On the Complexity of PAC Learning in Hilbert Spaces

Sergei Chubanov

Bosch Center for Artificial Intelligence, Germany
sergei.chubanov@de.bosch.com

Abstract

We study the problem of binary classification from the point of view of learning convex polyhedra in Hilbert spaces, to which one can reduce any binary classification problem. The problem of learning convex polyhedra in finite-dimensional spaces is sufficiently well studied in the literature. We generalize this problem to that in a Hilbert space and propose an algorithm for learning a polyhedron which correctly classifies at least $1 - \varepsilon$ of the distribution, with a probability of at least $1 - \delta$, where ε and δ are given parameters. Also, as a corollary, we improve some previous bounds for polyhedral classification in finite-dimensional spaces.

Introduction

In general, the classification problem we are dealing with can be formulated as follows: Find a binary classifier which consistently classifies the training data such that the number of elementary operations involved in the description of the classifier is as small as possible. The intuition behind restricting the complexity of the classifier is based on the well-known Occam's razor principle; it has been proved (see e.g. (Blumer et al. 1987, 1989)) that there is a relationship between the complexity of a classifier and its prediction accuracy, in the sense of the probably approximately correct (PAC) classification.

We study this problem from the point of view of polyhedral classification, where the concept class to be learned consists of polyhedra in a given inner-product space. Polyhedral separability is always realizable by choosing a suitable kernel; i.e., our results are universally applicable to the general case of binary classification.

One should note that, unless $P = NP$, we cannot hope for a polynomial-time method for finding a polyhedral classifier defined by the minimum possible number of halfspaces because even the problem of polyhedral separation in finite-dimensional spaces by means of two halfspaces is NP-hard (Megiddo 1996).

Our results and related work. We propose an algorithm for both *proper* and *improper* learning of polyhedral concepts in inner-product spaces that can be finite-dimensional Euclidean spaces or infinite-dimensional Hilbert spaces. In the context of our paper, proper learning means learning a polyhedron defined by t halfspaces (t-polyhedron), under the assumption that there exists a t-polyhedron consistently classifying the entire instance space. Similarly to the previous publications with which we compare our results, we assume that the halfspaces defining a hypothetical ground-truth polyhedron do not contain any instance of the instance space in their γ-margins, for some $\gamma > 0$.

Consistent polyhedral separation is a well-known problem which has been proved to be intractable even in the case $t = 2$, as we have already mentioned. Despite to the NP-hardness of exact separation with a given number of inequalities in finite-dimensional spaces, there are different non-trivial results concerning exponential bounds on the running time of learning algorithms; see (Gottlieb et al. 2018). Our contribution to this line of research is the following: First, we propose a new algorithm allowing to improve existing bounds on the running time of PAC learning algorithms for the finite-dimensional case. Second, our algorithm is applicable to infinite-dimensional Hilbert spaces, in which case it guarantees similar complexity bounds in terms of a model of computation including inner products as elementary operations.

To the best of our knowledge, the current tightest bounds for both proper and improper learning have been obtained in (Gottlieb et al. 2018). There, the authors proposed algorithms for both proper and improper learning of polyhedral concepts in finite-dimensional spaces. A follow-up work related to (Gottlieb et al. 2018) appeared recently in (Gottlieb et al. 2022) with some corrections of the previous work. For a more exact comparison, we give a brief summary of their performance bounds compared to ours:

- (Gottlieb et al. 2022):
  - Running time for proper classification:
  $$m^{O(t\gamma^{-2}\log(\gamma^{-1}))},$$
  where m is the sample complexity.
  - Running time for improper classification:
  $$m^{O(\gamma^{-2}\log(\gamma^{-1}))},$$
  - Sample complexity:
  $$m = O(D\varepsilon^{-1}\log^2(\varepsilon^{-1}) + \log(\delta^{-1})),$$

where $D = (\gamma^{-2} t \log t)$.

  – Applicability to Hilbert spaces: Although this is not mentioned in (Gottlieb et al. 2022), their approach seems to be applicable to Hilbert spaces. However, it uses the Johnson-Lindenstrauss (JL) transform and, therefore, may need some additional analysis to be formulated in terms of a Hilbert space. At the same time, our method does not use the JL transform.

- Ours (with respect to a finite-dimensional space):
  – Running time for proper classification: $m^{O(t\gamma^{-2})}$.
  – Running time for improper classification:
$$O\left(d \cdot m \cdot (t\gamma^{-2})^{4\gamma^{-2}}\right).$$
  – Sample complexity m:
    * Proper learning: $O(dt \log t)$, where d is the dimension of the space.
    * Improper learning: $O(\varepsilon^{-1} D + \varepsilon^{-1} \log(\delta^{-1}))$ where $D = O((\gamma^{-2} + \log t)(t\gamma^{-2})^{4\gamma^{-2}})$.

For proper classification, we save $\log(\gamma^{-1})$ in the power of the exponent, compared to (Gottlieb et al. 2022). For improper classification, the running time of our algorithm is substantially better, but the sample complexity is exponential in γ^{-2}.

An improper learning algorithm for finite-dimensional spaces with the running time depending exponentially on the dimension of the space and without a margin assumption, but with additional assumptions about the probability distribution, was presented in (Vempala 2010) and (Klivans, O'Donnell, and Servedio 2008).

Support vector machines (Bennett and Bredensteiner 2000; Mavroforakis and Theodoridis 2006) as large margin classifiers are certainly also related to our work. In particular, in (Bennett and Bredensteiner 2000) a relevant geometric interpretation of SVM has been proposed. Beyond the kernel methods like SVM, research on large-margin classification directly in Hilbert (Rossi and Villa 2006) and Banach (Der and Lee 2007) spaces was initiated within the area of functional data analysis.

The problem of polyhedral classification in Euclidean spaces is known under various names, e.g., learning convex polytopes (Gottlieb et al. 2018), linear decision lists (Jeroslow 1975; Anthony 2004; Mangasarian 1968), neural lists (Long and Servedio 2006), or threshold functions (Anthony and Ratsaby 2012). Many of the methods for learning such structures (see (Anthony 2004) for overview) can be viewed as cutting-plane methods where inconsistencies are resolved by sequentially adding and reorganizing new halfspaces or hyperplanes at each iteration. These methods have several variations including a multisurface method proposed in (Mangasarian 1968; Mangasarian, Setiono, and Wolberg 1990), where at each stage two parallel hyperplanes located close to each other are identified, such that the points not enclosed between them have the same classification.

In (Manwani and Sastry 2010), the authors propose an approach for learning polyhedral classifiers based on logistic function. Paper (Kantchelian et al. 2014) proposes an algorithm for learning polyhedral concepts using stochastic gradient descent. The work (Anthony 2004) analyses generalization properties of techniques based on the use of linear decision lists and presents a bound on the generalization error in a standard probabilistic learning framework. The authors of (Chattopadhyay et al. 2019) have recently demonstrated a lower-bound technique for linear decision lists, which has been used to prove an explicit lower bound.

Preliminaries

In a Hilbert space $\mathcal{H}$, a halfspace can be represented as $\{z \in \mathcal{H} : \langle h, z \rangle + d \geq 0\}$ for some $h \in \mathcal{H}$ and $d \in \mathbb{R}$. Here, $\langle \cdot, \cdot \rangle$ is the inner product. Since it will always be known from the context what spaces the respective elements belong to, we will not use an additional notation to specify the space with which the respective inner product is associated. For instance, we will write $\langle (h_1, d_1), (h_2, d_2) \rangle$ for the inner product of elements (h_i, d_i), $i = 1, 2$, of the Cartesian product $\mathcal{H} \times \mathbb{R}$, where $h_i \in \mathcal{H}$ and $d_i \in \mathbb{R}$, meaning that $\langle (h_1, d_1), (h_2, d_2) \rangle = \langle h_1, h_2 \rangle + d_1 d_2$.

Large margins and polyhedra. Regarding large-margin polyhedral classification, we will partially rely on the terminology introduced in some previous works on polyhedral classification in Euclidean spaces. The outer γ-margin of a halfspace H is the set of points not in H whose distance to its boundary ∂H is not greater than γ. The inner γ-margin of H is the set of points in H whose distance to ∂H is not greater than γ. The γ-margin of H is the union of its inner and outer γ-margin. We denote by H^γ the union of H and its γ-margin.

We say that two sets X^- and X^+ are linearly γ-separable if there is a halfspace H such that $H \cap X^- = \emptyset$ and $X^+ \subset H$ and the γ-margin of H does not contain any point of $X^- \cup X^+$. For short, we will equivalently say that X is linearly γ-separable.

A polyhedron F is the intersection $\cap_\alpha H_\alpha$ of a finite collection of some halfspaces H_α. Let F^γ be the polyhedron $\cap_\alpha H_\alpha^\gamma$. The γ-margin of F is $F^\gamma \setminus F$. If for some set X polyhedron F contains X^+ and does not intersect with X^-, and at the same time the γ-margin of F does not intersect with the whole set X, we say that F γ-separates X^- from X^+. A t-polyhedron is a polyhedron being an intersection of t halfspaces.

Instance space. The instance space $\mathcal{X} \subset \mathcal{H}$ is assumed to be contained in a unit ball centered at the origin $\mathbf{0}$ of $\mathcal{H}$. That is, $\|x\| \leq 1$ for all $x \in \mathcal{X}$. We assume that there are two class labels -1 and 1 and that the instance space $\mathcal{X}$ is equipped with a function $y : \mathcal{X} \longrightarrow \{-1, 1\}$ with the property that there exists $\rho > 0$ such that
$$\forall x, x' \in \mathcal{X} : y(x) \neq y(x') \Longrightarrow \|x - x'\| \geq \rho. \qquad (1)$$
This condition tells that two instances of different classes cannot be arbitrarily close to each other. That is, the distance between $\mathcal{X}^-$ and $\mathcal{X}^+$ is nonzero, where $\mathcal{X}^-$ and $\mathcal{X}^+$ are the respective classes.

Further, when considering a subset X of $\mathcal{X}$, we will denote by X^- the set of all negatives, i.e., those $x \in X$ with $y(x) = -1$ and by X^+ the set of all positives in X, i.e., those with $y(x) = 1$.

Machine-learning framework. When talking about probability distributions, we assume that our space is a certain reproducing kernel Hilbert space of a sufficiently well-behaved kernel over an underlying Euclidean space such that our instance space in the Hilbert space is an image of a compact subset Ω of that Euclidean space. Drawing a sample (a finite subset of instances in $\mathcal{X}$) means drawing a sample from a fixed probability distribution defined on Ω and then considering its image in the Hilbert space.

We say that a subset H of $\mathcal{H}$ correctly classifies $x \in \mathcal{X}$ if $y(x) = 1$ implies that $x \in H$ and $y(x) = -1$ implies that $x \notin H$. In this role, we often refer to H as a classifier.

To estimate the prediction accuracy of polyhedral classifiers constructed by our algorithms, we use the classical framework of probably approximately correct (PAC) learning based on VC dimension; $1-\varepsilon$ and $1-\delta$ are the prediction accuracy and the confidence estimate, respectively, where ε and δ are given values in $(0, 1/2)$. For the finite-dimensional case, the VC dimension of linear concepts is bounded by the dimension d of the space. The VC dimension of the family of polyhedra defined by t halfspaces is bounded by $d \cdot t \log t$, which is a well-known fact. The infinite-dimensional case is more difficult, although we can also provide a similar estimate where in place of d we use $O(\gamma^{-2})$ under the assumption that there is a t-polyhedron that γ-separates the instance space. However, our algorithmic approach, though working for the finite-dimensional case, does not guarantee that the consistent polyhedron learned by the algorithm belongs to this concept class. To overcome this difficulty, we use discretization and an additional assumption that the Hilbert space in question is a reproducing kernel Hilbert space (RKHS) with a sufficiently well-behaved kernel. The discretization allows us to learn from a finite concept class.

Realizability. Polyhedral separability of the image of any finite-dimensional data with binary labels is always realizable in a suitable RKHS. E.g., we can choose an RBF kernel K or a suitable function K of a given kernel K' to guarantee that $K(x, x) = 1, K(x, x') < 1$, for all $x, x' \in \mathcal{X}$, $x \neq x'$. Then we can prove that the image of our original $\mathcal{X}$ in the RKHS is γ-separable by a t-polyhedron, for some t and γ, because of (1); see (Chubanov 2023) for details.

Model of computation. The elementary operations are computations of inner products and norms in the vector spaces involved, and arithmetic operations.

Linear Separation in Hilbert Spaces

In this section, we consider the following system $P(X)$ of strict linear inequalities associated with a subset X of $\mathcal{X}$:

$$P(X) : y(x)(\langle x, h\rangle + d) > 0, \forall x \in X.$$

where we are looking for $(h, d) \in \mathcal{H} \times \mathbb{R}$. A pair (h, d) defines a halfspace

$$H = \{z \in \mathcal{H} : \langle z, h\rangle + d \geq 0\},$$

which can serve as a linear classifier; If $x \in H$, it assigns label 1 to x, otherwise, label -1 is assigned. If (h, d) is feasible for $P(X)$, then H assigns correct labels for all $x \in X$.

To solve $P(X)$, we use Algorithm 1, which is a modification of von Neumann's algorithm for linear programming communicated in (Dantzig 1992). As input, it takes X and a *state* S defined as follows:

Algorithm 1: **Linear-programming (LP) algorithm**

```
Input: System $P(X)$, state $S$ compatible with $X$, and $\gamma$.
Output: State $S'$.
$S' := S$.
while $S' = \emptyset$ or $(h_{S'}, d_{S'})$ is not feasible for $P(X)$ and $\|(h_{S'}, d_{S'})\| \geq \gamma/2$ do
    if $S' \neq \emptyset$ then
        (Progress-contributing step)
        $(h, d) := (h_{S'}, d_{S'})$
        Find $x \in X$ with $y(x)(\langle x, h_{S'}\rangle + d_{S'}) \leq 0$.
        Let $(h', d')$ be the orthogonal projection of $(\mathbf{0}, 0)$ onto $[(h, d), y(x)(x, 1)]$ and $S'$ be defined by $(h', d')$.
    else
        Pick an arbitrary $x \in X$.
        $(h', d') := y(x)(x, 1)$
        Let $S'$ be defined by $(h', d')$.
    end if
    if $\|(h_{S'}, d_{S'})\| < \gamma/2$ then
        Report that $X$ is not linearly $\gamma$-separable.
    end if
end while
Return $S'$.
```

Definition 1 *A state S is an empty state (denoted as $\emptyset$) or a pair (h_S, d_S), where $h_S \in \mathcal{H}$ and $d_S \in \mathbb{R}$, defining a halfspace which we denote by H_S. For $S = \emptyset$, we set $H_\emptyset = \mathcal{H}$. That is, $H_\emptyset$ is the entire space.*

A state $S \neq \emptyset$ is compatible with X if (h_S, d_S) is a convex combination of a finite subset of $\{y(x)(x, 1) : x \in X\}$. Given a state S compatible with X, the LP algorithm returns another state S' compatible with X. The final state S' reached by the LP algorithm with the input (S, X, γ) will be denoted by LP(S, X). Here we omit γ because it is always the same.

The algorithm works in the product space $\mathcal{H} \times \mathbb{R}$ where the inner product is defined as $\langle (h_1, d_1), (h_2, d_2)\rangle = \langle h_1, h_2\rangle_\mathcal{H} + d_1 d_2$. Here, $\langle \cdot, \cdot\rangle_\mathcal{H}$ refers to the inner product in the original space $\mathcal{H}$. The norm in the product space is induced by its inner product.

In the while-loop of the LP algorithm there is a step which we will call a progress-contributing step. This name is motivated by the fact that, whenever the LP algorithm reaches such a step, it updates (h', d') so that $\|(h', d')\|^{-2}$ increases by a guaranteed value. Since $\|(h', d')\|^{-2}$ is upper bounded by some fixed value depending on γ, this allows us to estimate the number of iterations of the LP algorithm (i.e., the number of iterations of its while-loop); here, we adapt the analysis of the progress of the algorithm proposed in (Chubanov 2015) for the finite-dimensional case. The respective statements are formulated in the following lemma:

Lemma 1 *The following statements are true with respect to the LP algorithm:*

(a) At a progress-contributing step, the inverse squared

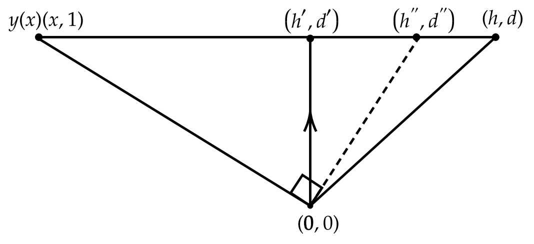

Figure 1: Iteration of the LP algorithm.

length of (h,d) *increases by at least that of* $y(x)(x,1)$:

$$\|(h',d')\|^{-2} \geq \|(h,d)\|^{-2} + \|y(x)(x,1)\|^{-2}. \qquad (2)$$

(b) The following property is invariant during the course of Algorithm 1, provided that X *is linearly* γ*-separable:*

$$\|(h',d')\| \geq \gamma/2 \qquad (3)$$

(c) If Algorithm 1 reports that X *is not linearly* γ*-separable, this decision is correct.*

(d) The number of iterations performed by the LP algorithm is bounded by $O(\gamma^{-2})$.

(e) If X *is linearly* γ*-separable, then the algorithm returns a state* S' *such that the associated halfpsace* $H_{S'}$ *correctly separates* X^- *and* X^+.

Proof. (a) This follows from the reciprocal Pythagorean theorem, which holds also for Hilbert spaces. Since x corresponds to a violated inequality of $P(X)$, we have

$$\langle (h,d), y(x)(x,1)\rangle = y(x)(\langle x,h\rangle + d) \leq 0.$$

Therefore, the triangle with vertices $(h,d), (\mathbf{0},0)$, and $y(x)(x,1)$ contains a right triangle with the right angle at $(\mathbf{0},0)$ and whose the other two vertices are $y(x)(x,1)$ (one of the vertices of the previous triangle) and some $(h'',d'') \in [(h,d), y(x)(x,1)]$, i.e., a point on the side opposite to $(\mathbf{0},0)$, of the previous triangle. Both triangles share the same height, which is the segment joining $(\mathbf{0},0)$ and its orthogonal projection (h',d') on $[(h,d), y(x)(x,1)]$. See Figure 1 for an illustration.

(b) Since the projection onto a line segment is a convex combination of the endpoints of the segment, at each iteration (h',d') belongs to the convex hull of $\{y(x)(x,1) : x \in X\}$, which means that there exists $c : X \longrightarrow [0,1]$ such that

$$(h',d') = \left(\sum_{x\in X} c(x)y(x)x, \sum_{x\in X} c(x)y(x)\right), \sum_{x\in X} c(x) = 1.$$

Since X is linearly γ-separable (by the assumption of (b)) and is contained in the unit ball centered at $\mathbf{0}$, there exists a solution (h^*,d^*) of $P(X)$ with $\|h^*\| = 1$ and $|d^*| \leq 1$ such that $y(x)(\langle x,h^*\rangle + d^*) \geq \gamma$ for all $x \in X$. It follows that

$$\gamma = \sum_{x\in X} c(x)\gamma \leq \sum_{x\in X} c(x)y(x)(\langle x,h^*\rangle + d^*)$$
$$= \langle (h',d'), (h^*,d^*)\rangle \leq \|(h',d')\|\|(h^*,d^*)\|.$$

Since $\|(h^*,d^*)\| \leq 2$ and (h^*,d^*) is feasible, and $\|x\| \leq 1$ for all $x \in \mathcal{X}$, this implies (3).

(c) follows from (b) and (d) follows from (a) and (b).

(e) In this case, the algorithm stops only when a feasible solution of $P(X)$ is found. ∎

Polyhedral Separation in Hilbert Spaces

From the theory of support vector machines, it follows that one can always find a suitable kernel such that the underlying data are linearly separable when represented in the respective RKHS. For this purpose, we can take, e.g., a radial-basis-function kernel. Such kernels are called universal kernels. So the case of linear separable classes is general enough. However, in many situations we would prefer a given kernel or it may be computationally infeasible to find one leading to linear separation, in a given class of kernels. This motivates us to study the case when $\mathcal{X}^-$ and $\mathcal{X}^+$ can be separated by a polyhedron. Another important aspect of polyhedral separation is the mentioned realizability in a suitable RKHS by an explicit choice of a kernel being a function of a given kernel.

Given a set $X \subset \mathcal{X}$, the major difficulty of the polyhedral classification problem is related to a correct representation of X^- as a union of t sets X_i^-, $i \in [t]$, in the sense that there exists a γ-separating polyhedron defined by some halfspaces H_i, $i \in [t]$, such that $X_i^- \cap H_i = \emptyset$. If we knew such a representation, we would solve the problem by simply applying Algorithm 1 t times. Unfortunately, the NP-hardness of proper learning even in the case $t = 2$ suggests that no polynomial algorithm exists for finding a correct representation of this type, unless $P = NP$.

At this stage we need to fix a hypothetical "ground-truth" polyhedron which we will need for theoretical purposes. That is, let us fix a γ-separating t-polyhedron F^* which correctly classifies the entire instance space $\mathcal{X}$. Polyhedron F^* contains $\mathcal{X}^+$ and is not intersected with $\mathcal{X}^-$. Let F^* be defined by halfspaces H_i, $i \in [t]$. Then $\mathcal{X}^+ \subset H_i$ for all $i \in [t]$. On the other hand, $\mathcal{X}^-$ can be represented as a union of some of its subsets $\mathcal{X}_i^-$, $i \in [t]$, such that $H_i \cap \mathcal{X}_i^- = \emptyset$ for each $i \in [t]$. Each set $\mathcal{X}^+ \cup \mathcal{X}_i^-$ is linearly γ-separated by the respective halfspace H_i.

From the above construction, only the union (where the subsets may overlap with each other)

$$\mathcal{X}^- = \mathcal{X}_1 \cup \ldots \cup \mathcal{X}_t$$

is assumed to be fixed. We will use this notation throughout this section. (The notation for halfspaces H_i is not fixed and may further have a different meaning depending on the context.)

There can be many ground-truth polyhedra with the above properties; we only assume that at least one exists. It should be stressed again that we need this hypothetical representation only for the theoretical analysis of the algorithm.

Definition 2 *A set* A *is called* i*-correct if* $A^- \subset \mathcal{X}_i^-$. *(Note that* A *can be* i*-correct for different* i.*)*

Further, by a t-partition of a set $\tilde{X}$ we mean a tuple $(\tilde{X}_1,\ldots,\tilde{X}_t)$ of its subsets whose union is $\tilde{X}$. In this definition, a set in the tuple is allowed to be empty.

Let $\mathcal{F}_i$, $i \in [t]$, be some families of sets in $\mathcal{H}$. Consider the following condition:

Condition 1 *There exists a* t*-partition* $(\tilde{X}_1,\ldots,\tilde{X}_t)$ *of* $\tilde{X}$ *such that for each* $i \in [t]$ *with* $\tilde{X}_i \neq \emptyset$ *there exists* $H_i \in \mathcal{F}_i$ *such that* $H_i \cap \tilde{X}_i = \emptyset$.

Algorithm 2: **Separation algorithm**

```
Input: $X$, $t$ and $\gamma$.
Output: A consistent $t$-polyhedron if there exists a $\gamma$-separating $t$-polyhedron.
$\mathcal{A} := \{X^+\}$
$S(X^+, 0) := \emptyset$ (initial state at the root of the search tree)
while no polyhedron found do
    $k := k + 1$
    Find out if there are $A_1, \dots, A_{t'} \in \mathcal{A}$ with $t' \leq t$ such
    that $F = \cap_{p \in [t']} H_{S(A_p, k-1)}$ correctly classifies $X$.
    if there is such $F$ then
        Return $F$
    end if
    (Branching step)
    Set $S(A, k) := S(A, k-1)$ for all $A \in \mathcal{A}$.
    for all $(A, x) \in \mathcal{A} \times X^-$ do
        if $x \in H_{S(A,k-1)}$ then
            $S(A \cup \{x\}, k) :=$ LP$(S(A, k-1), A \cup \{x\})$
            if the LP algorithm does not report that $A \cup \{x\}$
            is not linearly $\gamma$-separable then
                Add $A \cup \{x\}$ to $\mathcal{A}$.
            end if
        end if
    end for
end while
```

Relatively to polyhedral separation, Condition 1 has the following implication, when each element of $\mathcal{F}_i$ is a halfspace or the entire space $\mathcal{H}$: If Condition 1 is satisfied, then there exist sets $H_i \in \mathcal{F}_i$, $i \in [t]$, whose intersection does not contain any of the sets $\tilde{X}_i$. In the context of polyhedral separation of a given sample X, we will be interested in the case where each of the sets of the families $\mathcal{F}_i$ contains X^+ and $\tilde{X}$ is the set X^- of negative instances. Then, if Condition 1 is satisfied, the respective intersection $\cap_{i \in [t]} H_i$ yields a t'-polyhedron, with $t' \leq t$, consistently partitioning X. (We have $t' \leq t$ because H_i can be $\mathcal{H}$ according to our construction).

At each of its iterations, our separation algorithm (Algorithm 2) tries to construct a consistent t'-polyhedron with $t' \leq t$. In the case of success, it returns the polyhedron found in this way, otherwise Condition 1 is not satisfied for some suitably chosen families $\mathcal{F}_i$ (implicit in the algorithm, but used later for the theoretical analysis) and the algorithm continues. The logic of the algorithm is motivated by the following lemma which holds when Condition 1 is not satisfied:

Lemma 2 *If Condition 1 is not satisfied for $\tilde{X}$, then for each t-partition $(\tilde{X}_1, \dots, \tilde{X}_t)$ of $\tilde{X}$ there exists $j \in [t]$ with $\tilde{X}_j \neq \emptyset$ such that*

$$\forall H \in \mathcal{F}_j : H \cap \tilde{X}_j \neq \emptyset. \tag{4}$$

Proof. The lemma is obtained directly by negating Condition 1. ■

For the analysis of Algorithm 2, we introduce the notion of the *search tree*. Each level of the search tree corresponds to an iteration of the main loop of the algorithm, where a certain set family $\mathcal{A}$ is considered. The family $\mathcal{A}$ at the end of the kth iteration corresponds to the set of nodes forming the kth level of the search tree. Each node of the search tree is encoded by a pair (A, k) where A is contained in $\mathcal{A}$ at the end of iteration k, i.e., k is the level, of the search tree, containing (A, k). The nodes of the neighboring levels can be connected by arcs. There is an arc from $(A, k-1)$ to (A', k) if and only if $A = A'$ or $A' = A + \{x\}$ for some x.

When the algorithm decides whether a new node of the form $(A \cup \{x\}, k)$ is to be added to the search tree, based on a node $(A, k-1)$ at the previous level, it solves the respective problem $P(A \cup \{x\})$. Then, $(A \cup \{x\}, k)$ appears in the search tree if the LP algorithm finds a halfspace consistently partitioning $A \cup \{x\}$ (i.e., if the LP algorithm does not report that $A \cup \{x\}$ is not linearly γ-separable). The algorithm stores the states $S(A, k)$ reached by the LP algorithm at the respective nodes of the search tree. Informally, our algorithm works as follows:

- Given a sample X, the algorithm tries to construct a separating polyhedron defined by no more than t inequalities. To "assemble" such a polyhedron, it tries to find a suitable combination of halfspaces associated with the states stored at the current level.
- At each iteration, it expands the search tree by adding a new level. The nodes of the previous level stay in the new one. At the branching step, for each node $(A, k-1)$, the algorithm generates a new node $(A \cup \{x\}, k)$, provided that x is not separated by the halfspace $H_{S(A,k-1)}$ associated with the parent node $(A, k-1)$. The LP algorithm is then applied to the problem $P(A \cup \{x\})$ starting with the state $S(A, k-1)$ that has been previously reached for the parent node. Since $H_{S(A,k-1)}$ does not consistently separate x, it follows that the LP algorithm makes at least one progress-contributing step. We will use this property in the analysis of the running time of the algorithm.
- The search tree has the property that the state associated with a node (A, k) is compatible with the states associated with the other nodes of the subtree rooted at (A, k). This makes it possible to trace the progress of the LP algorithm along each path of the search tree. For this purpose, we introduce the notion of a progress-contributing arc, which is one corresponding to a transition via the LP algorithm from a state to another one with at least one progress-contributing step of the LP algorithm.

An example of a search tree constructed by Algorithm 2 is illustrated in Figure 2.

Recall that each nonempty state S consists of an associated pair (h_S, d_S) defining a halfspace H_S. When reaching another state $S' \neq S$ from S, the LP algorithm makes at least one progress-contributing step that increases $\|(h_S, d_S)\|^{-2}$ by a certain guaranteed value (1/4) following from inequality (2) of Lemma 1. So it is natural to introduce the following potential function π defined at each node of the search tree:

$$\pi(A, k) := \begin{cases} \|(h_{S(A,k)}, d_{S(A,k)})\|^{-2}, & \text{if } S(A,k) \neq \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

We call an arc $((A, k-1), (A', k))$ progress-contributing if

$$\pi(A', k) \geq \pi(A, k-1) + 1/4.$$

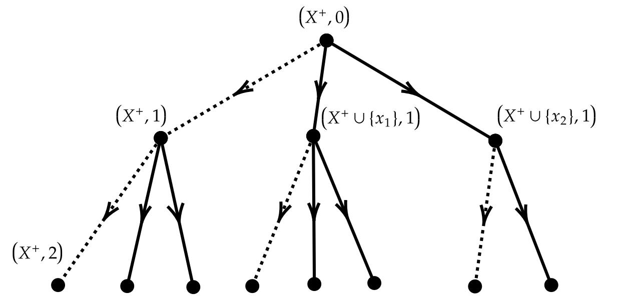

Figure 2: Search tree: solid arcs are progress-contributing; for each parent node, if more than one arc during the branching step, then one of them is not progress-contributing and the others should be progress-contributing.

This inequality takes place if and only if the LP algorithm makes at least one progress-contributing step when starting with $S(A, k-1)$ to solve the problem $P(A')$. If $A' = A$, then $S(A', k) := S(A, k-1)$. Therefore, for each arc $((A, k-1), (A', k))$ of the search tree, $\pi(A', k) \geq \pi(A, k-1)$. That is, π is non-decreasing along each path of the search tree.

On the other hand, the LP algorithm implies that

$$\pi(A, k) \leq 4\gamma^{-2}.$$

Therefore, the number of progress-contributing arcs on each path of the search tree is bounded by $O(\gamma^{-2})$.

Lemma 3 *At the end of iteration k, $|\mathcal{A}|$ is bounded by*

$$((m+1)k)^{O(\gamma^{-2})}, \tag{5}$$

where $m = |X|$.

Proof. Since each path of the search tree contains no more than $O(\gamma^{-2})$ progress-contributing arcs, it follows that each path from the root to the kth level of the tree can be uniquely encoded by a k-vector whose components take integer values from 0 to m and whose number of nonzero components is bounded by $O(\gamma^{-2})$. The (5) is an upper bound on the total number of such k-vectors. At the same time, this is an upper bound on the number of nodes at the kth level of the search tree, i.e., on $|\mathcal{A}|$ at the end of the kth iteration. ■

Now we come back to the notion of i-correctness (Definition 2) and extend it to the notion of i-correctness of a path: Further, we say that a path in the search tree is i-correct if A is i-correct for each node (A, k) on this path.

Lemma 4 *If (A, k) is a node of the search tree and A is i-correct, then the path connecting the root and (A, k) is i-correct.*

Proof. The path from the root has the form

$$(A_0, 0) = (X^+, 0), (A_1, 1), \ldots, (A_k, k) = (A, k).$$

Observe that $A_l \subseteq A_{l+1}$, $l = 0, \ldots, k-1$. This means that if some A_l is not i-correct then A_k is not i-correct because A_l is contained in A_k, which contradicts the assumption that A is i-correct. ■

Theorem 1 *If there is a consistent γ-separating t-polyhedron, then the running time of Algorithm 2 is bounded by*

$$m^{O(t\gamma^{-2}\lceil \log_m(\gamma^{-1})\rceil)}. \tag{6}$$

Proof. Consider iteration k. Let $\tilde{X} = X^-$. Consider a t-partition $(\tilde{X}_1, \ldots, \tilde{X}_t)$ of $\tilde{X}$ such that each nonempty $\tilde{X}_i$ is i-correct. For each $i \in [t]$ with $\tilde{X}_i \neq \emptyset$, let $\mathcal{F}_i$ be the family of all halfspaces associated with states $S(A, k-1)$ corresponding to i-correct sets A in $\mathcal{A}$. For each i with $\tilde{X}_i = \emptyset$, let $\mathcal{F}_i = \{\mathcal{H}\}$.

Let $\mathcal{A}_i$ denote the family of all i-correct sets in $\mathcal{A}$. Note that the family $\mathcal{A}_i$ is not empty for each $i \in [t]$ because it at least contains X^+.

Assume that the algorithm does not find a consistent t'-polyhedron with $t' \leq t$ at the current iteration. Then no consistent t-polyhedron can be assembled from the families $\mathcal{F}_i$, $i \in [t]$. It follows that Condition 1 is not satisfied. Then Lemma 2 implies that there exists $j \in [t]$ such that $\tilde{X}_j \cap H_{S(A,k-1)} \neq \emptyset$ for all j-correct sets $A \in \mathcal{A}$. This implies that, for each $A \in \mathcal{A}_j$, the current iteration k should construct a progress-contributing arc connecting $(A, k-1)$ with (A', k) where $A' = A \cup \{x\}$, $x \in \tilde{X}_j$. That is, A' is also j-correct. Denote this j by j_k, where k is the number of the current iteration.

Now let us consider the sequence of indices $j_1, j_2, \ldots$ over all iterations of the algorithm. Consider the subsequence of indices equal to some $r \in [t] : j_{k_1} = r, j_{k_2} = r, \ldots$. Here, $k_1, k_2, \ldots$ are the respective levels of the search tree. For each k_s, the previous level $k_s - 1$ in the search tree contains a node $(A, k_s - 1)$ where A is r-correct. Lemma 4 implies that the path from the root to $(A, k_s - 1)$ is r-correct. Our construction implies that each r-correct path from the root to level $k_s - 1$ is connected with level k_s by a set of arcs where at least one arc is progress-contributing and has the form $((A, k_s - 1), (A', k_s))$ where A' is r-correct, for each s. It follows that there exists an r-correct path, over all levels of the search tree, which contains at least as many progress-contributing arcs as the number of elements of the subsequence.

Assume that the number of iterations is infinite or exceeds $4t\gamma^{-2}$. Then, for some l, the subsequence of indices equal to l contains more than $4\gamma^{-2}$ elements. This implies a contradiction because in this case the search tree should contain an l-correct path with more than $4\gamma^{-2}$ progress-contributing arcs. Therefore, the number of iterations is not greater than $4t\gamma^{-2}$. When the algorithm stops, it returns a consistent t'-polyhedron with $t' \leq t$ because this is the only condition when it can stop.

The number of elementary operations needed for the LP algorithm whenever it is called from Algorithm 2 is bounded by $O(m\gamma^{-2})$ where the sample size $m = |X|$ appears because, at each of its iterations, the LP algorithm needs to find a violated constraint of a problem of the form $P(A \cup \{x\})$, where $(A \cup \{x\}) \subseteq X$. At each iteration, the time needed to construct F or make sure that none exists is bounded by $O(|\mathcal{A}|^t)$. Since the number of iterations of Algorithm 2 is bounded by $O(t\gamma^{-2})$, Lemma 3 implies that $|\mathcal{A}|$ is bounded by (5) with k replaced by $t\gamma^{-2}$. Replacing γ^{-2} by

Algorithm 3: **Improper-separation algorithm**

```
**Input:** $X$ and $\gamma$.
**Output:** A polyhedron consistent with $X$ if there exists a $\gamma$-separating polyhedron.
$\mathcal{A} := \{X^+\}$
$S(X^+, 0) := \emptyset$ (initial state at the root of the search tree)
while no polyhedron found do
    $k := k + 1$
    $F := \cap_{A \in \mathcal{A}} H_{S(A,k-1)}$
    if $F$ correctly separates $X^-$ and $X^+$ then
        Return $F$
    end if
    Pick an arbitrary $x \in X$ incorrectly classified by $F$. ($x \in X^-$ because $X^+ \subset F$.
    (Branching step)
    Set $S(A, k) := S(A, k-1)$ for all $A \in \mathcal{A}$.
    for all $A \in \mathcal{A}$ do
        $S(A \cup \{x\}, k) :=$ LP$(S(A, k-1), A \cup \{x\})$
        if the LP algorithm does not report that $A \cup \{x\}$ is not linearly $\gamma$-separable then
            Add $A \cup \{x\}$ to $\mathcal{A}$.
        end if
    end for
end while
```

$m^{\log_m(\gamma^{-2})}$ and rearranging the terms, we get the complexity bound (6). ∎

Now we formulate another algorithm based on similar ideas. In contrast to Algorithm 2, it constructs F as the intersection of all halfspaces available at the current iteration. If F is not consistent with the given sample X, then it chooses an arbitrary element x of X to which F assigns a false label. Since all the halfspaces contain X^+, it follows that $x \in X^-$. Since $x \in F$, it follows that $x \in H_{S(A,k-1)}$ for all $A \in \mathcal{A}$ at the beginning of the current iteration. This observation allows us to prove that the previous bound on the number of iterations of Algorithm 2 is also valid for Algorithm 3.

Theorem 2 *If there exists a γ-separating t-polyhedron correctly classifying the instance space, then Algorithm 3 finds a consistent polyhedron in time*

$$O\left(m \cdot (2t\gamma^{-2})^{4\gamma^{-2}}\right) \tag{7}$$

The number of halfspaces defining the polyhedron is bounded by

$$O((8t\gamma^{-2})^{4\gamma^{-2}}). \tag{8}$$

Proof. The number of iterations is bounded by the same value as that for Algorithm 2. This can be proved by the same argument as that we have used in the proof of Theorem 1. The only difference is how polyhedron F is constructed and what conclusions are made if no polyhedron F is found; in Algorithm 3, we simply check if the polyhedron F, built as the intersection of halfspaces associated with all states reached at the previous level, gives us a consistent polyhedron. If it doesn't, then each arc $((A, k-1), (A \cup \{x\}, k))$ is progress-contributing because $x \in X^-$ is contained in each of the halfspaces $H_{S(A,k-1)}$, which means that the LP algorithm makes at least one progress-contributing step during the transition from $S(A, k-1)$ to $S(A \cup \{x\}, k)$. Now we can derive an upper bound $4t\gamma^{-2}$ on the number of iterations in the way identical to that used in the proof of Theorem 1. To estimate $|\mathcal{A}|$ at the end of each iteration, we use (5) where k should be replaced by $4t\gamma^{-2}$ and $m+1$ by 2 because the search tree constructed by Algorithm 3 is binary, which means that the paths can be encoded by binary vectors. ∎

Remark. If none consistent γ-separating t-polyhedron exists, then the number of iterations of both algorithms will be infinite. If such a polyhedron exists, the number of iterations of both algorithms is bounded by $4t\gamma^{-2}$. So we can add a step verifying whether $k \leq 4t\gamma^{-2}$. If this is not the case, then no consistent γ-separating t-polyhedron exists.

PAC Polyhedral Classification

If $\mathcal{H}$ is finite-dimensional, the VC dimension of the t-polyhedron returned by Algorithm 2 is bounded by $O(dt \log t)$. To obtain the VC dimension of the polyhedron returned by Algorithm 3, the t in the above estimate should be replaced by the respective bound (8). Thus, we come to the following corollary:

Corollary 1 *Let $\mathcal{H}$ be a space of dimension $d < \infty$. Then the following statements are true:*

- *Algorithm 2 constructs a PAC polyhedron in time $m^{O(t\gamma^{-2})}$ from a concept class whose VC dimension D is bounded by $O(dt \log t)$.*
- *Algorithm 3 constructs a PAC polyhedron in time (7) from a concept class with VC dimension D bounded by $O((\gamma^{-2} + \log t)(t\gamma^{-2})^{4\gamma^{-2}})$.*

The above PAC learning is then possible with a sample complexity of $O(\varepsilon^{-1}(D + \log(\delta^{-1})))$, where D is the respective VC dimension.

For the infinite-dimensional case, we prove the following:

Corollary 2 *Let $\mathcal{H}$ be an RKHS with a continuous kernel K. Let $\mathcal{X}$ be the image of a compact set Ω in $\mathbb{R}^s$ under the feature map φ of the RKHS. Both Algorithm 2 and Algorithm 3 can be implemented so as to produce PAC polyhedra under the same assumptions as before. The sample complexity is polyhomially bounded in the dimension s of the space containing Ω, γ^{-1}, and t.*

Proof. The main idea is to use the well-known discretization trick which consists in replacing Ω by a finite set $\Omega^\#$ with a sufficiently small step size. Then the algorithms are modified by replacing elements x of $\mathcal{H}$ occurring in the course of the algorithms by their approximations from the image $\varphi(\Omega^\#)$. Of course, φ is not computed explicitly because we only need K to compute inner products. For more details, see (Chubanov 2023). ∎

Summary

We propose an algorithm for PAC learning in Hilbert spaces, such as RKHS's. The algorithm learns polyhedral concepts, which makes it universally applicable because a suitable kernel can always be explicitly chosen so that polyhedral concepts become realizable in the respective RKHS.

References

Anthony, M. 2004. Generalization Error Bounds for Threshold Decision Lists. *Journal of Machine Learning Research*, 5: 189–217.

Anthony, M.; and Ratsaby, J. 2012. Robust cutpoints in the logical analysis of numerical data. *Discrete Applied Mathematics*, 160(4-5): 355–364.

Bennett, K. P.; and Bredensteiner, E. J. 2000. Duality and Geometry in SVM Classifiers. In *ICML*, 57–64.

Blumer, A.; Ehrenfeucht, A.; Haussler, D.; and Warmuth, M. K. 1987. Occam's Razor. *Inf. Process. Lett.*, 24(6): 377–380.

Blumer, A.; Ehrenfeucht, A.; Haussler, D.; and Warmuth, M. K. 1989. Learnability and the Vapnik-Chervonenkis dimension. *J. ACM*, 36(4): 929–965.

Chattopadhyay, A.; Mahajan, M.; Mande, N. S.; and Saurabh, N. 2019. Lower Bounds for Linear Decision Lists. *CoRR*, abs/1901.05911.

Chubanov, S. 2015. A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*, 153: 687–713.

Chubanov, S. 2023. On the complexity of PAC learning in Hilbert spaces. *arXiv:2303.02047*.

Dantzig, G. 1992. An ϵ-precise feasible solution to a linear program with a convexity constraint in $1/\epsilon^2$ iterations independent of problem size. *Tech. rep. SOL 92-5*.

Der, R.; and Lee, D. 2007. Large-Margin Classification in Banach Spaces. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics, AISTATS 2007*, 91–98.

Gottlieb, L.; Kaufman, E.; Kontorovich, A.; and Nivasch, G. 2018. Learning convex polytopes with margin. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, 5711–5721.

Gottlieb, L.-A.; Kaufman, E.; Kontorovich, A.; and Nivasch, G. 2022. Learning Convex Polyhedra With Margin. *IEEE Transactions on Information Theory*, 68: 1976–1984.

Jeroslow, R. G. 1975. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11(2): 119–124.

Kantchelian, A.; Tschantz, M. C.; Huang, L.; Bartlett, P. L.; Joseph, A. D.; and Tygar, J. D. 2014. Large-Margin Convex Polytope Machine. In *Proceedings of the Neural Information Processing Systems conference, 2014*, 3248–3256.

Klivans, A. R.; O'Donnell, R.; and Servedio, R. A. 2008. Learning Geometric Concepts via Gaussian Surface Area. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, 541–550.

Long, P. M.; and Servedio, R. A. 2006. Attribute-efficient learning of decision lists and linear threshold functions under unconcentrated distributions. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems, 2006*, 921–928.

Mangasarian, O.; Setiono, R.; and Wolberg, W. H. 1990. Pattern Recognition Via Linear Programming: Theory And Application To Medical Diagnosis.

Mangasarian, O. L. 1968. Multisurface method of pattern separation. *IEEE Trans. Information Theory*, 14(6): 801–807.

Manwani, N.; and Sastry, P. S. 2010. Learning Polyhedral Classifiers Using Logistic Function. In *ACML*, volume 13 of *JMLR Proceedings*, 17–30. JMLR.org.

Mavroforakis, M. E.; and Theodoridis, S. 2006. A geometric approach to Support Vector Machine (SVM) classification. *IEEE Trans. Neural Networks*, 17(3): 671–682.

Megiddo, N. 1996. On the complexity of polyhedral separability. Technical Report RJ 5252, IBM Almaden Research Center.

Rossi, F.; and Villa, N. 2006. Support vector machine for functional data classification. *Neurocomputing*, 69(7-9): 730–742.

Vempala, S. S. 2010. A Random-Sampling-Based Algorithm for Learning Intersections of Halfspaces. *J. ACM*, 57(6).