18 Computing the Bayes Kernel Classifier

Pál Ruján
FB Physik and ICBM
Carl von Ossietzky Universität Oldenburg
Postfach 2503, D-26111 Oldenburg, Germany
rujan@neuro.uni-oldenburg.de
http://www.neuro.uni-oldenburg.de/∼rujan

Mario Marchand
SITE, University of Ottawa
Ottawa, K1N-6N5 Ontario, Canada
marchand@site.uottawa.ca
http://www.site.uottawa.ca/∼marchand

We present below a simple ray-tracing algorithm for estimating the Bayes classifier for a given class of parameterized kernels.

18.1 Introduction

Support Vector Machines try to achieve good generalization by computing the maximum margin separating hyperplane in a high-dimensional feature space. This approach effectively combines two very good ideas.

The first idea is to map the space of input vectors into a very high-dimensional feature space in such a way that nonlinear decisions functions on the input space can be constructed by using only separating hyperplanes on the feature space. By making use of kernels, we can implicitly perform such mappings without explicitly using high-dimensional separating vectors(Boser et al., 1992). Since it is very likely that the training examples will be linearly separable in the high-dimensional feature space, this method offers an elegant alternative to network growth algorithms as in(Ruján and Marchand, 1989; Marchand et al., 1990) which try to construct non-linear decision surfaces by combining perceptrons.

The second idea is to construct the separating hyperplane on the feature space which has the largest possible margin. Indeed, it was shown by Vapnik and others that this may give good generalization even if the dimension of the feature space

is infinite. We must keep in mind however that the optimal classifier is the Bayes decision rule.

Better learning takes more time

Hence, in an attempt to obtain classifiers with better generalization, we present here a method, based on ergodic billiards, for estimating the Bayes classifier on the high-dimensional feature space at the price of longer training times. We present empirical evidence that the resulting classifier can generalize better than the maximum margin solution.

18.2 A simple geometric problem

Here is the main message of this Chapter in a two-dimensional nutshell. Consider a convex polygon $\mathcal{P}$ in 2D. Given a direction $\mathbf{v}$, compute the line partitioning the polygon $\mathcal{P}$ into two parts of equal area $A_1 = A_2$ (see Fig. 18.1). Call this line the Bayes decision line for direction $\mathbf{v}$.

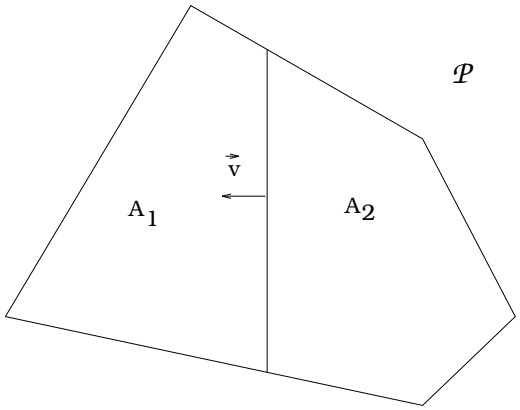

Figure 18.1 Partitioning a convex polyhedron in two equal volumes by a hyperplane with normal $\mathbf{v}$.

Now let us draw such Bayes lines in all possible directions, as shown in Fig. 18.2. Contrary to our triangle preconditioned expectations, these lines do *not* intersect in one point. Hence, no matter how we choose a point inside the polygon $\mathcal{P}$, there will be directions along which the $\mathbf{v}$ oriented line will not partition $\mathcal{P}$ into two equal areas. The *Bayes point* is defined here as the point for which the direction-averaged squared area-difference is minimal.

The Bayes point

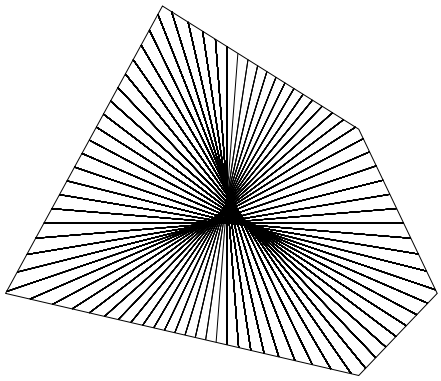

Figure 18.2 The set of Bayes-decision lines.

In general, it is rather difficult to compute the Bayes point. The convex polyhedra we will consider are defined through a set of inequalities, the vertices of the polyhedron are not known. Under such conditions one feasible approximation of the Bayes point is to compute the center of the largest inscribed circle or, better, the symmetry center of the largest area inscribed ellipse. As shown below, the center of the largest inscribed circle corresponds to the maximal margin perceptron. For strongly elongated polygons, a definitely better approach is to consider the center of mass of the polygon, as illustrated in Fig. 18.3 Jumping suddenly into N-dimensions, the

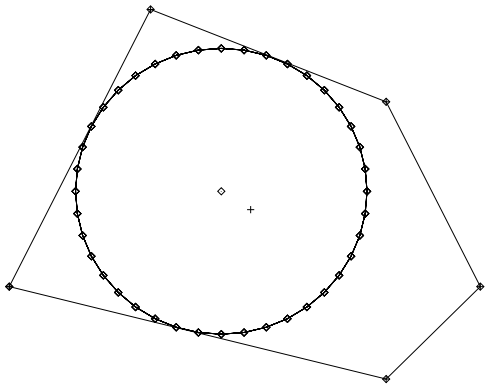

Figure 18.3 The largest inscribed circle and the center of mass (cross).

question is now how to sample effectively a high dimensional polyhedron in order to compute its center of mass. One possibility is to use Monte Carlo sampling (Neal, 1996) or sampling with pseudo-random sequences (Press et al., 1992).

As suggested by one of us, (Ruján, 1997), another viable alternative is to use a

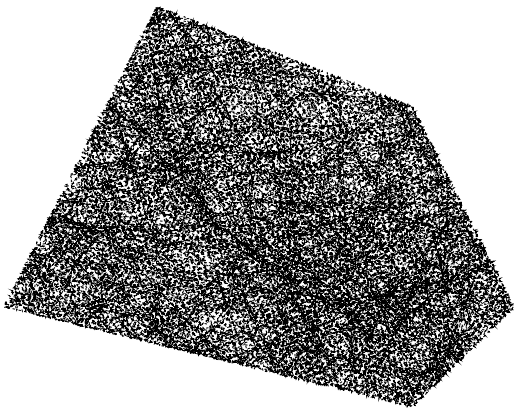

Figure 18.4 Trajectory (dashed line) after 1000 bounces.

Billiards

ray-tracing method. The bounding polygon will be considered as a billiard table inside which a ball bounces elastically on the walls. With few exceptions corresponding to fully integrable or rational angle billiards, the so defined Hamiltonian dynamics will be ergodic, implying that a typical trajectory will homogeneously cover the polygon, as illustrated in Fig. 18.4. It is known that the entropy of polygonal (and polyhedral) billiards vanish (Zemlyakov and Katok, 1975). However, by excluding from the billiard a spherical region inside the polyhedron, one can make the dynamics hyperbolic (Bunimovich, 1979). Such a billiard is somewhat similar to the Sinai-Lorenz billiard, which has strong mixing properties (Bunimovich and Sinai, 1980; Berry, 1981). The absence of a general theorem, however, does not prevent us from using this algorithm. In the following, we will simply *assume* that a typical classification problem leads to ergodic dynamics and will sample accordingly the phase space. The presence of limit cycles, typical for fully integrable systems or KAM-tori in soft chaos can be - in principle - detected numerically.

18.3 The maximal margin perceptron

Consider a set of N-dimensional m data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ belonging to two classes labeled by $\{y_1, \ldots, y_m\}$, $y_i = \pm 1$. We are seeking the plane normal $\mathbf{w}$ and two thresholds b_{+1}, b_{-1} such that

$$(\mathbf{w} \cdot \mathbf{x}_i) - b_{+1} > 0, \quad y_i = +1 \tag{18.1}$$

$$(\mathbf{w} \cdot \mathbf{x}_i) - b_{-1} < 0, \quad y_i = -1 \tag{18.2}$$

$$b_{+1} > b_{-1}$$

Maximal margin

This corresponds to finding two parallel hyperplanes passing between the two convex hulls of the positive and negative examples, respectively, such that their distance (the gap or margin) is maximal

$$G = \max_{\mathbf{w}} \frac{b_{+1} - b_{-1}}{(\mathbf{w} \cdot \mathbf{w})} \tag{18.3}$$

A set of linear inequalities...

This *primal* problem is the maximal margin or maximal 'dead zone' perceptron (Vapnik, 1979; Lampert, 1969). Eq. (18.1-18.2) and can be rewritten compactly as a set of linear inequalities

$$y_i(\mathbf{x}_i, 1)^T(\mathbf{w}, -b) \geq \Delta \geq 0, \quad i = 1, 2, \cdots, m \tag{18.4}$$

...and its stability

where $b = \frac{b_{+1}+b_{-1}}{2}$ is the threshold of the maximal margin perceptron and $\Delta = \frac{b_{+1}-b_{-1}}{2}$ the *stability* of the set of linear inequalities (18.4). Note that in this notation the normal vector $(\mathbf{w}, -b)$ is also $N+1$-dimensional.

A two dimensional illustration of these concepts is shown in Fig. 18.5.

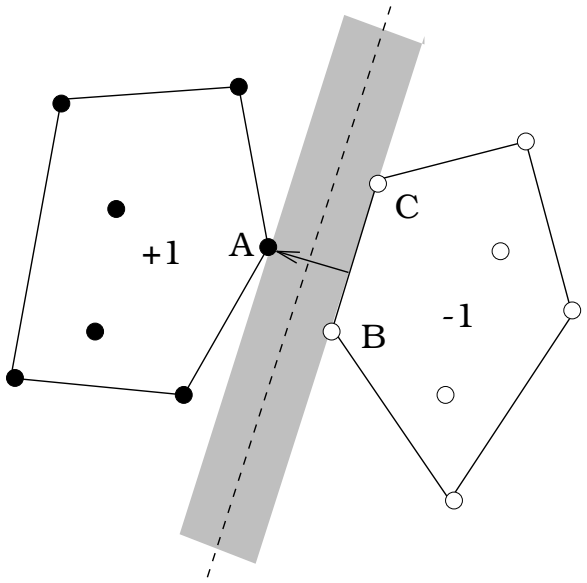

Figure 18.5 The maximal margin perceptron (once again!).

As shown by (Lampert, 1969) and geometrically evident from Fig. 18.5, the *minimal connector* problem is *dual* to the maximal margin problem: find two convex combinations

$$\mathbf{X}_+ = \sum_{\{i:y_i=+1\}} \alpha_i^+ \mathbf{x}_i; \quad \sum_{\{i:y_i=+1\}} \alpha_i^+ = 1; \quad \alpha_i^+ \geq 0 \tag{18.5}$$

$$\mathbf{X}_- = \sum_{\{i:y_i=-1\}} \alpha_i^- \mathbf{x}_i; \quad \sum_{\{i:y_i=-1\}} \alpha_i^- = 1; \quad \alpha_i^- \geq 0 \tag{18.6}$$

such that

$$L^2 = \min_{\{\alpha_i^+,\alpha_i^-\}} \|\mathbf{X}_+ - \mathbf{X}_-\|^2 \tag{18.7}$$

Minimal connector

The corresponding weight vector is given by $\mathbf{w}_{mm} = \mathbf{X}_+ - \mathbf{X}_-$. (18.7) together with the convexity constraints (18.5-18.6) defines a quadratic programming problem. In the mathematical programming literature the vertices A, B, and C in Fig. 18.5 are called *active* constraints. Only the Lagrange multipliers α_i^+ and α_i^- corresponding to active vertices i are strictly positive, all others vanish. The active constraints satisfy the inequalities Eq. (18.1-18.2) as equalities. Furthermore, the gap G, Eq. (18.3) and the minimal connector's length L, Eq. (18.7) are equal only at optimality. Vapnik calls the active constraints *support vectors*, since the expression for $\mathbf{w}_{mm}$ (Eqs. (18.5-18.6)) involves two convex combination of the active constraints only.

Active constraints

A better geometric picture can be obtained by considering the linear conjugate vector space, so that Eq. (18.4) describes now hyperplanes whose normals are given by $\mathbf{z}_i = y_i(\mathbf{x}_i, 1)$. In this space all $\mathbf{W} = (\mathbf{w}, -b)$ vectors satisfying the linear inequalities Eq. (18.4) lie within the convex cone shown in Fig. 18.6. The ray corresponds to the direction of the maximal margin perceptron. The point at which it intersects the unit sphere is at distance

The version space

$$d_i^{mm} = \left| \frac{(\mathbf{w}_{mm} \cdot \mathbf{x}_i)}{\sqrt{(\mathbf{x}_i \cdot \mathbf{x}_i)}} - b \right| = \Delta \tag{18.8}$$

The max margin perceptron as largest inscribed sphere

from the active example $\mathbf{x}_i$. Hence, *if all active examples have the same length*, the maximal margin perceptron corresponds to the center of the largest sphere inscribed into the spherical polyhedron defined by the intersection of the unit sphere with the version space polyhedral cone.

18.4 The Bayes perceptron

Given a set of data $\mathbf{z} = \{\mathbf{z}_i\}_{i=1}^m$ and an unknow example $\mathbf{x}$, the (optimal) Bayes decision rule for $\mathbf{x}$ is given by (see for example (Neal, 1996)):

$$f(\mathbf{x}) = \operatorname{sgn}\left(\int g(\mathbf{x}, \mathbf{W}) P(\mathbf{W}|\mathbf{z}) d\mathbf{W} \right) \tag{18.9}$$

where g is the perceptron output-function, $g(\mathbf{x}, \mathbf{W}) = \operatorname{sgn}\{(\mathbf{x} \cdot \mathbf{w}) - b\}$ and $P(\mathbf{W}|\mathbf{z})$ the posterior network distribution. As usual, the Bayes identity

$$P(\mathbf{W}|\mathbf{z}) = \frac{P(\mathbf{z}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{z})} \tag{18.10}$$

relates the posterior distribution to the prior $P(\mathbf{W})$. It was shown by (Watkin, 1993) that for the uniform posterior, the Bayes classifier Eq. (18.9) can be represented by a single perceptron corresponding to the center of mass of the version space V:

Bayes point = Bayes perceptron

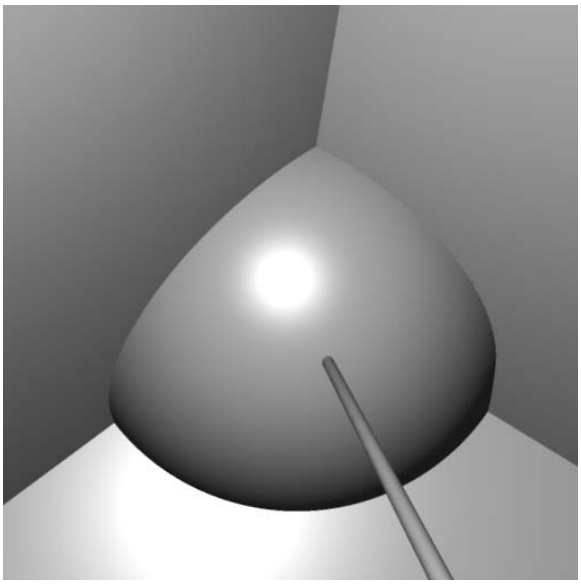

Figure 18.6 The version space is a convex polyhedral cone. The upper left plane corresponds to the active vertex A, the one upper right to the vertex B, and the lower one to the vertex C in Fig. (18.5). The version space is restricted to the intersection between the unit sphere and the convex cone by the normalization constant $(\mathbf{W}_i \cdot \mathbf{W}_i) \equiv 1.\ i = 1, \ldots, m$. The ray corresponds to the direction of the maximal margin perceptron.

$$\lim_{N\to\infty, M\to\infty, \frac{m}{N}=const} f(\mathbf{x}) = g(\mathbf{x}, \mathbf{W}^*) \tag{18.11}$$

where

$$\mathbf{W}^* = \frac{1}{\mathrm{vol}\,(\mathbf{W})} \int_{\mathbf{W}\in V} \mathbf{W} dV \tag{18.12}$$

Hence, the center of mass of the intersection of the version space polyhedral cone with the unit sphere defines the Bayes perceptron parameters as long as the posterior is uniform. It is, however, not difficult to generalize the sampling algorithm presented below to nonuniform posteriors. As described in Chapter 1, the Bayes point is the center of mass of a polyhedral hypersurface whose local mass density is proportional to the posterior distribution (18.10). Under the ergodicity assumption the ray-tracing algorithm provides a collection of homogeneously distributed sampling points. The center of mass can be then estimated by a weighed average of these vectors, where the weights are proportional to the posterior probability density. Before describing the implementation of the billiard algorithm in more detail, we consider the generalization of these ideas to SVM.

18.5 The Kernel-Billiard

As explained in the introduction to this book (Chapter 1), the support vector machines are somewhat similar to the method of potential functions and "boosting" architectures. ¿From a physicist's point of view, however, the support vector machines (SVM) operate on the quantum probability amplitude (wave-packets) $\mathbf{\Phi}$ with $\mathcal{L}^2$ metric, while the method of potential functions and the boosting on a classical probability density with $\mathcal{L}^1$ metric. To see this consider Fig. **??** in Chapter 1: the SVM architecture acts as an *operator* on the input $\mathbf{\Phi}(\mathbf{x})$ feature vector, while the method of potential functions propagates directly the input vector through the set of real functions $\mathbf{\Phi}(\mathbf{x}_i)$, the dot-product layer is missing. The boosting architecture requires, in addition, the weights connected to the output unit to be convex coefficients. Hence, they can be interpreted as the probability that the "hypothesis" function $\mathbf{\Phi}_i$ is correct.

In order to generalize the perceptron learning method described above to kernels, one must rewrite the algorithm in terms of dot-products of the form $q_{ij} = (\mathbf{x}_i \cdot \mathbf{x}_j)$. If this is possible, one makes the substitution $q_{ij} \leftarrow (\mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{\Phi}(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$, where $k(\mathbf{x}, \mathbf{y})$ is a Mercer-kernel. The main trick is thus to substitute this positive definite kernel for all scalar products. The explicit form of $\mathbf{\Phi}_i \equiv \mathbf{\Phi}(\mathbf{x}_i)$ is not needed.

Change in the dot-product

18.5.1 The Flipper Algorithm

Algorithm 18.1 : Flipper (Ruján, 1997)

1. Initialize center of mass vector, counters, etc., normalize example vectors according to Eq. (18.19).
2. Find a feasible solution $\mathbf{W}$ inside the version space,
3. Generate a random unit direction vector $\mathbf{V}$ in version space,
4. For iteration $n_B < N_{max}$ max-iterations, (flight time $\tau < \tau_{max}$ max-time)
   - compute flight-times to all bounding planes (m^2 dot-products $\sim$ kernel evaluations),
   - compute plane-index of the next bounce (corresponds to shortest positive flight time),
   - compute new position in version space $\mathbf{W}'$ and store it in center of mass vector,
   - compute the reflected direction vector $\mathbf{V}'$,
5. Test conditions : go back to 3) if escaped to ∞ or exit at N_{max}

Therefore, the typical number of operations is $O(M^2 N_{max})$ kernel evaluations. To find a feasible solution use any perceptron learning algorithm or the *light-trap method* described below. Fig. 18.7 illustrates the main step of the algorithm. We compute the flight times for all planes and choose the smallest positive one. This will be the plane first hit by the billiard trajectory. We move the ball to the new

Elastic scattering

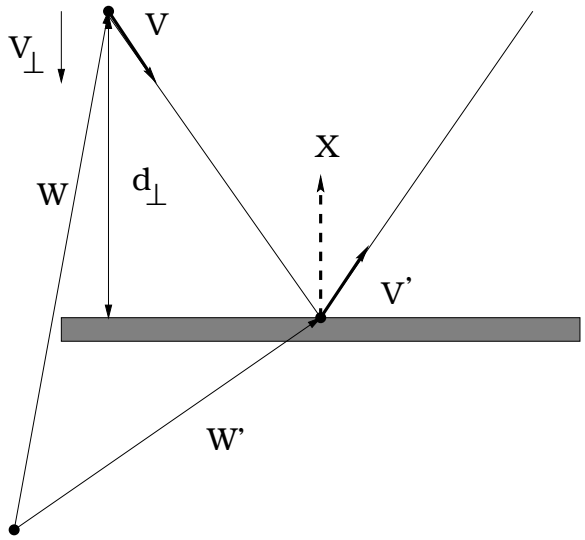

Figure 18.7 Bouncing on a plane. The flight time from point $\mathbf{W}$ to the plane can be computed from the scalar products $d_\perp = (\mathbf{W} \cdot \mathbf{X})$ and $V_\perp = (\mathbf{V} \cdot \mathbf{X})$ as $\tau = -d_\perp / V_\perp$. The collision takes place at $\mathbf{W}' = \mathbf{W} + \tau \mathbf{V}$ and the new direction is $\mathbf{V}' = \mathbf{V} + 2V_\perp \mathbf{X}$.

position and reflect the direction of flight. The arc between the old and the new position is added according to the rules derived below to the center of mass estimate. Since the version space is a polyhedral cone, the trajectory will escape sometimes to infinity. In such cases we restart it from the actual estimated center of mass. Note the analogy to the two-dimensional geometrical problem mentioned above.

How many bounces N_B do we need before the estimated center of mass vector will be within an ε distance from the true center of mass with probability $1 - \eta$? *If* the generated series of vectors $\mathbf{W}$ are identically and independently generated from the posterior distribution, then, as shown in Appendix A, it is sufficient to make

$$N_B = \frac{m(b-a)^2}{2\epsilon^2} \ln\left(\frac{2m}{\eta}\right) \tag{18.13}$$

Convergence estimate

bounces. Where m is the number of training vectors and for simplicity we have assumed that each component of $\mathbf{W}$ lies in the interval $[a, b]$. However, to this number, we must add the number of bounces needed for the generated series of vectors $\mathbf{W}$ to become identically and independently distributed.

Let us describe a given trajectory by the sequence of hyperplane indices hit by the billiard ball. Consider *two* trajectories started from neighboring points in slightly different directions. It seems reasonable to assume that the two trajectories will become fully uncorrelated once the two balls start to bounce on different planes. Hence, let us define the correlation length Λ as the average number of bounces after which two trajectories originally close in phase space bounce for the first time on

different planes. The efficiency of the flipper algorithm depends on how Λ scales with m. In comparison, note that the best known algorithm for estimating the center of mass of a convex polyhedron uses the largest volume inscribed ellipsoid and apart logarithmic terms scales with $O(m^{3.5})$ (Khachiyan and Todd, 1993). By introducing slight changes in the flipper algorithm we can enhance the mixing properties of the billiard and thus optimize Λ. A simple solution is, for instance, to introduce a reflecting sphere around the (estimated) center of mass, lying completely inside the polyhedron. Another possibility is to use a random, nonlinear scattering angle function simulating a dispersing boundary. Such a dynamics would dissipate energy but leads to better mixing properties. This interesting topic will be addressed elsewhere.

When generalizing the billiard algorithm to kernel methods, we have first to show that the center of mass of the now very high dimensional space lies in the span defined by the example set. Let $\mathbf{\Phi}_i \equiv \mathbf{\Phi}(\mathbf{x}_i)$ denote in feature space the image of the training example $\mathbf{x}_i$. The version space $\mathcal{V}$ is defined to be the following set of weight vectors:

$$\mathcal{V} = \{\mathbf{w} : (\mathbf{w} \cdot \mathbf{\Phi}_i) > 0 \text{ for } i = 1, \ldots, m\} \tag{18.14}$$

where m denotes the number of training examples.

Any vector $\mathbf{w}$ lying in the version space can be written as $\mathbf{w} = \mathbf{w}_{\|} + \mathbf{w}_{\perp}$ where $\mathbf{w}_{\|}$ lies in the space spanned by $\{\mathbf{\Phi}_i\}_{i=1}^m$:

$$\exists\ \alpha_1, \ldots, \alpha_m : \mathbf{w}_{\|} = \sum_{i=1}^{m} \alpha_i \mathbf{\Phi}_i \tag{18.15}$$

and $\mathbf{w}_{\perp}$ lies in the orthogonal complement of that space (with respect to $\mathcal{V}$):

$$(\mathbf{w}_{\perp} \cdot \mathbf{\Phi}_i) = 0\ \forall i = 1, \ldots, m \tag{18.16}$$

Hence, $(\mathbf{w}_{\|} + \mathbf{w}_{\perp}) \in \mathcal{V}$ if and only if $(\mathbf{w}_{\|} - \mathbf{w}_{\perp}) \in \mathcal{V}$. For each $\mathbf{w}$ the $\mathbf{w}_{\perp}$ components cancel and the center of mass $\mathbf{w}^*$:

$$\mathbf{w}^* = \frac{1}{\mathrm{vol}\,(\mathcal{V})} \int_{\mathcal{V}} \mathbf{w} dw \tag{18.17}$$

Center of mass is in examples's span

lies in the space spanned by $\{\mathbf{\Phi}_i\}_{i=1}^m$.

This simple result implies that instead of choosing a general direction in feature space we can restrict ourselves to a vector lying in the subspace spanned by the example vectors:

$$\mathbf{V} = \left(\sum_{i=1}^{m} \beta_i \mathbf{\Phi}_i, v_0 \right) \tag{18.18}$$

subject to the normalization condition

$$(\mathbf{V} \cdot \mathbf{V}) = \sum_{i=1}^{m} \sum_{j=1}^{m} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) + v_0^2 = 1 \tag{18.19}$$

The formulation of the flipper algorithm ensures that all subsequent direction

vectors will also lie in the linear span. As usual, we expand also the weight vector as

$$\mathbf{W} = \left(\sum_{i=1}^{m} \alpha_i \mathbf{\Phi}_i, -b \right) \tag{18.20}$$

The normal vectors of the version space polyhedron boundaries are given in this description by

$$\mathbf{Z}_p = \frac{1}{\sqrt{k(\mathbf{x}_p, \mathbf{x}_p) + 1}} \left(y_p \alpha_p \mathbf{\Phi}_p, y_p \right) \tag{18.21}$$

where $p \in P$ denotes the subset of the extended example vectors (hyperplanes), which bounds from inside the zero-error feature space. Support vectors as defined by the maximal margin algorithm (MMSV) are related to those training examples which touch the largest inscribed hypersphere. Support vectors as defined in a Bayesian context correspond to hyperplanes bounding the zero-error feature space (BSV). Obviously, $MMSV \subseteq BSV \subseteq \mathbf{X}$.

Feature space support vectors

In general, the vectors $\mathbf{\Phi}_i$, $i = 1, \ldots, m$ are neither orthogonal, nor linearly independent. Therefore, the expansions (18.18)-(18.20) are not unique. In addition, some of the example hyperplanes might (will !) lie outside the version space polyhedral cone and therefore cannot contribute to the center of mass we are seeking. *Strictly speaking, the expansions in Eqs. (18.18) - (18.21) should be only in terms of the $p \in P$ support vectors.* This set is unknown, however, at the beginning of the algorithm. This *ambiguity*, together with the problem of a drifting trajectory (discussed below in more detail), are specific to the kernel method.

As explained in Fig. 18.7, if we bounce on the j-th example hyperplane

$$V_j^{\perp} = \sum_{i=1}^{m} y_j \beta_i k(\mathbf{x}_i, \mathbf{x}_j) + v_0 y_j \tag{18.22}$$

and, similarly

$$W_j^{\perp} = \sum_{i=1}^{m} y_j \alpha_i k(\mathbf{x}_i, \mathbf{x}_j) - b y_j \tag{18.23}$$

Flight times

Once the example index j^* corresponding to the smallest positive flight time $\tau_{\min}$,

$$j^* = \operatorname*{argmin}_{j, \tau_{\min} > 0} \tau_{\min}, \quad \tau_{\min} = -\frac{W_j^{\perp}}{V_j^{\perp}} \tag{18.24}$$

has been calculated, the update rules for $\mathbf{V}$ and $\mathbf{W}$ are

$$\begin{aligned} \beta_i &\leftarrow \beta_i - 2 y_{j^*} \mathbf{V}^{\perp} \delta_{i,j^*} \\ v_0 &\leftarrow v_0 - 2 y_{j^*} \mathbf{V}^{\perp} \end{aligned} \tag{18.25}$$

and

$$\begin{aligned} \alpha_i &\leftarrow \alpha_i + \tau_{\min} \beta_i \\ b &\leftarrow b + \tau_{\min} v_0 \end{aligned} \tag{18.26}$$

In order to update the center of mass vector we define the sum of two unit vectors $\mathbf{a}$ and $\mathbf{b}$ as follows (see Fig. 18.8). Each vector has an associated weight, denoted by ρ_a and ρ_b, respectively. The sum of the two vectors is $\mathbf{c}$, with weight $\rho_c = \rho_a + \rho_b$. If we choose the vector $\mathbf{a}$ as the unit vector in direction x, then, according to Fig. 18.8, the coordinates of the three vectors are $\mathbf{a} = (1, 0)$, $\mathbf{b} = (\cos\phi, \sin\phi)$, and $\mathbf{c} = (\cos\phi_1, \sin\phi_1)$, respectively. Note that $\cos\phi = (\mathbf{a} \cdot \mathbf{b})$. We now require that the angle of the resultant vector $\mathbf{c}$ equals $\phi_1 = \frac{\rho_b}{\rho_c}\phi$, as when adding parallel forces.

Center of mass update

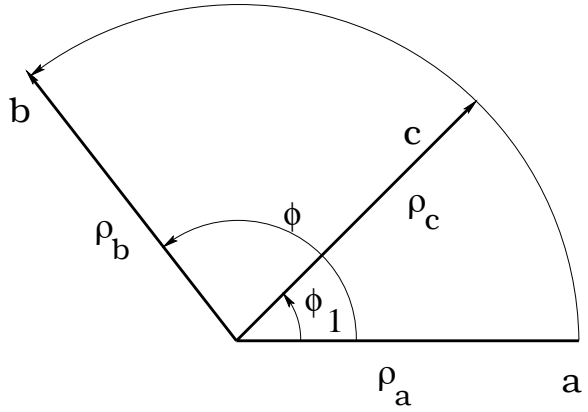

Figure 18.8 Adding two vectors lying on the unit hypersphere. The weights are denoted by ρ and $\mathbf{c} = \mathbf{a} + \mathbf{b}$ implies $\rho_c = \rho_a + \rho_b$.

This leads to the following addition rule

$$\mathbf{c} = \cos\left(\frac{\phi\rho_b}{\rho_a + \rho_b}\right)\mathbf{a} + \frac{\sin\left(\frac{\phi\rho_b}{\rho_a+\rho_b}\right)}{\sin(\phi)}[\mathbf{b} - \cos(\phi)\mathbf{a}] \tag{18.27}$$

where $\cos\phi = \mathbf{ab} \leftarrow \mathbf{k}(\mathbf{a}, \mathbf{b})$. As explained in (Ruján, 1997), we can add recursively using Eq. (18.27) the middle points of the arcs between the two bouncing points $\mathbf{W}$ and $\mathbf{W}'$, with a weight given by the length of the arc, which is proportional to the angle ϕ. If the new direction points towards the free space, $\tau_{\min} = \infty$, we restart the billiard at the actual center of mass in a randomly chosen direction. If the number of bounces and the dimension are large enough, which is almost always the case, then it is enough to add the bouncing points with the weight determined by the posterior probability density.

Now consider the case when the number of BSV's is less than m. This means that in the example set there are some hyperplanes which can not contribute to the center of mass. The simplest such example occurs when the ball bounces for a long time between two given planes while moving (drifting) along with a constant speed, while the other examples are inactive. According to the update rules Eqs. (18.25-18.26), *all* examples contribute to both $\mathbf{w}$ and the threshold b! However, when using the center of mass for classification the contributions from inactive examples cancel out.

18.5.2 The light-trap algorithm

We can extend the main idea behind the billiard dynamics in several ways. One variant is the *light-trap* algorithm, which can be used to generate a feasible solution. Instead of the full reflecting mirror in Fig. 18.7, consider a half-reflecting mirror, schematically shown in Fig. 18.9

Trapping billiards

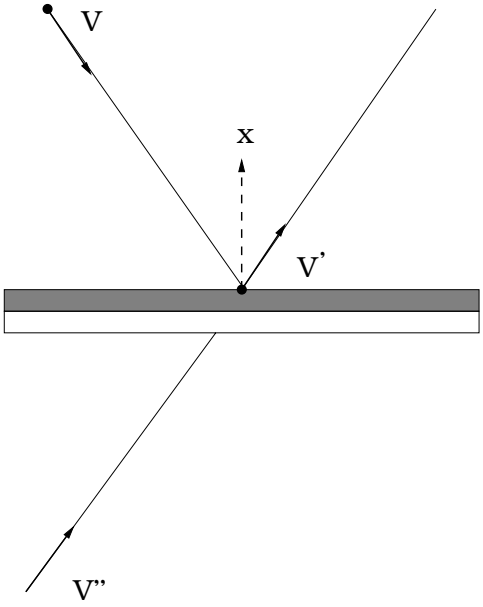

Figure 18.9 Bouncing on a semi-transparent plane. If the "light" comes from above ($\mathbf{V}_\perp < 0$) the trajectory is elastically reflected. If, however, the light shines from below, ($\mathbf{V}_\perp > 0$), the trajectory is allowed to pass through the mirror.

Therefore, if we start the trajectory at a point making few errors in version space, any time we encounter a non satisfied linear inequality we will "pass" through it, reducing by one the number of errors. Eventually, the trajectory will be trapped in the version space with zero errors. This algorithm is particularly simple to

implement for the Gaussian kernel

$$k(\mathbf{x}, \mathbf{y}) = \mathrm{e}^{\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} \tag{18.28}$$

Gauss kernels

where the starting point corresponds to the limit $\sigma \to 0$. In this limit all $\mathbf{\Phi}_i$ vectors are pairwise orthogonal and $\mathbf{X}_-$, $\mathbf{X}_+$ are given by the center of mass of negative and positive example vectors in feature space: $\alpha_i = 1/m_-$ for negative and $\alpha_i = 1/m_+$ for positive examples, respectively. $m_{\pm 1}$ is the number of positive (negative) training examples.

This is nothing else than a Bayesian decision for fully uncorrelated patterns, the only useful information being the occurrence frequency of the two classes. This solution is a good starting point, making relatively few errors also for finite σ values. This idea can be easily implemented and provides in this context a new perceptron-learning method, thus making the billiard method self contained.

18.5.3 The soft billiard

Another natural extension is to "soften" the billiard, by allowing for one or more errors, as illustrated in the Fig. (18.10)

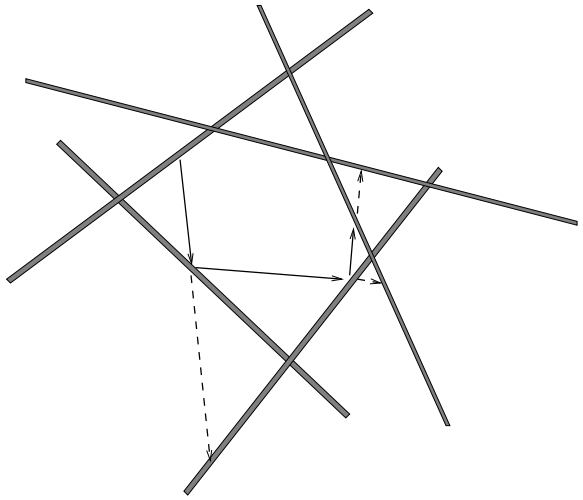

Figure 18.10 Projecting a trajectory beyond the zero-error version space.

Soft billiards

The "onion-algorithm" *sorts* the positive flight times in increasing order. While the shortest flight time corresponds to the boundary of the zero-error feature space polyhedron, we can sample now the position this ray would have on the second (one-error version space), third (two-error version space), etc., bounce. Therefore, while keeping the trajectory inside the zero-error cone, we generate at a minimal additional cost simultaneously estimates of the zero, one-error, two-errors, etc.

version spaces. By keeping track of the original labels of the planes the trajectory is bouncing upon, we produce a set of 'center-of-mass' vectors corresponding to a given (n_-, n_+) number of negative and positive errors, respectively. After convergence, we can try different ways of weighting these results without having to rerun the sampling procedure again.

This method provides a powerful tool of searching for good 'mass-centers' in addition to changing the kernel parameter σ. It has also the same drawback, namely it requires a careful testing according to the 'k out on m'-estimate protocol.

18.6 Numerical Tests

We did not yet perform exhaustive tests on the SVM variant of these algorithms. While in the process of editing this manuscript we received an article (Herbrich et al., 1999) where numerical results obtained on a large set of data strongly support our conclusions. For illustration purposes we present below results obtained for normal perceptrons, were also analytic results are available (Opper and Haussler, 1991). The training examples have been generated at random and classified according to a randomly chosen but fixed "teacher" perceptron. Fig. (18.11) shows the theoretical Bayes learning curve. The experimental results were obtained using the billiard algorithm and represents an average over 10 different runs.

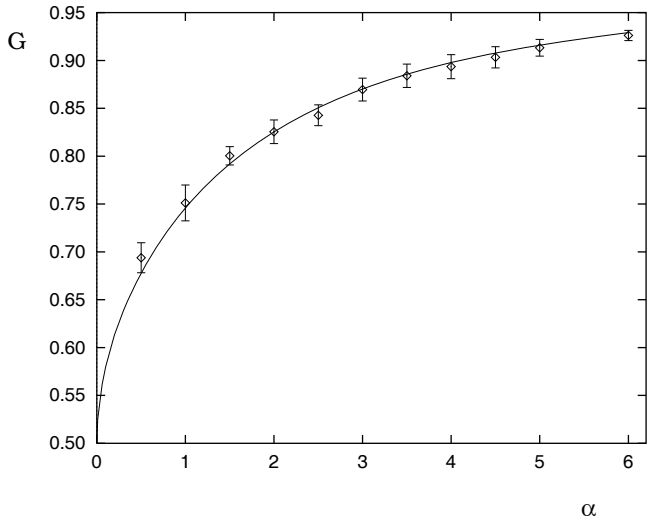

Figure 18.11 The learning curve (generalization probability) as a function of $\alpha = \frac{M}{D}$, where M is the number of randomly generated examples and $D = 100$ the input dimension. The continuous curve is the Bayes result of Opper and Haussler, the points with error bars represent billiard results.

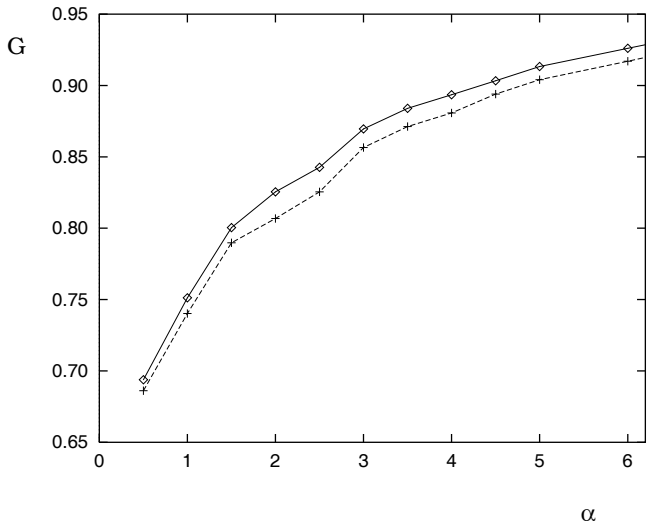

Figure 18.12 Comparison between average generalization values obtained with the maximal margin perceptron and the flipper algorithm, respectively. Same parameter values as in Fig. 18.11.

Fig. (18.11) shows a comparison between the average generalization error of the maximal margin perceptron (lower curve) vs. the billiard results. Not shown is the fact that in each single run the maximal margin perceptron was worse than the billiard result. We present below the results obtained with the Gaussian kernel Eq. (18.28) for two real-life data sets. In Fig. 18.13 are the sonar data (Gorman and Sejnowsky, 1988), split into two sets according to the aperture angle. Note that different versions of this split are in circulation. Our split can be found in P.R.'s www-home page (http://www.neuro.uni-oldenburg/~rujan)

These results show that using the onion-algorithm one can obtain very good, stable results for a rather wide range of σ values.

We also tested our algorithms on the Wisconsin breast cancer data collected by W. H. Wolberg (Breast Cancer Database, University of Wisconsin Hospitals, Madison). After removing all incomplete examples we are left with 673 cases. Each has 9 attributes and 2 classes. Using the leave-10-out cross-validation method we obtain for $\sigma = 1.75$ 25 errors for the maximal margin SVM and 23 errors by the billiard algorithm (out of 670 tests).

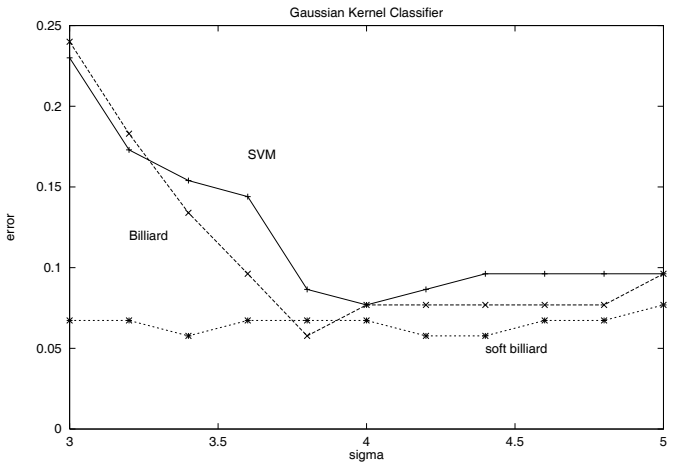

Figure 18.13 The aspect-angle dependent sonar data split: comparison between the generalization error obtained with the maximal margin SVM, the flipper, and the onion algorithms, respectively. Note that different data splits are available in the literature.

18.7 Conclusions

Although we have not yet tested in great detail the billiard algorithm for SVM's, we believe that this algorithm and its variants provide good estimates for the Bayesian kernel classifiers. Another interesting observation is that the best results were obtained just before the billiard dynamics "closed". In general, for small values of the kernel parameter σ, the trajectories are rather short, the ball escapes often to ∞. As σ increases the trajectory length increases as well. A phase-transition like change happens for even larger values of σ: the length of the trajectory seems to diverge, the billiard is closed.

Further theoretical work is necessary for determining correctly the typical correlation length Λ in Eq. (18.13). If it turns out that Λ does not depend on the dimension (number of training examples), then the estimate (18.13) would suggest that the billiard method is superior to any known algorithm for solving convex programming problems.

Ideal gas of balls

In the present implementation the billiard based algorithms are slow in comparison to the QP algorithms. However, since they can be run in parallel on different processors without having to exchange a lot of data, they are well suited for massively parallel processing.

err	+0	+1	+2	+3	+4	+5
-0	8 (3,5)	9 (1,8)	13 (1,12)	18 (1,17)	20 (0,20)	21 (0,21)
-1	10 (5,5)	7 (2,5)	8 (1, 7)	8 (1, 7)	9 (1, 8)	11 (1,10)
-2	10 (5,5)	8 (3,5)	8 (2, 6)	6 (1, 5)	10 (1, 9)	16 (1,15)
-3	10 (5,5)	9 (4,5)	10 (3, 7)	6 (1, 5)	7 (1, 6)	9 (1, 8)
-4	12 (7,5)	10 (5,5)	10 (4, 6)	9 (3, 6)	8 (1, 7)	8 (1, 7)
-5	10 (8,2)	10 (5,5)	9 (4, 5)	9 (4, 5)	8 (3, 5)	7 (1, 6)

Table 18.1 Typical error table delivered by the soft billiard (onion) algorithm. The matrix indices represent the number of errors made allowed when computing the weight vector for the $-$ and the $+$ class, respectively. The matrix elements contain the total number of errors made on the test set and their split into $-$ and $+$ class errors, respectively. The results shown are for the aspect-dependent angle sonar data. A gaussian kernel with $\sigma = 4.4$ has been used, the total number of test examples was 104.

Acknowledgments

This work has been performed during P.R.'s visit at SITE, University of Ottawa in September 1998. During the editing of this manuscript we were informed about a similar work by R. Herbrich, T. Graepel, and C. Campbell, arriving at similar conclusions (Herbrich et al., 1999). We thank them for sending us a reprint of their work. The kernel implementation of the billiard algorithm as described above can be found at http://www.neuro.uni-oldenburg/~rujan.

18.8 Appendix

In this section, we first present a sampling lemma (that directly follows from Hoeffding's inequality) and then discuss its implications on the number of samples needed to find a good estimate of the center of mass of the version space.

Lemma 18.1
Let $\mathbf{v}$ be a n-dimensional vector $(v_1, v_2, \ldots, v_n)$ where each component v_i is confined to the real interval $[a, b]$. Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be k independent vectors that are identically distributed according to some unknown probability distribution P. Let μ be the true mean of $\mathbf{v}$ (*i.e.* $\mu = \int \mathbf{v} dP(\mathbf{v})$) and let $\hat{\mu}$ be the empirical estimate of μ obtained from k samples (*i.e.* $\hat{\mu} = (1/k) \sum_{j=1}^{k} \mathbf{v}_j$). Let $||\hat{\mu} - \mu||$ denote the Euclidean distance between $\hat{\mu}$ and μ. Then with probability at least $1-\eta$, $||\hat{\mu} - \mu|| < \epsilon$ whenever

$$k > \frac{n(b-a)^2}{2\epsilon^2} \ln\left(\frac{2n}{\eta}\right) \tag{18.29}$$

Proof To have $||\hat{\mu} - \mu|| < \epsilon$, it is sufficient to have $|\hat{\mu}_i - \mu_i| < \epsilon/\sqrt{n}$ simultaneously for each component $i = 1, \ldots, n$. Let A_i be the event for which $|\hat{\mu}_i - \mu_i| < \epsilon/\sqrt{n}$ and let $\overline{A}_i$ be the complement of that event so that $\Pr(A_1 \cap A_2 \cdots \cap A_n) = 1 - \Pr(\overline{A}_1 \cup \overline{A}_2 \cdots \cup \overline{A}_n)$. Hence we have $\Pr(A_1 \cap A_2 \cdots \cap A_n) > 1 - \eta$ if and only if $\Pr(\overline{A}_1 \cup \overline{A}_2 \cdots \cup \overline{A}_n) < \eta$. However, from the well known union bound, we have $\Pr(\overline{A}_1 \cup \overline{A}_2 \cdots \cup \overline{A}_n) \leq \sum_{i=1}^{n} \Pr(\overline{A}_i)$. Hence to have $\Pr(A_1 \cap A_2 \cdots \cap A_n) > 1 - \eta$, it is sufficient to have $\Pr(\overline{A}_i) < \eta/n$ for $i = 1, \ldots, n$. Consequently, in order to have $||\hat{\mu} - \mu|| < \epsilon$ with probability at least $1 - \eta$, it is sufficient to have $|\hat{\mu}_i - \mu_i| > \epsilon/\sqrt{n}$ with probability at most η/n for each component i. Now, for any component i, Hoeffding's inequality (Hoeffding, 1963) states that:

$$\Pr\{|\hat{\mu}_i - \mu_i| \geq \alpha\} \leq 2e^{-2k\alpha^2/(b-a)^2} \tag{18.30}$$

The lemma then follows by choosing $\alpha = \epsilon/\sqrt{n}$ and by imposing that η/n be larger than the right-hand side of Hoeffding's inequality. ■

To apply this lemma to the problem of estimating the center of mass of the version space, we could just substitute for the vector $\mathbf{v}$, the separating weight vector (that would include an extra component for the threshold). However, we immediately run into a difficulty when the separating vector lies in an infinite-dimensional feature space. In that case, we just apply the lemma for the m-dimensional vector $(\alpha_1, \alpha_2, \ldots, \alpha_m)$, dual to the separating weight vector. Hence, because we must now replace n by the number m of training examples, the number k of samples needed becomes:

$$k > \frac{m(b-a)^2}{2\epsilon^2} \ln\left(\frac{2m}{\eta}\right) \tag{18.31}$$

Regardless of whether we are sampling directly the weight vectors or the duals, the bound obtained from this lemma for estimating the center of mass applies only when we are sampling according to the true Bayesian posterior distribution.

References

M. V. Berry. Quantizing a classically ergodic system: Sinai's billiard and the KKR method. *Annals of Physics*, 131:163–216, 1981.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

L. A. Bunimovich. On the ergodic properties of nowhere dispersing billiards. *Commun. Math. Phys.*, 65:295–312, 1979.

L. A. Bunimovich and Ya.G. Sinai. Markov partitions for dispersed billiards. *Commun. Math. Phys.*, 78:247–280, 1980.

R. P. Gorman and T. J. Sejnowsky. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1, 1988.

R. Herbrich, T. Graepel, and C. Cambell. Bayes point machines: Estimating the bayes point in kernel space. *IJCAI 99*, 1999.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

L. G. Khachiyan and M. J. Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. *Mathematical Programming*, 61:137–159, 1993.

P. R. Lampert. Designing pattern categories with extremal paradigm information. In M. S. Watanabe, editor, *Methodologies of Pattern Recognition*. Academic Press, N.Y., 1969.

M. Marchand, M. Golea, and P. Ruján. Convergence theorem for sequential learning in two layer perceptrons. *Europhysics Letters*, 11:487–492, 1990.

R. Neal. *Bayesian Learning in Neural Networks*. Springer Verlag, 1996.

M. Opper and D. Haussler. Generalization performance of bayes optimal classification algorithm for learning a perceptron. *Physical Review Letters*, 66:2677, 1991.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, Cambridge, 1992.

P. Ruján. Playing billiard in version space. *Neural Computation*, 9:99–122, 1997.

P. Ruján and M. Marchand. Learning by minimizing resources in neural networks. *Complex Systems*, 3:229–242, 1989.

V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

T. Watkin. Optimal learning with a neural network. *Europhysics Letters*, 21:871, 1993.

A. N. Zemlyakov and A. .B. Katok. The topological transitivity of billiards in polygons. *mat. zametki*, 18:291–301, 1975.