LETTER Communicated by Christopher Williams

Prototype Classification: Insights from Machine Learning

Arnulf B. A. Graf
arnulf.graf@nyu.edu
Max Planck Institute for Biological Cybernetics, 72076 Tübingen, Germany, and
New York University, Center for Neural Science, New York, NY 10003, U.S.A.

Olivier Bousquet*
obousquet@gmail.com
Gunnar Rätsch
Gunnar.Raetsch@tuebingen.mpg.de
Bernhard Schölkopf
bernhard.schoelkopf@tuebingen.mpg.de
Max Planck Institute for Biological Cybernetics, 72076 Tübingen, Germany

We shed light on the discrimination between patterns belonging to two different classes by casting this decoding problem into a generalized prototype framework. The discrimination process is then separated into two stages: a projection stage that reduces the dimensionality of the data by projecting it on a line and a threshold stage where the distributions of the projected patterns of both classes are separated. For this, we extend the popular mean-of-class prototype classification using algorithms from machine learning that satisfy a set of invariance properties. We report a simple yet general approach to express different types of linear classification algorithms in an identical and easy-to-visualize formal framework using generalized prototypes where these prototypes are used to express the normal vector and offset of the hyperplane. We investigate non-margin classifiers such as the classical prototype classifier, the Fisher classifier, and the relevance vector machine. We then study hard and soft margin classifiers such as the support vector machine and a boosted version of the prototype classifier. Subsequently, we relate mean-of-class prototype classification to other classification algorithms by showing that the prototype classifier is a limit of any soft margin classifier and that boosting a prototype classifier yields the support vector machine. While giving novel insights into classification per se by presenting a common and unified formalism, our generalized prototype framework also provides an efficient visualization and a principled comparison of machine learning classification.

*Olivier Bousquet is now at Google in Zürich, Switzerland. Gunnar Rätsch is now at the Friedrich Miescher Laboratory of the Max Planck Society in Tübingen, Germany.

Neural Computation **21, 272–300** (2009)

1 Introduction

Discriminating between signals, or patterns, belonging to two different classes is a widespread decoding problem encountered, for instance, in psychophysics, electrophysiology, and computer vision. In detection experiments, a visual signal is embedded in noise, and a subject has to decide whether a signal is present or absent. The two-alternative forced-choice task is an example of a discrimination experiment where a subject classifies two visual stimuli according to some criterion. In neurophysiology, many decoding studies deal with the discrimination of two stimuli on the basis of the neural response they elicit, in either single neurons or populations of neurons. Furthermore, in many engineering applications such as computer vision, pattern recognition and classification (Duda, Hart, & Stork, 2001; Bishop, 2006) are some of the most encountered problems. Although most of these applications are taken from different fields, they intrinsically deal with a similar problem: the discrimination of high-dimensional patterns belonging to two possibly overlapping classes.

We address this problem by developing a framework—the prototype framework—that decomposes the discrimination task into a data projection, followed by a threshold operation. The projection stage reduces the dimensionality of the space occupied by the patterns to be discriminated by projecting these high-dimensional patterns on a line. The line on which the patterns are projected is unambiguously defined by any two of its points. We propose to find two particular points that have a set of interesting properties and call them *prototypes* by analogy to the mean-of-class prototypes widely used in cognitive modeling and psychology (Reed, 1972; Rosch, Mervis, Gray, Johnson, & Boyes-Braem, 1976). The projected patterns of both classes then define two possibly overlapping one-dimensional distributions. In the threshold stage, discrimination (or classification) simply amounts to setting a threshold between these distributions, similar to what is done in signal detection theory (Green & Swets, 1966; Wickens, 2002). Linear classifiers differ by their projection axis and their threshold, both of them being explicitly computed in our framework. While dimensionality reduction per se has been extensively studied, using, for instance, principal component analysis (Jolliffe, 2002), locally linear embedding (Roweis & Saul, 2000), non-negative matrix factorization (Lee & Seung, 1999), or neural networks (Hinton & Salakhutdinov, 2006), classification-specific dimensionality reduction as considered in this letter has surprisingly been ignored so far.

As mentioned above, the data encountered in most applications are high-dimensional and abstract, and both classes of exemplars are not always well separable. Machine learning is ideally suited to deal with such classification problems by providing a range of sophisticated classification algorithms (Vapnik, 2000; Duda et al., 2001; Schölkopf & Smola, 2002; Bishop, 2006). However, these more complex algorithms are sometimes hard to interpret and visualize and do not provide good intuition as to the nature of the solution. Furthermore, in the absence of a rigorous framework, it is hard to compare and contrast these classification methods with one other. This letter introduces a framework that puts different machine learning classifiers on the same footing—namely, that of prototype classification. Although classification is still done according to the closest prototype, these prototypes are computed using more sophisticated and more principled algorithms than simply averaging the examples in each class as for the mean-of-class prototype classifier.

We first present properties that linear classifiers, also referred to as hyperplane classifiers, must satisfy in order to be invariant to a set of transformations. We show that a linear classifier with such invariance properties can be interpreted as a generalized prototype classifier where the prototypes define the normal vector and offset of the hyperplane. We then apply the generalized prototype framework to three classes of classifiers: non-margin classifiers (the classical mean-of-class prototype classifier, the Fisher classifier, and the relevance vector machine), hard margin classifiers (the support vector machine and a novel classifier—the boosted prototype classifier), and soft margin classifiers (obtained by applying a regularized preprocessing to the data, and then classifying these data using hard margin classifiers). Subsequently we show that the prototype classifier is a limit of any soft margin classifier and that boosting a prototype classifier yields the support vector machine. Numerical simulations on a two-dimensional toy data set allow us to visualize the prototypes for the different classifiers, and finally the responses of a population of artificial neurons to two stimuli are decoded using our prototype framework.

2 Invariant Linear Classifiers

In this section, we define several requirements that a general linear classifier—a hyperplane classifier—should satisfy in terms of invariances. For example, the algorithm should not depend on the choice of a coordinate system for the space in which the data are represented. These natural requirements yield nontrivial properties of the linear classifier that we present below.

Let us first introduce some notation. We assume a two-class data set $\mathcal{D} = \{\boldsymbol{x}_i \in \mathcal{X}, y_i = \pm 1\}_{i=1}^{n}$ of n examples. We denote by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ the input patterns (in finite dimensional spaces, these are represented as column vectors), elements of an inner product space $\mathcal{X}$, and by $y_1, \ldots, y_n$ their labels in $\{-1, 1\}$ where we define by $\mathcal{Y}_{\pm} = \{i \mid y_i = \pm 1\}$ the two classes resulting from $\mathcal{D}$ and by $n_{\pm} =\mid \mathcal{Y}_{\pm} \mid$ their size. Let $\boldsymbol{y}$ be the vector of labels and $\mathbf{X}$ denote the set of input vectors; in finite-dimensional spaces, $\mathbf{X} = \{\boldsymbol{x}_i\}_{i=1}^{n}$ is represented as a matrix whose columns are the $\boldsymbol{x}_i$. A classification algorithm A takes as input a data set $\mathcal{D}$ and outputs a function $f : \mathcal{X} \to \mathbb{R}$ whose sign is the predicted class. We are interested in specific algorithms, typically called *linear classifiers*, that produce a signed affine decision function:

$$g(\boldsymbol{x}) = \operatorname{sign}(f(\boldsymbol{x})) = \operatorname{sign}(\boldsymbol{w}^t\boldsymbol{x} + b)\,, \tag{2.1}$$

where $\boldsymbol{w}^t\boldsymbol{x}$ stands for the inner product in $\mathcal{X}$ and the sign function takes values $\operatorname{sign}(z) = -1, 0, 1$ according to whether $z < 0$, $z = 0$ or $z > 0$, respectively. For such classifiers, the set of patterns $\boldsymbol{x}$ such that $g(\boldsymbol{x}) = 0$ is a hyperplane called the *separating hyperplane* (SH), which is defined by its normal vector $\boldsymbol{w}$ (sometimes also referred to as the weight vector) and offset b. A pattern $\boldsymbol{x}$ belongs to either side of the SH according to the class $g(\boldsymbol{x})$ (a pattern on the SH does not get assigned to any class). The function $f(\boldsymbol{x})$ is proportional to the signed distance $\frac{f(x)}{\|\boldsymbol{w}\|}$ of the example to the separating hyperplane. Since $\mathcal{X}$ is a (subset of a) vector space, we can consider that the data set $\mathcal{D}$ is composed of a matrix $\mathbf{X}$ and a vector $\boldsymbol{y}$. We can now formulate the notion of invariance of the classifiers we consider.

Definition 1 (invariant classifier). *Invariance of $A(\mathbf{X}, \boldsymbol{y})(\boldsymbol{x})$ with respect to a certain transformation (T_x, T_y) (where T_x applies to the $\mathcal{X}$ space while T_y applies to the $\mathcal{Y}$ space) means that for all $\boldsymbol{x}$ and all $(\mathbf{X}, \boldsymbol{y})$,*

$$A\left(T_x(\mathbf{X}), T_y(\boldsymbol{y}))(T_x(\boldsymbol{x})\right) = T_y(A(\mathbf{X}, \boldsymbol{y})(\boldsymbol{x}))\,.$$

Put in less formal words, an algorithm is invariant with respect to a transformation if the produced decision function does not change when the transformation is applied to all data to be classified by the decision function. We conjecture that a "reasonable" classifier should be invariant to the following transformations:

- **Unitary transformation**. This is a rotation or symmetry, that is, a transformation that leaves inner products unchanged. Indeed if $\mathbf{U}$ is a unitary matrix, $(\mathbf{U}\boldsymbol{x})^t(\mathbf{U}\boldsymbol{y}) = \boldsymbol{x}^t\boldsymbol{y}$. This transformation affects the coordinate representation of the data but should not affect the decision function.
- **Translation**. This corresponds to a change of origin. Such a transformation $\boldsymbol{u}$ changes the inner products $(\boldsymbol{x} + \boldsymbol{u})^t(\boldsymbol{y} + \boldsymbol{u}) = \boldsymbol{x}^t\boldsymbol{y} + (\boldsymbol{x} + \boldsymbol{y})^t\boldsymbol{u} + \boldsymbol{u}^t\boldsymbol{u}$ but should not affect the decision function.
- **Permutation of the inputs**. This is a reordering of the data. Any learning algorithm should in general be invariant to permutation of the inputs.
- **Label inversion**. In the absence of information on the classes, it is reasonable to assume that the positive and negative classes have an equivalent role, so that changing the signs of the data should simply change the sign of the decision function.

- **Scaling**. This corresponds to a dilation or a retraction of the space. It should also not affect the decision function since in general, the scale comes from an arbitrary choice of units in the measured quantities.

When we impose these invariances to our classifiers, we get the following general proposition (see appendix A for the proof):

Proposition 1. *A linear classifier that is invariant with regard to unitary transformations, translations, inputs permutations, label inversions, and scaling produces a decision function g that can be written as*

$$g(\boldsymbol{x}) = sign\left(\sum_{i=1}^{n} y_i \alpha_i \boldsymbol{x}_i^t \boldsymbol{x} + b\right), \tag{2.2}$$

with

$$\sum_{i=1}^{n} y_i \alpha_i = 0, \quad \sum_{i=1}^{n} |\alpha_i| = 2,$$

and where α_i depends on only the relative values of the inner products and the differences in labels and b depend on only the inner products and the labels. Furthermore, in the case where $\boldsymbol{x}_i^t \boldsymbol{x}_j = \lambda \delta_{ij}$, for some $\lambda > 0$, we have $\alpha_i = 1/n_{\pm}$.

The normal vector of the SH is then expressed as $\boldsymbol{w} = \sum_i y_i \alpha_i \boldsymbol{x}_i$. For a classifier satisfying the assumptions of proposition 1, we call the representation of equation 2.2 the *canonical* representation. In the next proposition (see appendix B for the proof), we fix the classification algorithm and vary the data, as, for example, when extending an algorithm from hard to soft margins (see section 6):

Proposition 2. *Consider a linear classifier that is invariant with regard to unitary transformations, translations, input permutations, label inversions, and scaling. Assume that the coefficients α_i of the canonical representation in equation 2.2 are continuous at $\mathbf{K} = \mathbf{I}$ (where $\mathbf{K}$ is the matrix of inner products between input patterns and $\mathbf{I}$ the identity matrix). If the constant δ_{ij}/C is added to the inner products, then, as $C \to 0$, for any data set, the decision function returned by the algorithm will converge to the one defined by $\alpha_i = 1/n_{\pm}$.*

For most classification algorithms, the condition $\sum_i \mid \alpha_i \mid = 2$ can be enforced by rescaling the coefficients α_i. Furthermore, most algorithms are usually rotation invariant. However, they depend on the choice of the origin and are thus not a priori translation invariant, and in the most general case, the dual algorithm may not satisfy the condition $\sum_i y_i \alpha_i = 0$. One way to ensure that the coefficients returned by the algorithm do satisfy this condition directly is to center the data, the prime denoting a centered parameter:

$$x_i' = x_i - c \quad \text{where} \quad c = \frac{1}{n}\sum_i x_i. \tag{2.3}$$

Setting $\gamma_i = \alpha_i y_i$, we can write:

$$\begin{aligned} w' &= \sum_i \gamma_i' x_i' = \sum_i \gamma_i' x_i - \frac{1}{n}\sum_i \gamma_i' \sum_j x_j = \\ &= \sum_i \left(\gamma_i' - \frac{1}{n}\sum_j \gamma_j' \right) x_i \doteq \sum_i \gamma_i x_i = w, \end{aligned}$$

where $\gamma_i = \gamma_i' - \frac{1}{n}\sum_j \gamma_j'$. Clearly, we then have $\sum_i \gamma_i = 0$. The equations of the SH on the original data are then

$$w = w' \quad \text{and} \quad b = b' - w^t c \tag{2.4}$$

since we have $0 = (w')^t x' + b' = w^t(x - c) + b' = w^t x + b' - w^t c$. Because of the translation invariance, centering the data does not change the decision function.

3 On the Universality of Prototype Classification

In the previous section we showed that a linear classifier with invariance to a set of natural transformations has some interesting properties. We here show that linear classifiers satisfying these properties can be represented in a generic form, our so-called *prototype framework*.

In the prototype algorithm, one "representative" or prototype is built for each class from the input vectors. The class of a new input is then predicted as the class of the prototype that is closest to this input (nearest-neighbor rule). Denoting by $p_\pm$ the prototypes, we can write the decision function of the classical prototype algorithm as

$$g(x) = \text{sign}\left(\|x - p_-\|^2 - \|x - p_+\|^2 \right). \tag{3.1}$$

This is a linear classifier since it can be written as $g(x) = \text{sign}(w^t x + b)$ with

$$w = p_+ - p_- \quad \text{and} \quad b = \frac{\|p_-\|^2 - \|p_+\|^2}{2} \tag{3.2}$$

In other words, once the prototypes are known, the SH passes through their average $(\boldsymbol{p}_+ + \boldsymbol{p}_-)/2$ and is perpendicular to them. The prototype classification algorithm is arguably simple, and also intuitive since it has an easy geometrical interpretation. We now introduce a generalized notion of prototype classifier, where a shift is allowed in the decision function.

Definition 2 (generalized prototype classifier). *A generalized prototype classifier is a learning algorithm whose decision function can be written as*

$$g(\boldsymbol{x}) = sign\left(\|\boldsymbol{x} - \boldsymbol{p}_-\|^2 - \|\boldsymbol{x} - \boldsymbol{p}_+\|^2 + S\right), \tag{3.3}$$

where the vectors $\boldsymbol{p}_+$ and $\boldsymbol{p}_-$ are elements of the convex hulls of two disjoint subsets of the input data and where $S \in \mathbb{R}$ is an offset (called the shift of the classifier).

From definition 2, we see that $g(\boldsymbol{x})$ can be written as $g(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^t\boldsymbol{x} + b)$ with $\boldsymbol{w} = \boldsymbol{p}_+ - \boldsymbol{p}_-$ and $b = \frac{\|\boldsymbol{p}_-\|^2 - \|\boldsymbol{p}_+\|^2 + S}{2}$. Using proposition 1, we get the following proposition:

Proposition 3. *Any linear classifier that is invariant with respect to unitary transforms, translations, input permutations, label inversion, and scaling is a generalized prototype classifier. Moreover, if the classifier is given in canonical form by α_i and b, then the prototypes are given by*

$$\begin{cases} \boldsymbol{p}_+ = + \sum\limits_{y_i\alpha_i > 0} y_i\alpha_i\boldsymbol{x}_i \\ \boldsymbol{p}_- = - \sum\limits_{y_i\alpha_i < 0} y_i\alpha_i\boldsymbol{x}_i, \end{cases} \tag{3.4}$$

and the shift is given by

$$S = 2b + \|\boldsymbol{p}_+\|^2 - \|\boldsymbol{p}_-\|^2. \tag{3.5}$$

Clearly we have $\boldsymbol{w} = \boldsymbol{p}_+ - \boldsymbol{p}_- = \sum_i y_i\alpha_i\boldsymbol{x}_i$. In the next three sections, we explicitly compute the parameters α_i and b of some of the most common hyperplane classifiers that are invariant with respect to the transformations mentioned in section 2 and can thus be cast into the generalized prototype framework. These algorithms belong to three distinct classes: non-margin, hard margin, and soft margin classifiers.

4 Non-Margin Classifiers

We consider in this section three common classification algorithms that do not allow a margin interpretation: the mean-of-class prototype classifier that inspired this study; the Fisher classifier, which is commonly used in statistical data analysis; and the relevance vector machine, which is a sparse probabilistic classifier. For convenience, we use the notation $\gamma_i = y_i \alpha_i$ throughout this section.

4.1 Classical Prototype Classifier. We study here the classification algorithm that inspired this study. One of the simplest and most basic example classification algorithms is the mean-of-class prototype learner (Reed, 1972), which assigns an unseen example $\boldsymbol{x}$ to the class whose mean, or prototype, is closest to it. The prototypes are here simply the average example of each class and can be seen as the center of mass of each class assuming a homogeneous punctual mass distribution on each example. The parameters of the hyperplane in the dual space are then

$$\boldsymbol{w} = \sum_i \gamma_i \boldsymbol{x}_i \quad \text{and} \quad b = -\frac{1}{2} \sum_{i=\pm} \frac{\sum_{y_i} \boldsymbol{w}^t \boldsymbol{x}_k}{n_i} \tag{4.1}$$

where

$$\gamma_i = \frac{y_i + 1}{2n_+} + \frac{y_i - 1}{2n_-}. \tag{4.2}$$

In the above, we clearly have $\sum_i \gamma_i = 0$, implying that the data do not need centering. Moreover, the SH is centered ($S = 0$). One problem arising when using prototype learners is the absence of a way to refine the prototypes to reflect the actual structure (e.g., covariance) of the classes. In section 5.2, we remedy this situation by proposing a novel algorithm for boosted prototype learning.

4.2 Fisher Linear Discriminant. The Fisher linear discriminant (FLD) finds a direction in the data set that allows best separation of the two classes according to the Fisher score (Duda et al., 2001). This direction is used as the normal vector of the separating hyperplane, the offset being computed so as to be optimal with respect to the least mean square error. Following Mika, Rätsch, Weston, Schölkopf, and Müller (2003), the FLD is expressed in the dual space as

$$\boldsymbol{w} = \sum_i \gamma_i \boldsymbol{x}_i \quad \text{and} \quad b = -\frac{1}{2} \sum_{i=\pm} \frac{\sum_{y_i} \boldsymbol{w}^t \boldsymbol{x}_k}{n_i}. \tag{4.3}$$

The vector $\boldsymbol{\gamma}$ is the leading eigenvector of $\mathbf{M}\boldsymbol{\gamma} = \lambda \mathbf{N} \boldsymbol{\gamma}$, where the between-class variance matrix is defined as $\mathbf{M} = (\boldsymbol{m}_- - \boldsymbol{m}_+)(\boldsymbol{m}_- - \boldsymbol{m}_+)^t$ and the within-class variance matrix as $\mathbf{N} = \mathbf{K}\mathbf{K}^t - \sum_{i=\pm} n_i \boldsymbol{m}_i \boldsymbol{m}_i^t$. The Gram matrix of the data is computed as $\mathbf{K}_{ij} = \boldsymbol{x}_i^t \boldsymbol{x}_j$, and the means of each class are defined as $\boldsymbol{m}_\pm = \frac{1}{n_\pm} \mathbf{K} \boldsymbol{u}_\pm$, where $\boldsymbol{u}_\pm$ is a vector of size n with value 0 for

$i \mid y_i = \mp 1$ and value 1 for $i \mid y_i = \pm 1$. In most applications, in order to have a well-conditioned eigenvalue problem, it may be necessary to regularize the matrix $\mathbf{N}$ according to $\mathbf{N} \to \mathbf{N} + C\mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

4.3 Relevance Vector Machine. The relevance vector machine (RVM) is a probabilistic classifier based on sparse Bayesian inference (Tipping, 2001). The offset is included in $\boldsymbol{w} = \sum_{i=0}^{n} \gamma_i \boldsymbol{x}_i$ using the convention $\gamma_0 = b$ and extending the dimensionality of the data as $\boldsymbol{x}_i|_0 = 1 \quad \forall i = 1, \ldots, n$, yielding,

$$\boldsymbol{w} = \sum_{i=1}^{n} \gamma_i \boldsymbol{x}_i \quad \text{and} \quad b = w_0. \tag{4.4}$$

The two classes of inputs define two possible "states" that can be modeled by a Bernoulli distribution,

$$p(\boldsymbol{y} \mid \mathbf{X}, \boldsymbol{\gamma}) = \prod_{i=1}^{n} s_i^{\frac{1+y_i}{2}} [1 - s_i]^{\frac{1-y_i}{2}} \quad \text{with} \quad s_i = \frac{1}{1 + \exp(-[\mathbf{C}\boldsymbol{\gamma}]_i)}, \tag{4.5}$$

where $\mathbf{C}_{ij} = [1 \mid \boldsymbol{x}_i^t \boldsymbol{x}_j]$ is the "extended" Gram matrix of the data. An unknown gaussian hyperparameter $\boldsymbol{\beta}$ is introduced to ensure sparsity and smoothness of the dual space variable $\boldsymbol{\gamma}$:

$$p(\boldsymbol{\gamma} \mid \boldsymbol{\beta}) = \prod_{i=1}^{n} \mathcal{N}\big(\gamma_i \mid 0, \beta_i^{-1}\big). \tag{4.6}$$

Learning of $\boldsymbol{\gamma}$ then amounts to maximizing the probability of the targets $\boldsymbol{y}$ given the patterns $\mathbf{X}$ with respect to $\boldsymbol{\beta}$ according to

$$p(\boldsymbol{y} \mid \mathbf{X}, \boldsymbol{\beta}) = \int p(\boldsymbol{y} \mid \mathbf{X}, \boldsymbol{\gamma}) p(\boldsymbol{\gamma} \mid \boldsymbol{\beta}) d\boldsymbol{\gamma}. \tag{4.7}$$

The Laplace approximation is used to approximate the integrand locally using a gaussian function around its most probable mode. The variable $\boldsymbol{\gamma}$ is then determined from $\boldsymbol{\beta}$ using equation 4.6. In the update of $\boldsymbol{\beta}$, some $\beta_i \to \infty$, implying an infinite peak of $p(\gamma_i \mid \beta_i)$ around 0, or equivalently, $\gamma_i = 0$. This feature of the RVM ensures sparsity and defines the relevance vectors (RVs): $\beta_i < \infty \Leftrightarrow \boldsymbol{x}_i \in RV$.

5 Hard Margin Classifiers

In this section we consider classifiers that base their classification on the concept of margin stripe between the classes. We consider the state-of-the-art support vector machine and also develop a novel algorithm based on boosting the classical mean-of-class prototype classifier. As presented here, these classifiers need a linearly separable data set (Duda et al., 2001).

5.1 Support Vector Machine. The support vector machine (SVM) is rooted in statistical learning theory (Vapnik, 2000; Schölkopf & Smola, 2002). It computes a separating hyperplane that separates best both classes by maximizing the margin stripe between them. The primal hard margin SVM algorithm is expressed as

$$\begin{aligned}&\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|^2\\ &\text{subject to } y_i(\boldsymbol{w}^t\boldsymbol{x}_i + b) \geq 1 \quad \forall i.\end{aligned} \tag{5.1}$$

The saddle points of the corresponding Lagrangian yield the dual problem:

$$\begin{aligned}&\max_{\alpha}\left[\sum_i \alpha_i - \frac{1}{2}\sum_{ij} y_i y_j \alpha_i \alpha_j \boldsymbol{x}_i^t \boldsymbol{x}_j\right]\\ &\text{subject to } \sum_i \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0 \quad \forall i.\end{aligned} \tag{5.2}$$

The Karush-Kuhn-Tucker conditions (KKT) of the above problem are written as

$$\alpha_i[y_i(\boldsymbol{w}^t\boldsymbol{x}_i + b) - 1] = 0 \quad \forall i.$$

The SVM algorithm is sparse in the sense that typically, many $\alpha_i = 0$. We then define the support vectors (SVs) as $\boldsymbol{x}_i \in SV \Leftrightarrow \alpha_i \neq 0$. The SVM algorithm can be cast into our prototype framework as follows:

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i \quad \text{and} \quad b = \left\langle y_i - \boldsymbol{w}^t\boldsymbol{x}_i \right\rangle_{i|\alpha_i \neq 0}, \tag{5.3}$$

where b is computed using the KKT condition by averaging over the SVs. The update rule for $\boldsymbol{\alpha}$ is given by equation 5.2. Using one of the saddle points of the Lagrangian, multiplying each term of the KKT conditions by $\sum_i y_i\cdot$, we obtain

$$b = -\frac{\sum_i \alpha_i \boldsymbol{w}^t\boldsymbol{x}_i}{\sum_i \alpha_i}. \tag{5.4}$$

Since $\sum_i \alpha_i y_i = 0$, no centering of the data is required. Furthermore, the shift of the offset of the SH is zero:

$$S = 2b + \| \boldsymbol{p}_+ \|^2 - \| \boldsymbol{p}_- \|^2 = 2b + (\boldsymbol{p}_+ - \boldsymbol{p}_-)^t (\boldsymbol{p}_+ + \boldsymbol{p}_-) =$$
$$= 2b + \sum_i \alpha_i \boldsymbol{w}^t \boldsymbol{x}_i = 2 \left(- \frac{\sum_i \alpha_i \boldsymbol{w}^t \boldsymbol{x}_i}{\sum_i \alpha_i} \right) + \sum_i \alpha_i \boldsymbol{w}^t \boldsymbol{x}_i = 0,$$

using $\sum_i \alpha_i = \sum_i \mid \alpha_i \mid = 2$ since $\alpha_i > 0$.

5.2 Boosted Prototype Classifier. Generally boosting methods aim at improving the performance of a simple classifier by combining several such classifiers trained on variants of the initial training sample. The principle is to iteratively give more weight to the training examples that are hard to classify, train simple classifiers so that they have a small error on those hard examples (i.e., small weighted error), and then make a vote of the obtained classifiers (Schapire & Freund, 1997). We consider below how to boost the classical mean-of-class prototype classifiers in the context of hard margins. The boosted prototype algorithm that we will develop in this section cannot exactly be cast into our prototype framework since it is still an open problem to determine the invariance properties of the boosted prototype algorithm. However, the boosted prototype classifier is an important example of how the concept of prototype can be extended.

Boosting methods can be interpreted in terms of margins in a certain feature space. For this, let $\mathcal{H}$ be a set of classifiers (i.e., functions from $\mathcal{X}$ to $\mathbb{R}$) and define the set of convex combinations of such basis classifiers as

$$\mathcal{F} = \left\{ f = \sum_{i=1}^{l} v_i h_i : l \in \mathbb{N},\ v_i \geq 0,\ \sum_{i=1}^{l} v_i = 1,\ h_i \in \mathcal{H} \right\}.$$

For a function $f \in \mathcal{F}$ and a training sample $(\boldsymbol{x}_i, y_i)$, we define the margin as $y_i f(\boldsymbol{x}_i)$. It is non-negative when the training sample is correctly classified and negative otherwise, and its absolute value gives a measure of the confidence with which f classifies the sample. The problem to be solved in the boosting scenario is the maximization of the smallest margin in the training sample (Schapire, Freund, Bartlett, & Lee, 1998):

$$\max_{f \in \mathcal{F}} \min_{i=1,\ldots,n} y_i f(\boldsymbol{x}_i). \tag{5.5}$$

It is known that the solution of this problem is a convex combination of elements of $\mathcal{H}$ (Rätsch & Meir, 2003). Let us now consider the case where the base class $\mathcal{H}$ is the set of linear functions corresponding to all possible prototype learners. In other words, $\mathcal{H}$ is the set of all affine functions that can be written as $h(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + b$ with $\|\boldsymbol{w}\| = 1$. It can easily be seen that equation 5.5, using hypotheses of the form $h(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + b$, is equivalent to

$$\max_{f\in\mathcal{F},b\in\mathbb{R}}\min_{i=1,\ldots,n} y_i(f(\boldsymbol{x}_i) + b) \tag{5.6}$$

using hypotheses of the form $h(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x}$ (i.e., without bias). We therefore consider for simplicity the hypothesis set $\mathcal{H} := \{\boldsymbol{x} \mapsto \boldsymbol{w}^t\boldsymbol{x} \mid \|\boldsymbol{w}\| = 1\}$.

Several iterative methods for solving equation 5.5 have been proposed (Breiman, 1999; Schapire, 2001; Rudin, Daubechies, & Schapire, 2004). We will not pursue this idea further, but what we want to emphasize is the interpretation of such methods. In order to ensure the convergence of the boosting algorithm when using the prototype classifier as a weak learner, we have to find, for any weights on the training examples, an element of $\mathcal{H}$ that (at least approximately) maximizes the weighted margin:

$$\operatorname*{argmax}_{h\in\mathcal{H}}\sum_i \alpha_i y_i h(\boldsymbol{x}_i), \tag{5.7}$$

where $\boldsymbol{\alpha}$ represents the weighting of the examples as computed by the boosting algorithm. It can be shown (see section 7.2) that under the condition $\sum_i \alpha_i y_i = 0$, the solution of equation 5.7 is given by the prototype classifier with

$$\boldsymbol{p}_\pm = \frac{\sum_{y_\pm}\alpha_i\boldsymbol{x}_i}{\sum_i\alpha_i}. \tag{5.8}$$

We can now state an iterative algorithm for our boosted prototype classifier. This algorithm is an adaptation of AdaBoost* (Rätsch & Warmuth, 2005), which includes a bias term. A convergence analysis of this algorithm can be found in Rätsch and Warmuth (2005). The patterns have to be normalized to lie in the unit ball (i.e., $|\ \boldsymbol{w}^t\boldsymbol{x}_i\ |\leq 1 \quad \forall\boldsymbol{w}, i$ with $\|\boldsymbol{w}\| = 1$). The first iteration of our boosted prototype classifier is the classical mean-of-class prototype classifier. Then, during boosting, the boosted prototype classifier maintains a distribution of weights α_i on the input patterns and at each step computes the corresponding weighted prototype. Then the patterns where the classifier makes mistakes have their weight increased, and the procedure is iterated until convergence. This algorithm maintains a set of weights that are separately normalized for each class, yielding the following pseudocode:

```
1. Determine the scale factor of the whole data set $\mathcal{D}$: $s = \max_i(\|\boldsymbol{x}_i\|)$.
2. Scale $\mathcal{D}$ such that $\|\boldsymbol{x}_i\| \leq 1$ by applying $\boldsymbol{x}_i \to \frac{\boldsymbol{x}_i}{s} \quad \forall i$.
3. Set the accuracy parameter $\epsilon$ (e.g., $\epsilon = 10^{-2}$).
4. Initialize the weights $\alpha_i^1 = \frac{1}{n_\pm}$ and the target margin $\rho_0 = 1$.
```

5. Do $k = 1, \ldots, k_{\max}$; compute:
   (a) The weighted prototypes: $\boldsymbol{p}_\pm{}^k = \sum_{i\in\mathcal{Y}_\pm} \alpha_i^k \boldsymbol{x}_i$
   (b) The normalized weight vector: $\boldsymbol{w}^k = \frac{\boldsymbol{p}_+{}^k - \boldsymbol{p}_-{}^k}{\|\boldsymbol{p}_+{}^k - \boldsymbol{p}_-{}^k\|}$
   (c) The target margin $\rho_k = \min(\rho_{k-1}, \frac{\gamma_k^+ + \gamma_k^-}{2} - \epsilon)$ where $\gamma_k^\pm = \sum_{i\in\mathcal{C}_\pm} \alpha_i^k y_i (\boldsymbol{w}^k)^t \boldsymbol{x}_i$
   (d) The weight for the prototype:
   $$v^k = \frac{1}{8} \log \frac{(2 + \gamma^+ - \rho_k)(2 + \gamma^- - \rho_k)}{(2 - \gamma^+ + \rho_k)(2 - \gamma^- + \rho_k)}$$
   (e) The bias shift parameter:
   $$\beta^k = \frac{1}{2} \log \frac{\sum_{i\in\mathcal{C}_+} \alpha_i^k e^{-v^k y_i (\boldsymbol{w}^k)^t \boldsymbol{x}_i}}{\sum_{i\in\mathcal{C}_-} \alpha_i^k e^{-v^k y_i (\boldsymbol{w}^k)^t \boldsymbol{x}_i}}$$
   (f) The weight update:
   $$\alpha_i^{k+1} = \frac{\alpha_i^k e^{-v^k y_i (\boldsymbol{w}^k)^t \boldsymbol{x}_i + \rho_k v^k}}{Z_k^\pm},$$
   where the normalization $Z_k^\pm$ is such that $\sum_{i\in\mathcal{C}_\pm} \alpha_i^{k+1} = 1$.
6. Determine the aggregated prototypes, normal vector, and bias:
$$\boldsymbol{p}_\pm = s \frac{\sum_{k=1}^{k_{\max}} v^k \boldsymbol{p}_\pm{}^k}{\sum_{k=1}^{k_{\max}} v^k} \quad \text{and} \quad \boldsymbol{w} = \frac{\sum_{k=1}^{k_{\max}} v^k \boldsymbol{w}^k}{\sum_{k=1}^{k_{\max}} v^k} \quad \text{and} \quad b = s \frac{\sum_{k=1}^{k_{\max}} \beta^k}{\sum_{k=1}^{k_{\max}} v^k},$$

In the final expression for $\boldsymbol{p}_\pm$, $\boldsymbol{w}$, and b, the factor $\sum_k v^k$ ensures that these quantities are in the convex hull of the data. Moreover, since the data are scaled by s, the bias and the prototypes have to be rescaled according to $\boldsymbol{w}^t \boldsymbol{x} + b \leftrightarrow \boldsymbol{w}^t(s\boldsymbol{x}) + sb$. In practice, it is important to note that the choice of ϵ must be coupled with the number of iterations of the algorithm.

We can express the prototypes as a linear combination of the input examples:

$$\boldsymbol{p}_\pm = \sum_{i\in\mathcal{Y}_\pm} \left(\sum_k \frac{s v^k}{\sum_l v^l} \alpha_i^k \right) \boldsymbol{x}_i,$$

where the scale factor s, the weight update α_i^k, and the weight v_k are defined above. The weight vector $\boldsymbol{w}$, however, is not a linear combination of the patterns since there is a normalization factor in the expression of $\boldsymbol{w}^k$. The decision function at each iteration is implicitly given by $h^k(\boldsymbol{x}) = \text{sign}(\|\boldsymbol{x} - \boldsymbol{p}_-{}^k\|^2 - \|\boldsymbol{x} - \boldsymbol{p}_+{}^k\|^2)$, while at the last iteration of the algorithm, it reverts to the usual form: $f(\boldsymbol{x}) = \boldsymbol{w}^t \boldsymbol{x} + b$.

6 Soft Margin Classifiers

The problem with the hard margin classifiers is that when the data are not linearly separable, these algorithms will not converge at all or will converge to a solution that is not meaningful (the non-margin classifiers are not affected by this problem). We deal with this problem by extending the hard margin classifiers to soft margin classifiers. For this, we apply a form of "regularized" preprocessing to the data, which then become linearly separable. The hard margin classifiers can subsequently be applied on these processed data. Alternatively, in the case of the SVM, we can also rewrite its formulation in order to allow nonlinearly separable data sets.

6.1 From Hard to Soft Margins. In order to classify data that are not linearly separable using a classifier that assumes linear separability (such as the hard margin classifiers), we preprocess the data by increasing the dimensionality of the patterns $\boldsymbol{x}_i$ in the data:

$$\boldsymbol{x}_i \rightarrow \boldsymbol{X}_i = \begin{pmatrix} \boldsymbol{x}_i \\ \frac{\boldsymbol{e}_i}{\sqrt{C}} \end{pmatrix}, \tag{6.1}$$

where $\frac{1}{\sqrt{C}}$ appears at the ith row after $\boldsymbol{x}_i$ ($\boldsymbol{e}_i$ is the ith unit vector) and C is a regularization constant. The (hard margin) classifier then operates on the patterns $\boldsymbol{X}_i$ instead of the original patterns $\boldsymbol{x}_i$ using a new scalar product:

$$\boldsymbol{X}_i^t \boldsymbol{X}_j = \boldsymbol{x}_i^t \boldsymbol{x}_j + \frac{\delta_{ij}}{C}. \tag{6.2}$$

The above corresponds to adding a diagonal matrix to the Gram matrix in order to make the classification problem linearly separable. The soft margin preprocessing allows us to extend hard margin classification to accommodate overlapping classes of patterns. Clearly, the hard margin case is obtained by setting $C \rightarrow \infty$. Once the SH and prototypes are obtained in the space spanned by $\boldsymbol{X}_i$, their counterparts in the space of the $\boldsymbol{x}_i$ are computed by simply ignoring the components added by the preprocessing.

6.2 Soft Margin SVM. In the case of the SVM, we can change the formulation of the algorithm in order to deal with nonlinearly separable data sets (Vapnik, 2000; Schölkopf & Smola, 2002). For this, we first consider the 2-norm soft margin SVM with quadratic slacks. The primal SVM algorithm is expressed as

$$\begin{aligned} &\min_{\boldsymbol{w},b,\xi} \left[\|\boldsymbol{w}\|^2 + C \sum_i \xi_i^2 \right] \\ &\text{subject to } y_i(\boldsymbol{w}^t \boldsymbol{x}_i + b) \geq 1 - \xi_i, \end{aligned} \tag{6.3}$$

where C is a regularization parameter and $\boldsymbol{\xi}$ is the slack variable vector accounting for outliers: examples that are misclassified or lie in the margin stripe. The saddle points of the corresponding Lagrangian yield the dual problem:

$$\max_{\alpha}\left[\sum_i \alpha_i - \frac{1}{2}\sum_{ij} y_i y_j \alpha_i \alpha_j \left(\boldsymbol{x}_i^t \boldsymbol{x}_j + \frac{\delta_{ij}}{C}\right)\right] \\ \text{subject to } \textstyle\sum_i \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0 \quad \forall i. \tag{6.4}$$

In the above formulation, the addition of the term $\frac{\delta_{ij}}{C}$ to the inner product $\boldsymbol{x}_i^t \boldsymbol{x}_j$ corresponds to the preprocessing introduced in equation 6.2. The KKT conditions of the above problem are written as

$$\alpha_i[y_i(\boldsymbol{w}^t \boldsymbol{x}_i + b) - 1 + \xi_i] = 0 \quad \forall i.$$

The SVM algorithm is then cast into our prototype framework as follows:

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i \quad \text{and} \quad b = \left\langle y_i\left(1 - \frac{\alpha_i}{C}\right) - \boldsymbol{w}^t \boldsymbol{x}_i \right\rangle_{i|\alpha_i \neq 0}, \tag{6.5}$$

where b is computed using the first constraint of the primal problem applied on the margin SVs given by $0 < \alpha_i < C$ and $\xi_i = 0$. The update rule for $\boldsymbol{\alpha}$ is given by equation 6.4. Using one of the saddle points of the Lagrangian $\boldsymbol{\alpha} = C\boldsymbol{\xi}$ and applying $\sum_i y_i \cdot$ to the KKT conditions, we get

$$b = -\frac{\sum_i \alpha_i \boldsymbol{w}^t \boldsymbol{x}_i}{\sum_i \alpha_i} - \frac{\sum_i \alpha_i^2 y_i}{C \sum_i \alpha_i}. \tag{6.6}$$

Setting $C \to \infty$ in the above equations yields the expression for the hard margin SVM obtained in equation 5.4.

We now discuss the case of the 1-norm soft margin SVM, which is more widespread than the SVM with quadratic slacks. The primal SVM algorithm is written as

$$\min_{\boldsymbol{w},b,\xi}\left[\|\boldsymbol{w}\|^2 + C\sum_i \xi_i\right] \\ \text{subject to } y_i(\boldsymbol{w}^t \boldsymbol{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0, \tag{6.7}$$

where C is a regularization parameter and $\boldsymbol{\xi}$ is the slack variable vector. The saddle points of the corresponding Lagrangian yield the dual problem:

$$\max_{\alpha}\left[\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \boldsymbol{x}_i^t \boldsymbol{x}_j\right] \\ \text{subject to } \sum_i \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C. \tag{6.8}$$

The KKT conditions are then written as $\alpha_i[y_i(\boldsymbol{w}^t\boldsymbol{x}_i + b) - 1 + \xi_i] = 0 \quad \forall i$. The above allows us to cast the SVM algorithm into our prototype framework,

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i \quad \text{and} \quad b = \frac{1}{\mid 0 < \alpha_i < C \mid} \sum_{i \mid 0 < \alpha_i < C} (y_i - \boldsymbol{w}^t\boldsymbol{x}_i), \tag{6.9}$$

where b is computed using the first constraint of the primal problem applied on the margin SVs given by $0 < \alpha_i < C$ and $\xi_i = 0$. The update rule for $\boldsymbol{\alpha}$ is given by equation 6.8. In the hard margin case, also obtained for $C \to \infty$, the KKT conditions becomes $\alpha_i(y_i(\boldsymbol{w}^t\boldsymbol{x}_i + b) - 1) = 0 \quad \forall i$. From this, we deduce by application of $\sum_i y_i \cdot$ to the KKT conditions the expression of the bias obtained for the hard margin case in equation 5.4. Finally, we notice that the 1-norm SVM does not naturally yield the scalar product substitution of equation 6.2 when going from hard to soft margins.

7 Relations Between Classifiers

In this section we outline two relations between the prototype classification algorithm and the other classifiers considered in this letter. First, in the limit where $C \to 0$, we show that the soft margin algorithms converge to the classical mean-of-class prototype classifier. Second, we show that the boosted prototype algorithm converges to the SVM solution.

7.1 Prototype Classifier as a Limit of Soft Margin Classifiers. We deduce the following proposition as a direct consequence of proposition 2:

Proposition 4. *All soft margin classifiers obtained from linear classifiers whose canonical form is continuous at* $\mathbf{K} = \mathbf{I}$ *by the regularized preprocessing of equation 6.1 converge toward the mean-of-class prototype classifier in the limit where* $C \to 0$.

7.2 Boosted Prototype Classifier and SVM. While the analogy between boosting and the SVM has been suggested previously (Skurichina & Duin, 2002), we here establish that the boosting procedure applied on the classical prototype classifier yields the hard margin SVM as a solution when appropriate update rules are chosen:

Proposition 5. *The solution of the problem in equation 5.5 when* $\mathcal{H} = \{h(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + b$ *with* $\|\boldsymbol{w}\| = 1\}$ *is the same as the solution of the hard margin SVM.*

Proof. Introducing non-negative weights α_i, we first rewrite the problem of equation 5.5 in the following equivalent form:

$$\max_{f\in\mathcal{F}} \min_{\boldsymbol{\alpha}\geq 0,\, \sum\alpha_i=1} \sum_i \alpha_i y_i f(\boldsymbol{x}_i)\,.$$

Indeed, the minimization of a linear function of the α_i is achieved when one α_i (the one corresponding to the smallest term of the sum) is one and the others are zero. Now notice that the objective function is linear in the convex coefficients α_i and also in the convex coefficients representing f, so that by the minimax theorem, the minimum and maximum can be permuted to give the equivalent problem:

$$\min_{\boldsymbol{\alpha}\geq 0,\, \sum\alpha_i=1} \max_{f\in\mathcal{F}} \sum_i \alpha_i y_i f(\boldsymbol{x}_i)\,.$$

Using the fact that we are maximizing a linear function on a convex set, we can rewrite the maximization as running over the set $\mathcal{H}$ instead of $\mathcal{F}$, which gives

$$\min_{\alpha\geq 0,\, \sum\alpha_i=1} \max_{\|\boldsymbol{w}\|=1,\, b\in\mathbb{R}} \sum_i \alpha_i y_i(\boldsymbol{w}^t\boldsymbol{x}_i + b).$$

One now notices that when $\sum_i \alpha_i y_i \neq 0$, the maximization can be achieved by taking b to infinity, which would be suboptimal in terms of the minimization in the α's. This means that the constraint $\sum_i \alpha_i y_i = 0$ will be satisfied by any nondegenerate solution. Using this and the fact that

$$\max_{\|\boldsymbol{w}\|=1} \sum_i \alpha_i y_i \boldsymbol{w}^t\boldsymbol{x}_i = \left\| \sum_i \alpha_i y_i \boldsymbol{x}_i \right\|^2, \tag{7.1}$$

we finally obtain the following problem:

$$\min_{\boldsymbol{\alpha}\geq 0,\, \sum\alpha_i=1} \left\| \sum_i \alpha_i y_i \boldsymbol{x}_i \right\|^2, \quad \text{subject to } \sum_i \alpha_i y_i = 0.$$

This is equivalent to the hard margin SVM problem of equation 5.1.

In other words, in the context of hard margins, boosting a mean-of-class prototype learner is equivalent to a SVM. It is then straightforward to extend this result to the soft margin case using the regularized preprocessing of equation 6.1. Thus, without restrictions, the SVM is the asymptotic solution of a boosting scheme applied on mean-of-class prototype classifiers. The above developments also allow us to state the following:

Proposition 6. *Under the condition $\sum_i \alpha_i y_i = 0$, the solution of equation 5.7 is given by the prototype classifier defined by*

$$\boldsymbol{p}_{\pm} = \frac{\sum_{y_\pm} \alpha_i \boldsymbol{x}_i}{\sum_i \alpha_i}\,.$$

Proof. This is a consequence of the proof of proposition 5. Indeed, the vector $\boldsymbol{w}$ achieving the maximum in equation 7.1 is given by $\boldsymbol{w} = \sum_i y_i \alpha_i \boldsymbol{x}_i / \| \sum_i y_i \alpha_i \boldsymbol{x}_i \|$, which shows that $\boldsymbol{w}$ is proportional to $\boldsymbol{p}_+ - \boldsymbol{p}_-$. The choice of b is arbitrary since one has $\sum_i \alpha_i y_i = 0$, so that there exists a choice of b such that the corresponding function h is the same as the prototype function based on $\boldsymbol{p}_+$ and $\boldsymbol{p}_-$.

8 Numerical Experiments

In the numerical experiments of this section, we first illustrate and visualize our prototype framework on a linearly separable two-dimensional toy data set. Second, we apply the prototype framework to discriminate between two overlapping classes (nonlinearly separable data set) of responses from a population of artificial neurons.

8.1 Two-Dimensional Toy Data Set. In order to visualize our findings, we consider in Figure 1 a two-dimensional linearly separable toy data set where the examples of each class were generated by the superposition of three gaussian distributions with different means and different covariance matrices. We compute the prototypes and the SHs for the classical mean-of-class prototype classifier, the Fisher linear discriminant (FLD), the relevance vector machine (RVM), and the hard margin support vector machine (SVM HM). We also study the trajectories taken by the "dynamic" prototypes when using our boosted prototype classifier and when varying the soft margin regularization parameter for the soft margin SVM (SVM SM). We can immediately see that the prototype framework introduced in this letter allows one to visualize and distinguish at a glance the different classification algorithms and strategies. While the RVM algorithm per se does not allow an intuitive geometric explanation as, for instance, the SVM (the margin SVs lie on the margin stripe) or the classical mean-of-class prototype classifier, the prototypes are an intuitive and visual interpretation of sparse Bayesian learning. The different classifiers yield different SHs and consequently also a different set of prototypes. As foreseen in theory, the classical prototype and the SVM HM have no shift in the decision function $S = 0$, indicating that the SH passes through the middle of the prototypes. This shift is largest for the RVM, reflecting the fact that one of the prototypes is

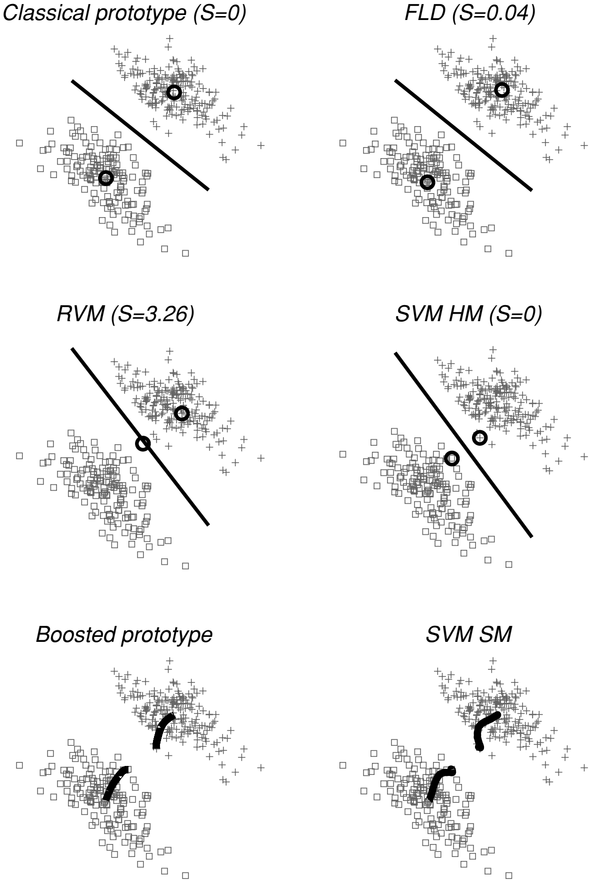

Figure 1: Classification on a two-dimensional linearly separable toy data set. For the classical prototype classifier, FLD, RVM, and SVM HM, the prototypes are indicated by the open circles, the SH is represented by the line, and the offset in the decision function is indicated by the variable S. For the boosted prototype and the SVM SM, the trajectories indicate the evolution of the prototypes during boosting and when changing the soft margin regularization parameter C, respectively.

close to the center of mass of the entire data set. This is due to the fact that the RVM algorithm usually yields a very sparse representation of the γ_i. In our example, a single γ_i, which corresponds to the prototype close to the center of one of the classes, strongly dominates this distribution, such that the other prototype is bound to be close to the mean across both classes (the center of the entire data set). The prototypes of the SVM HM are close to

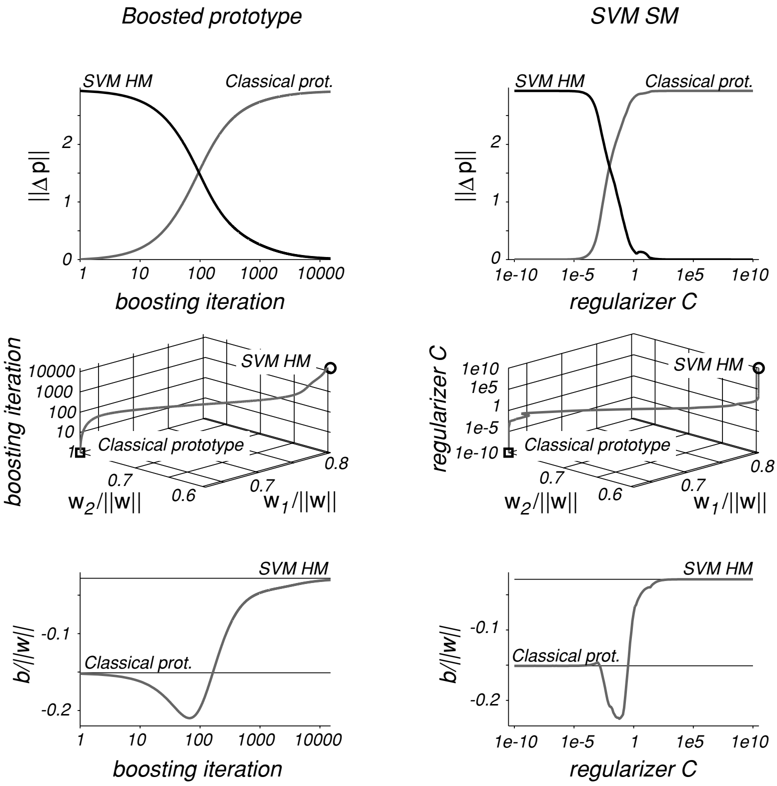

Figure 2: Dynamic evolution and convergence of the boosted prototype classifier (first column) and the soft margin SVM classifier (second column) for the two-dimensional linearly separable toy data set. The first row shows the norm of the difference between the "dynamic" prototypes and the prototype of either the classical mean-of-class prototype classifier or the hard margin SVM. The second row illustrates the convergence behavior of the normal vector $\boldsymbol{w}$ of the SH, and the third row shows the convergence of the offset b of the SH.

the SH, which is due to the fact that they are computed using only the SVs corresponding to exemplars lying on the margin stripe. When considering the trajectories of the "dynamic" prototypes for the boosted prototype and the soft margin SVM classifiers, both algorithms start close to the classical mean-of-class prototype classifier and converge to the hard margin SVM classifier. We further study the dynamics associated with these trajectories in Figure 2. The prototypes and the corresponding SH have a similar behavior in all cases. As predicted theoretically, the first iteration of boosting is identical to the classical prototype classifier. However, while the iterations proceed, the boosted prototypes get farther apart from the classical ones and finally converge as expected toward the prototypes of the hard margin

SVM solution. Similarly, when $C \to 0$, the soft margin SVM converges to the solution of the classical prototype classifier, while for $C \to \infty$, the soft margin SVM converges to the hard margin SVM.

8.2 Population of Artificial Neurons. To test our prototype framework on more realistic data, we decode the responses from a population of six independent artificial neurons. The responses of the neurons are assumed to have a gaussian noise distribution around their mean response, the variance being proportional to the mean. We use our prototype framework to discriminate between two stimuli using the population activity they elicit. This data set is not linearly separable, and the pattern distributions corresponding to both classes may overlap. We thus consider the soft margin preprocessing for the SVM and the boosted prototype classifier. We first find the value of C minimizing the training error of the SVM SM and then use this value to compute the soft margin SVM and the boosted prototype classifiers. As expected from the hard margin case, we find in Figure 3 that the boosted prototype algorithm starts as a classical mean-of-class prototype classifier, and converges toward the soft margin SVM. In order to visualize the discrimination process, we project the neural responses onto the axis defined by the prototypes (i.e., the normal vector $\boldsymbol{w}$ of the SH). Equivalently, we compute the distributions of the distances $\delta(\boldsymbol{x}) = \frac{\boldsymbol{w}^t \boldsymbol{x}+b}{\|\boldsymbol{w}\|}$ of the neural responses to the SH. Figure 4 shows these distance distributions for the classical prototype classifier, the FLD, the RVM, the soft margin SVM, and the boosted prototype classifier. The projected prototypes have locations similar to what we observed for the toy data set for the prototype classifier and the FLD. For the SVM, they can be even closer to the SH ($\delta = 0$) since they depend on only the SVs, which may here also include exemplars inside the margin stripe (and not only on the margin stripe as for the hard margin SVM). For the RVM, however, the harder classification task (high-dimensional and nonlinearly separable data set) yields a less sparse distribution of the γ_i than for the toy data set. This is reflected by the fact that none of its prototypes lies in the vicinity of the mean over the whole data set ($\delta = 0$). As already suggested in Figure 3, we can clearly observe how the boosted prototypes evolve from the prototypes of the classical mean-of-class prototype classifier to converge toward the prototypes of the soft margin SVM. Most important, the distance distributions allow us to compare our prototype framework directly with signal detection theory (Green & Swets, 1966; Wickens, 2002). Although the neural response distributions were constructed using gaussian distributions, we see that the distance distributions are clearly not gaussian. This makes most analysis such as "receiver operating characteristic" not applicable in our case. However, the different algorithms from machine learning provide a family of thresholds that can be used for discrimination, independent of the shape of the distributions. Furthermore, the distance distributions are dependent on

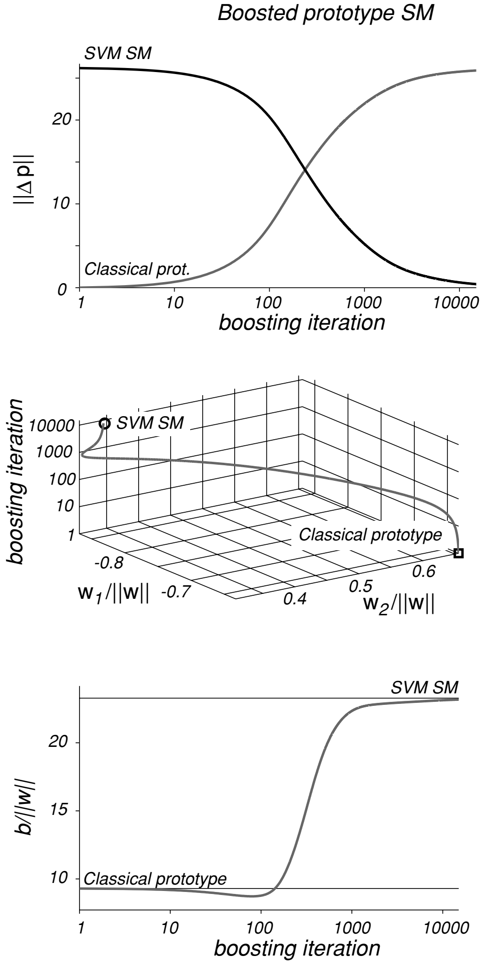

Figure 3: Dynamical evolution and convergence of the boosted prototype classifier in the soft margin case. See the caption for Figure 2.

the classifier used to compute the SH. This example illustrates one of the novelties of our prototype framework: a classifier-specific dimensionality reduction. In other words, we here visualize the space the classifiers use to discriminate: the cut through the data space provided by the axis spanned by the prototypes. As a consequence, the amount of overlap between the distance distributions is different across classifiers. Furthermore, the shape of these distributions varies: the SVM tends to cut the data such that many exemplars lie close to the SH, while for the classical prototype, the distance

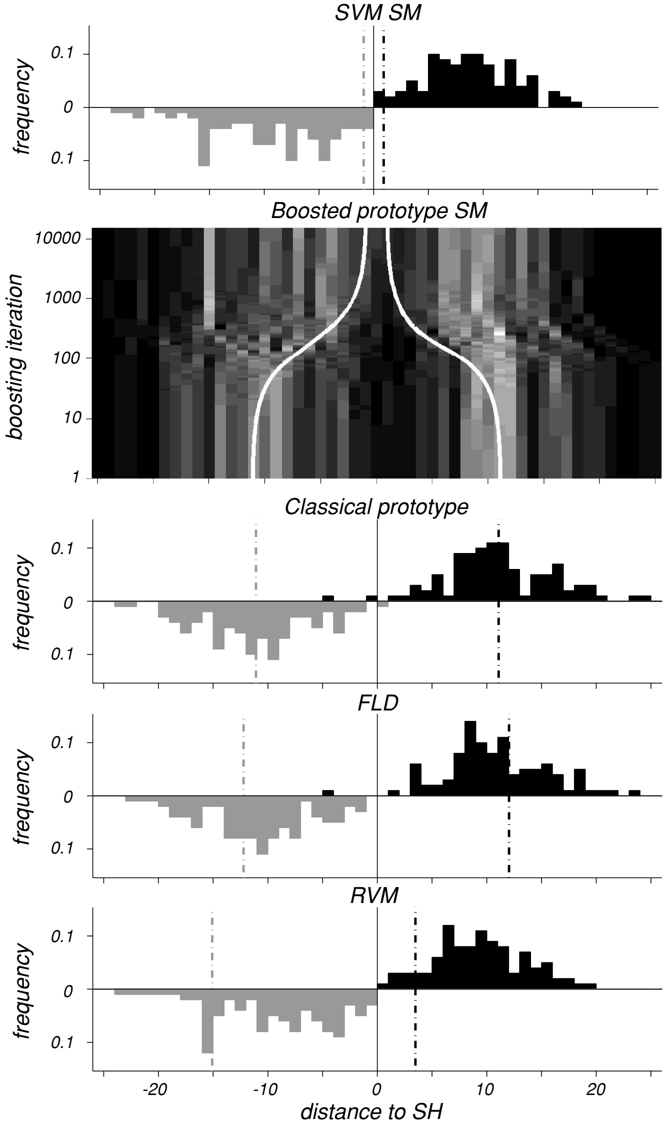

Figure 4: Distance distributions of the neural responses to the SH. For the boosted prototype classifier (second row), we indicate the distance distributions as a function of the iterations of the boosting algorithm. The trajectory of the projected "dynamic" prototypes is represented by the white line. For the remaining classifiers, we plot the distributions of distances for both classes separately and also the position of the projected prototypes (vertical dotted lines).

distributions of the same data are more centered around the means of each class. The boosted prototype classifier gives us here an insight on how the distance distribution of the mean-of-class prototype classifier evolves iteratively into the distance distribution of the soft margin SVM. This illustrates how the different projection axes are nontrivially related to generate distinct class-specific distance distributions.

9 Discussion

We introduced a novel classification framework—the prototype framework—inspired by the mean-of-class prototype classifier. While the algorithm itself is left unchanged (up to a shift in the offset of the decision function), we computed the generalized prototypes using methods from machine learning. We showed that any linear classifier with invariances to unitary transformations, translations, input permutations, label inversions, and scaling can be interpreted as a generalized prototype classifier. We introduced a general method to cast such a linear algorithm into the prototype framework. We then illustrated our framework using some algorithms from machine learning such as the Fisher linear discriminant, the relevance vector machine (RVM), and the support vector machine (SVM). In particular, we obtained through the prototype framework a visualization and a geometrical interpretation for the hard-to-visualize RVM. While the vast majority of algorithms encountered in machine learning satisfy our invariance properties, the main class of algorithms that are ruled out are online algorithms such as the perceptron since they depend on the order of presentation of the input patterns.

We demonstrated that the SVM and the mean-of-class prototype classifier, despite their very different foundations, could be linked: the boosted prototype classifier converges asymptotically toward the SVM classifier. As a result, we also obtained a simple iterative algorithm for SVM classification. Also, we showed that boosting could be used to provide multiple optimized examples in the context of prototype learning according to the general principle of divide and conquer. The family of optimized prototypes was generated from an update rule refining the prototypes by iterative learning. Furthermore, we showed that the mean-of-class prototype classifier is a limit of the soft margin algorithms from learning theory when $C \to 0$. In summary, both boosting and soft margin classification yield novel sets of "dynamic" prototypes paths: through time (the boosting iteration) and though the soft margin trade-off parameters C, respectively. These prototype paths can be seen as an alternative to the "chorus of prototypes" approach (Edelman, 1999).

We considered classification of two classes of inputs, or equivalently, we discriminated between two classes given the responses corresponding to each one. However, when faced with an estimation problem, we need to choose one class among multiple classes. For this, we can readily extend our prototype framework by considering a one-versus-the-rest strategy (Duda et al., 2001; Vapnik, 2000). The prototype of each class is then computed by discriminating this class against all the remaining ones. Repeating this procedure for all the classes yields an ensemble of prototypes—one for each class. These prototypes can then be used for multiple class classification, or estimation, using again the nearest-neighbor rule.

Our prototype framework can be interpreted as a two-stage learning scheme. First, from a learning perspective, it can be seen as a complicated and time-consuming training stage that computes the prototypes. This stage is followed by a very simple and fast nearest-prototype testing stage for classification of new patterns. Such a scheme can account for a slow training phase followed by a fast testing phase. Albeit it is beyond the scope of this letter, such a behavior may be argued to be biologically plausible. Once the prototypes are computed, the simplicity of the decision function is certainly one advantage of the prototype framework. This letter shows that it is possible to include sophisticated algorithms from machine learning such as the SVM or the RVM into the rather simple and easy-to-visualize prototype formalism. Our framework then provides an ideal method for directly comparing different classification algorithms and strategies, which could certainly be of interest in many psychophysical and neurophysiological decoding experiments.

Appendix A: Proof of Proposition 1

We work out the implications for a linear classifier to be invariant with respect to the transformations mentioned in section 2.

Invariance with regard to scaling means that the pairs $(\boldsymbol{w}_1, b_1)$ and $(\boldsymbol{w}_2, b_2)$ correspond to the same decision function, that is, $\text{sign}(\boldsymbol{w}_1{}^t\boldsymbol{x} + b_1) = \text{sign}(\alpha)\text{sign}(\boldsymbol{w}_2{}^t\boldsymbol{x} + b_2)$, $\forall x \in \mathcal{X}$, if and only if there exists some $\alpha \neq 0$ such that $\boldsymbol{w}_1 = \alpha\boldsymbol{w}_2$ and $b_1 = \alpha b_2$.

We denote by $(\boldsymbol{w}_X, b_X)$ the parameters of the hyperplane obtained when trained on data $\mathbf{X}$. We show below that invariance to unitary transformations implies that the normal vector to the decision surface $\boldsymbol{w}_X$ lies in the span of the data. This is remarkable since it allows a dual representation and it is a general form of the representer theorem (see also Kivinen, Warmuth, & Auer, 1997).

Lemma 1 (unitary invariance). *If A is invariant by application of any unitary transform* $\mathbf{U}$*, then there exists* $\boldsymbol{\gamma}$ *such that* $\boldsymbol{w}_X = \mathbf{X}\boldsymbol{\gamma}$ *is in the span of the input data and* $b_X = b_{UX}$ *depends on the inner products between the patterns of* $\mathcal{X}$ *and on the labels.*

Proof. Unitary invariance can be expressed as

$$\boldsymbol{w}_X^t\boldsymbol{x} + b_X = \boldsymbol{w}_{UX}^t\mathbf{U}\boldsymbol{x} + b_{UX}.$$

In particular, this implies $b_{UX} = b_X$ (take $x = 0$), and thus b_X does not depend on $\mathbf{U}$. This shows that b_X can depend on only inner products between the input vectors (only the inner products are invariant by $\mathbf{U}$ since $(\mathbf{U}\boldsymbol{x})^t(\mathbf{U}\boldsymbol{y}) = \boldsymbol{x}^t\boldsymbol{y}$) and on the labels. Furthermore we have the condition

$$\boldsymbol{w}_X^t\boldsymbol{x} = \boldsymbol{w}_{UX}^t\mathbf{U}\boldsymbol{x},$$

which implies (since $\mathbf{U}$ is self-adjoint)

$$\boldsymbol{w}_{UX} = \mathbf{U}\boldsymbol{w}_X,$$

so that $\boldsymbol{w}$ is transformed according to $\mathbf{U}$. We now decompose $\boldsymbol{w}_X$ as a linear combination of the patterns plus an orthogonal component:

$$\boldsymbol{w}_X = \mathbf{X}\boldsymbol{\gamma} + \boldsymbol{v},$$

where $\boldsymbol{v} \perp \text{span}\{\mathbf{X}\}$, and similarly we decompose

$$\boldsymbol{w}_{UX} = \mathbf{U}\mathbf{X}\boldsymbol{\gamma}_U + \boldsymbol{v}_U$$

with $\boldsymbol{v}_U \perp \text{span}\{\mathbf{U}\mathbf{X}\}$. We are using $\boldsymbol{w}_{UX} = \mathbf{U}\boldsymbol{w}_X$:

$$\mathbf{U}\mathbf{X}\boldsymbol{\gamma}_U + \boldsymbol{v}_U = \mathbf{U}\mathbf{X}\boldsymbol{\gamma} + \mathbf{U}\boldsymbol{v},$$

and since $\mathbf{U}\boldsymbol{v} \perp \text{span}\{\mathbf{U}\mathbf{X}\}$, then $\boldsymbol{v}_U = \mathbf{U}\boldsymbol{v}$ and $\mathbf{X}\boldsymbol{\gamma} = \mathbf{X}\boldsymbol{\gamma}_U$.

Now we introduce two specific unitary transformations. The first, $\mathbf{U}$, performs a rotation of angle π along an axis contained in span$\{\mathbf{X}\}$, and the second, $\mathbf{U}'$, performs a symmetry with respect to a hyperplane containing this axis and $\boldsymbol{v}$. Both transformations have the same effect on the data. However, they have the opposite effect on the vector $\boldsymbol{v}$. This means that in order to guarantee invariance, we need to have $\boldsymbol{v} = 0$, which shows that $\boldsymbol{w}$ is in the span of the data: $\boldsymbol{w}_X = \mathbf{X}\boldsymbol{\gamma}$.

Next, we show that in addition to the unitary invariance, invariance with respect to translations (change of origin) implies that the coefficients of the dual expansion of $\boldsymbol{w}_X$ sum to zero.

Lemma 2 (translation and unitary invariance). *If A is invariant by unitary transforms* $\mathbf{U}$ *and by translations* $\boldsymbol{v} \in \mathcal{X}$*, then there exists* $\boldsymbol{u}$ *such that* $\boldsymbol{w}_X = \mathbf{X}\boldsymbol{u}$ *and* $\boldsymbol{u}^t\boldsymbol{i} = 0$ *where* $\boldsymbol{i}$ *denotes a column vector of size n whose entries are all 1. Moreover, we also have* $b_{X+\boldsymbol{v}\boldsymbol{i}^t} = b_X - \boldsymbol{w}_X^t\boldsymbol{v}$.

Proof. The invariance condition means that for all $\mathbf{X}$, $\boldsymbol{v}$, and $\boldsymbol{x}$, we can write

$$\boldsymbol{w}_X^t \boldsymbol{x} + b_X = \boldsymbol{w}_{X+vi^t}^t (\boldsymbol{x} + \boldsymbol{v}) + b_{X+vi^t} = \boldsymbol{w}_{X+vi^t}^t \boldsymbol{x} + \boldsymbol{w}_{X+vi^t}^t \boldsymbol{v} + b_{X+t1^T}.$$

We thus obtain

$$(\boldsymbol{w}_X - \boldsymbol{w}_{X+vi^t})^t \boldsymbol{x} = -b_X + b_{X+t1^T} + \boldsymbol{w}_{X+vi^t}^t \boldsymbol{v},$$

which can be true only if $\boldsymbol{w}_X = \boldsymbol{w}_{X+vi^t}$ and $b_{X+t1^T} = b_X - \boldsymbol{w}_{X+vi^t}^t \boldsymbol{v}$. In particular, since we can write by the previous lemma $\boldsymbol{w}_X = \mathbf{X}\boldsymbol{\gamma}_X$ and $\boldsymbol{w}_{X+vi^t} = (\mathbf{X} + \boldsymbol{v}\boldsymbol{i}^t)\boldsymbol{\gamma}_{X+vi^t}$, we have for all $\boldsymbol{v}$:

$$\boldsymbol{w}_X = \mathbf{X}\boldsymbol{\gamma}_X = \mathbf{X}\boldsymbol{\gamma}_{X+vi^t} + \boldsymbol{v}\boldsymbol{i}^t\boldsymbol{\gamma}_{X+vi^t}.$$

Taking the center of mass of the data, $\boldsymbol{t} = -\frac{1}{n}\mathbf{X}\boldsymbol{i}$, we obtain

$$\boldsymbol{w}_X = \mathbf{X}\boldsymbol{\gamma}_X = \mathbf{X}\left(\boldsymbol{\gamma}_{X+vi^t} - \frac{1}{n}\boldsymbol{i}\boldsymbol{i}^t\boldsymbol{\gamma}_{X+vi^t}\right) = \mathbf{X}\boldsymbol{u},$$

where, denoting by $\boldsymbol{u}$ the parenthetical factor of $\mathbf{X}$ on the right-hand side, we can then compute that $\boldsymbol{u}^t\boldsymbol{i} = 0$, which concludes the proof.

For clarity of notation, from now on we omit the explicit dependency of the separating hyperplane on the data set and write $(\boldsymbol{w}, b)$ instead of $(\boldsymbol{w}_X, b_X)$. As a consequence from the above lemmas, a linear classifier that is invariant with respect to unitary transformations and translations produces a decision function g that can be written as

$$g(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{n} \gamma_i \boldsymbol{x}_i^t \boldsymbol{x} + b\right),$$

with

$$\sum_{i=1}^{n} \gamma_i = 0, \quad \sum_{i=1}^{n} |\ \gamma_i\ | = 2.$$

Since the decision function is not modified by scaling, one can normalize the γ_i to ensure that the sum of their absolute values is equal to 2.

Invariance with respect to label inversion means the γ_i are proportional to y_i, but then the α_i are not affected by an inversion of labels, which means that they depend on only the products $y_i y_j$ (which indicate the differences in label).

Invariance with respect to input permutation means that in the case where $x_i^t x_j = \delta_{ij}$, since the patterns are indistinguishable, so are the α_i. Hence, the α_i corresponding to duplicate training examples that have the same label should be the same value, and from the other constraints, we immediately deduce that $\alpha_i = 1/n_{\pm}$. This finally proves proposition 1.

Appendix B: Proof of Proposition 2

Notice that adding δ_{ij}/C to the inner products means replacing $\mathbf{K}$ by $\mathbf{K} + \mathbf{I}/C$. The result follows from the continuity and from the invariance by scaling, which means that we can as well use $\mathbf{I} + C\mathbf{K}$, which converges to $\mathbf{I}$, when $C \to 0$, and for $\mathbf{I}$, the obtained α_i were computed in proposition 1.

Acknowledgments

We thank E. Simoncelli, G. Cottrell, M. Jazayeri, and C. Rudin for helpful comments on the manuscript. A.B.A.G was supported by a grant from the European Union (IST 2000-29375 COGVIS) and by an NIH training grant in Computational Visual Neuroscience (EYO7158).

References

Bishop, C. (2006). *Pattern recognition and machine learning*. New York: Springer.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation*, *11*(7), 1493–1518.

Duda, R., Hart, P., & Stork, D. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Edelman, S. (1999). *Representation and recognition in vision*. Cambridge, MA: MIT Press.

Green, D., & Swets, J. (1966). *Signal detection theory and psychophysics*. New York: Wiley.

Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Jolliffe, I. (2002). *Principal component analysis* (2nd ed.). New York: Springer.

Kivinen, J., Warmuth, M., & Auer, P. (1997). The perceptron algorithm vs. winnow: Linear vs. logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, *97*(1–2), 325–343.

Lee, D., & Seung, H. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401, 788–791.

Mika, S., Rätsch, G., Weston, J., Schölkopf, B., & Müller, K.-R. (2003). Constructing descriptive and discriminative non-linear features: Rayleigh coefficients in kernel feature spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(5), 623–628.

Rätsch, G., & Meir, G. (2003). An introduction to boosting and leveraging. In *Advanced lectures on machine learning* (Vol. LNAI 2600, pp. 119–184). New York: Springer.

Rätsch, G., & Warmuth, M. (2005). Efficient margin maximization with boosting. *Journal of Machine Learning Research*, *6*, 2131–2152.

Reed, S. (1972). Pattern recognition and categorization. *Cognitive Psychology*, *3*, 382–407.

Rosch, E., Mervis, C., Gray, W., Johnson, D., & Boyes-Braem, P. (1976). Basic objects in natural categories. *Cognitive Psychology*, *8*, 382–439.

Roweis, S., & Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*, 2323–2326.

Rudin, C., Daubechies, I., & Schapire, R. (2004). The dynamics of Adaboost: Cyclic behavior and convergence of margins. *Journal of Machine Learning Research*, *5*, 1557–1595.

Schapire, R. (2001). Drifting games. *Machine Learning*, *43*(3), 265–291.

Schapire, R., & Freund, Y. (1997). A decision theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*, 119–139.

Schapire, R., Freund, Y., Bartlett, P., & Lee, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, *26*(5), 1651–1686.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge, MA: MIT Press.

Skurichina, M., & Duin, R. (2002). Bagging, boosting and the random subspace method for linear classifiers. *Pattern Analysis and Applications*, *5*, 121–135.

Tipping, M. (2001). Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, *1*, 211–214.

Vapnik, V. (2000). *The nature of statistical learning theory* (2nd ed.). New York: Springer.

Wickens, T. (2002). *Elementary signal detection theory*. New York: Oxford University Press.

Received January 22, 2007; accepted April 9, 2008.