^* can be obtained as

$$b^* = y_i - w^* \cdot x^i = y_i - \sum_{j=1}^N \alpha_i^* y_i x^j \cdot x^i, \tag{2.1.18}$$

and so the solution of the primal problem is fully obtained by only solving the dual one.

A final remark that should be made about the dual problem is that some of the α_i may be 0. These patterns x^i with $\alpha_i = 0$ do not make any difference when classifying an unseen pattern x, as the classification function of the SVM hyperplane can be expressed as

$$f(x) = w \cdot x + b = \sum_{i=1}^N \alpha^i y_i x^i \cdot x + b, \tag{2.1.19}$$

where patterns with $\alpha_i = 0$ can be removed from the summation. After this removal, only patterns with $\alpha_i > 0$ are left, and thus the SVM is represented in a sparse way based on the relevant training patterns. Those patterns are called the “Support Vectors”, and they comprise the information about the trained SVM model. The support vectors will form the base of the conformal transformation methods explained in this work.

2.1.5 Dealing with noisy and overlapping data

Unfortunately, in real world problems we cannot aim to find a perfect classification, since the training data may be noisy and the classes may overlap, or even they could overlap anyway due to the nature of the problem. As far as explained, the SVM tries to maximize tha margin while classifying all the training points perfectly, so the SVM optimization problem can not be solved when the classes overlap. Fortunately, an extension to the SVM can overcome this caveat, by using the known as “slack variables”.

The main idea is to keep the aim of the SVM of maximizing the margin, but this time allowing some patterns to be misclassified or not beyond the margin. However, in order to maintain a good accuracy level, these misclassifications are penalized, and the SVM will try to keep these penalties low. Formally, the SVM problem is reformulated as

$$\begin{aligned} \min_{w,b,\xi,\mu} \quad & \|w\|^2 + C\left(\sum_{x^i \in X_+} \xi_i + \sum_{x^i \in X_-} \mu_i\right) \\ \text{s.t.} \quad & \left\{ \begin{array}{c} w \cdot x^i + b + \xi_i \geq 1 \quad \forall \quad x^i \in X_+ \\ w \cdot x^i + b - \mu_i \leq -1 \quad \forall \quad x^i \in X_- \end{array} \right\}, \end{aligned} \tag{2.1.20}$$

where X_+ stands for the set of patterns with $y_i = 1$, X_- for the set of patterns with $y_i = -1$, ξ for the slack variables of patterns in X_+, μ for the slack variables of patterns in X_-, and C for a SVM parameter that controls the tradeoff between maximizing the margin and reducing the allowed errors. While usually no distinction is used between the slack variables of both classes, we will adopt here this more general notation. Observe that the slack variables are compensating the violation of the restrictions by adding or subtracting a value to the “margin” $w \cdot x^i + b$ of the patterns. A graphical example of this type of SVM is shown in figure 2.1.4.

Again, this problem is hard to solve, and so we are willing to obtain a dual formulation. The good news are that all the preceding theory can be applied again in this kind of SVM, since the function to minimize is still convex and the constraints are affine. Therefore the dual problem expression can be obtained in the same way, resulting in

$$\begin{aligned} \max_{\alpha} \quad & L_D = -\tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x^i \cdot x^j + \sum_{i=1}^{N} \alpha_i \\ \text{s.t.} \quad & \left\{ \begin{array}{c} 0 \leq \alpha_i \leq C \quad \forall i \\ \sum_{i=1}^{N} \alpha_i y_i = 0 \end{array} \right\}. \end{aligned} \tag{2.1.21}$$

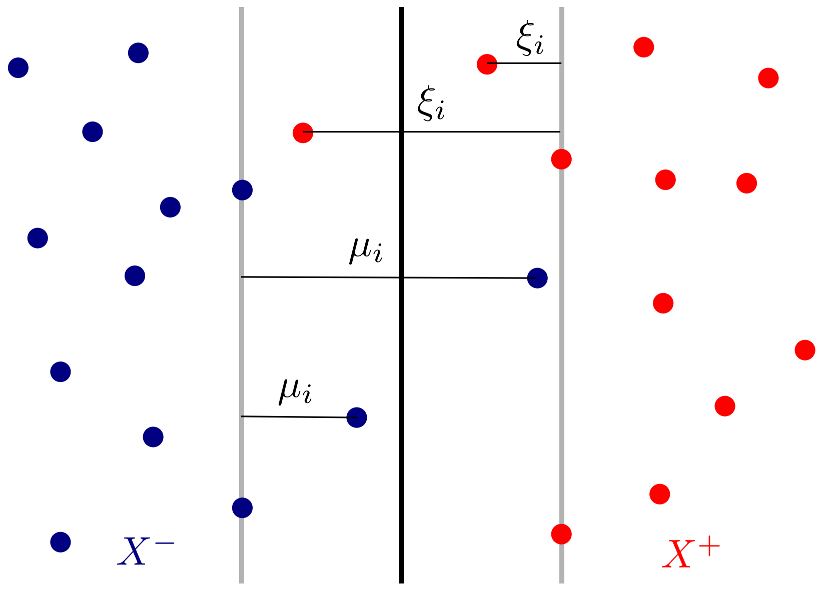

Figure 2.1.4: A typical example of a SVM with slack variables. The patterns that are misclassified or not beyond the margin have slack variables > 0 in order to meet the constraints.

We will not show explicitly how to obtain this expression, as essentially we follow the same procedure that in the "no slacks" case. Once the dual problem is solved we can obtain the SVM parameters in a similar fashion. The only difference is that the KKT conditions are not the same in this case, and so b^* must be computed using the following ones:

$$\begin{aligned} &\alpha_i(1 - w \cdot x^i - b - \xi_i) = 0 \quad \forall x^i \in X_+, \\ &\alpha_i(1 + w \cdot x^i + b - \mu_i) = 0 \quad \forall x^i \in X_-. \end{aligned} \tag{2.1.22}$$

It can be shown (via KKT conditions) that if a support vector x^i has $\alpha_i < C$, then it lies on the margin, an so its associated slack variable has value 0. Therefore from the above conditions the value of b^* can be obtained using a pattern x^i with $0 < \alpha_i < C$. Generally, all the patterns which meet those conditions are used to compute estimates of b^* and a mean is used.

In this way a SVM can deal with a broader range of problems without losing its dual problem properties. Alas, another extension is required to be capable of working with real world problems.

2.1.6 The kernel trick

In the previous section an extension to the SVM was introduced in order to deal with overlapping data. However, the decision function of the SVM, that is, the separating boundary found between both classes had to be necessarily a hyperplane. Hence, a SVM of the explained form will not be

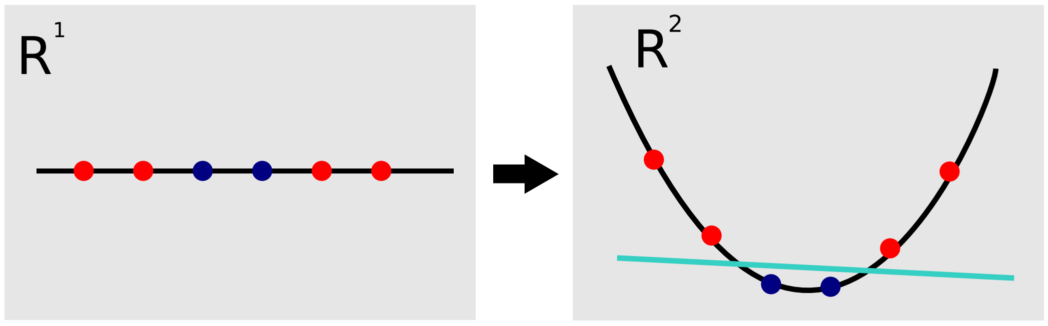

Figure 2.1.5: An illustrative example of the kernel trick. The original problem has only one input variable, and a linear classifier is unable to solve it. However, by using a suitable mapping function to 2-dimensional space we can find a line that makes a perfect classification.

able to correctly classify problems that are not linearly separable. In the real world problems this is not due to happen, and so another extension to allow a non–lineal SVM should be introduced: the kernel trick.

The kernel trick is based on the idea that, when a suitable transformation of the input variables is made, we could obtain a new set of variables that express the original patterns in a new space, and in this new space the classes might be linearly separable. More formally, suppose we have a function $\Phi(x^i)$ such that it receives a pattern x^i with d input variables, and returns another vector of D variables, $D >> d$. We can think of this function as a mapping from the input space (a d-dimensional space) to a wider "feature space" (a D-dimensional space). If we appropriately choose Φ we can obtain a linearly separable problem in the feature space, even if it was not separable in the input space. An illustrative example is shown in figure 2.1.5, where a 1-dimensional problem can be linearly solved in a 2-dimensional space.

Problems arise if we try to compute the explicit value of $\Phi(x^i)$ for all the training patterns, since the feature space could have a really high dimensionality (even infinite). But the fact is that we will not need to compute them explicitly, and that is the main advantage of the kernel trick. Observe that if we substitute x^i with $\Phi(x^i)$ in the SVM dual 2.1.16 we obtain the following optimization problem:

$$\begin{array}{ll} \max\limits_{\alpha} & L_D = -\frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i\alpha_j y_i y_j \Phi(x^i)\cdot\Phi(x^j) + \sum_{i=1}^N \alpha_i \\ \text{s.t.} & \left\{ \begin{array}{c} \alpha_i \geq 0 \quad \forall i \\ \sum_{i=1}^N \alpha_i y_i = 0 \end{array} \right\}. \end{array} \tag{2.1.23}$$

We find that the mapped patterns $\Phi(x^i)$ only appear in pairs, combined by an inner product. This is relevant, as in the end we will only need to know the inner product of pairs of $\Phi(x^i)$, not their explicit values. Therefore, a "kernel function" k is defined as

$$k(x^i, x^j) = \Phi(x^i) \cdot \Phi(x^j), \tag{2.1.24}$$

and so the problem can be stated as

$$\begin{array}{ll} \max\limits_{\alpha} & L_D = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j k(x^i, x^j) + \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} & \left\{ \begin{array}{c} \alpha_i \geq 0 \quad \forall i \\ \sum_{i=1}^{N}\alpha_i y_i = 0 \end{array} \right\}. \end{array} \tag{2.1.25}$$

So we do not really need to know how the function $\Phi(x^i)$ is computed, but only k. In fact, the usual approach is to just choose some function k and forget about the underlying $\Phi(x^i)$. But not any function can be chosen as k, because it may not be possible to break it down into the inner product of two Φ functions. Fortunately, the Mercer's Theorem states what kind of functions could be used for this purpose.

Theorem 3 *(Mercer's theorem) If a scalar function $k(x, x')$ is positive semi-definite, then it can always be decomposed as an inner product $k(x, x') = \Phi(x) \cdot \Phi(x')$ where $\Phi(x)$ are vectors which potentially may have infinite dimensions.*

Definition 1 *(Positive semi-definite kernel) A kernel function $k(x, x')$ is positive semi-definite if for any other function $f \in L^2(\mathcal{X})$ it meets*

$$\int_{\mathcal{X}\times\mathcal{X}} k(x, x')f(x)f(x')dxdx' \geq 0, \tag{2.1.26}$$

where $\mathcal{X}$ is the set of all possible x. Alternatively, a kernel function is positive semi-definite if for any $x \neq 0$

$$x^T K x \geq 0, \tag{2.1.27}$$

where K stands for the kernel matrix containing all the possible kernel products of the training patterns

$$K_{ij} = k(x^i, x^j). \tag{2.1.28}$$

Definition 2 *(L^p space) A L^p space is a mathematical space containing all the functions that are p times integrable.*

We will not prove this theorem here (for more details, see [9]). Although we could design some kind of adapted kernel to each problem, generally the Gaussian or RBF kernel is used:

$$k(x^i, x^j) = e^{\left(-\frac{\|x^i - x^j\|^2}{2\sigma^2}\right)}, \tag{2.1.29}$$

where σ is a parameter which controls the width of the kernel function. As we will show later, this parameter actively modifies the shape of the feature space, and so the classification accuracy will heavily depend on it. Nevertheless, it is usually chosen by cross-validation techniques. Other common kernel function is the inhomogeneous polynomial kernel:

$$k(x^i, x^j) = (x^i \cdot x^j + c)^p, \tag{2.1.30}$$

with c and p parameters of the kernel, although usually c is fixed to 1.

As the dual problem can be solved using the kernel trick, we can obtain the optimal support weights α^*. However, we can not compute w^* explicitly, as it would take the form $w^* = \sum_{i=1}^N \alpha_i y_i \Phi(x^i)$ and $\Phi(x^i)$ is not known. But in fact, now that we have found α^* and the training is finished, we only need to evaluate the decision function of the SVM, which we can manipulate in the following way:

$$f(x) = w \cdot \Phi(x) + b = \sum_{i=1}^{N} \alpha_i y_i \Phi(x^i) \cdot \Phi(x) + b = \sum_{i=1}^{N} \alpha_i y_i k(x^i, x) + b, \tag{2.1.31}$$

and again we only need to compute the kernel function. Hence, we can effectively train and classify with a SVM in the feature space without computing explicitly the patterns in this space. Even if we are performing a linear classification in the feature space, the separating boundary in the input space would be non-linear.

2.1.7 The full SVM

Finally, the kernel trick can be applied along the slack variables extension, and so the full SVM formalization is obtained. In this way the following primal problem appears as

$$\begin{array}{cc} \displaystyle\min_{w,b,\xi,\mu} & \|w\|^2 + C\left(\sum_{x^i \in X_+} \xi_i + \sum_{x^i \in X_-} \mu_i\right) \\ \text{s.t.} & \left\{ \begin{array}{c} w \cdot \Phi(x^i) + b + \xi_i \geq 1 \quad \forall x^i \in X_+ \\ w \cdot \Phi(x^i) + b - \mu_i \leq -1 \quad \forall x^i \in X_- \end{array} \right\}, \end{array} \tag{2.1.32}$$

whose dual problem is

$$\begin{array}{cc} \displaystyle\max_{\alpha} & L_D = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j k(x^i, x^j) + \sum_{i=1}^N \alpha_i \\ \text{s.t.} & \left\{ \begin{array}{c} 0 \leq \alpha_i \leq C \quad \forall i \\ \sum_{i=1}^N \alpha_i y_i = 0 \end{array} \right\}. \end{array} \tag{2.1.33}$$

We will not enter in further details here, as a different point of view will be used, which we explain in the following section.

2.2 Nearest Points in Convex Hulls Problem

In the previous section the formalization of the SVM was explained up to some detail, as it will be necessary in the following chapters. However, the point of view we will take at the training time of a SVM will be quite different, and so in this section we explain it.

2.2.1 A geometrical alternative: CH-SVM

Although the solving of the dual problem of the SVM is usually performed directly, other alternatives exist. One of those alternatives is to solve the SVM in a geometrical way, that is, creating algorithms that can train the SVM using geometrical notions about the problem. To explain this, we will first show some geometrical intuitions that seem to follow the SVM philosophy, and then we will prove that this geometrical point of view is in fact equivalent to the original.

To start, we first define the convex hull of a set of points

Definition 3 *We call $\mathcal{C}(X)$ the convex hull of the set of points X, and is defined as the minimum convex polyhedron such that all the points in X are inside it. An alternative definition states that a point x^0 belongs to $\mathcal{C}(X)$ only if it can be expressed as a convex combination of the points in X, that is:*

$$x^0 = \textstyle\sum_{i=1}^N c_i x^i,$$
$$\textstyle\sum_{i=1}^N c_i = 1,$$
$$c_i \geq 0 \, \forall i.$$

Now, in a linearly separable problem where the classes do not overlap, we can construct the convex hulls of both classes ($\mathcal{C}(X_+)$ and $\mathcal{C}(X_-)$) and they will not overlap. If we look for the nearest points of both convex hulls, which we would call $\hat{w}_+$ and $\hat{w}_-$, we can see that we can obtain a separating hyperplane by joining both points with a line and tracing the hyperplane bisecting it. Formally, the normal vector of this hyperplane is

$$\hat{w} = \hat{w}_+ - \hat{w}_-, \tag{2.2.1}$$

and the bias $\hat{b}$ should be such that the hyperplane is located in the middle point between $\hat{w}_+$ and $\hat{w}_-$, so

$$\begin{aligned} 0 &= \hat{f}\left(\frac{\hat{w}_+ + \hat{w}_-}{2}\right) \\ 0 &= \hat{w} \cdot \left(\frac{\hat{w}_+ + \hat{w}_-}{2}\right) + \hat{b} \\ \hat{b} &= -\frac{1}{2}(\hat{w} \cdot \hat{w}_+ + \hat{w} \cdot \hat{w}_-) \end{aligned} \tag{2.2.2}$$

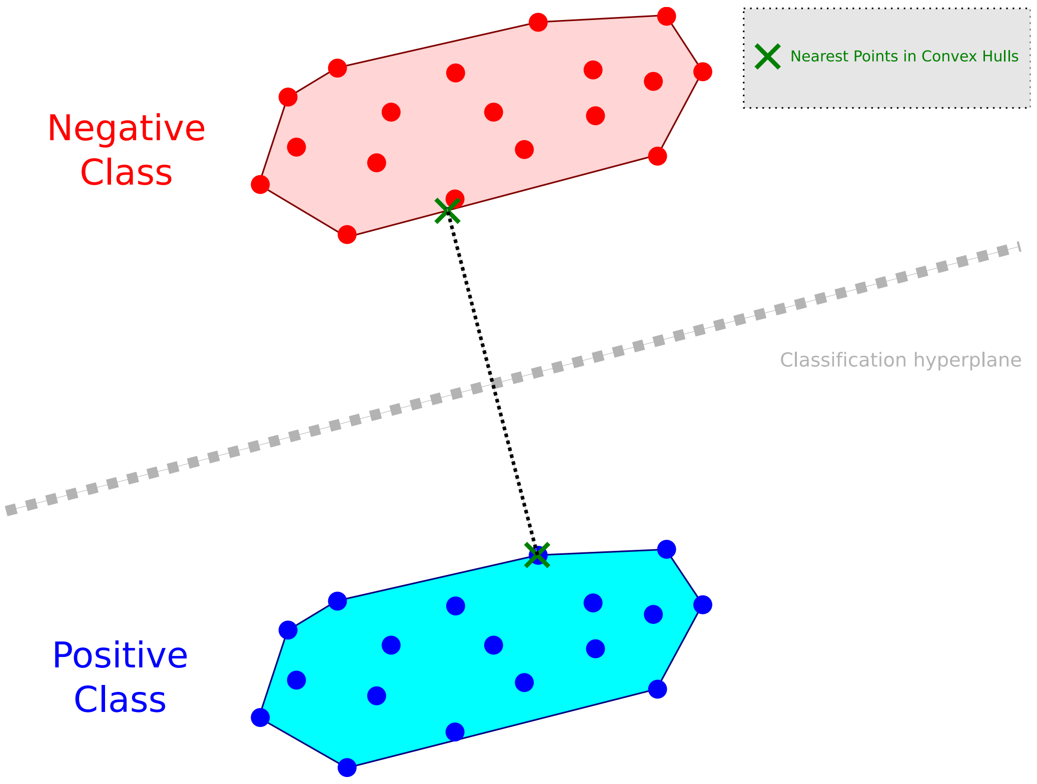

Figure 2.2.1: In this example the convex hulls of both classes are shown. By finding the nearest points and the bisecting hyperplane between them, the solution of the SVM is found.

What is more, this hyperplane $(\hat{w}, \hat{b})$ looks quite similar to the solution that would be obtained by a SVM. In fact it can be proved that the obtained hyperplane is the same that the one that solves the "classic" dual problem, up to some rescaling. An example is shown in figure 2.2.1.

The problem of finding the closest points can be formally stated as an optimization problem:

$$\begin{aligned} \min_{\hat{w}_+, \hat{w}_-} \quad & \|\hat{w}\|^2 \\ \text{s.t.} \quad & \hat{w}_+ \in \mathcal{C}(X_+) \quad , \quad \hat{w}_- \in \mathcal{C}(X_-). \end{aligned} \tag{2.2.3}$$

In order be sure that neither $\hat{w}_+$ nor $\hat{w}_-$ are outside their respective convex hulls, we should keep track of the coefficients that can express them as a convex combination. If we call $\hat{\alpha}_i$ the convex coefficient provided by the i-th point of the dataset, we can write $\hat{w}_+$ and $\hat{w}_-$ as

$$\hat{w}_+ = \sum_{x^i \in X_+} \hat{\alpha}_i x^i \quad , \quad \hat{w}_- = \sum_{x^i \in X_-} \hat{\alpha}_i x^i. \tag{2.2.4}$$

Also, as the $\hat{\alpha}_i$ are coefficients of a convex combination, they should meet the constraints

$$\sum_{i|x^i \in X_+} \hat{\alpha}_i = 1 \quad , \quad \sum_{i|x^i \in X_-} \hat{\alpha}_i = 1 \quad , \quad \hat{\alpha}_i \geq 0 \, \forall i. \tag{2.2.5}$$

Hence the problem of finding the closest points in the convex hulls, which we shall call CH-SVM, can be written in terms of these convex coefficients, as

$$\begin{array}{ll} \min\limits_{\hat{\alpha}_i} & \| \sum_{x^i \in X_+} \hat{\alpha}_i x^i - \sum_{x^i \in X_-} \hat{\alpha}_i x^i \|^2 \\ \text{s.t.} & \left\{ \begin{array}{c} \sum_{i|x^i \in X_+} \hat{\alpha}_i = 1 \\ \sum_{i|x^i \in X_-} \hat{\alpha}_i = 1 \\ \hat{\alpha}_i \geq 0 \quad \forall i. \end{array} \right\}, \end{array} \tag{2.2.6}$$

Again, we have a quadratic programming problem. But this time it is not necessary to find a dual or to use a quadratic optimizer, as we can use geometric notions to tailor an algorithm to solve it. In fact in the geometry literature one can find some algorithms which find the closest point to the origin inside a polyhedron, like the Gilbert-Schlesinger-Kozinec (GSK) algorithm [10] or the Mitchell-Dem'yanov-Malozemov (MDM) [11]. These algorithms can be adapted to solve this nearest points problem, therefore obtaining a separating hyperplane. They also can be adapted in order to perform the kernel trick, and to include slack variables (as we will see later); for more details about them look for instance [12].

2.2.2 Equivalency of SVM and CH-SVM

As stated in the previous subsection, solving a CH-SVM we obtain a hyperplane $(\hat{w}, \hat{b})$ that can be proved to be equivalent to the hyperplane (w, b) obtained by solving a SVM, up to some rescaling. This result was first shown by Bennet and Bredensteiner [13], although alternative proofs have been developed, like in our previous work [14]. Since the demonstration is not useful for the purpose of this work, we just state its results.

Suppose that the optimal solution for a SVM (2.1.16) is obtained as (w^*, b^*), and the optimal solution for the CH-SVM (2.2.6) is obtained as $(\hat{w}^*, \hat{b}^*)$. Then these optimal solutions are related by

$$w^* = \frac{2}{\|\hat{w}^*\|^2} \hat{w}^* \quad , \quad b^* = \frac{2}{\|\hat{w}^*\|^2} \hat{b}^*; \tag{2.2.7}$$

that is, they are equivalent under a normalization. Although usually the original approach is used, we shall follow in this work this geometric approach, as it gives a better intuition of the "behaviour" of the algorithms.

2.2.3 CH-SVM with slack variables and the kernel trick

Although the SVM is equivalent to the CH-SVM, the nearest points problem only makes sense when the two convex hulls are linearly separable, that is, when they do not overlap. If overlapping appears then the nearest points would be, in fact, the same point, and so $\hat{w} = 0$, which is not a real solution.

Hence, we should use the slack variables and the kernel trick of the SVM in CH-SVM in order to assure we deal with a linearly separable problem. However, it is not easy to deduce at first glance how to introduce these extensions into CH-SVM. In spite of this, in fact the answer is fairly simple, and it involves the use of a special kernel function, but to arrive at it we must first make some considerations.

Firstly, we define some extended patterns $\overline{x^i}$ and weight vector $\overline{w}$ as

$$\begin{aligned} \overline{w} &= (w, \sqrt{C}\xi_1, \dots, \sqrt{C}\xi_N, \sqrt{C}\mu_1, \dots, \sqrt{C}\mu_N), \\ \overline{x}^i_+ &= (x^i,\ 0, \dots, \tfrac{1}{\sqrt{C}}, \dots, 0,\ \ 0, \dots, 0), \\ \overline{x}^j_- &= (x^j,\ 0, \dots, 0,\ \ 0, \dots, \tfrac{-1}{\sqrt{C}}, \dots, 0), \end{aligned} \tag{2.2.8}$$

where $\overline{x}^i_-$ is the extension for patterns in the negative class and $\overline{x}^i_+$ is the extension for patterns in the negative class. The extra attributes that are introduced in the patterns are all zero except one that has value $\frac{y_i}{\sqrt{C}}$. This non-zero attribute depends on the pattern itself: for the first pattern of the positive class, the first extra attribute is non-zero, for the second one, the second extra attribute is non-zero... and so on, proceeding with the patterns of the negative class when there are no more patterns of the positive one. Therefore, N extra attributes are added.

By using these extended vectors in the SVM primal problem (2.1.4) we obtain

$$\begin{array}{ll} \min\limits_{\overline{w}} & \|\overline{w}\|^2 = \|w\|^2 + C\sum_{i=1}^N(\xi_i^2 + \mu_i^2) \\ \text{s.t.} & \left\{ \begin{array}{l} \overline{w} \cdot \overline{x}^i + b = w \cdot x^i + \xi_i + b \geq 1 \quad \forall x^i \in X_+ \\ \overline{w} \cdot \overline{x}^i + b = w \cdot x^i - \mu_i + b \leq -1 \quad \forall x^i \in X_- \end{array} \right\}. \end{array} \tag{2.2.9}$$

The optimization problem that appears is quite similar to the SVM with slacks (2.1.20), but with squared penalties in the objective function. This is in fact another possible way of introducing slacks into the SVM. Therefore we can see that if squared penalties are used, we can avoid solving an optimization problem with slacks by extending the patterns and the weights vector in this way. What is more, in fact we do not really need to perform this extension. Observe that the dual problem would be

$$\begin{array}{ll} \max\limits_{\alpha} & L_D = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \overline{x}^i\cdot\overline{x}^j + \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} & \left\{ \begin{array}{c} \alpha_i \geq 0 \quad \forall i \\ \sum_{i=1}^{N}\alpha_i y_i = 0 \end{array} \right\}. \end{array} \tag{2.2.10}$$

In the dual only the extended patterns appear, but due to the nature of this extension note that

$$\overline{x}^i\cdot\overline{x}^j = \overline{k}(x^i,x^j) = x^i\cdot x^j + \frac{\delta_{ij}}{C}, \tag{2.2.11}$$

where δ_{ij} is the Kronecker's delta function, which is defined as

$$\delta_{ij} = \left\{ \begin{array}{c} 1\,,\; i=j \\ 0\,,\; i\neq j \end{array} \right\}. \tag{2.2.12}$$

Through this simple kernel function $\overline{k}$ we are able to solve the dual of a SVM with slacks as if it did not have them. Furthermore if we are wishing to use another kernel function k in order to compute the SVM in a higher dimensional space, we can do the following

$$\begin{aligned} \hat{k}(x^i,x^j) &= \hat{\Phi}(x^i)\cdot\hat{\Phi}(x^j) \\ &= (\Phi(x^i),0,\cdots,\frac{y_i}{C},\cdots,0)\cdot(\Phi(x^j),0,\cdots,\frac{y_j}{C},\cdots,0) \qquad (2.2.13) \\ &= k(x^i,x^j) + \frac{\delta_{ij}}{C}. \qquad (2.2.14) \end{aligned}$$

And so we solve the dual problem

$$\begin{array}{ll} \max\limits_{\alpha} & L_D = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^i y^j \hat{k}(x^i,x^j) + \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} & \left\{ \begin{array}{c} \alpha_i \geq 0 \quad \forall i \\ \sum_{i=1}^{N}\alpha_i y_i = 0 \end{array} \right\}. \end{array}$$

Summing up, if we use squared penalties and a special kernel function $\hat{k}$ we can solve a SVM as if it had no slacks. The advantage of this approach is that in the corresponding CH-SVM the classes will not overlap, and so a solution is possible. Hence, solving a CH-SVM with a kernel in the form $\hat{k}$ we can obtain the solution for an equivalent SVM with both the slacks and the kernel trick extension. Therefore, both SVM and CH-SVM have the same generalization capability. However in this work we will use the second approach, as it is easier to understand due to its geometrical interpretation.

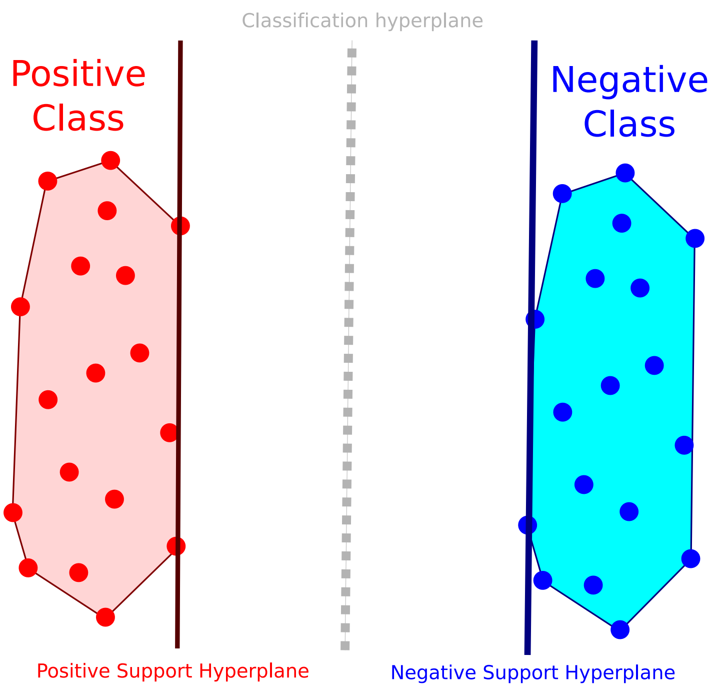
Classification hyperplane
Positive Class
Negative Class
Positive Support Hyperplane
Negative Support Hyperplane

Figure 2.2.2: An example of the support hyperplanes obtained in a CH-SVM problem.

2.2.4 Support hyperplanes

The main reason of explaining the CH-SVM concepts in this work is the useful definition of support hyperplanes that arises under this point of view. In a linearly separable problem the support hyperplanes are defined as those which are the farthest away from the separating hyperplane and, at the same time, would perform a correct classification (although not with the best margin). These support hyperplanes must be parallel to the separating hyperplane found by the SVM. An example is shown in figure 2.2.2.

Another definition could be that the support hyperplanes are those which intersect the nearest points w_+ and w_- (one each) and are parallel to the SVM separating hyperplane. In this way w_+ and w_- determine the location of the support hyperplanes. This observation will prove to be useful when trying to move these hyperplanes, as we will do in our proposed methods for imbalance correction.

Chapter 3

Imbalanced datasets and SVMs

3.1 The problem of imbalanced datasets

In real world problems where we have to perform a binary classification it is quite common that the number of available patterns for training belonging to one of the classes outnumbers significantly the other. This is known as class imbalance, or we talk about imbalanced datasets. When this situation arises most of the common machine learning methods will perform poorly when attempting to classify new patterns of the minority class. This is mainly due to the fact that these methods usually pursue the minimization of an error measure, such as the misclassification error or mean squared error. With almost no effort, any machine learning method could simply classify all the points as if they belonged to the majority class, and nevertheless obtain a good error measure, since there are very few patterns of the minority class in the training or the validation set. However, when attempting to classify new, unseen patterns, the model will perform poorly [15].

A usual measure to overcome this problem is to use an asymmetric error function; asymmetric in the way that it uses different weights to penalize errors from one or the other class. In the context of classification tasks, a possible error function could be:

$$E = R_+ \sum_{i|y_i=1} (1 - \hat{y}_i) + R_- \sum_{i|y_i=-1} (1 + \hat{y}_i), \tag{3.1.1}$$

where the $\hat{y}_i$ stand for the classifications made by the model, the y_i for the actual classes, R_+ for the risk factor of misclassifying an instance of X_+, and R_- for the risk factor of misclassifying an instance of X_-. Supposing that X_+ is the minority class, we would like to have $R_+ > R_-$ in order to "force" the learning method to focus on those data points. Another error function that is commonly used in imbalanced problems is the Kubat's g-means accuracy measure. It takes the form

$$g = \sqrt{a^+ a^-}, \tag{3.1.2}$$

where a^+ and a^- are the obtained accuracies for the patterns in X_+ and X_-, respectively. However it is not possible to use these kind of error functions on all the machine learning methods; for example it is not possible (up to our knowledge) to use the g criterion to train a multilayer perceptron or a SVM. Therefore, other techniques should be used to correct the method itself in order to avoid this problem.

An artificial example of an imbalanced problem, which is used in [15], is the checkers board problem. In it a squared region is divided into four smaller subregions, arranged as tiles in a checkers or chess board. While the upper-left and lower-right tiles contain patterns of the negative class, scattered uniformly, the upper-right and lower-left have patterns of the positive class, also generated uniformly but in much smaller number. A depiction of the layout of this problem is shown in figure 3.1.1.

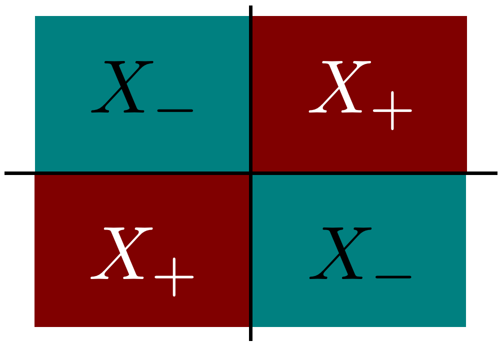

Figure 3.1.1: Layout of the checkers board problem. X_- points are randomly generated in the upper-left and lower-right areas, while X_+ points are generated in the upper-right and lower-left areas

As an example, we create a training dataset for this problem in where the ratio of class instances is 1:100, with 20 instances of the minority class. The obtained dataset is shown in figure 3.1.2. When training a SVM in which parameters are chosen by cross-validation techniques in order to reduce misclassification error the separating frontier depicted in figure 3.1.3 is obtained. It can be observed that the frontier is biased towards the minority class region, and thus in a balanced test set a considerable amount of misclassifications is made when dealing with patterns of the minority class.

In the particular case of using a SVM, there are three main reasons that explain why the frontier is biased [16, 17]:

1. As there are few instances of the minority class, it is unlikely that one of them appears near the "true" class frontier, so it is impossible for the SVM to infere where that real frontier is

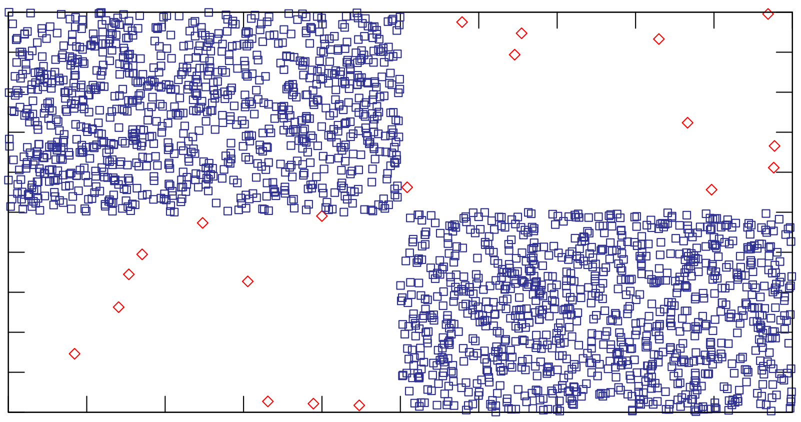

Figure 3.1.2: Training set of the checkers board problem. The patterns belonging to the positive class (in red) are heavily outnumbered by the patterns of the negative class (in blue).

located. Hence, as the SVM selects the frontier such as to maximize the margin, it would be biased towards the minority class because the respective support hyperplane is badly estimated.

2. Even if some data points are near the true frontier, the SVM may treat them as noise, using a non-zero slack in them, so as to obtain a larger margin. An example of this is depicted in figure 3.1.4.

3. Again, because of the low number of instances in the minority class, a little number of support vectors appear in this class. Even if they have a bigger weight, as for any point in the real frontier there will be a much larger number of negative support vectors, the SVM is prone to classify the point as belonging to the negative (majority) class.

Some methods have been developed in order to overcome this problem. We will briefly review some of them in the next section to provide a general idea of the standard techniques used.

3.2 Basic methods for imbalance correction in SVMs

In this section we will briefly explain three methods widely used to alleviate the problem of imbalanced datasets in SVMs: boundary movement, biased penalties and kernel modification.

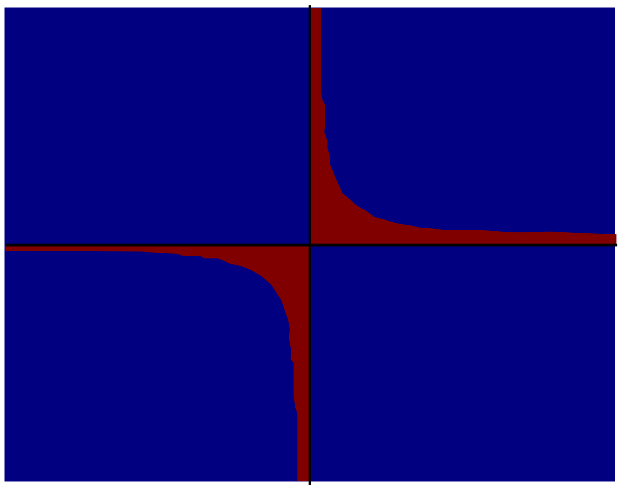

Figure 3.1.3: The results of training a SVM over an imbalanced dataset. The correctly classified region is depicted in blue, while the wrongly classified is depicted in red. As it can be seen, the separating frontier is biased towards the minority class, producing a region of error.

3.2.1 Boundary movement

Boundary movement is a naive method that tries to correct the biased separating frontier by moving it through the bias term b. An additional term Δb is added to the SVM function, in the form

$$f(x) = \sum_{i=1}^{N} y_i \alpha_i k(x, x^i) + b + \Delta b. \tag{3.2.1}$$

Tuning Δb provides a tradeoff between false positives and false negatives. If the orientation (w) of the hyperplane is correct, it may work up to some point, but generally this will not be the case. Additionally and as we shall explain later, boundary movement can be regarded as just shifting the whole frontier along w, support hyperplanes included. This is not generally a good idea, since the negative support hyperplane is likely to be well estimated and by moving it we are losing this estimation.

3.2.2 Biased penalties

A strategy that sounds similar to using an error function with risk factors is the one proposed in [18], where different SVM penalty factors C^+ and C^- are used for the points in X_+ and X_- respectively. In this way the SVM problem appears as

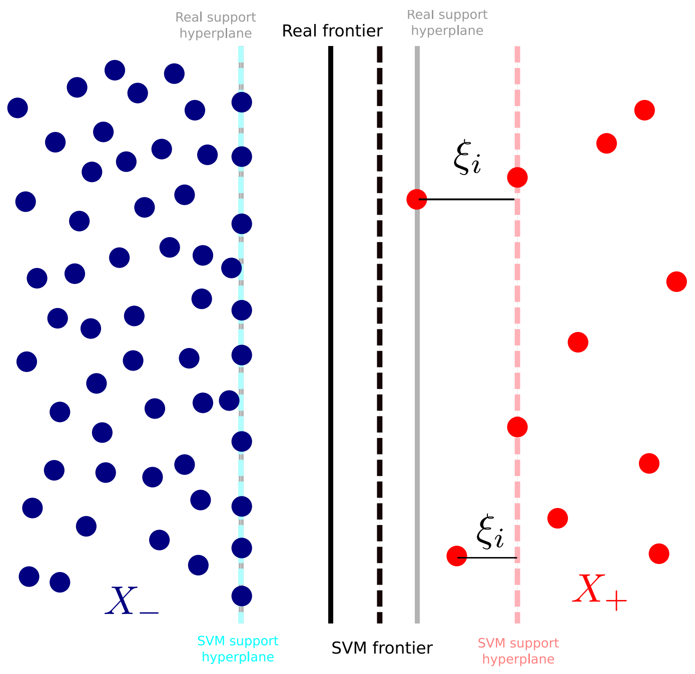

Figure 3.1.4: An example of a dataset where the SVM finds a biased classification frontier. Even if two data points of the minority class are in the real support hyperplane, the SVM regards them as noise, taking a farther hyperplane.

$$\begin{aligned}
&\min_{w,b,\xi,\mu} && \|w\|^2 + C^+ \textstyle\sum_{i\in X_+} \xi_i + C^- \sum_{i\in X_-} \mu_i \\
&\text{s.t.} && \left\{ \begin{array}{c} w\cdot\Phi(x^i) + b + \xi_i \geq 1 \quad \forall i \in X_+ \\ w\cdot\Phi(x^i) + b - \mu_i \leq -1 \quad \forall i \in X_- \end{array} \right\},
\end{aligned} \tag{3.2.2}$$

and the dual problem takes the form

$$\begin{aligned}
&\max_{\alpha} && L_D = -\tfrac{1}{2}\textstyle\sum_{i=1}^N \sum_{j=1}^N \alpha_i\alpha_j y_i y_j k(x^i,x^j) + \sum_{i=1}^N \alpha_i \\
&\text{s.t.} && \left\{ \begin{array}{c} 0 \leq \alpha_i \leq C^+ \quad \forall i \in X_+ \\ 0 \leq \alpha_i \leq C^- \quad \forall i \in X_- \\ \sum_{i=1}^N \alpha_i y^i = 0 \end{array} \right\}.
\end{aligned}$$

In our case, we would choose $C^+ >> C^-$, since we want to give a bigger penalty for misclassified points in the minority class. However, as stated in [15] this method does not help as much as expected. By looking at the constraints of the dual problem we realize that incrementing C^+ does not necessary mean that an increase in the value of the support weights will appear, since the C^+ only acts as an upper bound. Moreover, the third constraint means that some increase in the support weights of points in X_+ will also increase the weights in some points of X_-. This means that in the end the support vectors of the minority class will have more or less the same influence over the decision boundary than the support vectors of the majority class, so no improvement may be obtained.

3.2.3 Kernel modification

The kernel function directly affects the way the SVM is built, as the geometry of the feature space is a consequence of it. If a non-suitable kernel function is chosen the dataset might not be linearly separable in the feature space (up to the use of slacks), and a poor model would be obtained. Usually a kernel function adjustable through parameters is used, such as the Gaussian kernel (2.1.29), in which the kernel width $2\sigma^2$ has to be chosen. This choice is generally made via cross-validation techniques in order to reduce the misclassification error over a validation dataset. By doing this we are implicitly adapting the feature space induced by the kernel function in order to obtain a better model. But more advanced methods exist for this purpose.

In this work we shall focus on this kind of methods, specifically on those who make use of conformal transformations in order to obtain a more suitable feature space. We will introduce the general theory of conformal transformations in the next section.

3.3 Adapting the feature space

In this section we will show how the choice of a kernel function induces a particular geometry in the feature space that can be understood under the theory of Riemaniann manifolds, which we will introduce up to some point. We will also see how we can modify a given kernel function via conformal transformations in order to modify the structure of the feature space, and so make it more suitable for our problem.

3.3.1 Riemannian geometry

Before explaining how the geometry in the feature space depends on the kernel function, some theory about Riemaniann geometry should be introduced, as this feature space follows a Non-Euclidean geometry. We will start introducing some mathematical concepts regarding the topic. Please note that we do not pretend to explain Riemannian geometry in detail, and so some of the following definitions may not be complete, as they are only intended to provide a basis for further reasoning.

One way to describe complex geometrical structures is using topological manifolds. A topological manifold is an abstract mathematical space in which, for every point, a neighbourhood could be defined such as $\exists$ an homeomorphism that maps it into an open interval in $\mathcal{R}^n$ that is an Euclidean space; however, the global structure could be more complex. An homeomorphism is a continuous bijective function with a continuous inverse.

A simple example of a manifold is a circle [1], which can be regarded as a one-dimensional manifold embedded in a two-dimensional space. To understand why it is in fact a one-dimensional space, suppose we are constrained to move only in the space defined by the circle. Then the only feasible movements are clockwise or counterclockwise displacements along the circle, which are in fact movements along opposite directions. That is, only movements along one dimension are allowed. What is more, this space is Non-Euclidean, since we cannot measure the distance between one point of the circle and another by using an Euclidean distance (i.e. a straight line), because we can not exit the circle. A suitable measure of distance would be the arc formed by the two points, but obviously this is not an Euclidean distance. A depiction of this example is shown in figure 3.3.1.

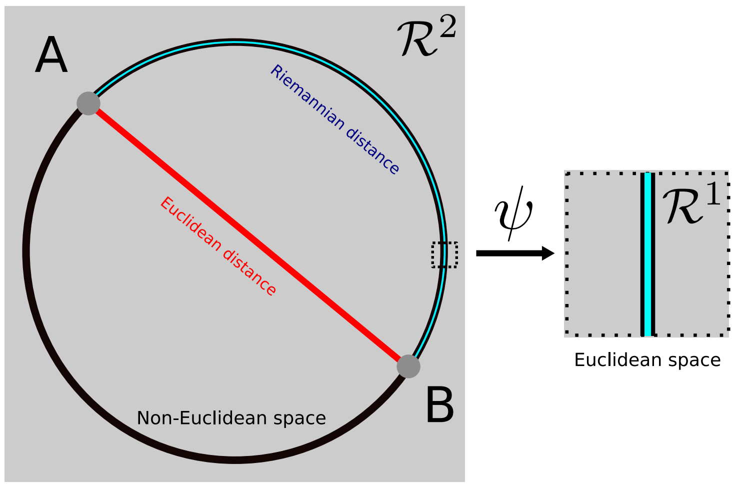

Figure 3.3.1: The circle is a simple example of a Riemannian manifold. Even if the circle is embedded in a two-dimensional space, it is in a fact a one-dimensional manifold. Distances between points can not be measured using an Euclidean distance, but a Riemannian one. Under a suitable mapping function ψ the neighbourhood of a point in the circle can be regarded as an euclidean space.

If we are able to map the neighbourhood of a point into an Euclidean space $\mathcal{R}^n$ then we would be able use the tools provided by the Euclidean geometry to obtain measures in that neighbourhood. Unfortunately there might not be a way to generalize this local measures to the whole geometry of the manifold, so more characteristics are needed in order to make measurements along the manifold.

One of those characteristics is the requirement of having a differentiable manifold. A topolog-

[1]By circle we refer to the mathematical space of points that meet the formula $(x-a)^2+(y-b)^2=r^2$. In this way the points "inside" the shape of the circle (the disk) do not belong to the circle.

ical manifold is differentiable if the mapping functions of the homeomorphisms are all differentiable. If the manifold is differentiable we would be able to use tools from calculus to propagate the results obtained in the mapped Euclidean spaces to the manifold. But we will focus on the particular case of the Riemannian manifolds.

A differentiable manifold is a Riemannian manifold if Riemannian distances can be measured along it. The Riemannian distance is expressed as

$$\|dz\|^2 = \sum_i \sum_j g_{ij} dx_i dx_j, \tag{3.3.1}$$

where dz stands for a differential vector of displacement inside the manifold, dx is the corresponding displacement in the mapped Euclidean space and g_{ij} is the Riemannian tensor, which defines the structure of the distance in the manifold. This tensor may change its values depending on the considered point of the manifold.

Therefore in a Riemannian manifold the distance between two points can be measured as some kind of integration of differential displacements through it. The circle used as an example is, in fact, a Riemannian manifold, as the proper distance between two points (the arc) can be measured with infinitesimal straight displacements along the circle. Also observe that the standard Euclidean distance is a particular case of the Riemannian distance, in which the Riemannian tensor is the unit matrix. We will not extend further in the theory of Riemannian manifolds, as the notion of Riemannian tensor is enough to understand the motivation under the conformal transformation methods.

3.3.2 Kernels and Riemannian manifolds

Now we will explain what the Riemannian geometry theory has to do with SVMs. As known, a SVM makes use of the kernel trick to map the input space $\mathcal{I}$ of dimension d into a feature space $\mathcal{F}$ of dimension D, with $D >> d$ (even infinite). But in fact what the transformation function Φ is doing is to map the points of $\mathcal{I}$ to a manifold of dimension d which is embedded into $\mathcal{F}$. So, under the theory of Riemaniann geometry, Φ defines the homeomorphism between the manifold in $\mathcal{F}$ and an Euclidean space in $\mathcal{I}$. A clear example is shown in figure 2.1.5, where a one-dimensional input space is mapped into a parabola (a one-dimensional manifold) in a two-dimensional feature space. We shall prove that in fact the kernel function generates a Riemannian metric in the following theorem:

Theorem 4 *A valid kernel function $k(x^i, x^j)$ maps an input space $\mathcal{I}$ of d dimensions into a Riemannian manifold of d dimensions embedded into a feature space $\mathcal{F}$ of $D >> d$ dimensions (which may be infinite), and so a Riemannian metric in the form* (3.3.1) *exists, whose Riemannian tensor* *can be computed using the kernel function as*

$$g_{ij} = \frac{\partial^2}{\partial x_i \partial x'_j} k(x, x') \mid_{x'=x} . \tag{3.3.2}$$

A kernel function is considered valid when it meets the Mercer's Theorem (3).

Proof First of all, consider a differential displacement vector dz in the mapped space. We have to show that the norm of this displacement can be measured via a Riemannian metric. As a simplification, we will assume D finite. Observe that the squared norm of that differential vector is simply the inner product

$$\|dz\|^2 = dz \cdot dz. \tag{3.3.3}$$

In order to express dz, we realize that any point z in the manifold is in fact obtained from a point x in the input space through $\Phi(x)$. So, it can be shown that a differential dz can be obtained as

$$z = \Phi(x) \quad , \quad dz = \nabla\Phi(x)dx. \tag{3.3.4}$$

Now, to compute the gradient of $\Phi(x)$, observe that we can decompose it into D functions $\phi_i(x)$, each one providing the position of the mapped point of x in one of the dimensions of the feature space:

$$\Phi(x) = \begin{pmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_D(x) \end{pmatrix}, \tag{3.3.5}$$

and so the gradient would be

$$\nabla\Phi(x) = \begin{pmatrix} \frac{\partial\phi_1(x)}{\partial x_1} & \cdots & \frac{\partial\phi_1(x)}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial\phi_D(x)}{\partial x_1} & \cdots & \frac{\partial\phi_D(x)}{\partial x_d} \end{pmatrix}. \tag{3.3.6}$$

In this way dz will take the form

$$\begin{aligned} dz &= \nabla\Phi(x)dx = \begin{pmatrix} \sum_{i=1}^{d} \frac{\partial\phi_1(x)}{\partial x_i} dx_i \\ \vdots \\ \sum_{i=1}^{d} \frac{\partial\phi_D(x)}{\partial x_i} dx_i \end{pmatrix} \\ &= \sum_{i=1}^{d} \frac{\partial\Phi(x)}{\partial x_i} dx_i, \end{aligned} \tag{3.3.7}$$

because

$$\frac{\partial \Phi(x)}{\partial x_i} = \begin{pmatrix} \frac{\partial \phi_1(x)}{\partial x_i} \\ \vdots \\ \frac{\partial \phi_D(x)}{\partial x_i} \end{pmatrix}. \tag{3.3.8}$$

Now that we have a way to express dz as a function of Φ we can compute the norm of dz using it:

$$\begin{aligned} \|dz\|^2 &= dz \cdot dz = \sum_{i=1}^{d} \frac{\partial \Phi(x)}{\partial x_i} dx_i \cdot \sum_{i=1}^{d} \frac{\partial \Phi(x)}{\partial x_i} dx_i \\ &= \sum_{i=1}^{d} \sum_{j=1}^{d} \frac{\partial \Phi(x)}{\partial x_i} \cdot \frac{\partial \Phi(x)}{\partial x_j} dx_i dx_j \\ &= \sum_{i=1}^{d} \sum_{j=1}^{d} g_{ij} dx_i dx_j \end{aligned} \tag{3.3.9}$$

where

$$g_{ij} = \frac{\delta \Phi(x)}{\delta x_i} \cdot \frac{\delta \Phi(x)}{\delta x_j}, \tag{3.3.10}$$

and so we have an expression for g_{ij}. Now, to prove the statement of the theorem we simply substitute the expression

$$\begin{aligned} \frac{\partial^2}{\partial x_i \partial x'_j} k(x, x') \mid_{x'=x} &= \frac{\partial^2}{\partial x_i \partial x'_j} \Phi(x) \cdot \Phi(x') \mid_{x'=x} \\ &= \frac{\partial \Phi(x)}{\partial x_i} \cdot \frac{\partial \Phi(x')}{\partial x'_j} \mid_{x'=x} \\ &= g_{ij}. \end{aligned} \tag{3.3.11}$$

□

3.3.3 Geometry induced by the Gaussian kernel

As in this work we are going to use the Gaussian kernel (2.1.29) as a yardstick to compare the performance of the studied methods, we will now show how to obtain the Riemannian tensor for it. First, consider only a generic kernel in the form

$$k(x, x') = f\left(\frac{1}{2}\|x - x'\|^2\right). \tag{3.3.12}$$

If we compute the Riemannian tensor for this kernel, we have

$$\begin{aligned}
g_{ij} &= \frac{\partial^2}{\partial x_i \partial x'_j} f\left(\frac{1}{2}\|x - x'\|^2\right) |_{x'=x} \\
&= \frac{\partial^2}{\partial x_i \partial x'_j} f\left(\frac{1}{2}\sum_{k=1}^{d}(x_k - x'_k)^2\right) |_{x'=x} \\
&= \frac{\partial}{\partial x'_j} f'\left(\frac{1}{2}\sum_{k=1}^{d}(x_k - x'_k)^2\right)(x_i - x'_i) |_{x'=x} \\
&= -f''\left(\frac{1}{2}\sum_{k=1}^{d}(x_k - x'_k)^2\right)(x_j - x'_j)(x_i - x'_i) |_{x'=x} - f'\left(\frac{1}{2}\sum_{k=1}^{d}(x_k - x'_k)^2\right)\delta_{ij} |_{x'=x} \\
&= -f'(0)\delta_{ij}, \qquad (3.3.13)
\end{aligned}$$

where δ_{ij} stands for the Kronecker's delta (2.2.12). In the particular case of the Gaussian kernel, we have

$$f(z) = e^{-\frac{z}{\sigma^2}}, \qquad (3.3.14)$$

and so the Riemannian tensor is

$$\begin{aligned}
g_{ij} &= -f'(0)\delta_{ij} && (3.3.15) \\
&= \left[-e^{\frac{z}{\sigma^2}}\frac{1}{\sigma^2}\right] |_{z=0}\ \delta_{ij} && (3.3.16) \\
&= \frac{\delta_{ij}}{\sigma^2}. && (3.3.17)
\end{aligned}$$

The obtained metric is Euclidean, although the image of Φ might be curved [19]. Note that the parameter of the kernel σ heavily affects the induced geometry. In particular, if $\sigma \simeq 0$ then all the points are located at large distances from each other, while on the other hand if $\sigma \longrightarrow \infty$ all the points in the input space are mapped to the same point in the feature space. In this way the choice of σ affects the model learnt by the SVM, as the feature space will be different. Intuitively, a good feature space would be the one in which a large margin can be obtained, while the class points are not very scattered; that is, we would have large class separability but small intra-class variability. We will further explore this idea later.

3.3.4 Conformal kernel transformations

Although the parameters of a kernel function allow the manipulation of the geometry of the feature space up to some point, greater adaptability can be obtained by applying conformal transformations. A conformal transformation of a geometrical space could be defined as a function that maps that space into a new one in which the angles between curves are locally preserved.

A proper transformation could be able to turn a non–linearly separable problem into a separable one, thus obtaining a better model. An example of a conformal transformation is depicted in figure 3.3.2 [2], where a grid of lines is transformed into another space. In the new space the curves still have angles of 90^o.

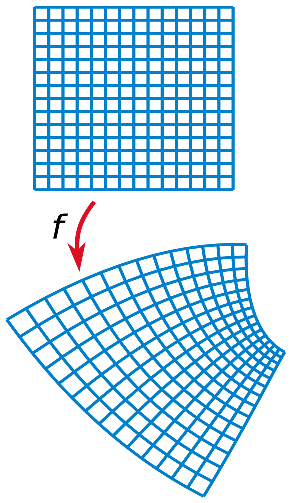

Figure 3.3.2: An example of a conformal transformation. The space is adapted so that it takes a new shape, but locally the angles between curves are still the same.

The idea is to obtain a new kernel function $\tilde{k}(x, x')$ by a conformal transformation of the original $k(x, x')$. Specifically, we will use a $\tilde{k}(x, x')$ in the form

$$\tilde{k}(x, x') = c(x)c(x')k(x, x'), \tag{3.3.18}$$

where c is an scalar positive function, which we will call the transformation function. $\tilde{k}$ is chosen in this way in order to be able to break down the kernel function as

$$\tilde{k}(x, x') = \tilde{\Phi}(x) \cdot \tilde{\Phi}(x') = c(x)\Phi(x) \cdot c(x')\Phi(x'). \tag{3.3.19}$$

It can be proved that this kind of transformation is conformal, and what is more, the resulting $\tilde{k}$ meets Mercer's Theorem requirements if k meets them [20].

Now we will see how the feature space is affected by the conformal transformation. If we call $\tilde{g}_{ij}$ the Riemannian tensor after the transformation, we have

Theorem 5 *When k is the Gaussian kernel* (2.1.29) *and under a conformal transformation of the form* (3.3.19)*, the metric in the transformed space is*

$$\tilde{g}_{ij} = c_i(x)c_j(x) + c(x)^2 g_{ij}(x), \tag{3.3.20}$$

[2]Original image by Oleg Alexandrov (`http://en.wikipedia.org/wiki/Image:Conformal_map.svg`)

where $c_i(x) = \frac{\partial c(x)}{\partial x_i}$.

Proof

$$\begin{aligned}
\tilde{g}_{ij} &= \frac{\partial^2}{\partial x_i \partial x'_j} c(x)c(x')k(x,x')|_{x'=x} \\
&= \frac{\partial}{\partial x'_j}\left[c_i(x)c(x')k(x,x') + c(x)c(x')\frac{\partial}{\partial x_i}k(x,x')\right]|_{x'=x} \\
&= c_i(x)c_j(x')k(x,x')|_{x'=x} + c_i(x)c(x')\frac{\partial}{\partial x'_j}k(x,x')|_{x'=x} + \\
&\quad +c(x)c_j(x')\frac{\partial}{\partial x_i}k(x,x')|_{x'=x} + c(x)c(x')\frac{\partial^2}{\partial x_i \partial x'_j}k(x,x')|_{x'=x}. \qquad (3.3.21)
\end{aligned}$$

Now note that when performing the substitution $x' = x$ the expression is simplified in a similar way as the one made in (3.3.13). By substituting in the first three terms we get

$$\begin{aligned}
\tilde{g}_{ij} &= c_i(x)c_j(x') \times 1 + c_i(x)c(x') \times 0 + c(x)c_j(x) \times 0 + c(x)c(x')\frac{\partial^2}{\partial x_i \partial x'_j}k(x,x')|_{x'=x} \\
&= c_i(x)c_j(x) + c(x)^2 g_{ij} \qquad (3.3.22)
\end{aligned}$$

□

Therefore the transformation function $c(x)$ actively modifies the metric in the induced feature space. In this way, if we have a fixed kernel function k giving poor classification results, we might be able to improve it by selecting a suitable transformation $c(x)$, so that the new kernel $\tilde{k}$ provides a “better” feature space and hence a better classification.

Finally note that, in fact, we do not need to actually know the kernel function at training time if we have the kernel matrix, which is defined as

$$K_{ij} = k(x^i, x^j). \qquad (3.3.23)$$

Then the kernel matrix corresponding to the modified kernel function $\tilde{k}$ can be computed as

$$\tilde{K}_{ij} = c(x^i)c(x^j)K_{ij}, \qquad (3.3.24)$$

and a SVM could be trained by directly using this kernel matrix.

In the next chapter we will review some of the proposed transformation functions and the methods to choose them available in the literature.

Chapter 4

Previous work

In this chapter we shall review the proposed methods in the literature to conformally transform the kernel function in order to obtain a better classification rate. While some of these proposals are aimed to alleviate the problems produced by unbalanced datasets, other are thought to be applicable under a general scenario. Nevertheless all these methods are based on the previously explained theory, each one choosing a particular transformation function $c(x)$ and showing how to choose their parameters.

4.1 Augmenting the margin via conformal transformations through SV

Amari and Si Wu [19] were the first to propose the use of conformal transformations to obtain a data-dependent kernel function. The main idea in their work is that the transformation function $c(x)$ should augment the spatial resolution near the SVM frontier, and reduce it far away from the frontier; in this way a better margin might be obtained by a SVM trained with the new kernel function.

First of all, this augmentation of the spatial resolution has to be formalized in some way. For this task, the Riemannian volume is used, which is defined as

$$dV(x) = \sqrt{|g(x)|}dx_1 \dots dx_d = \mathcal{V}(x)dx_1 \dots dx_d, \tag{4.1.1}$$

where $g(x)$ is the Riemannian tensor in the point x, and $|\cdot|$ is the determinant. This Riemannian volume expresses how a local area in the input space is augmented (or compressed) in the feature space due to the transformation Φ. $\mathcal{V}(x) = \sqrt{|g(x)|}$ is called the magnification factor, and quantifies how much the volume is augmented in the feature space. Hence it proves to be useful in order to measure the augmentation in a particular area of the input space under the transformation given by the initial kernel k.

Now we define the magnification factor after the transformation as

$$\tilde{\mathcal{V}}(x) = \sqrt{|\tilde{g}(x)|}, \tag{4.1.2}$$

where $\tilde{g}(x)$ is the Riemannian tensor in x after the conformal transformation. By comparing the ratio of both magnification factors we can get a measure of the change in spatial resolution made by the conformal transformation:

$$R = \frac{\tilde{\mathcal{V}}}{\mathcal{V}}, \tag{4.1.3}$$

which we shall call magnification ratio. If the spatial resolution of the neighbourhood of a point x is augmented after the conformal transformation we will have $\tilde{\mathcal{V}} > \mathcal{V} \rightarrow R > 1$, or in the other hand if it is reduced we will have $R < 1$.

As stated before, we would like to have $R > 1$ near the SVM separating hyperplane, so that in the new feature space we can obtain a larger margin. But also we would like to have $R < 1$ far away from the separating hyperplane, since if all the feature space is augmented in the same way no improvements can be made, as the SVM will be working in a similar space but with a different scaling, obtaining essentially the same separating hyperplane.

The main problem of this approach is that the separating hyperplane is not explicitly known. Hence, Amari and Wu propose to use the support vectors as a base for the transformation function $c(x)$, since SVs are most likely to be located in the margins of the SVM. In this way the proposed transformation function is

$$c(x) = \sum_{i \in SV} \alpha_i e^{-\frac{\|x - x^i\|^2}{2\tau^2}}, \tag{4.1.4}$$

where α_i are the support weights and τ is a parameter of the function. With this kind of transformation function it can be shown [19] that the magnification factor after the transformation in the neighbourhood of a SV x^i is

$$\tilde{\mathcal{V}}(x) \simeq \frac{\alpha_i^d}{\sigma^d} e^{-\frac{d r_i^2}{2\tau^2}} \sqrt{1 + \frac{\sigma^2}{\tau^2} r_i^2}, \tag{4.1.5}$$

where $r_i^2 = \|x - x^i\|^2$, and d is the dimension of the input space. By studying the behaviour of the derivative of $\tilde{\mathcal{V}}(x)$ the authors estimate that the optimal value for the parameter τ is

$$\tau \simeq \frac{\sigma}{\sqrt{d}}, \tag{4.1.6}$$

which is chosen in this way in order to maximize $\tilde{\mathcal{V}}(x)$ near the support vectors, and at the same time to keep the augmentation restricted to a local area.

Algorithm 1 Amari's Method

```
1: Train a SVM $svm$ with kernel matrix $K$ and the patterns in $X$.
2: Compute $\tau$ using (4.1.6).
3: Compute $c(x)\ \forall\, x \in X$ using (4.1.4).
4: Compute the new kernel matrix $\tilde{K}$ using $c(x)$ and $K$.
5: Train a new SVM $s\tilde{v}m$ with kernel matrix $\tilde{K}$ and the patterns in $X$.
6: return $s\tilde{v}m$.
```

Summing up the method, an algorithmic representation is shown in algorithm 1. Once the "transformed" SVM is obtained after the process, predictions can be made using this SVM and the modified kernel function $\tilde{k}$ (not the kernel matrix, as it only contains info about the training patterns), as

$$\begin{aligned} f(x) &= \sum_{i=1}^{N} \tilde{\alpha}_i y_i \tilde{k}(x, x^i) + \tilde{b} \\ &= \sum_{i=1}^{N} \tilde{\alpha}_i y_i c(x) c(x^i) k(x, x^i) + \tilde{b}, \end{aligned} \tag{4.1.7}$$

where the $\tilde{\alpha}_i$ stand for the support weights of the transformed SVM and $\tilde{b}$ for its bias. Note that we have to compute the transformation function for an unseen pattern x, hence all the parameters of the algorithm (α_i,$\tilde{\alpha}_i$,$\tilde{b}$,τ) should be available at prediction time. As we will see, this subtle detail may create difficulties in more complex algorithms.

4.2 An iterative approach

Another paper by Si Wu and Amari [21] extends the original one, proposing a generalization of the transformation function and an iterative way to concatenate conformal transformations. First, the transformation function is made more general by removing the support weights α_i from the expression, and by using a different τ_i for each support vector, resulting in

$$c(x) = \sum_{i \in SV} e^{-\frac{\|x - x^i\|^2}{\tau_i^2}}. \tag{4.2.1}$$

The use of individual factors for each support vector is introduced to solve a pathological problem of the original method: when support vectors appear in higher density in some areas of the space, the magnification factor after the transformation is prone to be excessively high; that is, it is not uniform along the separating boundary as desired. These τ_i factors are intended to fix this problem by choosing their values as a function of the density of support vectors:

$$\tau_i^2 = \frac{1}{M} \sum_{q \in \alpha_M} \|x^q - x^i\|^2, \tag{4.2.2}$$

where α_M is the set of the M support vectors nearest to x^i, and M is a parameter of the method.

Secondly, a way to iterate conformal transformations is proposed. This iteration is based in training a SVM with a kernel function, modify it using $c(x)$ and train the SVM again, until a number of iterations is reached. The method is shown in algorithm 2.

Algorithm 2 Si Wu's Method

```
1: Train a SVM $svm_0$ with kernel matrix $K_0 = K$ and the patterns in $X$.
2: for $t = 1$ to $MAX$ do
3:     Compute $\tau_i^t$ $\forall i \in SV$ using (4.2.2).
4:     Compute $c_{t-1}(x)$ $\forall\, x \in X$ using (4.2.1).
5:     Compute the new kernel matrix $K_t$ using $c_{t-1}(x)$ and $K_{t-1}$.
6:     Train a new SVM $svm_t$ with kernel matrix $K_t$ and the patterns in $X$.
7: end for
8: return $svm_{MAX}$.
```

Although the training algorithm seems reasonable, problems arise at prediction time when trying to classify an unseen point x as

$$f(x) = \sum_{i=1}^{N} \alpha_i^{MAX} y_i k_{MAX}(x, x^i) + b_{MAX}, \tag{4.2.3}$$

where the α_i^{MAX} stand for the support weights of the SVM trained in the last iteration of the algorithm, and b_{MAX} for its bias. The problem in this equation is that $k_{MAX}(x, x^i)$ is not known. However, note that it can be obtained using

$$\begin{aligned} k_{MAX}(x, x^i) &= c_{MAX-1}(x) c_{MAX-1}(x^i) k_{MAX-1}(x, x^i) && (4.2.4) \\ &= c_{MAX-1}(x) \dots c_0(x) c_{MAX-1}(x^i) \dots c_0(x^i) k(x, x^i), && (4.2.5) \end{aligned}$$

that is, a recursive formula for $k_{MAX}(x, x^i)$ appears, with $k_0(x, x^i) = k(x, x^i)$ as the base case. Therefore a way to compute the obtained kernel function for an unseen pattern exists, but we will have to store all the c transformation coefficients generated in the training procedure. In order to simplify the notation, we define the cumulative transformation coefficients d as

$$d_t(x^i) = \prod_{j=0}^{t} c_j(x^i). \tag{4.2.6}$$

Algorithm 3 Si Wu's Modified Method

```
1: Train a SVM $svm_0$ with kernel matrix $K_0 = K$ and the patterns in $X$.
2: Initialize $d(x) = 1 \, \forall \, x \in X$.
3: for $t = 1$ to $MAX$ do
4:     Compute $\tau_i \, \forall i \in SV$ using (4.2.2).
5:     Compute $c_{t-1}(x) \, \forall \, x \in X$ using (4.2.1).
6:     Compute $d(x) = d(x)c_{t-1}(x)$, $d_{t-1}(x) = d(x) \, \forall \, x \in X$.
7:     Compute the new kernel matrix $K_t$ using $c_{t-1}(x)$ and $K_{t-1}$.
8:     Train a new SVM $svm_t$ with kernel function $K_t$ and the patterns in $X$.
9: end for
10: return $svm_{MAX}$.
```

In this way we can rewrite the method as shown in algorithm 3. With this modification now we could classify an unseen pattern as

$$f(x) = \sum_{i=1}^{N} \alpha_i^{MAX} y_i d_{MAX}(x) d_{MAX}(x^i) k(x, x^i) + b_{MAX}; \tag{4.2.7}$$

however $d_{MAX}(x)$ is unknown a priori, but it can be computed if all the parameters used in the training method (τ_i^t,α_i^t) are stored, using

$$d_t(x) = \prod_{j=0}^{t} c_j(x), \tag{4.2.8}$$

where the $c_j(x)$ involve the mentioned parameters. As stated in the previous section, these kind of methods are prone to present difficulties at prediction time due to the amount of parameters that have to be stored, as in this case.

4.3 Using conformal kernel transformations to get rid of imbalanced datasets

Gang Wu and Chang [15, 16] proposed the idea of applying Amari and Si Wu methods to imbalanced datasets. Their method, known as Adaptive Conformal Transformation (ACT) gives more "radius" in the transformation to the patterns of the minority class, so that the spatial resolution is enlarged far away from the minority class. This is consistent with the skewness of the SVM boundary, since the ideal data boundary should be closer to the majority class. An example of how the area of maximum spatial augmentation is not the ideal data boundary is depicted in figure 4.3.1.

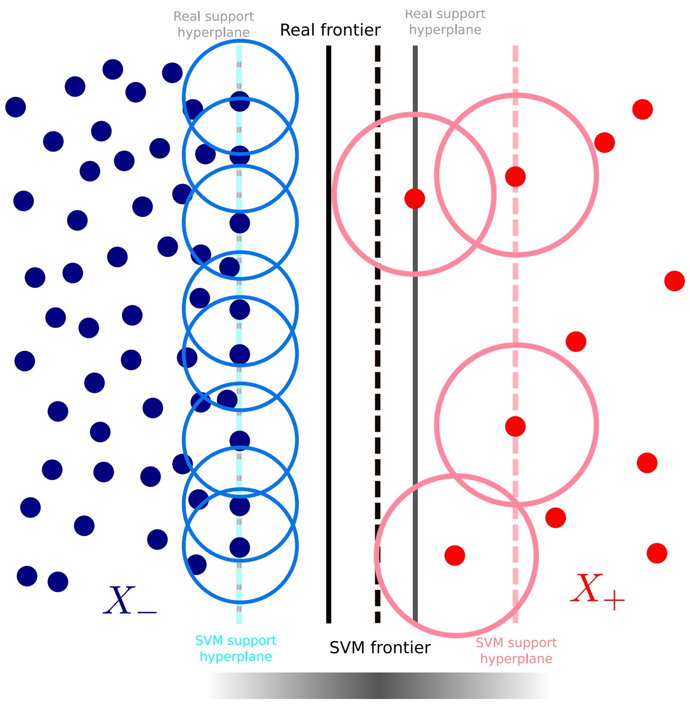

Figure 4.3.1: An example of Si Wu's method in an unbalanced dataset. The magnitude of the τ_i parameters of each SV is depicted as a radius around the SV. An schematic representation of spatial augmentation is depicted as a gray gradient at the bottom, where darker areas are those of most augmentation (obviously this is a simplification, since augmentation would be two-dimensional). As observed the area of maximum augmentation is near the SVM boundary, even if the smaller density of positive SV gives them a larger τ_i.

To properly choose the τ_i coefficients in order to deal with this skewness two modifications are introduced. The first one comes from the fact that when choosing these coefficients the density of SV in the input space is considered, while it would be better to measure this density in the feature space. Hence the τ_i are chosen as

$$\tau_i^2 = AVG_{j \in \{\|\Phi(x^i)-\Phi(x^j)\|^2 < M_i \, , \, y_i \neq y_k\}} \left\{ \|\Phi(x^i) - \Phi(x^j)\|^2 \right\}, \tag{4.3.1}$$

with

$$M_i = AVG_{i=\{SV^{MAX}, SV^{MIN}\}} \left\{ \|\Phi(x^i) - \Phi(x^j)\|^2 \right\}, \tag{4.3.2}$$

where SV^{MAX} and SV^{MIN} are the farthest and nearest SV to x^i.

The other modification explicitly tries to compensate the frontier skewness by correcting the τ_i values by a quantity that depends on the class of the SV, namely

$$\left\{ \begin{array}{c} \tilde{\tau}_i^2 = \eta_+ \tau_i^2 \ , \ y_i = 1 \\ \tilde{\tau}_i^2 = \eta_- \tau_i^2 \ , \ y_i = -1 \end{array} \right\}, \tag{4.3.3}$$

where η_+ and η_- stand for the correction factors, which should be taken as

$$\eta_+ = O\left(\frac{|SV^-|}{|SV^+|}\right) \quad , \quad \eta_- = O\left(\frac{|SV^+|}{|SV^-|}\right),$$

with SV^+ and SV^- the number of positive and negative support vectors. In this way the SV of the minority class will have a greater radius τ_i in the transformation, and so the area of maximum augmentation is more likely to be close to the ideal data frontier. An example of this method is shown in figure 4.3.2.

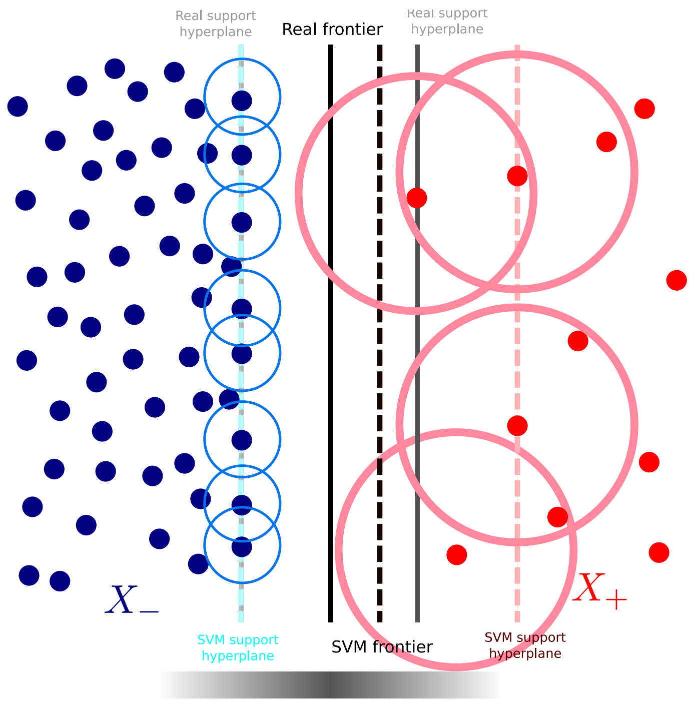

Figure 4.3.2: An example of ACT in an unbalanced dataset. The magnitude of the $\tilde{\tau}_i$ parameters of each SV is depicted as a radius around the SV. An schematic representation of spatial augmentation is depicted as a gray gradient at the bottom, where darker areas are those of most augmentation. Compared with figure 4.3.1, the support vectors of the majority class have now a reduced influence, while those of the minority class have a larger radius. This results in a displacement of the area of maximum spatial resolution augmentation, now more due to lie near the ideal data frontier.

Therefore, the transformation function is modified to use these corrected factors:

$$c(x) = \sum_{i \in SV} e^{-\frac{\|x - x^i\|^2}{\tilde{\tau}_i^2}}. \tag{4.3.4}$$

Algorithm 4 ACT

```
1: Train a SVM $svm_0$ with kernel matrix $K_0 = K$ and the patterns in $X$.
2: Obtain the misclassification error of $svm_0$ over $V$: $\epsilon_{new}$.
3: Initialize $d(x) = 1 \, \forall \, x \in X$ , $t = 1$, $\epsilon_{old} = \infty$
4: while $\epsilon_{old} - \epsilon_{new} > \theta$ && $t < MAX$ do
5:     Compute $\tau_i \, \forall i \in SV$ using (4.3.1).
6:     Compute $\tilde{\tau}_i \, \forall i \in SV$ using (4.3.3).
7:     Compute $c_{t-1}(x) \, \forall \, x \in X$ using (4.3.4).
8:     Compute $d(x) = d(x)c_{t-1}(x)$, $d_{t-1}(x) = d(x) \, \forall \, x \in X$.
9:     Compute the new kernel matrix $K_t$ using $c_{t-1}(x)$ and $K_{t-1}$.
10:    $\epsilon_{old} = \epsilon_{new}$.
11:    Train a new SVM $svm_t$ with kernel matrix $K_t$ and the patterns in $X$.
12:    Obtain the misclassification error of $svm_t$ over $V$: $\epsilon_{new}$.
13:    $t++$.
14: end while
15: return $svm_{t-1}$.
```

The pseudocode of ACT is quite similar to one shown before for Si Wu's method, having some slight differences in the calculation of the τ_i factors, as detailed in algorithm 4. Also the stopping criterion has been modified in order to stop when no improvement is made over a validation set V. Again, all the training parameters and computed values must be stored in order to be able to classify unseen patterns.

4.4 Estimating the real data boundary

In another work by Gang Wu and Chang [20] a more refined method to deal with imbalance is proposed. This time an estimation of the location of the ideal boundary is made, and then artificial support vectors are created along this boundary, which will be the center of the transformation function. This method is known as the Kernel Boundary Alignment (KBA) algorithm, and we will review how it works.

First, an estimation of the ideal boundary must be made. To do this a corrected SVM function is created as

$$\tilde{f}(x^i) = f(x^i) + \eta \, , \; 0 \leq \eta \leq 1, \tag{4.4.1}$$

which clearly resembles the technique of boundary movement (section 3.2.1). To choose η an error criterion is used, in the form

$$\eta^* = \text{argmin}_\eta \sum_{i \in E_+} C_+(y_i \tilde{f}(x^i)) + \sum_{i \in E_-} C_-(y_i \tilde{f}(x^i)), \tag{4.4.2}$$

where E_+ and E_- are the sets of the misclassified patterns of X^+ and X^- (respectively) with $-1 \leq f(x) \leq 1$, and C_+ and C_- are error functions. These error functions can be chosen at will, but with the idea of imposing a heavier penalty to misclassifications in the minority class. As an example the authors propose using

$$\begin{aligned} C_+(y_i \tilde{f}(x^i)) &= e^{-y_i \tilde{f}(x^i)}, \\ C_-(y_i \tilde{f}(x^i)) &= ln(1 + e^{-y_i \tilde{f}(x^i)}), \end{aligned} \tag{4.4.3}$$

which have an exponential tendency to classification error when misclassification happens in the minority class, but a linear one when it happens in the majority class. Once η has been determined solving this minimization problem (i.e. with gradient descent methods), an interpolation factor β is computed:

$$\beta = \frac{1 + \eta^*}{2}. \tag{4.4.4}$$

With this interpolation factor a new set of points $\mathcal{S}^b$ is created using the equation

$$\Phi(x^b) = (1 - \beta)\Phi(x^+) + \beta\Phi(x^-), \tag{4.4.5}$$

where $(\Phi(x^+), \Phi(x^-))$ are any possible pair of support vectors in the margin belonging to X^+ and X^-, respectively. With this procedure an artificial set of support vectors is created along the estimated boundary, as interpolations of real support vectors. An example of this procedure is shown in figure 4.4.1.

Once this set has been created, the transformation function will be based on it:

$$c(x_i) = \frac{1}{|\mathcal{S}^b(i)|} \sum_{x^j \in \mathcal{S}^b(i)} e^{\frac{\|\Phi(x^i) - \Phi(x^j)\|^2}{\tau_i^2}}, \tag{4.4.6}$$

where $\mathcal{S}^b(i)$ is the dataset of points $x^j \in \mathcal{S}^b$ with $\|\Phi(x^i) - \Phi(x^j)\|^2 < M$, with M a parameter of the algorithm. This time the τ_i factors are obtained as

$$\tau_i^2 = AVG_{j \in \{\|\Phi(x^i) - \Phi(x^j)\|^2 < M\}} \left\{\|\Phi(x^i) - \Phi(x^j)\|^2\right\}. \tag{4.4.7}$$

Apart from the fact that the transformation function is based on the artificial support vectors, the method is quite similar to ACT, as shown in algorithm 5.

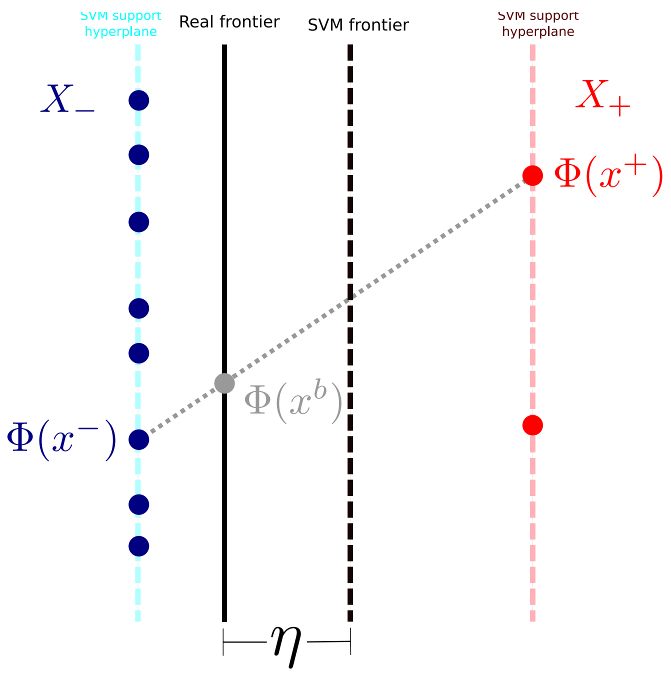

Figure 4.4.1: KBA estimates the ideal frontier by looking for a η that minimizes an error criterion. Once found, artificial support vectors are created in the intersections of the estimated boundary and each line joining opposite support vectors. In the figure one of these artificial support vectors is shown. Only the support vectors in the margin have been drawn.

4.5 Conformal transformations using the SVM function

Taking a different approach, Williams et al. [22] propose the use of a transformation function not based in the support vectors, but in the SVM output function itself, $f(x)$. In the end f determines the separating hyperplane of the SVM, and so augmenting the spatial resolution near the boundary can be made by incorporating f in the transformation function c in the following way

$$c(x) = e^{-\kappa f(x)^2}, \tag{4.5.1}$$

where κ is a parameter of the method. This transformation function has its maximum in those x with $f(x) = 0$, that is, in the boundary of the SVM, and decreases in an exponential way as we move far away from it. The recommended value for κ is

$$\kappa = \frac{1}{|f|_{max}}, \tag{4.5.2}$$

where $|f|_{max}$ is the maximum output value of the SVM for the training patterns, in absolute value. This transformation function is much simpler than the previously presented ones, although it is more costly to compute since the evaluation of $f(x)$ in each pattern x is quite expensive. Also,

Algorithm 5 KBA

```
1: Train a SVM $svm_0$ with kernel matrix $K_0 = K$ and the patterns in $X$.
2: Obtain the misclassification error of $svm_0$ over $V$: $\epsilon_{new}$.
3: Initialize $d(x) = 1 \; \forall \, x \in X$ , $t = 1$, $\epsilon_{old} = \infty$
4: while $\epsilon_{old} - \epsilon_{new} > \theta$ && $t < MAX$ do
5:     Compute $\eta^*$, $\beta$ using (4.4.2) and (4.4.4).
6:     Create the dataset of artificial support vectors using (4.4.5).
7:     Compute $\tau_i \; \forall i \in \mathcal{S}^b$ using (4.4.7).
8:     Compute $c_{t-1}(x) \; \forall \, x \in X$ using (4.4.6).
9:     Compute $d(x) = d(x)c_{t-1}(x)$, $d_{t-1}(x) = d(x) \; \forall \, x \in X$.
10:    Compute the new kernel matrix $K_t$ using $c_{t-1}(x)$ and $K_{t-1}$.
11:    $\epsilon_{old} = \epsilon_{new}$.
12:    Train a new SVM $svm_t$ with kernel matrix $K_t$ and the patterns in $X$.
13:    Obtain the misclassification error of $svm_t$ over $V$: $\epsilon_{new}$.
14:    $t + +$.
15: end while
16: return $svm_{t-1}$.
```

this method does not deal with imbalanced datasets, and it is not thought as an iterative procedure where several transformations may be applied; hence it is very similar to Amari's original method. The pseudocode for this method is shown in algorithm 6.

Algorithm 6 Williams' Method

```
1: Train a SVM $svm$ with kernel matrix $K$ and the patterns in $X$.
2: Compute $c(x) \; \forall \, x \in X$ using (4.5.1).
3: Compute the new kernel matrix $\tilde{K}$ using $c(x)$ and $K$.
4: Train a new SVM $s\tilde{v}m$ with kernel matrix $\tilde{K}$ and the patterns in $X$.
5: return $s\tilde{v}m$.
```

4.6 Computing class separability in the feature space

Finally, Xiong et al. [23] show a novel point to define a conformal transformation. They claim that instead of trying to maximize or minimize the spatial resolution in some areas of the space, these methods should pursue the maximization of the class separability in the feature space. This idea is far more intuitive than the former ones, since class separability is a well-known concept used by popular machine learning methods such as Fisher's Discriminant Analysis [4]. Although the feature space is hardly tractable due to the fact of only knowing about it through the kernel function, the authors show that some characteristics of this space can be used to compute a separability criterion.

An alternative feature space, named “empirical feature space” can be defined in such a way that shares some properties with the theoretical feature space induced by the kernel function, but which can be handled more easily. First suppose that the kernel matrix K has rank r. As it is symmetric and semidefinite positive (because the kernel function k meets Mercer’s Theorem) it can be decomposed into

$$K = P\Lambda P^T, \tag{4.6.1}$$

where P is an $N \times r$ matrix containing the eigenvectors of K, Λ is a $r \times r$ diagonal matrix with the eigenvalues of K in decreasing order, and T denotes transposition. Also P is a unitary matrix, that is, $P^T P = I$, I the unit matrix. Now an “empirical mapping” Φ_r^e is defined as

$$\Phi_r^e(x) = \Lambda^{-\frac{1}{2}} P^T (k(x, x^1), \ldots, k(x, x^N))^T. \tag{4.6.2}$$

This transformation maps the points in the input space to the empirical feature space. To see the similarity between this space and the theoretical feature space, a matrix Z is defined as

$$Z = KP\Lambda^{-\frac{1}{2}}, \tag{4.6.3}$$

which contains in each row the result of applying $\Phi_r^e(x)\,\forall\, x \in X$. Now observe that by multiplying ZZ^T we would obtain the kernel matrix of Φ_r^e for the patterns of X:

$$\begin{aligned} ZZ^T &= KP\Lambda^{-\frac{1}{2}}\Lambda^{-\frac{1}{2}}P^T K^T \\ &= P\Lambda P^T P\Lambda^{-1}P^T P\Lambda P^T \\ &= P\Lambda\Lambda^{-1}\Lambda P^T \\ &= P\Lambda P^T \\ &= K, \end{aligned} \tag{4.6.4}$$

that is, the kernel matrix in this empirical feature space and in theoretical one are the same. As in SVM-like algorithms only the elements of the kernel matrix are needed to perform training, then there is no difference in performing the training in the theoretical or in the empirical feature space. This will allow the computation of the separability criterion in the empirical feature space, which will remain valid for the theoretical feature space.

The separability criterion used by the authors is the Fisher’s J criterion, which is defined as

$$J = \frac{tr(S_B)}{tr(S_W)}, \tag{4.6.5}$$

where S_B is the between-class scatter matrix, S_W is the within-class scatter matrix, and tr is the trace function of a matrix. To write down the expression for these matrices some definitions must be made before. Suppose there are N_1 patterns in the negative class, and N_2 in the positive one. Then we define:

$$\begin{aligned} z^i &= \Phi_r^e(x^i), && (4.6.6)\\ \bar{z} &= \frac{1}{N}\sum_{i=1}^{N} z^i, && (4.6.7)\\ \bar{z_1} &= \frac{1}{N_1}\sum_{i\in X_-} z^i, && (4.6.8)\\ \bar{z_2} &= \frac{1}{N_2}\sum_{i\in X_+} z^i. && (4.6.9)\end{aligned}$$

Now the traces of the scatter matrices are defined as

$$\begin{aligned} tr(S_B) &= \frac{1}{N}\sum_{i=1}^{2} N_i(\bar{z}_i - \bar{z})(\bar{z}_i - \bar{z})^T, && (4.6.10)\\ tr(S_W) &= \frac{1}{N}\sum_{i=1}^{2}\sum_{j=1}^{N_i}(z_j^i - \bar{z}_i)(z_j^i - \bar{z}_i)^T, && (4.6.11)\end{aligned}$$

where z_j^i stands for the i-th pattern of class j. In order to make the notation easier, we would assume that the patterns in X are arranged in such a way that the patterns of the negative (first) class are first, and those belonging to the positive (second) class are after them. Then the kernel matrix can be decomposed in blocks as

$$K = \begin{pmatrix} \mathcal{K}_{11} & \mathcal{K}_{12} \\ \mathcal{K}_{21} & \mathcal{K}_{22} \end{pmatrix}, \qquad (4.6.12)$$

where $\mathcal{K}_{ij}$ is the block of the kernel matrix which only contains the kernel products of patterns of classes i and j. With this, the following theorem can be stated:

Theorem 6 *Having a kernel matrix in the form* (4.6.12)*, the separability induced by the kernel function in the feature space is computed as*

$$J = \frac{J_1}{J_2} = \frac{1_N^T B\, 1_N}{1_N^T W\, 1_N}, \qquad (4.6.13)$$

where

$$\begin{aligned} B &= K_{intra} + \frac{1}{N}K, & (4.6.14)\\ W &= diag(K) - K_{intra} & (4.6.15)\\ K_{intra} &= \begin{pmatrix} \frac{1}{N_1}\mathcal{K}_{11} & 0 \\ 0 & \frac{1}{N_2}\mathcal{K}_{22} \end{pmatrix} & (4.6.16) \end{aligned}$$

with $diag(A)$ the matrix containing the diagonal entries of A (and the rest zeros), and 1_N a vector of length N with all elements of 1 value. Furthermore, if a conformal transformation in the form 3.3.19 is applied to the kernel function, the separability in the transformed feature space can be computed as

$$\tilde{J} = \frac{\tilde{J}_1}{\tilde{J}_2} = \frac{q^T B q}{q^T W q}, \qquad (4.6.17)$$

where q is defined as a vector of length N containing the results of applying the transformation function to every pattern in the training set

$$q = \begin{pmatrix} c(x^1) \\ \vdots \\ c(x^N) \end{pmatrix}. \qquad (4.6.18)$$

As the proof is quite extensive and is detailed in [23] we will not repeat it here, since in this work we will only use the consequences of the theorem. The main advantage it provides is the ability to tune the parameters of the transformation function $c(x)$ so as to maximize the separability $\tilde{J}$. By computing the gradient of $\tilde{J}$ with respect to the transformation parameters, standard optimization techniques such as gradient ascent can be applied to obtain optimal parameters.

The authors define a transformation function similar to (4.2.1), as

$$\begin{aligned} c(x) &= c_0 + \textstyle\sum_{i=1}^{N_{core}} c_i k_1(x, a_i), & (4.6.19)\\ k_1(x, a_i) &= e^{-\gamma \|x - a_i\|^2}, & (4.6.20) \end{aligned}$$

where c_i are the parameters of the transformation, a_i are the “empirical cores”, N_{core} is the number of empirical cores and γ is a free parameter. The empirical cores are some chosen points in the input space around which the transformation will be centered. While they are randomly chosen as some of the training patterns, the c_i are found by maximizing the separability. γ is chosen using cross–validation techniques.

As some of our proposed methods are based in this “metamodel” or technique to find the optimal parameters, we will not further explain the work of Xiong et al., since the derivations are quite similar. For more details, see [23].

Chapter 5

Proposed methods

After reviewing the basic concepts of SVMs, conformal transformations and the works about the topic found in the literature, in this chapter we shall present our proposed methods. First we will talk about the general scenario to face when dealing with conformal transformation methods, next we will explain the "metamodels" or optimizers used to select the methods' parameters and finally we will show in detail our proposals.

5.1 General scenario

The core of any conformal transformation method is its transformation function $c(x)$. As shown, most of the authors of works in this area put a great deal of effort into finding an appropriate function, while the main idea of the methods remains the same: train a SVM, apply the conformal transformation and then retrain. The choosing of this function also involves defining a way to select its parameters, as bad parameter choices may provide bad results.

Usually the value for those parameters is chosen following a more or less "heuristic" rule, where no strictly mathematical reasons are given. Among the reviewed works, only Xiong et al. [23] provides a framework for this parameter selection. We believe that some improvements could be made by using a more intelligent search for their values, that is, by using a metamodel or optimizer able to fix those parameters using some criterion to minimize/maximize. This criterion may be for instance misclassification rate in a validation set, or class separability as in [23].

In this work shall we propose different methods which mix some of the reviewed proposals with new ideas. To find the optimal parameters for them we will use several metamodels, which we detail in the next section. Finally, in the next chapter, we will experimentally test these new methods along with some of the reviewed ones in order to extract some conclusions about the methodology of conformal transformations itself.

5.2 Metamodels

In order to perform an intelligent selection of the transformation function parameters two metamodels will be used: a deterministic one, based on Xiong et al.'s idea of maximizing the class separability, and a stochastic one based on the genetic algorithm CMA-ES [24]. In this section we will explain both ideas under a generic transformation $c(x)$.

5.2.1 Gradient ascent of the class separability

Although in Xiong's work [23] the method for the parameter selection is explained for an specific transformation function $c(x)$, theorem 6 in section 4.6 is applicable under any c. We detail here how to compute the gradient of the class separability in the feature space $\tilde{J}$ for a generic $c(x)$. Firstly, recall that $\tilde{J}$ can be expressed as $\tilde{J} = \frac{\tilde{J}_1}{\tilde{J}_2}$. Supposing w is the vector of parameters of $c(x)$, the gradient would be

$$\begin{aligned} \nabla_w \tilde{J} &= \frac{1}{\tilde{J}_2^2}\left(\tilde{J}_2 \nabla_w \tilde{J}_1 - \tilde{J}_1 \nabla_w \tilde{J}_2\right) \\ &= \frac{1}{\tilde{J}_2}\left(\nabla_w \tilde{J}_1 - \tilde{J} \nabla_w \tilde{J}_1\right). \end{aligned} \tag{5.2.1}$$

Now, the gradients of $\tilde{J}_1$ and $\tilde{J}_2$ are

$$\begin{aligned} \nabla_w \tilde{J}_1 &= \nabla_q \tilde{J}_1 \cdot \nabla_w q \\ &= 2q^T B \cdot \nabla_w q, \qquad (5.2.2) \\ \nabla_w \tilde{J}_2 &= \nabla_q \tilde{J}_2 \cdot \nabla_w q \\ &= 2q^T W \cdot \nabla_w q, \qquad (5.2.3) \end{aligned}$$

and so the gradient of the separability depends on the gradient of the transformation function in the following way:

$$\nabla_w \tilde{J} = \frac{2}{\tilde{J}_2}\left(q^T B \cdot \nabla_w q - \tilde{J} q^T W \cdot \nabla_w q\right). \tag{5.2.4}$$

Therefore, if the selected transformation function is differentiable with respect to its parameters, then the gradient of $\tilde{J}$ can be computed. However, usually the transformation function is somewhat complex and we cannot simply compute this derivative, solve it for the parameter vector w and hence obtain the optimal values that give the maximum separability. Even if the transformation function is simple with respect to the parameters, like the one proposed in Xiong (4.6.19), solving the equation requires the inversion of a matrix which is prone to be ill-posed. So, a step-wise approach is used.

This step-wise approach can be a gradient ascent method like the one used in [23], which consists in iterating steps in the form

$$w^{new} = w + \eta \nabla_w \tilde{J}, \tag{5.2.5}$$

where η is a step size parameter. Although this method is fair simple, choosing η is not straightforward as large values could make it ineffective, and small values might turn it to be too slow.

Instead, we will use the Davidon-Fletcher-Powell [25] method of maximization, which estimates the Hessian matrix of second derivatives of the objective function in order to have more information about the function's landscape. Furthermore, in each step it uses a linear maximization algorithm in order to find the maximum along the line, so no step size parameter is used.

This method is quite fast at finding a maximum in the separability function, but it does not have necessarily to be a global maximum. If the algorithm moves to a local maximum, the gradient of the function will be zero (not considering second derivatives) and no further movement could be performed. Hence, gradient ascent will find a good set of parameters to improve class separability, but they might not be the best.

5.2.2 An evolutionary search: CMA–ES

Other possible approach when dealing with a parameter optimization problem is to use a black-box optimizer, that is, an algorithm that does not make use of the derivatives of the objective function. Among these kind of techniques, genetic algorithms [26] are well known methods for solving a problem where no further information than the evaluation of the objective function at the tested points is provided.

In this work we shall make use of the popular Covariance Matrix Adaption - Evolutionary Strategy (CMA–ES) [24] genetic algorithm. This method generates random test points using a multivariate normal distribution, whose covariance matrix is adapted as the algorithm evolves in order to generate with a higher probability those points which follow the gradient of the function. As the real gradient is not known, it is estimated through the evaluated points during the algorithm.

Although the information provided by the derivatives is not used at all, CMA–ES is able to find directions in which the objective function is improved. The main disadvantage of this approach is that it is slower than a gradient ascent, since ascent directions are not known and have to be estimated. But on the other hand, CMA–ES may not get trapped inside local maxima, since the evolutionary algorithm continues its search even if a local maximum has been reached. In spite of not having a guarantee of reaching the global maximum in a finite number of iterations, the probability of finding it increases with the time spent in the algorithm.

Another advantage of using a black box optimizer is that other criterions (apart from class separability) whose derivative can not be computed could be used. As an example, CMA-ES could be configured to minimize misclassification error over a validation subset.

5.3 Conformal transformations along a corrected SVM boundary: MDT

Among the reviewed conformal functions we think that the simplest of all, while promising, is the Williams' transformation function (4.5.1). However, this transformation does not deal with imbalanced data sets, and then the spatial augmentation will be focused over a separating boundary which is biased. We thus propose to combine Williams' transformation with an skew correction method.

5.3.1 Correction by margin displacement

While the KBA method (section 4.4) tries to correct this skew by doing a bias displacement, we feel this approach is not appropriate. By moving the bias we are moving the whole SVM boundary and support hyperplanes, but the support hyperplane of the majority class is most likely to be well estimated, and hence there is no reason for moving it. A more adequate correction would be to move the support hyperplane of the minority class (which is skewed) towards the support hyperplane of the majority class, and then recalculate the separating boundary. A depiction of both scenarios is shown in figure 5.3.1.

Although the obtained boundary could be the same in both methods, the distance between the support hyperplanes can only be correctly adapted in our proposal. Hence the transformation function will not have the same effect over a boundary movement approach than over a margin displacement one, since the output function of the SVM will be different.

To perform this "margin displacement", we do the following. Suppose the SVM is trained by solving its equivalent CH-SVM problem (2.2.6). In order to clarify the notation, we will not use here the hat marks over the terms of the CH-SVM (which nevertheless could be transformed to its equivalent SVM). As explained in section 2.2.4 the location of the support hyperplanes is fixed by the position of the nearest points in the convex hulls, w_+ and w_-. As we want to displace the positive support hyperplane towards the negative one, we will need to move w_+ towards w_- as

$$\tilde{w}_+ = w_+ + (w_- - w_+)\lambda, \tag{5.3.1}$$

where $\lambda \in [0, 1)$ is a parameter that controls the displacement. The resultant boundary $\tilde{w}$ of the movement can be written as

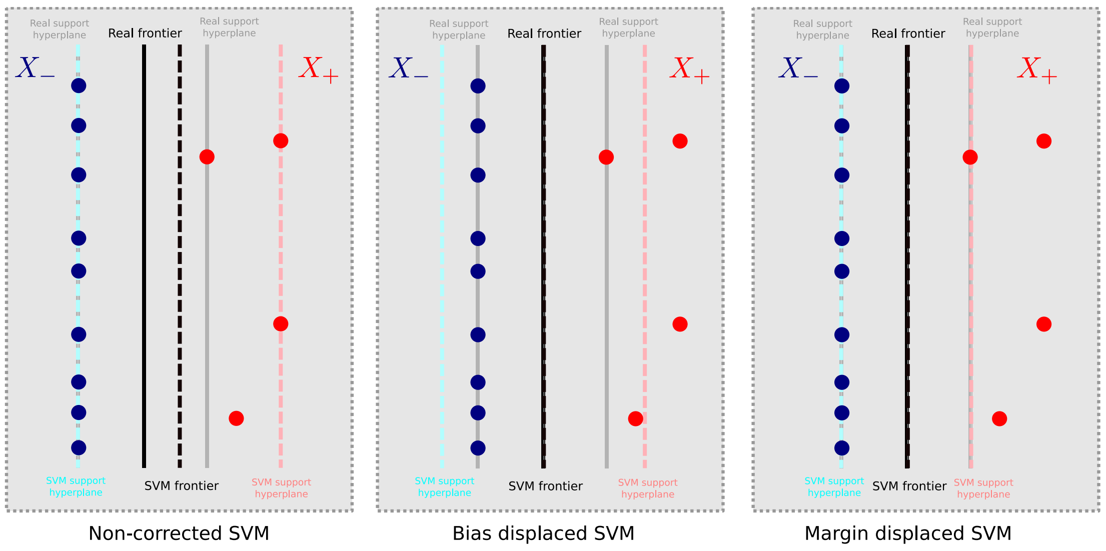

Figure 5.3.1: An illustration of two methods for computing a corrected separating boundary for the SVM. In the bias displacement case even if the boundary is corrected perfectly, the separating hyperplanes are not. However, in the margin displacement case by only moving the minority class support hyperplane a good estimation of both the boundary and the support hyperplanes could be achieved.

$$\begin{aligned}\tilde{w} &= \tilde{w}_+ - w_- = w_+ + (w_- - w_+)\lambda - w_- \\ &= w_+ + w_-\lambda - w_+\lambda - w_- = (1-\lambda)(w_+ - w_-) \\ &= \gamma w, \end{aligned} \tag{5.3.2}$$

with $\gamma = 1 - \lambda$. Now to calculate the bias of the new SVM frontier, $\tilde{b}$, we first have to define some quantities m_+ and m_- related to the support hyperplanes:

$$m_+ = w \cdot w_+, \tag{5.3.3}$$

$$m_- = w \cdot w_-. \tag{5.3.4}$$

Observe that the bias can be computed using these two quantities:

$$b = -\frac{1}{2} w \cdot (w_+ + w_-) = -\frac{1}{2}(m_+ + m_-). \tag{5.3.5}$$

Now if we compute these m_+ and m_- quantities after the displacement, we can write:

$$\begin{aligned}
\tilde{m_+} &= \tilde{w} \cdot \tilde{w}_+ = \gamma w \cdot \tilde{w_+} \\
&= \gamma w \cdot (w_+ + (w_- - w_+)\lambda) = \gamma w \cdot [w_+(1-\lambda) + \lambda w_-] \\
&= \gamma^2 m_+ + \gamma \lambda m_-, \qquad (5.3.6) \\
\tilde{m_-} &= \gamma w \cdot w_- = \gamma m_-. \qquad (5.3.7)
\end{aligned}$$

So the new bias $\tilde{b}$ is expressed as:

$$\begin{aligned}
\tilde{b} &= -\frac{1}{2}(\tilde{m_+} + \tilde{m_-}) = -\frac{1}{2}(\gamma^2 m_+ + \gamma\lambda m_- + \gamma m_-) \\
&= -\frac{1}{2}\gamma(\gamma m_+ + (\lambda+1)m_-) = -\frac{1}{2}\gamma(\gamma m_+ + (2-\gamma)m_-) \\
&= -\frac{1}{2}\gamma(\gamma m_+ + \gamma m_- - 2\gamma m_- + 2m_-) = \gamma^2 b + \gamma(\gamma m_- - m_-) \\
&= \gamma^2 b + \gamma(\gamma - 1)m_-. \qquad (5.3.8)
\end{aligned}$$

In this way the separating function of the "corrected" SVM will be

$$\tilde{f}(x) = \gamma w \cdot x + \gamma^2 b + \gamma(\gamma-1)m_- = \gamma(w \cdot x + \gamma b + (\gamma-1)m_-), \qquad (5.3.9)$$

which can be expressed in terms of the original $f(x)$ as

$$\tilde{f}(x) = \gamma(f(x) + (\gamma-1)(b + m_-)). \qquad (5.3.10)$$

Considering that, once the SVM has been trained, w, b and m_- can be regarded as constants, we have obtained an expression for a new SVM that only depends on the displacement parameter λ. This will allow us to perform the conformal transformation along the displaced boundary by simply using $\tilde{f}(x)$ instead $f(x)$ in the conformal transformation function, as will be shown next.

Finally, a fast computation for m_- can be made in the algorithm by observing that

$$\begin{aligned}
m_- &= w \cdot w_- = \sum_{i=1}^{N} \sum_{j \in C_2} \alpha_i \alpha_j k(x_i, x_j) y_i y_j \\
&= \sum_{i \in C_2} y_i \alpha_i \sum_{j=1}^{N} y_j \alpha_i \alpha_j k(x_i, x_j) \\
&= -\sum_{i \in C_2} \alpha_i (f(x_i) - b) \\
&= -\sum_{i \in C_2} \alpha_i f(x_i) + \sum_{i \in C_2} \alpha_i b \\
&= b \sum_{i \in C_2} \alpha_i - \sum_{i \in C_2} \alpha_i f(x_i) \\
&= b - \sum_{i \in C_2} \alpha_i f(x_i). \qquad (5.3.11)
\end{aligned}$$

5.3.2 Proposed method

Now that we have an estimation of the data boundary we will use Williams' choice as the transformation function, but centred on the corrected boundary instead of the standard SVM boundary, as

$$c(x) = e^{-\kappa \tilde{f}(x)^2} = e^{-\kappa(\gamma(w \cdot x + \gamma b + (\gamma - 1) m_-))^2}. \qquad (5.3.12)$$

$c(x)$ has its maximum along the corrected boundary, and decreases exponentially as we move away from it. The speed of this decrease is controlled by the parameter κ. Ideally we would like to perform an spatial resolution increase in the area between the two corrected support hyperplanes, and a decrease away from them, so we shall tune κ for that, but there is not an easy way to do it. This problem is also present in Williams' original method. Here we will find automatically the values for κ and γ using the metamodels explained before.

We shall name this method Margin Displacement Transformation (MDT). Its pseudocode is shown in algorithm 7.

5.3.3 Obtaining parameters values for MDT

Whenever we use gradient ascent as the metamodel for selecting the parameters, the gradients of $\tilde{J}$ with respect to those parameters have to be computed. We develop their calculations in this subsection. For the other metamodels there is no need to know these gradients, but nevertheless some interesting facts could be extracted from their calculations.

First, for the κ parameter we have

Algorithm 7 MDT

```
1: Train a SVM $svm_0$ with kernel matrix $K_0 = K$ and the patterns in $X$.
2: Obtain the misclassification error of $svm_0$ over $V$: $\epsilon_{new}$.
3: Initialize $d(x) = 1 \, \forall \, x \in X$ , $t = 1$, $\epsilon_{old} = \infty$
4: while $\epsilon_{old} - \epsilon_{new} > \theta$ && $t < MAX$ do
5:     Use a metamodel to obtain parameters $\kappa$ and $\lambda$.
6:     Compute $c_{t-1}(x) \, \forall \, x \in X$ using (5.3.12).
7:     Compute $d(x) = d(x)c_{t-1}(x)$, $d_{t-1}(x) = d(x) \, \forall \, x \in X$.
8:     Compute the new kernel matrix $K_t$ using $c_{t-1}(x)$ and $K_{t-1}$.
9:     $\epsilon_{old} = \epsilon_{new}$.
10:    Train a new SVM $svm_t$ with kernel matrix $K_t$ and the patterns in $X$.
11:    Obtain the misclassification error of $svm_t$ over $V$: $\epsilon_{new}$.
12:    $t++$.
13: end while
14: return $svm_{t-1}$.
```

$$[\nabla_\kappa q]_i = \frac{\partial c(x_i)}{\partial \kappa} = -e^{-\kappa \tilde{f}(x_i)^2} \tilde{f}(x_i)^2 = -\tilde{f}(x_i)^2 c(x_i), \tag{5.3.13}$$

$$\nabla_\kappa q = -\begin{pmatrix} \tilde{f}(x_1)^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{f}(x_N)^2 \end{pmatrix} q. \tag{5.3.14}$$

Defining

$$\tilde{F} = \begin{pmatrix} \tilde{f}(x_1) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{f}(x_N) \end{pmatrix}, \tag{5.3.15}$$

the gradient is compactly expressed as

$$\nabla_\kappa q = \tilde{F}\tilde{F}q. \tag{5.3.16}$$

The gradient with respect the other parameter to optimize (γ) can be obtained following a similar reasoning, noting that

$$\begin{aligned}
[\nabla_\gamma q]_i &= \frac{\partial c(x_i)}{\partial \gamma} = e^{-\kappa \tilde{f}(x_i)^2}(-\kappa)2\tilde{f}(x_i)(f(x_i)+(\gamma-1)(b+m_-))+\gamma(b+m_-) \\
&= e^{-\kappa \tilde{f}(x_i)^2}(-\kappa)2\tilde{f}(x_i)(f(x_i)+(2\gamma-1)(b+m_-)) \\
&= -2\kappa\, c(x^i)\tilde{f}(x_i)(f(x_i)+\tau), \qquad (5.3.17)
\end{aligned}$$

with $\tau = (2\gamma-1)(b+m_-)$. Then

$$\nabla_\gamma q = -2\kappa \begin{pmatrix} \tilde{f}(x_1)(f(x_1)+\tau) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{f}(x_N)(f(x_N)+\tau) \end{pmatrix} q. \qquad (5.3.18)$$

Again the derivative can be expressed more compactly by defining

$$\hat{F} = \begin{pmatrix} \tilde{f}(x_1)(f(x_1)+\tau) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{f}(x_N)(f(x_N)+\tau) \end{pmatrix}, \qquad (5.3.19)$$

and so we have

$$\nabla_\gamma q = -2\kappa \hat{F} q. \qquad (5.3.20)$$

Therefore the gradients have the form

$$\begin{aligned}
\nabla_\kappa \tilde{J} &= \frac{1}{\tilde{J}_2}\left(M_\kappa - N_\kappa \tilde{J}\right) \\
\nabla_\gamma \tilde{J} &= \frac{1}{\tilde{J}_2}\left(M_\gamma - N_\gamma \tilde{J}\right), \qquad (5.3.21)
\end{aligned}$$

where

$$\begin{aligned}
M_\kappa &= 2q^T B\tilde{F}\tilde{F}q, \\
N_\kappa &= 2q^T W\tilde{F}\tilde{F}q, \\
M_\gamma &= -4\kappa q^T B\hat{F}q, \\
N_\gamma &= -4\kappa q^T W\hat{F}q. \qquad (5.3.22)
\end{aligned}$$

Observe that there are redundant calculations in the procedure to obtain M_κ, M_γ, N_κ and N_γ. This is particularly evident is we write these down not using a matrix notation:

$$\begin{aligned}
M_\kappa &= -2q^T B\tilde{F}\tilde{F}q = -2\sum_{i=1}^{N}\sum_{j=1}^{N} c(x_i)c(x_j)B_{ij}\tilde{f}(x_i)^2 \\
&= -2\sum_{i=1}^{N}\tilde{f}(x_i)^2 c(x_i)\sum_{j=1}^{N} c(x_j)B_{ij}, \\
N_\kappa &= -2\sum_{i=1}^{N}\tilde{f}(x_i)^2 c(x_i)\sum_{j=1}^{N} c(x_j)W_{ij}, \\
M_\gamma &= -4\kappa\sum_{i=1}^{N}\tilde{f}(x_i)(f(x_i)+\tau)c(x_i)\sum_{j=1}^{N} c(x_j)B_{ij}, \\
N_\gamma &= -4\kappa\sum_{i=1}^{N}\tilde{f}(x_i)(f(x_i)+\tau)c(x_i)\sum_{j=1}^{N} c(x_j)W_{ij}.
\end{aligned}$$

We can avoid repeating the computation of some of these terms by defining

$$\begin{aligned}
\bar{B}_i &= c(x_i)\sum_{j=1}^{N} c(x_j)B_{ij}, \\
\bar{W}_i &= c(x_i)\sum_{j=1}^{N} c(x_j)W_{ij}; \qquad (5.3.23)
\end{aligned}$$

then we will have

$$\begin{aligned}
M_\kappa &= -2\sum_{i=1}^{N}\tilde{f}(x_i)^2\bar{B}_i, \\
N_\kappa &= -2\sum_{i=1}^{N}\tilde{f}(x_i)^2\bar{W}_i, \\
M_\gamma &= -4\kappa\sum_{i=1}^{N}\tilde{f}(x_i)(f(x_i)+\tau)\bar{B}_i, \\
N_\gamma &= -4\kappa\sum_{i=1}^{N}\tilde{f}(x_i)(f(x_i)+\tau)\bar{W}_i. \qquad (5.3.24)
\end{aligned}$$

Also observe that these quantities are helpful for computing $\tilde{J}_1$ and $\tilde{J}_2$ easily:

$$\begin{aligned}
\tilde{J}_1 &= \sum_{i=1}^{N}\bar{B}_i, \\
\tilde{J}_2 &= \sum_{i=1}^{N}\bar{W}_i,
\end{aligned}$$

which shall prove to be useful under any metamodel that needs to compute $\tilde{J}$.

5.3.4 Some remarks about the method

One might argue that MDT is a naive method, in the sense of that it assumes that the separating frontier found by the SVM is parallel to the real one, which is not likely to happen. If this is not the case, it is not clear whether the transformation could make any improvement. However, as in the original Williams' method [22] some improvements are claimed, we expect at least the same accuracy level. We shall test this in the experiments chapter.

5.4 Looking for optimal SV transformation weights: TSVT

Another method is proposed in this work, in order to analyse the influence of different kinds of transformation functions. This time an adaption of the function proposed by Gang Wu (4.2.1) is used, in the form

$$c(x) = \sum_{i \in SV} e^{-\bar{\tau}_i \|x - x^i\|^2}, \tag{5.4.1}$$

where

$$\bar{\tau}_i = \frac{1}{\tilde{\tau}_i^2}. \tag{5.4.2}$$

As stated before, we believe that a finer tuning of these parameters could lead to a better transformation, and so we use a metamodel to adjust them. The $\tilde{\tau}$ parameters have been inverted in the formula in order to avoid divisions by zero in the metamodel search.

The pseudocode of this method, which we shall name Tuned Support Vector Transformation (TSVT), is shown in algorithm 8. Although conceptually this method is simpler than MDT, in fact it is slower to compute as the number of parameters to be tuned by the metamodel is actually the number of support vectors, which depends on the number of patterns in the training set. However we would expect to obtain better results from it, since it provides a more general transformation. Similarly to the relationship between MDT and Williams' methods, we would also expect to obtain at least the same accuracy results with TSVT than with Gang Wu's method.

5.4.1 Obtaining parameters values for TSVT

Analogously to the MDT case, if we want to compute the gradients of the class separability with respect to the parameters of the transformation function of TSVT we must do some calculations. This time we have as many parameters to optimize as support vectors, since we have a τ_k for each

Algorithm 8 TSVT

```
1: Train a SVM $svm_0$ with kernel matrix $K_0 = K$ and the patterns in $X$.
2: Obtain the misclassification error of $svm_0$ over $V$: $\epsilon_{new}$.
3: Initialize $d(x) = 1 \, \forall \, x \in X$ , $t = 1$, $\epsilon_{old} = \infty$
4: while $\epsilon_{old} - \epsilon_{new} > \theta$ && $t < MAX$ do
5:     Use a metamodel to obtain parameters $\bar{\tau}_i$ for the Support Vectors.
6:     Compute $c_{t-1}(x) \, \forall \, x \in X$ using (4.2.1).
7:     Compute $d(x) = d(x)c_{t-1}(x)$, $d_{t-1}(x) = d(x) \, \forall \, x \in X$.
8:     Compute the new kernel matrix $K_t$ using $c_{t-1}(x)$ and $K_{t-1}$.
9:     $\epsilon_{old} = \epsilon_{new}$.
10:    Train a new SVM $svm_t$ with kernel matrix $K_t$ and the patterns in $X$.
11:    Obtain the misclassification error of $svm_t$ over $V$: $\epsilon_{new}$.
12:    $t++$.
13: end while
14: return $svm_{t-1}$.
```

one. The gradient of the class separability with respect to each parameter has a similar form than the one in MDT (5.3.21), namely

$$\nabla_{\bar{\tau}_k}\tilde{J} = \frac{1}{\tilde{J}_2}\left(M_{\bar{\tau}_k} - N_{\bar{\tau}_k}\tilde{J}\right), \tag{5.4.3}$$

where

$$\begin{aligned} M_{\bar{\tau}_k} &= \nabla_{\bar{\tau}_k}\tilde{J}_1 = 2q^T B \cdot \nabla_{\bar{\tau}_k} q, \\ N_{\bar{\tau}_k} &= \nabla_{\bar{\tau}_k}\tilde{J}_2 = 2q^T W \cdot \nabla_{\bar{\tau}_k} q. \end{aligned} \tag{5.4.4}$$

Supposing we want to compute the derivative of the transformation function over a pattern x^j with respect to a parameter τ_k, we have

$$\begin{aligned} \frac{\partial c(x^j)}{\partial \bar{\tau}_k} &= \sum_{i \in SV} \frac{\partial}{\partial \bar{\tau}_k} e^{-\bar{\tau}_i \|x^j - x^i\|^2} \\ &= -e^{-\bar{\tau}_k \|x^j - x^k\|^2} \|x^j - x^k\|^2, \end{aligned} \tag{5.4.5}$$

and so

$$\nabla_{\bar{\tau}_k} q = \begin{pmatrix} -e^{-\bar{\tau}_k \|x^1 - x^k\|^2} \|x^1 - x^k\|^2 \\ \vdots \\ -e^{-\bar{\tau}_k \|x^N - x^k\|^2} \|x^N - x^k\|^2 \end{pmatrix}. \tag{5.4.6}$$

Again, the gradients can be written as in (5.3.21)

$$\begin{aligned}
M_{\bar{\tau}_k} &= 2q^T B \nabla_{\bar{\tau}_k} q \\
&= 2\sum_{i=1}^{N}\sum_{j=1}^{N} B_{ij} c(x^i)(-e^{-\bar{\tau}_k \|x^j - x^k\|^2} \|x^j - x^k\|^2) \\
&= -2\sum_{j=1}^{N} e^{-\bar{\tau}_k \|x^j - x^k\|^2} \|x^j - x^k\|^2 \sum_{i=1}^{N} B_{ij} c(x_i) \\
&= -2\sum_{j=1}^{N} e^{-\bar{\tau}_k \|x^j - x^k\|^2} \|x^j - x^k\|^2 \frac{\bar{B}_j}{c(x^j)}, \\
N_{\tau_k} &= -2\sum_{j=1}^{N} e^{-\bar{\tau}^k \|x^j - x_k\|^2} \|x^j - x^k\|^2 \frac{\bar{W}_j}{c(x^j)}. \qquad (5.4.7)
\end{aligned}$$

This time these expressions are more complex, but nevertheless the calculation of $\bar{B}$ and $\bar{W}$ can be used to compute them as well as the class separability $\tilde{J}$.

Chapter 6

Experiments

In this chapter we shall put under test the proposed methods, as well as some of the reviewed ones for comparison purposes. A detail of these methods is shown in table 6.0.1. Along with the standard Support Vector Machine, three conformal transformation methods from the literature were implemented: Amari's original proposal, ACT and Williams' method. Regarding the proposed algorithms, both MDT and TSVT were implemented with three possible options for parameter selection: gradient ascent of the class separability, genetic ascent (CMA-ES) of the class separability, or genetic ascent of the g-means accuracy over a validation set.

SVM	Standard Support Vector Machine, without any conformal transformation.
AMA	Amari's original method (1).
ACT	Gang Wu's Adaptive Conformal Transformation (4).
WIL	Williams' method (6).
MDT–JGRAD	Margin Displacement Transformation (7) with gradient ascent on separability.
MDT–JMETA	Margin Displacement Transformation (7) with genetic ascent on separability.
MDT–VMETA	Margin Displacement Transformation (7) with genetic ascent on validation error.
TSVT–JGRAD	Tuned Support Vector Transformation (8) with gradient ascent on separability.
TSVT–JMETA	Tuned Support Vector Transformation (8) with genetic ascent on separability.
TSVT–VMETA	Tuned Support Vector Transformation (8) with genetic ascent on validation error.

Table 6.0.1: Methods tested in the experiments.

We will first explain the implementations made of these methods, next we shall show which datasets were used for the tests, and finally the results of the experiments will be presented.

6.1 Methodology

All the methods were coded in the C programming language. For the standard SVM, which is also the base of all the other methods, an implementation of the MDM algorithm for SVM classification was made [11]. Regarding the reviewed methods of the literature, as no public implementations are available (up to our knowledge), they were also coded along the new, proposed methods. The only module which was not necessary to be programmed was the CMA-ES algorithm, since a good implementation is downloadable from the author's webpage [27].

The Gaussian kernel (2.1.29) was used in all the experiments. Its $2\sigma^2$ parameter and the SVM penalty parameter C were chosen beforehand to a fixed value as $2\sigma^2 = 50$ and $C = 10$. As the tested methods are supposed to be able to optimize the kernel, these prefixed values should not appear as a major drawback (the values of these parameters are usually chosen by cross-validation techniques).

Regarding the implemented MDM algorithm, in fact an accelerated version is used, which was previously presented as part of another work. This implementation solves a Nearest Point Problem variant of the CH-SVM, which detects the appearance of cyclic updates along training and tries to avoid them. For further details, see [28].

About the CMA-ES algorithm, the default parameters of the implementation are used (see [27, 29]) with a prefixed number of generations of 30. The number of offsprings in each generation is automatically adjusted proportionally to the number of parameters to be optimized, which are two in MDT methods but variable in TSVT. This adjustment is automatically done by the CMA-ES formulae [24].

As we are dealing with imbalanced datasets, the g-means error criterion is used, which is more appropriate for this kind of problems, as a perfect classification in the majority class is useless if no pattern in the minority class is correctly classified. The g-means error is then defined in a similar way as the g-means accuracy (3.1.2), namely

$$g_{err} = 1 - \sqrt{a^+ a^-}. \tag{6.1.1}$$

Finally, all the tested conformal methods were adapted so as to be applicable in an iterative manner. Although most of the presented methods already include this capability, Amari's (algorithm 1) and Williams' (algorithm 6) only apply a unique transformation, and so they have been slightly modified to obtain a fair comparison. Also, all the methods keep a subset of the training patterns as a validation set in which to test the obtained accuracy after each transformation, in order to decide when the algorithm should stop applying transformations; this stopping criterion was first proposed in the ACT method (algorithm 4).

6.2 Experimental scenarios and datasets used

Two experimental scenarios were considered in this work:

- An artificial toy dataset where the separating boundaries found by the methods could be observed.
- Real problems datasets where class imbalance is present.

With these experiments we try to find out whether these methods are effective for alleviating the problems induced by the class imbalance, and whether they are also effective in improving the kernel function even if the dataset is not too much imbalanced. For the artificial scenario the checkers board problem previously presented is used. A detailed description of the datasets is shown in table 6.2.1. All the real problems datasets were obtained from the UCI repository [30]. For the datasets with more than two classes (Glass and Segment), the task presented to the SVM was to classify the smallest class (7 and 1, respectively) against the rest.

Dataset	N	N^+	N^-	d
Checkers board	2020	20	2000	2
Glass7	214	29	185	9
Segment1	2310	330	1980	19
Hepatitis	155	33	123	32
Sonar	208	97	111	60
Heart disease	297	137	160	13
Ionosphere	351	126	225	33
Wisconsin Breast Cancer Diagnosis	569	212	357	30

Table 6.2.1: Datesets used in the experiments. The total number of patterns (N), positive class patterns (N^+), negative class patterns (N^-) and input dimension (d) of the datasets are shown.

In the artificial scenario an instance of the checkers board problem was generated with 2000 patterns in the negative class and 20 in the positive class. This instance was used as the training set, generating another one with 20000 patterns in each class as the test set, uniformly spaced so as to observe the boundary computed by the methods.

On the other hand, when dealing with the real problems scenario a 7-fold cross-validation was performed on each dataset to estimate the generalization error. The folds were generated in a stratified manner, that is, maintaining the imbalance ratio that was present in the complete dataset. These decisions were taken in this way so as to follow as much as possible the experimental work presented in [15].

6.3 Results

6.3.1 Obtained results in the checkers board problem

In the checkers board problem we can observe a noticeable improvement when applying some of the conformal transformation methods, as is shown in table 6.3.1. However, in the remaining methods the error rate is unchanged. Recall that these methods use a validation set to test whether applying the transformation would produce any improvement, and they do not apply it if no advantage is obtained. Therefore it is not strange to observe that some of them generate the same error ratio as the standard SVM (no transformations are being applied).

Method	Misclassification rate	g-means error
SVM	0.45	0.69
AMA	0.15	0.16
ACT	0.14	0.16
WIL	0.45*	0.69*
MDT-JGRAD	0.45*	0.69*
MDT-JMETA	0.15	0.16
MDT-VMETA	0.45*	0.69*
TSVT-JGRAD	0.15	0.16
TSVT-JMETA	0.14	0.15
TSVT-VMETA	0.45*	0.69*

Table 6.3.1: Error levels obtained with the studied methods for the checkers board problem. Errors are measured in g-means error. An asterisk appears when the method decided not to apply any transformation, thus obtaining the same error rate as the SVM.

A depiction of the separating frontier found by the SVM is shown in figure 6.3.1. Observe how it does not follow the real frontier at all, as a separating hyperplane which classifies almost every point as belonging to the majority class has been chosen. The most reasonable explanation for this result is one of the three we pointed out in section 3.1: the SVM is intentionally incurring in misclassification of the points of the minority class to obtain a better margin. Thus, a model with poor generalization ability is obtained.

However, if we depict the separating frontier obtained after using TSVT-JMETA method, a better situation is observed (figure 6.3.2). Hence, the conformal transformation has been able to find a feature space in which the class imbalance is alleviated.

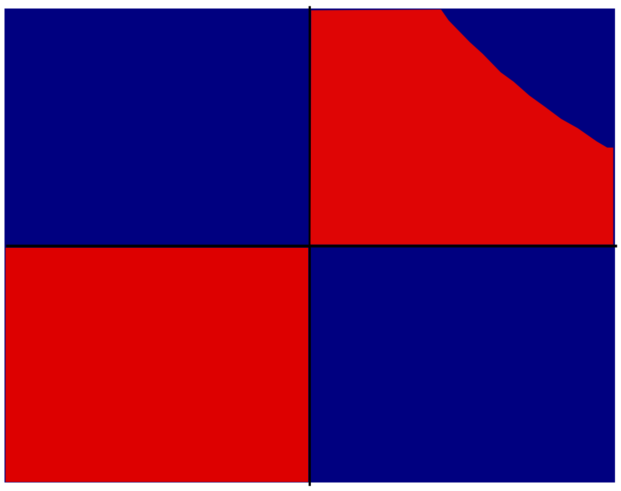

Figure 6.3.1: The results of training the SVM over the checkers board problem. The correctly classified region is depicted in blue, while the wrongly classified is depicted in red. So as to obtain a better margin the SVM is classifying mostly all the points as belonging to the majority class, placing the separating frontier in the top-right quadrant. Hence, a bad g-means error appears.

6.3.2 Obtained results in real problems datasets

In tables 6.3.2 and 6.3.3 the g-means errors obtained in each dataset for each method are shown. Marked in bold is the error rate of the best method. As it can be seen, results are not as promising as the ones obtained in the checkers board problem. The SVM without transformations is the best method in four of the seven datasets, while in the other three Amari's method is the best two times, while in the remaining the winner is MDT-VMETA. Regarding the magnitude of the improvement, only in the dataset Segment1 a great advantage is obtained when applying a conformal method. The best results on this and the Glass7 datasets are more or less in agreement with the ones performed in [15]. In spite of this, when applying the same methods to the rest of datasets only small improvements (if any) are obtained.

Analysing the obtained results in each dataset we can see the following. In Glass7 the SVM g-means error level seems to be good; however when trying to use the conformal methods, most of them decide not to apply any transformation. Only Amari's and ACT methods perform transformations, nevertheless the obtained errors are worse than in the SVM case. This fact may seem strange, as the methods do not apply any transformation if they think they will only worsen the accuracy of the SVM. However have in mind that a separate validation set is used for this decision, so it may well happen that a transformation which improves the accuracy on the validation set, worsens it on the test set. In the end no method is able to obtain a better model, although applying a Wilcoxon rank test [31] at the 10 % level shows that there is no statistical difference between them.

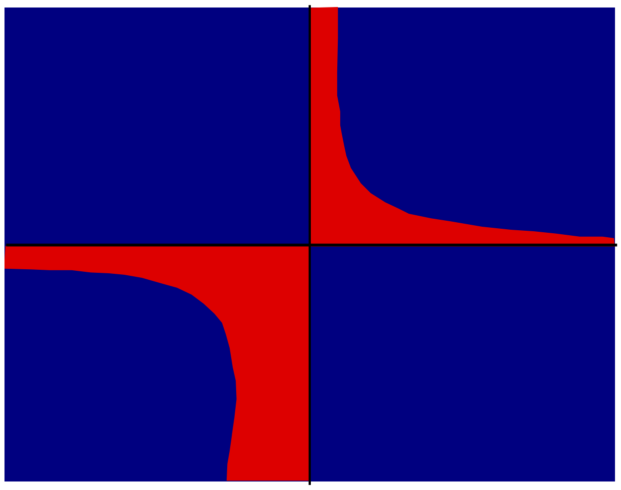

Figure 6.3.2: The results of training the SVM using TSVT-JMETA method over the checkers board problem. The correctly classified region is depicted in blue, while the wrongly classified is depicted in red. Although the boundary is still not perfect, a noticeable improvement has been obtained comparing with figure 6.3.1.

Regarding the Segment1 dataset, this time a great improvement is obtained when using Amari's method, which is confirmed by the Wilcoxon test. Also ACT and MDT-VMETA obtain noticeable improvements, although only the former has statistical significance. The remaining methods do not apply any transformation or perform similarly to the SVM, except TSVT-VMETA, which worsens the error rate (not significantly) by applying an inappropriate transformation. Observe also that the SVM g-means error is quite large, so the initial SVM is probably a bad model.

About the Hepatitis dataset, a slight improvement appears when using MDT-VMETA. The other methods that apply a transformation vary in small quantities the error level of the SVM, either worsening or improving it. In general, no statistical difference appears.

In the Sonar dataset every method which applies a transformation only worsens the results. This might indicate that the validation set is not representative of the problem, and so misleading hints about what transformation to apply are followed. The only methods that do not get a worse accuracy are those which do not apply any transformation. For the rest, the worsening showed a significant difference in the Wilcoxon test.

On the other hand, in the Heart Disease dataset only Amari's method applies the transformation, and obtains a slight improvement in accuracy. This time the validation set seems to have fulfilled its purpose, as when the decision of using a transformation is chosen no worsening appear. However, the improvement is not statistically significant.

Again, in the Ionosphere dataset the results are quite similar to the ones obtained in the Sonar problem, although this time only ACT and TSVT-JMETA methods apply a transformation. However, these transformations result in a worse accuracy, with statistical difference.

Method	Glass7	Segment1	Hepatitis	Sonar
SVM	**0.088 ± 0.100**	0.703 ± 0.296	0.499 ± 0.332	**0.104 ± 0.056**
AMA	0.091 ± 0.097	**0.168 ± 0.341**	0.499 ± 0.332*	0.161 ± 0.042
ACT	0.129 ± 0.119	0.381 ± 0.437	0.499 ± 0.332*	0.161 ± 0.042
WIL	0.088 ± 0.100*	0.703 ± 0.296*	0.504 ± 0.327	0.104 ± 0.056*
MDT-JGRAD	0.088 ± 0.100*	0.703 ± 0.296*	0.504 ± 0.327	0.104 ± 0.056*
MDT-JMETA	0.088 ± 0.100*	0.703 ± 0.296*	0.497 ± 0.323	0.189 ± 0.090
MDT-VMETA	0.088 ± 0.100*	0.615 ± 0.363	**0.477 ± 0.334**	0.169 ± 0.035
TSVT-JGRAD	0.088 ± 0.100*	0.703 ± 0.296*	0.497 ± 0.323	0.164 ± 0.041
TSVT-JMETA	0.088 ± 0.100*	0.701 ± 0.390	0.492 ± 0.328	0.165 ± 0.038
TSVT-VMETA	0.088 ± 0.100*	0.783 ± 0.279	0.499 ± 0.332*	0.161 ± 0.042

Table 6.3.2: Error levels obtained with the studied methods for Glass7, Segment1, Hepatitis and Sonar datasets. Errors are measured in g-means error. An asterisk appears when the method decided not to apply any transformation, thus obtaining the same error rate as the SVM.

Finally, in the Wisconsin Breast Cancer Diagnosis dataset most of the methods do not use any transformation, and the ones that do (Wilsons' and MDT-JGRAD) only attain a tiny increase in the error rate. Therefore, as expected, the Wilcoxon test shows no differences.

To understand this results, notice first that in the datasets where a great improvement appears (Checkers board and Segment1), the initial kernel choice seems to behave badly, as large g-means errors are obtained. These errors suggest that the SVM is classifying most of the patterns as belonging to the majority class. Starting with such a bad model, the conformal methods are able to improve the kernel function and therefore get a better error rate. However, when the base SVM model behaves fine (such as in Sonar or Ionosphere) no improvements seem to appear. In this way, conformal transformation methods seem to provide a way to improve bad initial kernel selections. This phenomenon also appears up to some point in the works of Si Wu [21] and Williams [22].

However, taking again as a example the checkers board problem, compare the optimized boundary obtained with TSVT-JMETA in figure 6.3.2 with the one previously shown in figure 3.1.3. In this second one no conformal method was used, but instead a Parameter Vector Grid Search (PVGS) method was used to find the optimal $2\sigma^2$ kernel parameter and the C penalty factor of the SVM. This method uses cross-validation procedures to select those parameters which maximize the estimated classification rate (for further details see [32]). As it can be observed, the boundary obtained using PVGS is more similar to the ideal one than the one obtained with TSVT-JMETA, and so is the accuracy of the model. However, it is worth noting that the computational costs of PVGS are much larger than those related to TSVT-JMETA, as the former involves training a great number of

Method	Heart Disease	Ionosphere	Breast Cancer
SVM	0.201 ± 0.072	**0.066** ± **0.045**	**0.021** ± **0.014**
AMA	**0.191** ± **0.068**	0.066 ± 0.045*	0.021 ± 0.014*
ACT	0.201 ± 0.072*	0.093 ± 0.054	0.021 ± 0.014*
WIL	0.201 ± 0.072*	0.066 ± 0.045*	0.026 ± 0.014
MDT-JGRAD	0.201 ± 0.072*	0.066 ± 0.045*	0.026 ± 0.014
MDT-JMETA	0.201 ± 0.072*	0.066 ± 0.045*	0.021 ± 0.014*
MDT-VMETA	0.201 ± 0.072*	0.066 ± 0.045*	0.021 ± 0.014*
TSVT-JGRAD	0.201 ± 0.072*	0.066 ± 0.045*	0.021 ± 0.014*
TSVT-JMETA	0.201 ± 0.072*	0.075 ± 0.053	0.021 ± 0.014*
TSVT-VMETA	0.201 ± 0.072*	0.066 ± 0.045*	0.021 ± 0.014*

Table 6.3.3: Error levels obtained with the studied methods for Heart Disease, Ionosphere and Wisconsin Breast Cancer Diagnosis datasets. Errors are measured in g-means error. An asterisk appears when the method decided not to apply any transformation, thus obtaining the same error rate as the SVM.

different SVMs in order to perform the cross-validations, while the latter only trains a SVM per iteration (which are usually less than 3).

Chapter 7

Conclusions and discussion

In this work we have reviewed the problem of unbalanced dataset classification and the techniques proposed in the literature to overcome it using Support Vector Machines, specifically those related with conformal kernel transformations. Furthermore, we have proposed several variants of these methods and tested them with artificial and real problems.

Although the theory of conformal transformations seems to provide a powerful and interesting way of adapting a kernel function to a particular training set so as to obtain a data-dependent kernel, our experiments show that the existing and proposed methods are quite limited. Real improvements were only observed when a poor initial kernel parameter selection was made. Better results were obtained in an artificial dataset by doing an "intelligent" (more or less thorough) search of those parameters. Despite these results, we should claim that methods like ACT or those based in JGRAD or JMETA approaches are way faster than a thorough search of kernel parameters, although their results are worse. So that, when time restrictions exists, those methods may well provide some improvement over just using a SVM with poor prefixed parameters.

Anyway, we must admit that these methods make some naive assumptions. As we are dealing with Gaussian kernels, the produced feature space has (potentially) infinite dimensions. In such a complex space, assuming that the support vectors are near the real data boundary or that the boundary found by the SVM has a similar orientation than the real boundary of the data might be unrealistic. In an infinite dimensional space all the training patterns may well become support vectors, giving no clues about the area where the spatial resolution should be augmented. A similar reasoning tells that a simple displacement of the SVM boundary does not necessarily lead us to that "optimal augmentation area".

Regarding the use of the separability criterion J, note that it has the same drawbacks that are present in Fisher's Discriminant Analysis [4], as pointed out in [33]. The J criterion assumes that the classes are linearly separable in the feature space, which may or may not be true depending on the chosen kernel. Also, the classes should be unimodal, other requisite which might not be met. Although the SVM itself assumes linear separability in the feature space, the J criterion may provide misleading hints about what kernel function we should choose, arriving to an inappropriate feature space.

In a nutshell, we think the geometry of the induced feature space is way too complex to make these kind of assumptions. Other factors apart from linear separability, such as the curvature of the manifold containing the patterns, may have a decisive role over the boundary obtained by the SVM, although they are not considered in the shown methods. Furthermore, pursuing linear separability in the feature space (and hence perfect classification) is not completely desirable, as overfitting is likely to appear.

Apart from that, its noticeable that in a great proportion of the experiments the conformal methods do not apply any transformation at all. This indicates that the stopping criterion used (the error in the validation set) might be too strict. However, in other experiments not presented here we have perceived that Amari's method tends to overfitting if we apply a fixed number of transformations (e.g. three or five). As this method increases the spatial resolution around SV and wrongly classified patterns are all in this group, in the new feature space the SVM will pay more attention to them, hence making efforts to classify every point correctly. We suppose that the other methods could suffer from the same problem, as they are directly or indirectly (through the decision function) using the information from every SV.

Concluding, we believe that the theory behind conformal transformations methods could provide ways to adapt and learn an appropriate data-dependent kernel matrix. However, more work is needed to find more suitable transformation functions so that they only focus in relevant patterns (e.g. the SV near the margin), as well as valid criterions to guide these transformations. Additionally procedures to control overfitting should be developed in order to obtain not only a data-dependent feature space, but also to guarantee its generalization ability.

Appendix A

Related work

During the Master course for which this thesis is presented, other works related with SVMs and the topic of machine learning were developed, primarily in the form of papers sent to several conferences. In this appendix we will briefly review those works so as to show the results of other research activities done along the course.

A.1 Metamodels for automatic parameter tuning

In our paper *Finding Optimal Parameters by Discrete Grid Search* [32] we developed a metamodel to find the optimal parameters for the training algorithm of a classification or regression model. This metamodel (named PVGS) performs a grid search in the parameter space so as to find the optimal choice over a discrete version of them. The grid search is dynamic in the sense that it starts using a coarse grid which gets focused around the area where better results are observed. Although the search cost is still exponential with respect the number of parameters, it is much smaller than an exhaustive cross-validation search, which is the usual method.

In our posterior work *Finding Optimal Parameters by Deterministic and Random Discrete Grid Search* [34] we develop further the method, describing a variant to deal with the optimization of a larger number of parameters. This variant proceeds in a similar fashion than the original one, but instead of evaluating all the possible points in each grid, a simulated annealing method is used to select which of them are worth to test. A comparison with the black box optimizer CMA-ES [24] is made, showing similar performance.

In both cases, the parameters of multilayer perceptrons and SVMs are tuned using the proposed methods, and the obtained models tested with some public available datasets, showing that a fine selection of those parameters can produce noticeable improvements in accuracy. In this work we have used PVGS to find the optimal C and $2\sigma^2$ parameters of the SVM in the chekers board problem, as comparison against the proposed conformal methods.

A.2 Relationships between SVM algorithms

In another work, *Square Penalty Support Vector Regression* [14], we study the relationships between the SVM for regression and classification problems. In this way we provide a framework for turning a regression task into a classification one which can be solved using CH-SVM. Although this transformations were known in the literature [13, 35], we provide an alternative proof of them. We also note that when using square penalties, a relationship can be established between the obtained support weights and their position respect to the regression tube. Finally we test the method in benchmark datasets and in a real wind power prediction problem.

Separately, in *On the equivalence of the SMO and MDM Algorithms for SVM Training* [36] we show that the popular method SMO [8] for training a SVM is in fact closely related to the geometric algorithm MDM [11] for CH-SVM training. Traditionally these two theoretical approaches (SVM and CH-SVM) were known to be equivalent, but no equivalences between their algorithms were shown.

Following a similar fashion, in *Simple Clipping Algorithms for Reduced Convex Hull SVM Training* [37] we propose a modified version of GSK [12] and MDM which solves the Reduced Convex Hulls SVM (RCH-SVM) [13], a theoretical version of the CH-SVM with linear penalties. Although other algorithms to solve RCH-SVM have been proposed [38], they are in general more complex than their CH-SVM counterpart, while ours are derived naturally from the original GSK and MDM algorithms.

A.3 Improvements of SVM training algorithms

An improvement to speed up the MDM algorithm [11] for SVM training is proposed in *An accelerated MDM algorithm for SVM training* [28]. Observing the evolution of MDM iterations, cyclical updates are detected, which can be avoided by keeping track of the last updates and performing a combined update when a cycle appears. Experimental results show that noticeable improvements in both number of iterations and kernel operations are achieved when using this method. In this work we have used this algorithm for the training of the SVMs involved in every tested method.

Also, another version of the MDM algorithm is proposed in *A 4 – vector MDM Algorithm for Support Vector Training* [39] where it is modified so as to perform an update in each class per iteration, while normally one has to choose between updating in one class or the other.

A.4 Wind Power Forecasting and Kernel Methods

As a practical application of the SVMs, a work concerning the prediction of the wind power production of wind farms all around Spain was developed: *Kernel Methods for Wide Area Wind Power*

Forecasting. Historical power production and weather prediction data provided by Red Eléctrica Española, the European Centre for Medium-Range Weather Forecasts and the Instituto de Ingeniería del Conocimiento was used for the experiments. A multilayer perceptron (a usual predicting method in this field) was compared against a SVM, the latter showing better results both in prediction accuracy and in computational costs for training.

Appendix B

Notation glossary

For clarity, we provide here a summary of the notation used in this work. Please note that in order to have a common notation in all the work, it may not follow the original notation of the reviewed methods and proposals, although we have tried to be as general as possible.

α_i Support weight of the i-th training pattern; i-th inequality Lagrange multiplier.

α_i^t Support weight of the i-th training pattern at transformation step t.

β Interpolation factor in KBA.

β_i i-th equality Lagrange multiplier.

γ Margin displacement parameter.

ϵ Model error.

κ Williams' transformation function parameter.

θ Error tolerance for stopping criterion.

Θ Dual function of an optimization problem.

Φ SVM mapping function to the feature space.

σ Gaussian kernel width parameter.

τ Width parameter (unique) of the conformal transformation function.

τ_i i-th width parameter of the conformal transformation function.

τ_i^t i-th width parameter at step t of the conformal transformation function.

$\tilde{\tau}_i$ i-th corrected width parameter of the conformal transformation function.

$\bar{\tau}_i$ i-th width parameter of the TSVT transformation function.

η Boundary displacement in KBA.

η_+ Positive SV transformation correction factor.

η_- Negative SV transformation correction factor.

μ_i Slack variable of the SVM for the i-th pattern in the negative class.

ξ_i Slack variable of the SVM for the i-th pattern in the positive class.

b Bias of the SVM.

$\hat{b}$ Bias of the CH-SVM.

$\tilde{b}$ Displaced bias in MDT.

Δb Boundary movement term.

B Between-class scatter matrix in the feature space.

c Conformal transformation function.

$c_t(x^i)$ Transformation coefficient in step t of the i-th pattern.

C Penalty factor of the SVM.

C^+ Biased penalty factor for the positive class of the SVM.

C^- Biased penalty factor for the negative class of the SVM.

CH-SVM Convex Hull - Support Vector Machine.

$\mathcal{C}(X)$ Convex Hull of the set of points X.

d Number of input variables of the patterns (dimension of the input space).

dx Differential displacement in the input space.

dz Differential displacement in the feature space.

$d_t(x^i)$ Cumulative transformation coefficient in step t of the i-th pattern.

D Dimension of the feature space.

f Output function of the SVM.

$\hat{f}$ Output function of the CH-SVM.

$\tilde{f}$ Corrected output function in MDT.

F Objective function of an optimization problem.

$\mathcal{F}$ Feature space of the SVM.

g Kubat's g-means error criterion value.

g_i i-th inequality restriction of an optimization problem.

g_{ij} Components of the Riemannian tensor of distances in a manifold.

h_i i-th equality restriction of an optimization problem.

$\mathcal{I}$ Input space of the SVM.

J Fisher's separability criterion.

J_1 Numerator of Fisher's separability criterion.

J_2 Denominator of Fisher's separability criterion.

$\tilde{J}$ Fisher's separability criterion after the transformation.

$\tilde{J}_1$ Numerator of Fisher's separability criterion after the transformation.

$\tilde{J}_2$ Denominator of Fisher's separability criterion after the transformation.

k Kernel function of the SVM.

$\tilde{k}$ Transformed kernel function of the SVM.

K Kernel matrix.

K_t Kernel matrix at step t.

$\tilde{K}$ Transformed kernel matrix.

KKT Karush-Kuhn-Tucker conditions.

L Lagrangian function of an optimization problem.

L_D Dual function of the SVM.

L_p Lagrangian function of the SVM.

m Margin of the SVM.

m_+ Positive hyperplane "margin" in MDT.

m_- Negative hyperplane "margin" in MDT.

M Distance parameter of the conformal transformation function.

MDT Margin Displacement Transformation.

N Number of patterns in the training set.

N_- Number of patterns from the negative class in the training set.

N_+ Number of patterns from the positive class in the training set.

q Vector containing the result of applying c to each patterns in X.

R Spatial magnification ratio.

SV Support Vector.

SV^+ Number of positive support vectors.

SV^- Number of negative support vectors.

SVM Support Vector Machine.

$\mathcal{S}^b$ Interpolated artificial support vectors in KBA.

V Validation set.

$\mathcal{V}$ Magnification factor of the kernel function.

$\tilde{\mathcal{V}}$ Magnification factor of the transformed kernel function.

w Normal vector of the separating hyperplane.

$\hat{w}$ Normal vector of the separating hyperplane found by CH-SVM.

$\tilde{w}$ Corrected normal vector of the separating hyperplane in MDT.

$\hat{w}_+$ Nearest point in the convex hull of the positive class.

$\hat{w}_-$ Nearest point in the convex hull of the negative class.

$\tilde{w}_+$ Corrected nearest point of the positive class in MDT.

$\tilde{w}_-$ Corrected nearest point of the negative class in MDT.

$||w||$ Norm of the vector w.

W Parameters vector of an optimization problem; within-class scatter matrix in the feature space.

x^i i-th training pattern of a set.

x_j j-th input variable of the training patterns.

x^i_j j-th input variable of i-th training pattern of a set.

X Set of training patterns.

X_+ Set of training patterns belonging to the positive (minority) class ($y_i = +1$).

X_- Set of training patterns belonging to the negative (majority) class ($y_i = -1$).

y_i Class of the i-th pattern of a set, $\in \{-1, 1\}$.

Bibliography

[1] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Statistics. Springer, 2001.

[2] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[3] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. pages 89–114, 1988.

[4] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, 2000.

[5] E. Castillo, A. J. Conejo, P. Pedregal, R. García, and N. Alguacil. *Formulación y resolución de modelos de programación matemática en ingeniería y ciencia*. 2002.

[6] T. Joachims. Making large-scale support vector machine learning practical. *Advances in Kernel Methods - Support Vector Machines*, pages 169–184, 1999.

[7] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, 2000.

[8] J.C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods - Support Vector Machines*, pages 185–208, 1999.

[9] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A(209):415–446, 1909.

[10] E.G. Gilbert. Minimizing the quadratic form on a convex set. *SIAM J. Contr.*, 4:61–79, 1966.

[11] B.F. Mitchell, V.F. Dem'yanov, and V.N. Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM J. Contr.*, 12:19–26, 1974.

[12] V. Franc and V. Hlaváč. An iterative algorithm learning the maximal margin classifier. *Pattern Recognition*, 36:1985–1996, 2003.

[13] K.P. Bennett and E.J. Bredensteiner. Duality and geometry in svm classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 57–64, 2000.

[14] A. Barbero, J. López, and J. Dorronsoro. Square penalty support vector regression. In *Intelligent Data Engineering and Automated Learning – IDEAL 2007*, pages 537–546. Springer, December 2007.

[15] G. Wu and E. Y. Chang. Adaptive feature-space conformal transformation for imbalanced-data learning. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, 2003.

[16] G. Wu and E. Y. Chang. Class-boundary alignment for imbalanced dataset learning. In *Workshop on Learning from Imbalanced Datasets II, ICML*, 2003.

[17] R. Akbani, S. Kwek, and N. Japkowicz. Applying support vector machines to imbalanced datasets.

[18] K. Veropoulos, N. Cristianini, and C. Campbell. Controlling the sensitivity of support vector machines, 1999.

[19] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12:783–789, 1999.

[20] G. Wu and E. Y. Chang. Kba: Kernel boundary alignment considering imbalanced data distribution. *IEEE Transactions on knowledge and data engineering*, 17(6), 2005.

[21] S. Wu and S. Amari. Conformal transformation of kernel functions: A data-dependent way to improve support vector machine classifiers. *Neural Processing Letters*, 15:59–67, 2002.

[22] P. Williams, S. Li, and S. Wu. A geometrical method to improve performance of the support vector machine. In *IEEE Transactions on neural networks*, volume 18, May 2007.

[23] H. Xiong, M. N. S. Swamy, and M. Omair Ahmad. Optimizing the kernel in the empirical feature space. *IEEE Transactions on neural networks*, 16(2), March 2005.

[24] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[25] W. H. Press, Saul A. Teukolsky, W. T. Vetterling, and Brian P. Flannery. *Numerical recipes in C*. Cambridge University Press, Cambridge, second edition, 1992. The art of scientific computing.

[26] D. E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, 1989.

[27] N. Hansen. The cma evolution strategy: Source code. http://www.bionik.tu-berlin.de/user/niko/cmaes_inmatlab.html.

[28] A. Barbero, J. López, and J. Dorronsoro. An accelerated mdm algorithm for svm training. In *Proceedings of the 11th European Symposium on Artificial Neural Networks*, pages 421–426, 2008.

[29] N. Hansen. The cma evolution strategy: A tutorial. http://www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf.

[30] University of California Irvine. Uci machine learning repository. http://archive.ics.uci.edu/ml.

[31] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[32] A. Barbero, J. López, and J. Dorronsoro. Finding optimal parameters by discrete grid search. In *Innovations in Hybrid Intelligent Systems – HAIS 2007*, pages 120–127. Springer Berlin – Heidelberg, December 2008.

[33] B. Chen, H. Liu, and Z. Bao. A kernel optimization method based on the localized kernel fisher criterion. *Pattern Recognition*, pages 1098–1109, 2008.

[34] A. Barbero, J. López, and J. Dorronsoro. Finding optimal parameters by deterministic and random discrete grid search. *Neurocomputing*, Yet to be published.

[35] J. Bi and K.P. Bennett. A geometric approach to support vector regression. *Neurocomputing*, 55:79–108, 2003.

[36] J. López, A. Barbero, and J. Dorronsoro. On the equivalence of the smo and mdm algorithms for svm training. In *Proceedings of the ECML 2008 conference*, page submitted, 2008.

[37] A. Barbero, J. López, and J. Dorronsoro. Simple clipping algorithms for reduced convex hull svm training. In *Proceedings of the HAIS 2008 conference*, page submitted, 2008.

[38] M.E. Mavroforakis and S. Theodoridis. A geometric approach to support vector machine (svm) classification. *IEEE Transactions on Neural Networks*, 17(3):671–682, May 2006.

[39] A. Barbero, J. López, and J. Dorronsoro. A 4 – vector mdm algorithm for support vector training. In *Proceedings of the ICANN 2008 conference*, page yet to be published, 2008.