Machine Learning, 46, 203–223, 2002

An Analytic Center Machine

THEODORE B. TRAFALIS ttrafalis@ou.edu
ALEXANDER M. MALYSCHEFF alexm@ou.edu
Laboratory for Optimization and Intelligent Systems, School of Industrial Engineering, The University of Oklahoma, 202 West Bord, Norman, OK 73019, USA

Editor: Nello Cristianini

Abstract. Support vector machines have recently attracted much attention in the machine learning and optimization communities for their remarkable generalization ability. The support vector machine solution corresponds to the center of the largest hypersphere inscribed in the version space. Recently, however, alternative approaches (Herbrich, Graepel, & Campbell, In Proceedings of ESANN 2000) have suggested that the generalization performance can be further enhanced by considering other possible centers of the version space like the center of gravity. However, efficient methods for calculating the center of gravity of a polyhedron are lacking. A center that can be computed efficiently using Newton's method is the analytic center of a convex polytope. We propose an algorithm, that finds the hypothesis that corresponds to the analytic center of the version space. We refer to this type of classifier as the analytic center machine (ACM). Preliminary experimental results are presented for which ACMs outperform support vector machines.

Keywords: support vector machines, analytic center, interior point methods

1. Introduction

Support vector machines (SVMs) (Vapnik, 1995; Schölkopf, Burges, & Smola, 1999; Smola, 1998) have been recently introduced as a new tool for machine learning applications. One of the most outstanding characteristics of this learning machine is its impressive performance in generalization on unseen data. However, in cases where the version space, i.e. the space of hypotheses consistent with the training data, is elongated or asymmetric SVMs are not very effective. Recently, an alternative approach has been proposed by Herbrich et al. (2000) and Ruján (1997), the so-called Bayes point machine, which is based on an approximation of the center of mass of the version space. In fact, the SVM classifier corresponds to the center of the largest inscribed hypersphere in version space. However, computing the center of gravity of a polyhedron in an n-dimensional space is hard, since it involves the computation of its volume, which is a #P-hard problem (Kaiser, Morin, & Trafalis, 1991). An easily computable center has been proposed by Sonnevend (1985), the so-called "analytic center" of a convex polytope. This point depends on the data analytically (i.e. rather smoothly), is invariant with respect to affine transformations, and can be computed effectively by minimizing a strictly convex function over the convex polytope. In an earlier study Bouten et al. (1995) have investigated the generalization performance of perceptrons and found that the Bayes point solution in version space can be approximated by computing the minimum of a special class of potential functions. As this research effort originated initially from the support vector machine approach, it seems most appropriate to follow a genetic approach in this article. Consequently, we will start by giving a brief review of support vector machines and expand from there to the analytic center machine.

Support vector machines can be implemented in two different environments. Specifically, regression analysis and pattern classification. In this study we will focus on pattern classification problems, which can be expressed as a quadratic programming problem.

Next, we cast this type of problem into a quadratic optimization problem. Let the training data consist of l points, each of which is represented by a vector $\mathbf{x}_j \in \Re^d$, where $j = 1, \ldots, l$ and assign to each of the points a label y_j with $y_j \in \{-1, +1\}$. Here, we only distinguish between two classes. Hence the training set can be written as $T = \{(\mathbf{x}_j, y_j)_{j=1}^l\} \subset \Re^d \times \{-1, +1\}$. Constructing a learning machine that will separate the data of two training sets in pattern space finds its geometrical equivalent in identifying a hyperplane that separates the data points labeled by $y_j = -1$ from the data points labeled by $y_j = +1$. As there are situations, where datasets can not be linearly separated, we will restrict ourselves for the moment to those problem sets that are linearly separable. This type of classifier is often also referred to as the hard margin classifier. Vapnik (1995) has shown that the linear support vector machine algorithm requires a solution of the following optimization problem:

$$
\begin{aligned}
(P) \qquad & \min && \frac{1}{2}\|\mathbf{w}\|^2 \\
& \text{subject to} && y_j(\mathbf{x}_j \cdot \mathbf{w} + b) \geq +1 \quad \forall j = 1, \ldots, l,
\end{aligned}
\tag{1}
$$

where $\mathbf{w} \in \Re^d$ is the vector normal to the separating hyperplane and $b \in \Re$ the offset with respect to the origin. By assigning a Lagrangian multiplier λ_j to each constraint and by introducing the variables $\mathbf{\Lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_l)$, $D_{ij} = (y_i y_j x_i x_j)$, $\mathbf{e} = (1, 1, \ldots, 1)$, and $\mathbf{y} = (y_1, y_2, \ldots, y_l)$, the dual problem can be formulated in closed form:

$$
\begin{aligned}
(D) \qquad & \max && -\mathbf{\Lambda}^T\mathbf{e} + \frac{1}{2}\mathbf{\Lambda}^T\mathbf{D}\mathbf{\Lambda} \\
& \text{subject to} && \mathbf{\Lambda}^T\mathbf{y} = 0 \\
& && \mathbf{\Lambda}_j \geq 0 \quad \forall j = 1, \ldots, l.
\end{aligned}
\tag{2}
$$

A nonlinear separable problem in pattern space becomes a linearly separable problem in feature space using the concept of a kernel function (Vapnik, 1995). As can be seen, the dual problem (D) allows expressing all data pertinent to the problem, $\mathbf{x}_j$ and y_j, to be expressed in terms of inner products. We will revisit this idea in a later section.

In this paper concepts from interior point methods will be employed to solve pattern classification problems. However, interior point methods suffer from the difficulty of finding an initial feasible solution, when all constraints that define the feasible region are considered simultaneously. An alternative approach would be to deal with every sample point (constraint) at a time. Hence, the problem is solved for the first data point, then the next data point enters the system and another optimization problem is solved using the previous solution and so forth. We might call this iterative type of approaching the problem an online approach. Moreover, recent research has suggested to verify the generalization ability of support vector machines, as some results point to alternative learning machines with powerful generalization performance (Herbrich, Graepel, & Campbell, 2000).

In order to implement this online-approach, the support vector algorithm will be modified reducing the constraints of problem (P) essentially to what is known as the perceptron. Then, turning our attention to the weight space, it can be seen that each pattern will correspond to a hyperplane with normal vector $y\mathbf{x} = (yx_1, yx_2, \ldots, yx_d)$, if we are in a d-dimensional space. Moreover, after a slight modification we will be able to let all hyperplanes corresponding to examples pass through the origin. Therefore, the collection of all patterns, i.e. l hyperplanes, will reduce the region containing acceptable results for $\mathbf{w}$ to a cone in a d-dimensional space. Some studies refer to this cone as the version space $V(T)$ (Mitchell, 1997). While the perceptron allows any of these feasible $\mathbf{w}$'s to be selected as a solution, our objective will be to choose a $\mathbf{w}^*$ which remains far away from all the edges of the cone, since we do not know how future test patterns will scatter. This refers to as a large margin classifier (Smola et al., 2000)

The paper is organized as follows: we will highlight ideas pertaining to the selection of $\mathbf{w}^*$ in Section 2 by explaining, what type of problem must be solved, in order to retrieve $\mathbf{w}^*$. In Section 3 we will make the connection to the solution in feature space, thus allowing nonlinear separable problems to be solved. Section 4 describes, how the analytic center is calculated using a projected Newton descent approach. These ideas will then be revisited and modified for an online version of the gradient Newton descent approach in Section 5. In Section 6 preliminary results are presented to illustrate the generalization performance of analytic center machines compared to support vector machines. Section 7 will close this paper outlining possible directions for future improvements.

2. Statement of the problem

2.1. *Geometry of linear learning machines*

We would like to construct a learning machine based on the concept of the analytic center, which behaves also as a large margin classifier. Later, it will be shown that a similar learning machine can be derived for the feature space using the concept of kernel functions. In order to convey the basic ideas, we review the concepts that characterize the perceptron. In contrast to support vector machines no optimization process is invoked and the right-hand-side of the constraints becomes zero. Specifically in the perceptron algorithm it is our objective to solve the feasibility problem:

$$y_j \cdot (\mathbf{x}_j \cdot \mathbf{w} + b) \geq 0 \quad \forall j = 1, \ldots, l, \tag{3}$$

where $\mathbf{x}_j, \mathbf{w} \in \Re^d$ and $y_j, b \in \Re$. Any feasible solution $(\mathbf{w}, b)$ of (3) will separate the training set T. Thus, for linearly separable problems the perceptron will arbitrarily identify one out of infinitely many linear classifiers and generalization will be rather poor in most instances. We will return to the issue of selecting a good hyperplane in Section 2.2. For the moment let us slightly rewrite the set of constraints. By introducing the variables $\tilde{\mathbf{w}} = (\mathbf{w}, b) \in \Re^{d+1}$ and $\tilde{\mathbf{x}}_j = (\mathbf{x}_j, 1) \in \Re^{d+1}$ the perceptron problem simplifies to:

$$y_j \cdot (\tilde{\mathbf{x}}_j \cdot \tilde{\mathbf{w}}) \geq 0 \quad \forall j = 1, \ldots, l. \tag{4}$$

Essentially, b is now being considered as an additional weight, say $\tilde{w}_{d+1}$. Therefore, the problem dimension is expanded by one. As can be seen each pattern j is now described by the simple linear inequality

$$y_j \tilde{\mathbf{x}}_j \cdot \tilde{\mathbf{w}} \geq 0 \quad \forall j = 1, \ldots, l, \tag{5}$$

which corresponds in weight space to a hyperplane with normal vector $y_j \tilde{\mathbf{x}}_j$. Note that all hyperplanes are passing now through the origin of the weight space.

When the first pattern $(\mathbf{x}_1, y_1)$ enters in the system of linear inequalities, the set of feasible solutions is reduced to a $(d + 1)$-dimensional halfspace. The second pattern $(\mathbf{x}_2, y_2)$ will reduce the halfspace to a cone with the apex of the cone at the origin. After that, each additional pattern will either leave the current version space unchanged or even further reduce it. Once all patterns have been learned, the cone of the training patterns has been identified. Note that some training points $(\mathbf{x}_j, y_j)$ might be redundant for the determination of the cone. The set of all $\tilde{\mathbf{w}}$'s, inside the cone, $V(T)$, constitute the set of feasible solutions; hence infinitely many solutions exist. This situation corresponds to the perceptron in pattern space. Keep in mind that here we consider problems, which can be linearly separated (hard margin classifier). In this case the set of feasible $\tilde{\mathbf{w}}$'s will never be empty. The advantage of this approach lies in the nice geometry of the problem, which allows a rather simple implementation. Once the final cone is identified, it is intuitively clear that we would like to identify a solution as close to the central trajectory of the cone as possible. In doing so, we guarantee that the selected optimal $\tilde{\mathbf{w}}^*$ will correspond to a separating hyperplane in the space of patterns (x-space) that lies far away from the data samples of both classes. The next question to be addressed is, how to select the optimal $\tilde{\mathbf{w}}^*$ from all feasible $\tilde{\mathbf{w}}$'s in the version space.

2.2. *Analytical center of version space*

Next, we will use an idea that stems from the concept of interior point methods, the so-called analytic center. In order to describe the analytic center, we first introduce the concept of logarithmic barrier functions. Define the slack $\tilde{s}_j \in \Re$ as a measure of how close or how far away the current solution is from constraint j. If the current solution violates constraint j, $\tilde{s}_j$ will be negative. If the current solution fulfills constraint j as an equality, $\tilde{s}_j$ will be zero. An *interior* point of the feasible region is defined as a point, for which all slacks $\tilde{s}_j$ are positive. For the set of constraints, which currently describe our feasible region the slacks can be expressed as follows:

$$\tilde{s}_j = y_j \tilde{\mathbf{x}}_j \cdot \tilde{\mathbf{w}} \geq 0 \quad \forall j = 1, \ldots, l. \tag{6}$$

Furthermore, we need to identify a function, called potential function (Nesterov & Nemirovskii, 1994), which goes to infinity as we approach the boundary of the feasible region. A logarithmic function $\Phi : \Re^+ \rightarrow \Re$ fulfills our requirements. More specifically,

$$\Phi(\tilde{\mathbf{s}}) = -\ln(\tilde{\mathbf{s}}). \tag{7}$$

In analogy to electrostatics the function $\Phi(\tilde{\mathbf{s}})$ could describe the potential of some electrostatic field, if we consider the hyperplanes defining the feasible region as infinite plates of electric charge. For l constraints the logarithmic barrier function Φ becomes:

$$\Phi(\tilde{\mathbf{s}}) = -\sum_{j=1}^{l} \ln(\tilde{s}_j), \quad \tilde{\mathbf{s}}^T = (\tilde{s}_1, \ldots, \tilde{s}_l). \tag{8}$$

Here, the vector $\tilde{\mathbf{s}} \in \Re^l$ represents the collection of slacks for all l constraints, $\tilde{\mathbf{s}} = (\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_l)$. The minimum of the logarithmic barrier function $\Phi(\tilde{\mathbf{s}})$ constitutes an excellent means of computing a point that will be far away from all constraints. The point at which the barrier function attains its minimum is defined as the *analytic center* of the feasible region.

Now, let us return to the problem we would like to solve. We consider an augmented weight space (keep in mind that the offset b is treated as the $(d + 1)$-th element of $\tilde{\mathbf{w}}$), in which the set of all patterns $(\tilde{\mathbf{x}}_j, y_j)$ is represented by hyperplanes passing through the origin. After all hyperplanes have been defined, a cone in $\Re^{d+1}$ will describe the set of feasible solutions in the augmented weight space. It is our objective, to retrieve a weight vector $\tilde{\mathbf{w}}^*$, which will be as far away from the "walls" of the cone as possible. The tool we are going to employ is the concept of the analytic center. Unfortunately, the minimum of the logarithmic barrier function can only be calculated, if the feasible region is compact, which is decidedly not the case for a cone in $\Re^{d+1}$. In order to compactify the feasible region, we add as a constraint a hypersphere in $\Re^{d+1}$ with radius R around the origin. In other words, the solution, which will lie without doubt very close to the central trajectory of the cone will be projected onto a hypersphere of radius R.

Therefore, our goal is to calculate the minimum of $\Phi(\tilde{\mathbf{s}})$ while making sure at the same time that the optimal weight vector $\tilde{\mathbf{w}}^*$ satisfy the spherical constraint. For calculation purposes we choose $R = \sqrt{2}$. Taking into account the labels y_j as well, the slacks can be expressed through

$$\tilde{s}_j = y_j \cdot (\tilde{\mathbf{x}}_j \cdot \tilde{\mathbf{w}}) \geq 0 \quad \forall j = 1, \ldots, l \tag{9}$$

and the logarithmic barrier function becomes

$$\Phi(\tilde{\mathbf{s}}) = -\sum_{j=1}^{l} \ln(\tilde{s}_j). \tag{10}$$

Following the above, the optimal weight vector can be computed by solving the optimization problem:

$$\begin{aligned} \min \quad & \Phi(\tilde{\mathbf{w}}) = -\sum_{j=1}^{l} \ln(y_j(\tilde{\mathbf{x}}_j \cdot \tilde{\mathbf{w}})) \\ \text{s.t.} \quad & \frac{1}{2}\tilde{\mathbf{w}}^T\tilde{\mathbf{w}} = 1. \end{aligned} \tag{11}$$

Since the slacks depend on $\tilde{\mathbf{w}}$, we will write from now on $\Phi(\tilde{\mathbf{w}})$ or in feature space $\Phi(\alpha)$, instead of $\Phi(\tilde{\mathbf{s}})$. Remember that by definition $\tilde{\mathbf{w}}$ also includes the offset b of the optimal separating hyperplane. Before we will continue our discussion and demonstrate, how the optimization problem in (11) can be solved, we will extend the current formulation using the concept of kernel functions to the case of nonlinear separation.

3. Kernelization

In the first section the reader was introduced to the basic concepts of linear separation of datasets. In Section 2 the classification problem was reformulated, such that each pattern can be interpreted as a hyperplane in weight space. We will now perform yet another transformation moving the problem to some high-dimensional feature space. The algorithm, which is discussed in this paper actually does not really solve nonlinear problems. However, it is possible to map classification problems that can not be linearly separated to the feature space using a function $\phi : \mathbf{x} \to \phi(\mathbf{x})$. The mapping creates the corresponding classification problem in feature space, which can be solved employing linear classifiers. The price we have to pay is to significantly increase the number of dimensions. Thus a nonlinear problem behaves in feature space as if it was linearly separable.

Unfortunately, very often the function $\phi(\mathbf{x})$ is not available, can not be computed, or does not even exist. However, the inner product of two vectors can be computed, both, in pattern and in feature space. In other words, while $\phi(\mathbf{x})$ might not be available, we can still compute the inner product $\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2)$ in feature space. This inner product can be expressed by the kernel function

$$k : \Re^d \times \Re^d \to \Re : k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2).$$

The reader might verify that besides the label y_j in Eq. (4) in Section 2 every constraint can be formulated using solely inner products. Nonetheless, the constraints still contain the weight vector, while it would be desirable to express the inner products only through the data vectors $\mathbf{x}_j$. However, if we express the weight vector by a linear combination of data vectors, we can successfully avoid this problem. Indeed, if we define $\mathbf{w} = \sum_{i=1}^{l} \alpha_i \mathbf{x}_i$ we have:

$$\begin{aligned} k(\mathbf{w}, \mathbf{x}_j) &= k\left(\sum_{i=1}^{l} \alpha_i \mathbf{x}_i, \mathbf{x}_j\right) = \left(\sum_{i=1}^{l} \alpha_i \phi(\mathbf{x}_i)\right)^T \cdot \phi(\mathbf{x}_j) \\ &= \sum_{i=1}^{l} \alpha_i k(\mathbf{x}_i, \mathbf{x}_j) \quad \forall j = 1, \ldots, l. \end{aligned} \tag{12}$$

Therefore, the set of l equations in (3) defining hyperplanes in weight space can be reformulated using inner products. Specifically,

$$y_j\left(\mathbf{x}_j \cdot \sum_{i=1}^{l} \alpha_i \mathbf{x}_i + b\right) = y_j\left(\sum_{i=1}^{l} \alpha_i \mathbf{x}_i^T \mathbf{x}_j + b\right) \geq 0 \quad \forall j = 1, \ldots, l. \tag{13}$$

After having preprocessed the set of constraints, it is now easy to replace the pure inner products by the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$. Therefore, the set of constraints representing the l patterns can be described in the dual space of α's as follows:

$$y_j\left(\sum_{i=1}^{l} \alpha_i k(\mathbf{x}_i, \mathbf{x}_j) + b\right) \geq 0 \quad \forall j = 1, \ldots, l. \tag{14}$$

Obviously, this new set of linear inequalities must be solved in the space of α's. Note, that the decision function to test for a new pattern $\mathbf{x}$ can be formulated and calculated based only on the information contained in the set of α_i's:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{l} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b\right). \tag{15}$$

We are now ready to define our learning problem in α-space. Define the matrix $\mathbf{K} \in \Re^{(l+1)\times l}$ as:

$$\mathbf{K} = \begin{bmatrix} y_1 k_{11} & y_2 k_{12} & \ldots & y_l k_{1l} \\ y_1 k_{21} & y_2 k_{22} & \ldots & y_l k_{2l} \\ \ldots & \ldots & & \ldots \\ y_1 k_{l1} & y_2 k_{l2} & \ldots & y_l k_{ll} \\ y_1 & y_2 & \ldots & y_l \end{bmatrix}. \tag{16}$$

We follow the convention that the k_{ij} represent the kernel evaluated at the points $\mathbf{x}_i$ and $\mathbf{x}_j$. Specifically, $k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Note that $\mathbf{K}$ is not symmetric. Moreover, introduce the vectors

$$\mathbf{k}_j = \begin{bmatrix} y_j k_{1j} \\ y_j k_{2j} \\ \ldots \\ y_j k_{lj} \\ y_j \end{bmatrix}, \forall j = 1, \ldots, l, \tag{17}$$

where $\mathbf{k}_j \in \Re^{l+1}$. Therefore the matrix $\mathbf{K}$ can also be expressed as $\mathbf{K} = (\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_l)$. Furthermore, if we let the vector $\alpha \in \Re^{l+1}$ to be

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \ldots \\ \alpha_l \\ b \end{bmatrix}, \tag{18}$$

we can write the set of equations for all l patterns as $\mathbf{K}^T \alpha \geq 0,\ \forall j = 1, \ldots, l$. Note that the offset b is included as the $(l + 1)$-element and the corresponding adjustments were made for $\mathbf{K}$ and $\mathbf{k}_j$. Alternatively, the j-th constraint in α-space can be defined by $\mathbf{k}_j^T \alpha \geq 0,\ \forall j = 1, \ldots, l$, since

$$\mathbf{k}_j^T \alpha = y_j \alpha_1 k(\mathbf{x}_1, \mathbf{x}_j) + y_j \alpha_2 k(\mathbf{x}_2, \mathbf{x}_j) + \cdots + y_j \alpha_l k(\mathbf{x}_l, \mathbf{x}_j) + y_j b \geq 0. \tag{19}$$

Replacing the kernel expressions by simple inner products, equivalence between Eq. (19) and the more commonly known perceptron formulation becomes more obvious as is described by the following inequality:

$$\begin{aligned} y_j \alpha_1 \mathbf{x}_1^T \mathbf{x}_j + y_j \alpha_2 \mathbf{x}_2^T \mathbf{x}_j + \cdots + y_j \alpha_l \mathbf{x}_l^T \mathbf{x}_j + y_j b &= y_j \cdot \left(\sum_{i=1}^{l} \alpha_i \mathbf{x}_i^T \right) \mathbf{x}_j + y_j b \\ &= y_j \cdot (\mathbf{w}^T \mathbf{x}_j + b) \geq 0. \end{aligned} \tag{20}$$

Since it is necessary to rewrite the optimization problem from Eq. (11) for the space of α's, computation of the slacks for each constraint is required. We have l slacks, which can be computed from $s_j = \mathbf{k}_j^T \cdot \alpha,\ \forall j = 1, \ldots, l$. Previously, each pattern was characterized by its normal vector $y_j \tilde{\mathbf{x}}_j$. This vector defined a hyperplane in the space of weights reducing the final feasible region to a cone in $\Re^{d+1}$. We have now moved the problem to the space of α's. Therefore, the normal vector defining each hyperplane in α-space is given by $\mathbf{k}_j$. Note that the offset is included in $\mathbf{k}_j$, consequently, each hyperplane passes again through the origin; this time however, in the space of α's. The logarithmic barrier function evaluates the summation over all slack values, which are given by $s_j = \mathbf{k}_j^T \alpha,\ \forall j = 1, \ldots, l$. Hence the barrier is a function of α, i.e. $\Phi = \Phi(\alpha)$. Again, as in Section 2, the feasible region has the shape of a cone, which implies that the analytic center without any projection or box limitations imposed on the variables, does not exist. The analytic center of the barrier function is therefore calculated with the additional requirement that the set of variables (the α's) be projected on a hypersphere of radius $R = \sqrt{2}$. Consequently, we have to solve the following optimization problem:

$$\begin{aligned} \min \quad & \Phi(\alpha) = -\sum_{j=1}^{l} \ln\left(\mathbf{k}_j^T \alpha\right) \\ \text{s.t.} \quad & h(\alpha) = \frac{1}{2} \alpha^T \alpha - 1 = 0. \end{aligned} \tag{21}$$

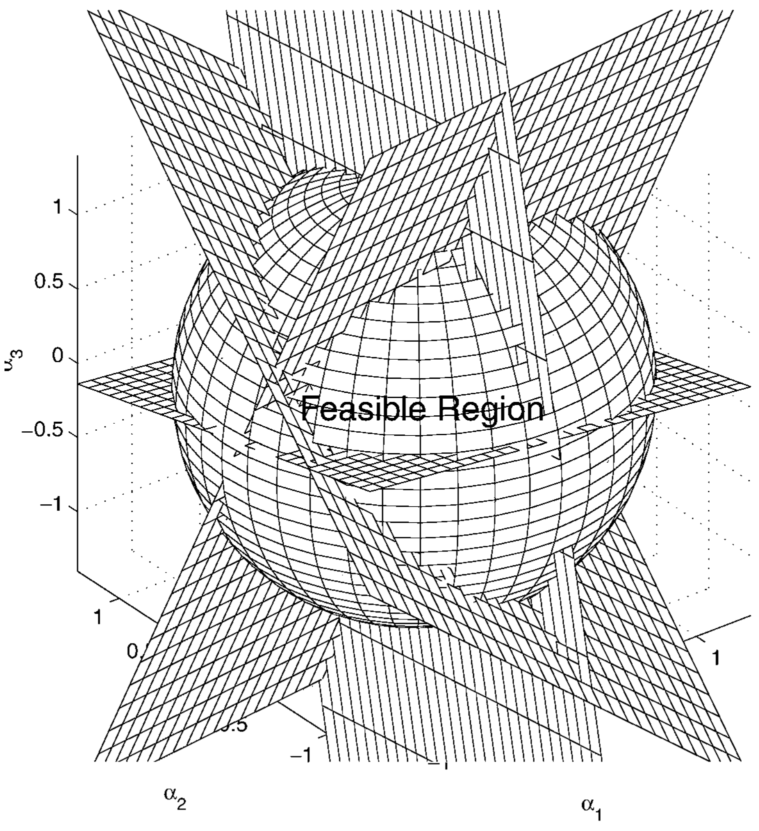

Figure 1. Feasible region in α-space.

Figure 1 illustrates this situation graphically. We need to find the analytic center of the feasible region, which is described by the intersection of several hyperplanes with the surface of the sphere. In the next section it will be shown, how the optimization problem can be solved using a projected Newton descent method.

4. A projected Newton descent method

In the previous section our discussion resulted in a constrained optimization problem, which must be solved by changing the set of α_i's. We started with the concept of a cone, which described all feasible **w**'s in weight space that would qualify to separate the two classes. It was then shown that a similar approach could be used to solve nonlinear instances resulting in roughly the same optimization problem. The two approaches differ in the space, in which the variables "live", as well as in the parameters. For the linear case information can be stored in $y_j\tilde{\mathbf{x}}_j$, while for the nonlinear case the $\mathbf{k}_j$'s need to be computed. The vector $\mathbf{k}_j$ depends on the kernel function as well as on the input data $(\mathbf{x}_j, y_j)$. In the following we have to address the optimization problem in (21) and implement an algorithm that allows solving this type of problem.

Let us start the analysis by introducing the Lagrangian of (21). With u being the Lagrangian multiplier we have:

$$L(\alpha) = \Phi(\alpha) + u \cdot h(\alpha). \tag{22}$$

The Karush-Kuhn-Tucker (KKT) optimality conditions (Bazaraa, Sherali, & Shetty, 1993) require that at optimality the gradient of the Lagrangian vanish (feasibility of the corresponding dual optimization problem). Moreover, primal feasibility is required as well, hence, the constraint in (21) represents an additional optimality condition. The following equations account for dual and primal feasibility and represent the optimality conditions:

$$\nabla_\alpha L(\alpha) = \nabla_\alpha \Phi(\alpha) + u \cdot \nabla_\alpha h(\alpha) \tag{23}$$

$$h(\alpha) = \frac{1}{2}\alpha^T \alpha - 1 = 0. \tag{24}$$

Next, define a new variable $\mathbf{Z}(\alpha, u)$ and let us rewrite the optimality conditions in matrix form. Thus,

$$\mathbf{Z}(\alpha, u) = \begin{bmatrix} \nabla_\alpha L(\alpha) \\ h(\alpha) \end{bmatrix} = \begin{bmatrix} \nabla_\alpha \Phi(\alpha) + u \cdot \nabla_\alpha h(\alpha) \\ \frac{1}{2}\alpha^T \alpha - 1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}. \tag{25}$$

In a more compact form the optimality conditions require $\mathbf{Z}(\alpha, u) = \mathbf{0}$. Employing the Newton-Raphson method to calculate α^* and u^*, the first-order approximation linear system

$$\mathbf{Z}(\alpha_k, u_k) + \nabla_\alpha \mathbf{Z}(\alpha_k, u_k) \begin{bmatrix} \alpha - \alpha_k \\ u - u_k \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix} \tag{26}$$

must be solved, in order to calculate the next iterate $(\alpha, u) = (\alpha_{k+1}, u_{k+1})$. Equation (26) allows to iteratively approach the solution (α^*, u^*), with which we can compute the classifier. Here, $\nabla \mathbf{Z}$ denotes the Jacobian of $\mathbf{Z}$ and (α_k, u_k) represents the current iterate. We find for the Jacobian of $\mathbf{Z}$

$$\nabla \mathbf{Z}(\alpha_k, u_k) = \begin{bmatrix} \nabla^2 L(\alpha_k, u_k) & \nabla h(\alpha_k, u_k) \\ \nabla h(\alpha_k, u_k)^T & 0 \end{bmatrix}. \tag{27}$$

Next, we need to compute the elements of $\nabla \mathbf{Z}$ and $\mathbf{Z}$ in order to move to the next iterate (α_{k+1}, u_{k+1}). Let us first examine the derivatives of the logarithmic barrier function and introduce a matrix $\mathbf{S} \in \Re^{l \times l}$ that carries in its diagonal the slack values $s_j = \mathbf{k}_j^T \alpha_k$, where α_k represents the current iterate. Moreover, let $\mathbf{e} = (1, 1, \ldots, 1)$ denote a vector of ones in an appropriate dimension, here $\mathbf{e} \in \Re^l$. Then the gradient of $\Phi(\alpha)$ becomes $\nabla \Phi(\alpha) = -\mathbf{K}\mathbf{S}^{-1}\mathbf{e}$. An alternative way of expressing the gradient of the barrier would be $\nabla \Phi(\alpha) = -\frac{\mathbf{k}_1}{s_1} - \frac{\mathbf{k}_2}{s_2} - \cdots - \frac{\mathbf{k}_l}{s_l}$. The Hessian of the barrier can be computed similarly, resulting in $\nabla^2 \Phi(\alpha) = \mathbf{K}\mathbf{S}^{-2}\mathbf{K}^T$ or alternatively, $\nabla^2 \Phi(\alpha) = \frac{\mathbf{k}_1 \mathbf{k}_1^T}{s_1^2} + \frac{\mathbf{k}_2 \mathbf{k}_2^T}{s_2^2} + \cdots + \frac{\mathbf{k}_l \mathbf{k}_l^T}{s_l^2}$.

Considering next the gradient and the Hessian of h, we find for the gradient $\nabla h(\alpha) = \alpha$ and for the Hessian $\nabla^2 h(\alpha) = \mathbf{I}$, with $\mathbf{I} \in \Re^{(l+1)\times(l+1)}$ representing the unit matrix. Now it is easy to extend the discussion to the gradient and to the Hessian of the Lagrangian. In fact,

$$\nabla^2 L(\alpha, u) = \nabla^2 \Phi(\alpha) + u \cdot \nabla^2 h(\alpha) = \mathbf{KS}^{-2}\mathbf{K}^T + u \cdot \mathbf{I} \tag{28}$$

$$\nabla L(\alpha, u) = \nabla \Phi(\alpha) + u \cdot \nabla h(\alpha) = -\mathbf{KS}^{-1}\mathbf{e} + u \cdot \alpha \tag{29}$$

Putting it all together we find for $\mathbf{Z}$:

$$\mathbf{Z}(\alpha_k, u_k) = \begin{bmatrix} -\mathbf{KS}^{-1}\mathbf{e} + u_k \cdot \alpha_k \\ \frac{1}{2}\alpha_k^T \alpha_k - 1 \end{bmatrix} \tag{30}$$

and for the Jacobian of $\mathbf{Z}$:

$$\nabla \mathbf{Z}(\alpha_k, u_k) = \begin{bmatrix} \mathbf{KS}^{-2}\mathbf{K}^T + u_k \cdot \mathbf{I} & \alpha_k \\ \alpha_k^T & 0 \end{bmatrix}. \tag{31}$$

We have now presented a modified Newton procedure, which allows finding the minimum value of the barrier function subject to the constraint that all α's be elements of a hypersphere in $\Re^{l+1}$. Note the dimensions of $\mathbf{Z} \in \Re^{l+2}$ and $\nabla \mathbf{Z} \in \Re^{(l+2)\times(l+2)}$, since the $(l+1)$-th element corresponds to the offset b, while the $(l+2)$-th element refers to the Lagrangian multiplier u.

5. Implementation issues

Until now, we have proposed a new method of selecting the optimal weight $\mathbf{w}^*$ for learning linear classifiers. Hereafter, the idea of selecting a value close to the central trajectory of a cone was extended to the nonlinear case. We have studied ways to obtain the optimal value ($\boldsymbol{\alpha}^*$ or $\mathbf{w}^*$) from an open cone using concepts based on the analytic center. However, several remarks should be made about the current version of the algorithm. First, it is necessary to identify an interior point of the feasible region. Keep in mind that the barrier function is logarithmic; hence, it is not defined for α's yielding negative slacks. It might not be trivial to find an initial feasible solution, for which the Newton algorithm will work, in particular in the nonlinear case. Instead, it might be wiser to solve the problem in a sequential way. What do we mean by that? So far, the algorithm is confronted with all available data right from the beginning. We might call this the batch approach. If we initiate on the other hand the Newton procedure pattern by pattern, we will be able to circumvent the problem of finding an initial feasible solution. Therefore, we will next turn our attention to the issue of implementation of this new approach in machine learning.

5.1. *How to process a "good" pattern*

We propose an online approach in computing the analytic center α-space by evaluating an approximation of the center for some current set of constraints. Once a pattern is classified correctly, which means that the current iterate is inside the current cone, we let a new pattern (a new hyperplane) enter the system of linear inequalities. Hence, for each new pattern feasibility of the current solution is verified and adjusted. In the next few paragraphs we will outline the mechanics behind this way of solving the problem.

We initiate the algorithm with arbitrary starting values, α_0 and u_0. Next, the normal vector for the hyperplane in α-space is calculated:

$$\begin{aligned}\mathbf{k}_1^T &= (y_1k_{11}, y_1k_{21}, \ldots, y_1k_{l1}, y_1)\\ &= (y_1k(\mathbf{x}_1,\mathbf{x}_1), y_1k(\mathbf{x}_2,\mathbf{x}_1), \ldots, y_1k(\mathbf{x}_l,\mathbf{x}_1), y_1). \end{aligned}\tag{32}$$

Upon evaluating $\mathbf{k}_1^T\alpha_0$ two alternative outcomes are possible. If the value for $\mathbf{k}_1^T\alpha_0$ is negative, which by definition also means that the slack $s_1 = \mathbf{k}_1^T\alpha_0$ is negative, we have to correct our current solution α_0. Let us postpone for the moment the discussion of this case by assuming that α_0 is feasible, hence the slack is strictly positive. Remember that even for slack values equal to zero, the Hessian might be ill-conditioned as the logarithmic barrier approaches $+\infty$ in those cases. The gradient and the Hessian of the barrier function can be found respectively from

$$\nabla\Phi(\alpha_0) = -\frac{\mathbf{k}_1}{s_1}\tag{33}$$

$$\nabla^2\Phi(\alpha_0) = \frac{\mathbf{k}_1\mathbf{k}_1^T}{s_1^2}.\tag{34}$$

Next, evaluate $h(\alpha_0) = \frac{1}{2}\alpha_0^T\alpha_0 - 1$ as well as the gradient $\nabla h(\alpha_0) = \alpha_0$. With this in mind we find for $\mathbf{Z}$:

$$\mathbf{Z}(\alpha_0, u_0) = \begin{bmatrix} -\frac{\mathbf{k}_1}{s_1} + u_0\cdot\alpha_0 \\ \frac{1}{2}\alpha_0^T\alpha_0 - 1 \end{bmatrix}.\tag{35}$$

Since the Hessian of $h(\alpha)$ equals the unit matrix with dimension $(l+1)\times(l+1)$, the Jacobian of $\mathbf{Z}$ is easily computed using

$$\nabla\mathbf{Z}(\alpha_0, u_0) = \begin{bmatrix} \frac{\mathbf{k}_1\mathbf{k}_1^T}{s_1^2} + u_0\cdot\mathbf{I} & \alpha_0 \\ \alpha_0^T & 0 \end{bmatrix}.\tag{36}$$

Based on the calculations of $\mathbf{Z}$ and $\nabla\mathbf{Z}$, the next iterate can be computed by solving the linear Newton approximation in Eq. (26). However, we would like to point out that this subproblem will not be solved to optimality, as we are not interested in the solution after having processed only one pattern.

Consequently, after having successfully computed what amounts to α_1 and u_1 the normal vector corresponding to the second pattern is evaluated:

$$\begin{aligned}\mathbf{k}_2^T &= (y_2k_{12}, y_2k_{22}, \ldots, y_2k_{l2}, y_2)\\ &= (y_2k(\mathbf{x}_1,\mathbf{x}_2), y_2k(\mathbf{x}_2,\mathbf{x}_2), \ldots, y_2k(\mathbf{x}_l,\mathbf{x}_2), y_2). \end{aligned}\tag{37}$$

Again, one needs to verify, whether the slack $s_2 = \mathbf{k}_2^T \alpha_1$ is positive and if so, the new values for $\mathbf{Z}$ and $\nabla \mathbf{Z}$ are computed:

$$\mathbf{Z}(\alpha_1, u_1) = \begin{bmatrix} -\frac{\mathbf{k}_1}{s_1} - \frac{\mathbf{k}_2}{s_2} + u_1 \cdot \alpha_1 \\ \frac{1}{2}\alpha_1^T \alpha_1 - 1 \end{bmatrix} \tag{38}$$

$$\nabla \mathbf{Z}(\alpha_1, u_1) = \begin{bmatrix} \frac{\mathbf{k}_1 \mathbf{k}_1^T}{s_1^2} + \frac{\mathbf{k}_2 \mathbf{k}_2^T}{s_2^2} + u_1 \cdot \mathbf{I} & \alpha_1 \\ \alpha_1^T & 0 \end{bmatrix}. \tag{39}$$

Hereafter, we conduct an additional Newton search and update the current solution (α_2, u_2) using Eq. (26) before repeating this procedure for the remaining patterns. Starting from an initial solution at $\alpha_0 = (0.5, 0.5)$ figure 2 illustrates, how the current solution is adjusted, when we process successively the patterns $\mathbf{k}_1, \mathbf{k}_2$, and $\mathbf{k}_3$. The updated solution is given by α_3.

Next, we will explain, how to adapt the algorithm for the case where the cone is significantly reduced by a new pattern such that the previous solution lies outside the new cone.

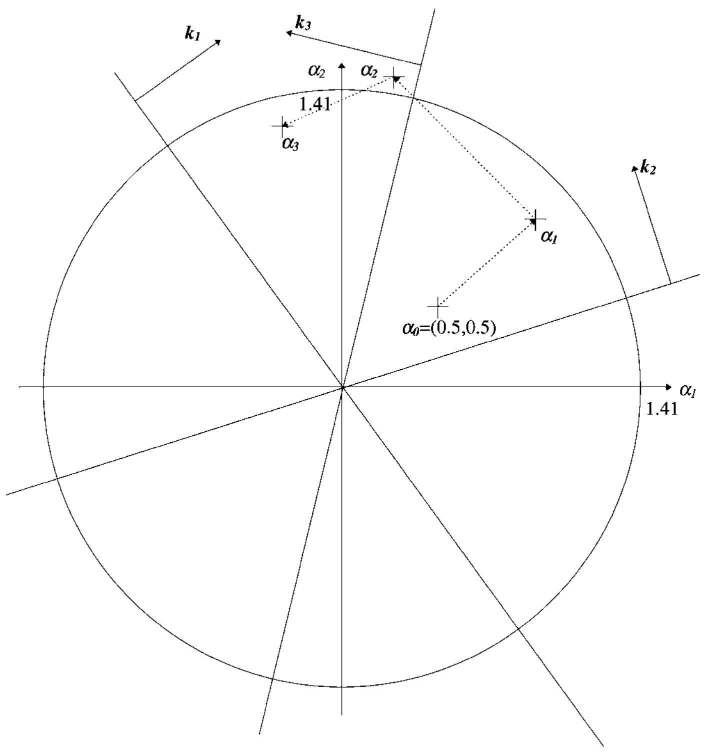

Figure 2. The previous solution α_{j-1} is always feasible with respect to the pattern $\mathbf{k}_j$.

5.2. *How to correct a "bad" pattern*

If patterns are not correctly classified, the current iterate represents an infeasible starting point for the Newton optimization process, which excludes using this type of optimization technique. To be more precise, suppose that we are at iteration j and that the j-th pattern reduces the feasible region so significantly that for the $(j-1)$-th solution (α_{j-1}, u_{j-1}), α_{j-1} lies outside of the cone. Looking at the set of slacks, this implies that the j-th slack s_j will be negative. However, all remaining slacks $s_1, \dots, s_{j-1}$ are still positive. Hence, we either need to move the current iterate into the new cone or alternatively relax the shape of the feasible region, such that the new feasible set still comprises the α_{j-1} of the previous iterate (α_{j-1}, u_{j-1}). Since we have only very limited information about the current position of (α_{j-1}, u_{j-1}) we decided to pursue the latter approach of relaxing the j-th constraint of the feasible region.

Since there is only one constraint (constraint j, the last one that entered the system of patterns), which is violated by the current iterate, we only need to focus on the j-th slack value. Keep in mind that s_j is a measure of distance between α_{j-1} and the hyperplane $\mathbf{k}_j^T\alpha = 0$. Modification of the feasible region will be realized as follows. Since s_j is the sole parameter defining the position of the hyperplane, we can shift the hyperplane parallel to itself by changing the value for s_j. For instance, the hyperplane $\mathbf{k}_j^T\alpha = s_j$ passes exactly through the current iterate α_{j-1}. Obviously, the parallel shift is not yet sufficient, as the barrier function approaches $+\infty$ for this situation. For this reason, we need to shift the last hyperplane a little farther, say, about ten percent of the slack value to guarantee that the current iterate α_{j-1} lies inside this "extended" cone. Consequently, the j-th shifted hyperplane is defined by the set of all α's for which $\mathbf{k}_j^T\alpha = 1.1s_j$. This hyperplane will change the entry for the j-th slack to $-0.1s_j$. Remember that s_j is negative, therefore $-0.1s_j$ is positive. We have now adjusted the feasible region such that α_{j-1} is an element of the feasible perturbed set.

Next, we need to invoke Newton's algorithm to virtually "push" α_{j-1} to the analytic center of the feasible region. This will be realized by conducting five consecutive Newton steps. Let $i = 1, \dots, 5$ count the Newton steps on the relaxed problem. At each step, we will examine, whether the updated solution $(\alpha_{j-1,i}, u_{j-1,i})$ still violates the original constraint $\mathbf{k}_j^T\alpha_{j-1,i} \geq 0$. If so, the remaining Newton steps are executed. Of course, one must be very careful about updating the slack s_j during this process.

Store the overall shift of the j-th hyperplane in the variable $total = -1.1s_j$. That way, once Newton's algorithm offers a new $\alpha_{j-1,i}$, the j-th slack value is calculated with respect to the original (unrelaxed) problem, $\mathbf{s} = \mathbf{K}^T \cdot \alpha$. However, the Newton procedure requires a positive slack value $s_{j,i}$, which means that the j-th slack must be separately updated with respect to the relaxed problem. Therefore, s_j is replaced by an intermediate slack valid only for the i-th Newton iteration on the relaxed problem $s_{j,i} = total + s_{j,i-1}$ in order to yield the slacks for the i-th Newton iteration on the relaxed problem (define $s_{j,0} = s_j$). This approach essentially measures the distance between $\alpha_{j-1,i}$ from the relaxed hyperplane $\mathbf{k}_j^T\alpha = -total$.

Nonetheless, it is possible that even after five Newton steps the analytic center of the relaxed feasible region does not yet satisfy the j-th constraint. Keeping in mind that *total* $= -1.1s_j$ defined the position of the relaxed j-th hyperplane we can calculate a new relaxed problem with a relaxed hyperplane that will lie between the first relaxed hyperplane and the original unrelaxed hyperplane (all parallel, we did not modify $\mathbf{k}_j$). Obviously, our situation will have improved after five Newton iterations, however, it is possible that the analytic center of the relaxed problem still violates the original problem. The new offset for the second relaxed problem can be computed as total $= -1.1s_{j-1,5}$ yielding the hyperplane $\mathbf{k}_j^T\alpha = -\textit{total}$. In other words the first relaxed hyperplane is moved to a new position reducing the relaxed cone. Hereafter, we invoke once more Newton's descent algorithm. Either one of the intermediate iterates $(\alpha_{j-1,i}, u_{j-1,i})$ finally lies within the original cone, or we have to construct a third intermediate hyperplane after five Newton steps. Recapitulating we have to deal with two different counters. Specifically, j indicates the pattern and i indicates the Newton step executed. To increase readability we have refrained from introducing a third index denoting the number of relaxed hyperplanes needed until we have moved the current iterate $(\alpha_{j-1,i}, u_{j-1,i})$ inside the original cone.

Note that for each newly computed solution $(\alpha_{j-1,i}, u_{j-1,i})$ ones needs to verify *all* slacks s_j, since it is possible that a pure Newton step leaves the feasible region. In our case the new α could violate a constraint other than the one (s_j), which has just been relaxed. This case arises particularly, if α_{j-1} of the current solution $(\alpha_{j-1,i}, u_{j-1,i})$ is located already rather close to one or more constraints (Wright, 1995). We decided to halve the step size of a pure Newton step and verify whether feasibility is given. If not, the step size is cut in half again and constraint violation is checked again.

Figure 3 displays the procedure of adjusting infeasible α_j's. As can be seen α_0 is feasible for the first pattern $\mathbf{k}_1$ moving the current solution to α_1. Hereafter, the second training pattern $\mathbf{k}_2$ reduces the feasible region significantly so that α_1 is no longer feasible. Thus the second constraint is relaxed by *total*$' = -1.1s_2' = -1.1s_{2,0}'$. Conducting five successive Newton steps results in the new iterate $\alpha_{1,5}' = \alpha_{1,0}''$. Obviously, the current solution is not yet feasible with respect to the original problem. Therefore, a second relaxation (indicated by a second prime) has to be conducted. Hence, $s_{2,5}' = s_{2,0}''$ and the Newton method is invoked again. Feasibility is achieved at $i = 2$ and the training pattern can now be considered as a "good pattern". Therefore, one more Newton step is conducted with respect to the original feasible region to classify $\mathbf{k}_2$.

We will now recapitulate our discussion until this point. The following generic pseudo code provides an overview of all steps necessary for the implementation of an online analytic center machine.

```
Step 1: Initialize
    $L$ - Number of patterns
    $j$ - Current pattern, $j = 1, \dots, L$
    $i$ - Index to identify all patterns from $1, \dots, j$
    $\alpha_0$ - Initial starting point, vector of ones
    $u$ - Initial Lagrangian multiplier for projection on sphere

Step 2a: Center and Add
    Set $j = 1$
```

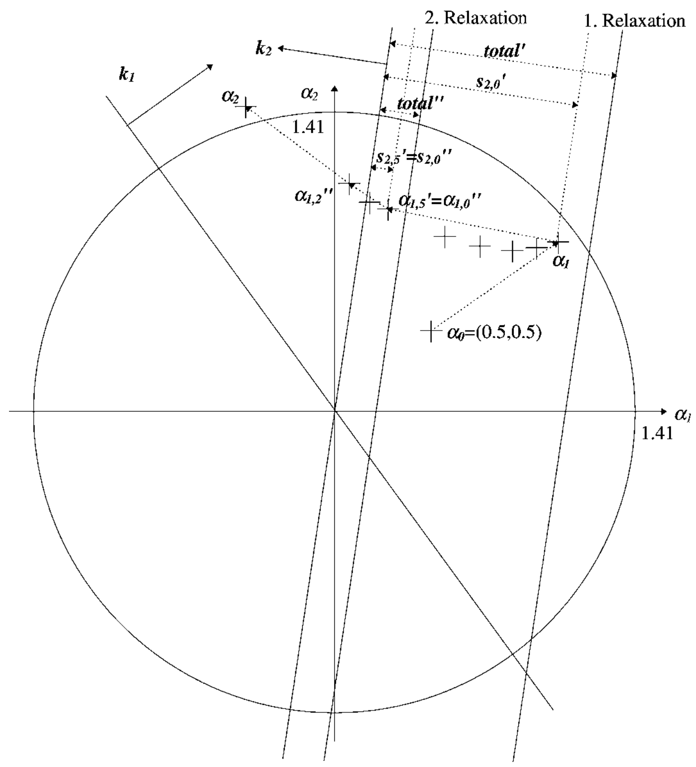

Figure 3. The current solution α_1 is infeasible for the second pattern $\mathbf{k}_2$.

```
    Compute the newest $\mathbf{k}_j$ according to $\mathbf{k}_j^T = (y_j k_{1j}, y_j k_{2j}, \ldots, y_j k_{lj}, y_j)$,
    the slacks $s_i = \mathbf{k}_i^T \alpha,\ \forall i = 1, \ldots, j$, and let $\mathbf{S} = diag(s_i), \forall i = 1, \ldots, j$
    Let $\mathbf{K} = (\mathbf{k}_1, \ldots, \mathbf{k}_j)$
    if $s_j \le 0$
        go to step 2b: Correct
    else
        go to step 2c: Update
Step 2b: Correct
    Set $total = -1.1 \cdot s_j$ and $s_{relax} = -0.1 \cdot s_j$ (Problem relaxation)
    Let $\mathbf{S}_{relax} = diag(s_1, s_2, \ldots, s_{j-1}, s_{relax})$
    while $(s_j = \mathbf{k}_j^T \alpha \le 0)$
        for $newtoncounter = 1$ to 5
            compute $\mathbf{Z}$ and $\nabla\mathbf{Z}$ using equations (30) and (31)
            calculate new $(\alpha, u)$ using Newton's approximation (26)
            if $(\mathbf{s} = \mathbf{K}^T \cdot \alpha \ge 0)$
```

```
                go to step 2c: Update
            else
                update the j-th slack, $s_j = total + s_j$ ($s_j$ is still negative!)
                check feasibility of all other slacks $(s_1, \ldots, s_{j-1})$
            **while** $min(s_1, \ldots, s_{j-1}) \leq 0$
                halve the pure Newton step
                compute $\mathbf{s} = \mathbf{K}^T \cdot \alpha$ and replace $s_j$, $s_j = total + s_j$
            **endwhile**
        **endfor**
        Set $total = -1.1 \cdot s_j$ (Construct new relaxation)
        Replace $s_j$, $s_j = -0.1 \cdot s_j$
    **endwhile**

Step 2c: Update
    compute $\mathbf{Z}$ and $\nabla\mathbf{Z}$ using equations (30) and (31)
    calculate new $(\alpha, u)$ using the newton approximation (26)
    $j = j + 1$, return to step 1
```

6. Computational results

6.1. Tests on synthetic data

Preliminary experiments were conducted comparing the performance of the analytic center machine to support vector machines. Both approaches were implemented in MATLAB and compared on two datasets, which have been previously used by Herbrich, Graepel, and Campbell (2000). Ten thousand points were randomly generated in $[-1, 1]^3 \subset \Re^3$ and assigned a random label of $+1$ or -1. Hereafter, a sample of ten points was selected for training. The size of the test set comprised all 10,000 points for each one of the experiments. An exponential kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$ with $\sigma = 1$ was used for the analysis. Table 1 displays the percentage values for the generalization error. As can be seen analytic center machines clearly outperform support vector machines for both experiments. Moreover, analytic center machines demonstrate a similar generalization behavior as Bayes point machines (Herbrich, Graepel, & Campbell, 2000).

Furthermore, we solved some problems graphically indicating the change of the decision surface. Figures 4 and 5 display the solutions for the analytic center machine and

Table 1. Generalization error.

Dataset	SVM (Herbrich, Graepel, & Campbell, in press)	BPM (Herbrich, Graepel, & Campbell, in press)	ACM
Dataset 1	6.50	6.10	6.22
Dataset 2	15.10	8.00	8.68

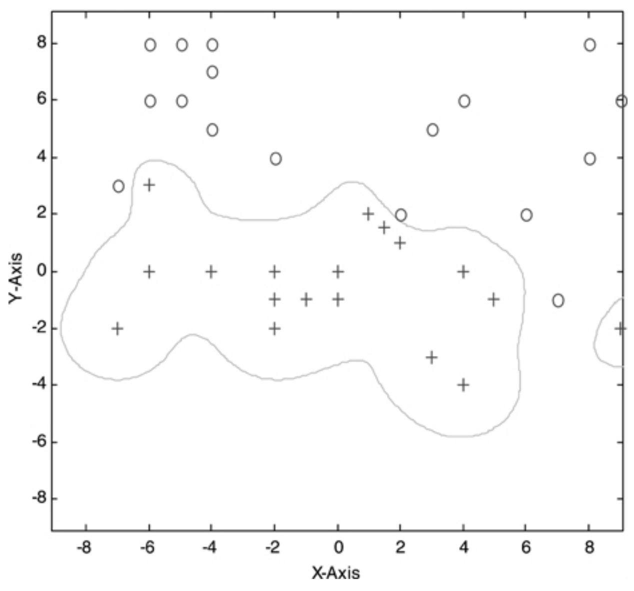

Figure 4. Analytic center solution.

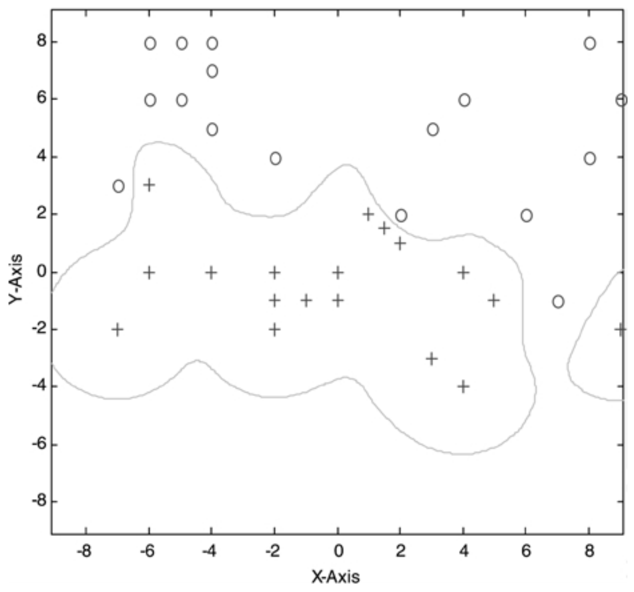

Figure 5. Support vector solution.

Table 2. Generalization error on standard benchmark data.

Dataset	SVM (Herbrich, Graepel, & Campbell, 2000)	BPM (Herbrich, Graepel, & Campbell, 2000)	ACM	σ	p-value
Heart	25.40 ± 0.40	22.80 ± 0.34	21.87 ± 3.51	10.00	1.0000
Thyroid	5.30 ± 0.24	4.40 ± 0.21	4.91 ± 2.28	3.00	0.9548
Banana	16.20 ± 0.15	15.10 ± 0.14	14.73 ± 1.69	0.50	1.0000

a support vector machine, respectively. Two classes were separated using again an exponential kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$ with $\sigma = 1$. It can be seen that the separation achieved by the analytic center machine is much smoother than for the support vector machine. This is due to the fact that the analytic center solution is influenced by all patterns, while the support vector machine includes only information from the support vectors.

6.2. *Tests on standard benchmark data*

In addition, performance of the analytic center machine was examined using three real-world datasets. The datasets "Heart" and "Thyroid" originate from the UCI Repository (Blake & Merz, 1998), while the dataset "Banana" was previously used as a benchmark in Herbrich, Graepel, and Campbell (2000) and Rätsch, Onoda, and Müller (2001). Each of the datasets consists of 100 training and test sets, which are randomly partitioned as follows. The datasets "Heart" and "Thyroid" exhibit a training to test set ratio of $60\% : 40\%$, while the dataset "Banana" is partitioned based on a ratio of $10\% : 90\%$[1] Table 2 shows the means and standard deviations for the support vector machine, the Bayes point machine, as well as for the analytic center machine, with the analytic center machine experiments being conducted using ACCPM (Péton & Vial, 2000), an analytic center optimization software. The fourth column indicates σ as it was used in the kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$. Finally, the fifth column represents the p-values for a pooled-variance t-test using a level of significance of 0.05 and verifying the hypothesis *SVM is greater than ACM*.

7. Conclusion

Preliminary experimentation has demonstrated that analytic center machines possess a promising potential for generalization. Future research might focus on an algorithm which deletes some or all of the nonbinding constraints of the cone. This could lead to a considerable improvement in performance, when large-scale problems are discussed. In our current implementation the analytic center of the version space is computed using all patterns. Reducing the set of characteristic patterns to those that are essential in the definition of the version space would increase the sparsity of the matrices and could lead to a significant speed-up of the algorithm. We are currently investigating heuristics to delete redundant constraints that are not necessary for the description of the version space. Efficient factorization techniques can be used for solving Newton's system of equations for large-scale implementations.

Acknowledgments

The authors would like to thank Nello Cristianini, the guest editor of this special issue, as well as two anonymous referees for comments that greatly improved the paper. The authors also would like to express their gratitude to Ralf Herbrich for providing some of the datasets and to Huseyin Ince for helping with the implementation. This research has been supported partially by the National Science Foundation, NSF Grant ECS-9978813.

Note

1. These datasets are publically available at http://www.first.gmd.de/raetsch/.

References

Bazaraa, M. Z., Sherali, H. D., & Shetty C. M. (1993). *Nonlinear Programming: Theory and Algorithms*, New York, NY: John Wiley & Sons.

Blake, C. L. & Merz, C. J. (1998). UCI Repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html.

Bouten, M., Schietse, J., & van den Broeck, C. (1995). Gradient descent learning in perceptrons: A review of its possibilities, *Physical Review, E, 52:2*, 1958–1967.

Herbrich, R., Graepel, T., & Campbell, C. (2000). Robust bayes point machines. In *Proceedings of ESANN 2000*. pp. 49–54.

Kaiser, M. J., Morin, T. L., & Trafalis, T. B. (1991). Centers and invariant points of convex bodies. In P. Gritzmann & B. Sturmfels (Eds), *Applied geometry and discrete mathematics: The victor klee festschrift*. AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science (pp. 367–385). American Mathematical Society, Providence, RI.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Nesterov, Y. E. & Nemirovskii, A. S. (1994). *Interior point polynomial methods in convex programming*, Philadelphia: SIAM Publications.

Péton, O. & Vial, J.-Ph. (2000). A tutorialon ACCPM. Technical report, HEC/Logilab, University of Geneva. http://ecolu-info.unige.ch/logibab/software/accpm/index.html.

Rätsch G., Onoda, T., & Müller, K.-R. (2001). Soft margins for AdaBoost. *Machine Learning, 42:2*, 287–320. In press.

Ruján, P. (1997). Playing billard in version space. *Neural Computation, 9*, 99–122.

Schölkopf, B., Burges, C. J. C., & Smola, A. J. (1999). *Advances in kernel methods: Support vector learning.* Cambridge, Massachusetts: The MIT Press.

Smola, A. J. (1998). Learning with kernels, Ph.D. Thesis, Technical University, Berlin.

Smola, A., Bartlett, P., Schölkopf B., & Schuurmans, D. (2000). *Advances in large margin classifiers*. Cambridge, Massachusetts: The MIT Press.

Sonnevend, G. (1985). An analytical center for polyhedrons and then new classes of global algorithms for linear (smooth, convex) programming. *Lectures Notes in Control Information Sciences* (pp. 866–876). New York: Springer-Verlag.

Vapnik, V. (1995). *The nature of statistical learning theory.* Berlin: Springer Verlag.

Wright, M. H. (1995). Why a pure primal newton barrier step may be infeasible. *SIAM Journal on Optimization, 5:1*, 1–12.

Received March 23, 2000
Revised March 9, 2001
Accepted March 9, 2001
Final manuscript March 6, 2001