On Undecided LP, Clustering and Active Learning

Stav Ashur
Department of Computer Science, University of Illinois, Urbana, IL, USA

Sariel Har-Peled
Department of Computer Science, University of Illinois, Urbana, IL, USA

Abstract

We study colored coverage and clustering problems. Here, we are given a colored point set, where the points are covered by k (unknown) clusters, which are monochromatic (i.e., all the points covered by the same cluster have the same color). The access to the colors of the points (or even the points themselves) is provided indirectly via various oracle queries (such as nearest neighbor, or separation queries). We show that one can correctly deduce the color of all the points (i.e., compute a monochromatic clustering of the points) using a polylogarithmic number of queries, if the number of clusters is a constant.

We investigate several variants of this problem, including *Undecided Linear Programming* and covering of points by k monochromatic balls.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Linear Programming, Active learning, Classification

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.12

Related Version *Full Version*: https://arxiv.org/abs/2103.09308

Funding *Sariel Har-Peled*: Work on this paper was partially supported by a NSF AF award CCF-1907400.

Acknowledgements The authors thank Pankaj Agarwal for useful discussions. We thank Lev Reyzin for pointing out the work by Maass and Turán [15, 16].

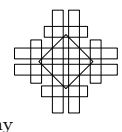
1 Introduction

Given a set of points P in $\mathbb{R}^d$ that are labeled (say, colored as red and blue), the problem of learning a classifier that labels the points correctly is a standard machine learning question. In the active learning settings, querying/exposing the label of an input is an expensive endeavor, and one tries to minimize such queries while performing the learning task.

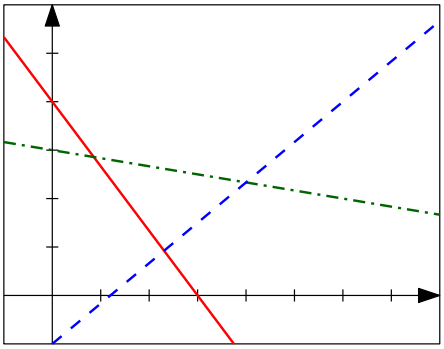
We are interested in a somewhat related question: If the input point set has a "simple" structure, but we are given access to the input via oracles that performs more "interesting" queries than just exposing the label of a point, can one classify correctly all the input points using relatively few oracle queries?

Implicit input model. In particular, consider the situation that instead of the algorithm reading the input, as in the classical settings, the access to the input is via *input primitives*, or *oracles*. Such indirect access to the data rises naturally if the data is already stored in a preexisting database or data-structure. This approach is of relevance nowadays as large amount of data makes even the basic task of reading the input infeasible or prohibitively expensive.

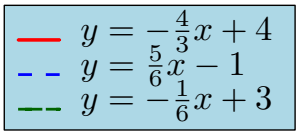
This gives rise to the main motivation for this work – what input primitives/oracles one needs, so that one can derive efficient algorithms. Here, of special interest are algorithms with running times that are sublinear in the input size.



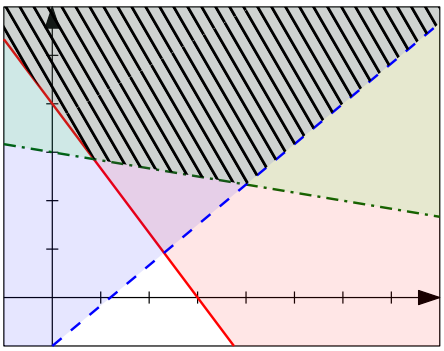

Problem I: Undecided linear programs. An instance of linear programming is a set of n linear inequalities on d variables, where one needs to find an assignment of real values to the variables, such that all the inequalities hold. We consider a new variant of LP, first studied by Maass and Turán [14], where the n linear constraints are given, but we do not know a priori whether the inequality is $\leq$ or $\geq$ for each one of them. Geometrically, this corresponds to being given n hyperplanes in $\mathbb{R}^d$, each having two closed halfspaces associated with it. Which of the two halfspaces is the one used in the LP, can be revealed by querying an oracle. For example, the "standard" *separation oracle*, which returns for a given query point p, a violated constraint of the LP, or alternatively returns that p is feasible.



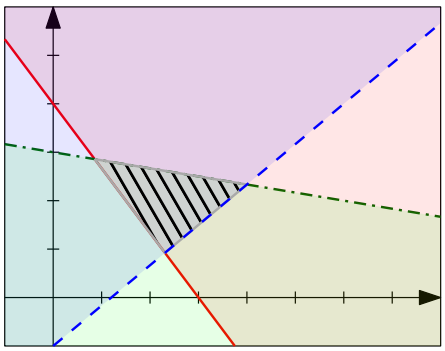
(i) A set of "undecided" constraints in 2d...



(ii) ... is a set of lines in the plane.



(iii) A possible commitment of the constraints, and the feasible polygon induced.



(iv) An alternative commitment of the underlying constraints, and the induced polygon.

Figure 1 An instance of 3 undecided constraints (bottom) with two possible sets of underlying decided constraints (top left and right).

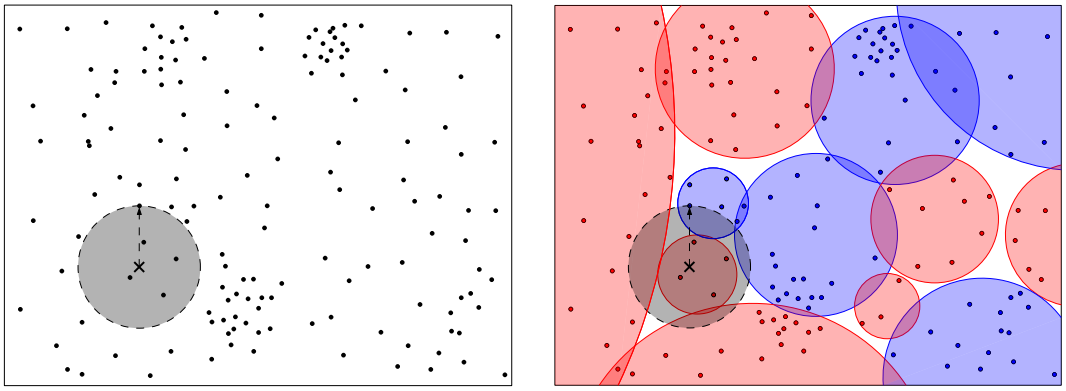
Problem II: Separating red and blue points, with a counterexample oracle. The above problem, in the dual, is the following: The input is a set of unlabeled points, and the task is to compute a hyperplane separating the blue points from the red points. The "separation" oracle here, is given an oriented hyperplane that is supposed to separate the points, and the oracle returns a misclassified point. A natural question is how many queries of this type one has to perform until classifying all the points correctly.

The learning model. Our model seems to be Angluin's equivalence query model for active learning [1]. In particular, Maass and Turán [16, 15] studied the above two problems. See Remark 7 for more details.

Problem III: Covering/clustering points with a ball using proximity oracle. Consider the situation where the input is a set of colored points in $\mathbb{R}^d$, where all the (say) red points are inside a ball, and all the points outside the ball are blue. Here, our access to the color of the points is via an oracle that can answer colored nearest-neighbor (NN) or furthest-neighbor (FN) queries (that is, the oracle can return the closest red [or blue] point to the query point). The task at hand is to label (i.e., color the points) correctly using a minimal number of oracle queries.

The challenge. To appreciate the difficulty in solving the above problem, consider the natural naive algorithm – pick a random sample, expose the colors of the points in the sample (in this case the colored NN queries can do that), compute a ball separating the red points and blue points in the sample, and feed it into the "counterexample" oracle (which returns a colored point that is on the wrong side of the ball – this oracle can be implemented using the NN/FN queries). The algorithm adds this counterexample to the current set of points that their color is known. Now the algorithm repeats the process finding an updated separating circle for the points their colors are known. This algorithm does arrive to the right answer, but it is easy to come up with examples where it has to do a linear number of iterations. As such, the challenge is to get a sublinear number of iterations.

Problem IV: Covering points by monochromatic balls. Consider the situation where the input is a set of points that can be covered by k balls (i.e., clusters). All the points covered by the same ball, have the same label/color (i.e., red or blue). Furthermore, given a query point, the oracle returns the closest point of a prespecified color.



(a) A NN (blue) oracle query marked by an "x" reveals that the points in the interior of the disk are red, and the returned point is blue.

(b) The same query with the underlying colors of the points, and an optimal solution with $k = 12$ monochromatic disks.

Figure 2 An instance of Problem IV.

Related work. Linear programming has a long history, see the survey by [5].

In *active learning*, also known as query learning or experimental design, the purpose of the algorithm is learning a concept but by querying specific input entries for their label. The main criteria for efficiency is minimizing the number of queries. The basic premise is that asking a specialist to label a specific example is an expensive operation. See Settles [17] for a survey on the topic of active learning.

Table 1 A summary of the results on ULP. Here δ can be chosen to be any constant in $(0, 1)$. The O_d hides constants that depends (probably exponentially or worse) on the dimension.

dim	ref	RT	# queries	Oracle
$d = 2$	Lemma 11	$O(n \log n)$	$O(\log n)$	Separation
	Lemma 5	$O(n)$	$O(\log^2 n)$	
	Lemma 13	$O(n)$	$O(\log n)$	Separation + labeling
$d = 3$	Lemma 15	$\widetilde{O}(n^{3/2})$	$O(\log n)$	Separation
	Theorem 18	$\widetilde{O}(n^{1+\delta})$	$O(\delta^{-1} \log n)$	
d	[16, 15] Lemma 2	$n^{O(d^2)}$	$O(d^2 \log n)$	Separation
	Lemma 2	$\widetilde{O}\left(n^d\right)$	$O(d^3 \log n)$	
	Lemma 5	$O_d(n)$	$O_d(\log^d n)$	
	Remark 12	$O_d(n \log n)$	$O_d(\log^{d-1} n)$	
	Theorem 14	$O_d(n)$	$O_d(\log^{d-1} n)$	Separation + labeling
$d > 3$	Remark 19	$\widetilde{O}\left(n^{1+\delta}\right)$	$O_d(\log^{d-2} n)$	Separation

Closer to our settings, Har-Peled et al. [10], studied algorithms for actively learning a convex body using a separation oracle. Such an oracle either confirms that that the query point is within the convex body, or alternatively, returns a hyperplane separating the point and the convex region.

Kane et al. [13] studied half plane classifiers using comparison queries, and showed an exponential query complexity improvement over learning with only membership queries. Their model is somewhat similar to the model of Problem II above (of separating red and blue points), except that their model assumes that the oracle can return the distance of a query point to the optimal separating hyperplane, while our model is somewhat different, only assuming that the oracle can identify a misclassified point.

In general, computational models that involve various oracles as algorithmic building blocks have been studied in computational geometry, as they represent algorithms in which the input is given implicitly, and access to any information provided by the input is done by oracle queries. See Har-Peled et al. [11] and references therein for such examples.

As mentioned above, Maass and Turán [16, 15] studied Problems I and II – our results are better, but only in constant dimensions, see Remark 7 for details.

1.1 Our results

(A) Undecided LP. We present several algorithms for solving undecided LPs. In Section 2.1, we revisit the algorithm of Maass and Turán, showing ULPs can be solved using $O(d^2 \log n)$ oracle queries, the main tool is repeatedly computing a centerpoint and feeding it to the oracle to further truncate the search space. This bound is polynomial in d, but the algorithm itself is doubly exponential in the dimension d. The running time can be improved to (roughly) $O(n^d)$, with the number of queries deteriorating to $O(d^3 \log n)$.

We present a linear time algorithm, for constant dimension, that uses cutting in Section 2.2, but the number of separation oracle queries is now $O(\log^d n)$.

In Section 2.3, we show that in the plane, if one is allowed also to use an exposure oracle (i.e., an oracle that returns the color of a specific point), then one can reduce the number of oracle queries to $O(\log n)$ (instead of $O(\log^2 n)$ described above). The algorithm is a (potentially interesting) combination of the two previous algorithms.

(B) Covering (red) points by a single ball. In Section 3.1 we study Problem III, we present an algorithm that uses $O(\log^2 n)$ colored NN/FN queries, and computes the single ball that covers (say) all the red points, and avoids all the blue points. The algorithm works by lifting the input point set to three dimensions, and then using the algorithm for undecided LP.

(C) Covering points by k monochromatic balls. In Section 3.2 we address Problem IV above, where the input is covered by k monochromatic balls, and we have access to a colored NN oracle. Inspired by the one ball case, we show a greedy algorithm that finds a ball that covers $O(1/\mathsf{k})$ faction of the uncovered points. This leads to an algorithm that performs $O(k^{d+2}\log^{d+2} n)$ queries (see Theorem 24) and correctly classifies all the points.

2 Algorithms for solving undecided linear programs

▶ Remark 1. All the algorithms described below for undecided LP work in the same fashion – they generate a sequence of separation oracle queries. Specifically, if any of the query points are feasible, the algorithm immediately stops, and output the ULP is feasible, with the queried point as a proof. As such, for the clarity of description, in the following we always assume a separation oracle that returns a separating hyperplane.

2.1 Centerpoint based algorithm for solving ULPs

We review the algorithm of Maass and Turán [16, 15] for solving undecided LP (they did not provide running time bounds for their algorithm, which we do).

Observe that for a set of d-dimensional (closed) hyperplanes H, if there exists a feasible point, then, under general position assumption, there exists a vertex of the arrangement $\mathcal{A}(H)$ that is feasible.

Let P be a set of n points in $\mathbb{R}^d$. For a parameter $\alpha \in (0,1)$, a point $c \in \mathbb{R}^d$ is an **α-centerpoint** if all halfspaces containing c also contain at least αn points of P. A classical implication of Helly's theorem, is that for any set P of n points in $\mathbb{R}^d$, there is a $1/(d+1)$-centerpoint. Such a point is simply a **centerpoint** of P. Such a centerpoint can be computed in $O(n^{d-1})$ time [12, 3]. A $2/d^2$-centerpoint can be computed in near linear time [8] (here, the running time is polynomial in d).

The algorithm. The input is a set H of n hyperplanes in d dimensions. The first step of the algorithm is to compute the set P of vertices of the arrangement $\mathcal{A}(H)$. The number of such vertices is $\leq \binom{n}{d}$. As long as P has more than $d^2 \log n$ points, the algorithm computes a centerpoint c of P. The algorithm then queries the separation oracle on c to decide whether c is feasible. If it is, then the algorithm is done, as it computed a feasible point. Otherwise, the oracle returned a violated constraint of the given LP (this also provides the algorithm with the direction of the constraint for the associated input hyperplane). The algorithm removes all the points of P that violate this constraint, and repeats until $|P| = O(d^2 \log n)$. Once P is that small, the algorithm simply checks the feasibility of each of the remaining vertices by querying it with the separation oracle.

▶ **Lemma 2** (proof in full version [2]). *The above algorithm computes a feasible point of H using $O(d^2 \log n)$ separation oracle queries. The runtime of this algorithm, ignoring the oracle calls, is $O(n^{d(d-1)})$ time.*

Alternatively, the algorithm can be modified so that it computes a feasible point using $O(d^3 \log n)$ separation oracle queries. The running time of this algorithm, ignoring the oracle calls, is $\widetilde{O}(n^d)$ time, where $\widetilde{O}$ hides polylogarithmic terms.

2.2 Cutting based algorithm

Here, we present the new algorithm to solved undecided LP using cuttings.

▶ **Definition 3.** *For a set of n d-dimensional hyperplanes H, a **$1/r$-cutting** of H is a partition Ξ of $\mathbb{R}^d$ into $O(r^d)$ simplices, such that the interior of each simplex of the cutting intersects at most n/r hyperplanes of H. The list of hyperplanes intersecting the interior of a simplex $\nabla \in \Xi$, is the **conflict list** of ∇.*

A $1/r$-cutting, and its conflict list can be computed in $O(nr^{d-1})$ time [7].

▶ Remark 4. The algorithm we present next call recursively on subsets of the constraints, and it also calls recursively on lower dimensional subspaces. In particular, the oracle can be applied to any lower dimensional affine subspace F, by using the original oracle in the ambient space – a returned constraint can be intersected with F to get a constraint in F. We emphasize that the oracle always work on the whole original input set of constraints – the recursive calls on subsets of the constraints are done for efficient bookkeeping, and do not effect how the oracle works.

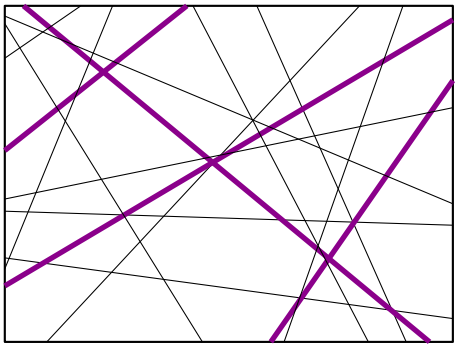
The Algorithm. The algorithm computes a $1/r$-cutting of H. This is a partition of $\mathbb{R}^d$ into $O(r^d)$ simplices, such that each simplex intersects at most n/r of the hyperplanes of H, where $n = |H|$. Let Ξ be this set of simplices, and let $\Xi_{d-1}(\Xi)$ be the set of $O(r^d)$ $(d-1)$-dimensional simplices that form the faces of the simplices of Ξ. The algorithm now solves the problem recursively on each of the $(d-1)$-dimensional simplices of $\Xi_{d-1}(\Xi)$, and the hyperplanes of H that intersects it. Each recursive call is on a problem that is one dimensional lower, and involves only n/r constraints, the oracle still applies to the whole set of constraints), see Remark 4.

If any of these recursive calls finds a feasible point, then we are done. Otherwise, the recursive calls performed involved the oracle, and forced some of the constraints to expose themselves. Let $\mathcal{C}$ be the set of these committed halfspaces (i.e., all the halfspaces returned by the separation oracle). If the intersection of all these constraints is empty (i.e., the associated LP is infeasible), then this can be discovered, in $O(|\mathcal{C}|)$ time, by invoking a standard LP solver on $\mathcal{C}$. If the LP is feasible, then it returns us a point p inside the polytope

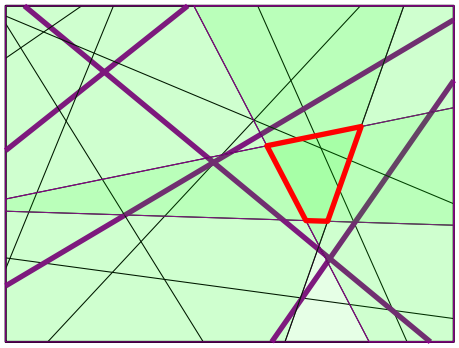
$$\mathcal{K} = \bigcap_{h^+ \in \mathcal{C}} h^+.$$

Furthermore, this polytope must be fully contained in the interior of one of the simplices of Ξ (otherwise, the algorithm would have found a feasible point in one of the recursive calls). By scanning Ξ, we discover the simplex $\nabla \in \Xi$ that contains p. The algorithm now call recursively on ∇ and its conflict list (passing $\mathcal{C}$ as the current set of committed constraints).

Algorithm in one dimension. The 1-dimensional case is solved using a binary search. Indeed, we have n uncommitted rays on the real line, and our purpose is to find an atomic interval that is feasible. To this end, the algorithm computes a median among the points



Cells of a cutting of an arrangement of undecided constraints.



A single cell of the cutting might contain a possibly feasible region in its interior.

Figure 3 Illustration of an iteration of the cutting-based algorithm. A cutting of an arrangement of undecided constraints is depicted on the left. After solving the problem recursively on the cuttings of the edges, some of the constraints are now committed. Their feasible region (the red polygon) they induce is now a polygon that must be fully contained in one of the cells of the cutting.

defining the rays. The algorithm then asks the oracle to commit the direction of the ray. This decrease the number of potential atomic intervals that might be feasible by half. In the end of the process, the algorithm is left with a single atomic interval that might be feasible – the algorithm asks the oracle whether the middle of this interval is feasible or not.

As such, after $O(\log n)$ iterations, the algorithm is done, as each iteration either finds a feasible point, or throws away half the rays as being irrelevant. The running time is $O(n)$, and the number of oracle queries performed is $O(\log n)$.

Analysis: Number of oracle queries. The query complexity of this algorithm is

$$Q_d(n) = O(r^d) Q_{d-1}\Big(\frac{n}{r}\Big) + Q_d\Big(\frac{n}{r}\Big),$$

with $Q_1(n) = O(\log n)$. The solution of this recurrence is $O(\log^d n)$, for r chosen to be a sufficiently large constant. Indeed, using induction, we have $Q_d(n) = O(r^d \log^{d-1} \frac{n}{r}) + Q_d\big(\frac{n}{r}\big)$,

Running time. As for the running time, we have

$$T_d(n) = O(nr^{d-1}) + O(r^d) T_{d-1}(n/r) + T_d(n/r),$$

with $T_1(n) = O(n)$. Assuming $T_{d-1}(n) \leq c_{d-1} n$, and for two constants c'_d and c''_d, we have

$$T_d(n) \leq c'_d n r^{d-1} + c''_d r^d c_{d-1} \frac{n}{r} + c_d \frac{n}{r} \leq \Big(c'_d r^{d-1} + c''_d r^{d-1} c_{d-1} + \frac{c_d}{r}\Big) n \leq c_d n,$$

which holds if $c_d \geq 2(c'_d r^{d-1} + c''_d r^{d-1} c_{d-1})$. This implies that $T_d(n) = O(n)$. We thus get the following result.

▶ **Lemma 5.** *Undecided LP with n constraints in $\mathbb{R}^d$, can be solved in $O_d(n)$ time, using $O(\log^d n)$ separation oracle queries.*

▶ Remark 6 (An implicit undecided LP). In the following, we need a variant of the above problem – the input is a set of undecided constraints, but the real undecided LP instance $\mathcal{I}$ corresponds to an (unknown) subset of these constraints. Fortunately, the given separation oracle works on the "real" instance of constraints $\mathcal{I}$. It is easy to verify that the above algorithm of Lemma 5 works verbatim in this case.

▶ Remark 7. The work of Maass and Turán [16, 15] also studied the dual settings of our problem (i.e., Problem II of separating red and blue points) They assume the input points are taken from a grid of bounded spread. They use the ellipsoid algorithm to get an efficient algorithm with small number of queries.

2.3 Better algorithms in two dimensions

The algorithm of Lemma 5 above uses only a separation oracle. One can get a faster algorithm if one is allowed to use a **labeling oracle** – this oracle, given an input point, returns its label/color. In our case, the labeling oracle reveals the underlying halfspace constraint defined by (undecided) input hyperplane.

We need the following lemma, due to Har-Peled and Mitchell [9].

▶ **Lemma 8** ([9]). *Let D be a fixed polygon with k edges, and let L be a set of m lines that intersect the interior of D. After $O(m(\log k + \log m))$ preprocessing, one can compute the number of vertices of $\mathcal{A}(L)$ that lie inside D, or sample such a vertex in $O(\log m)$ time.*

2.3.1 Polygon and point set reduction

▶ **Lemma 9** (proof in full version [2]). *Consider an instance of undecided LP in the plane, and a polygon D with t edges, such that the feasible region is contained in D. Using $O(\log t)$ separation queries, and $O(t)$ time, one can either compute a feasible point, or compute a polygon $D' \subseteq D$ with (say) at most 10 edges, such that the feasible region must be contained in D'.*

▶ **Lemma 10** (proof in full version [2]). *Consider an instance of undecided LP in $\mathbb{R}^d$, and a set P of n points. In $O(n)$ time, using $O(\log n)$ separation oracle queries, one can compute either: (i) a feasible point to the ULP, or alternatively, (ii) a polytope D with $O(\log n)$ faces, such that the feasible solution for the ULP lies inside D, and D contains no point of P.*

2.3.2 Near linear running time in two dimensions

Let L be the input set of n lines (i.e., undecided constraints). Let $L_0 = L$, and $D_0 = \mathbb{R}^2$.

Let V_i denote the set of vertices of $\mathcal{A}(L_{i-1})$ that lie in D_{i-1}. The algorithm computes $n_i = |V_i|$, using the algorithm of Lemma 8. There are two possibilities:

(A) If $n_i = 0$, then the algorithm picks any point p_i in D_{i-1}, and calls the separation oracle on p_i. If p_i is feasible the algorithm is done, otherwise, the algorithm returns that the given instance is infeasible.

(B) If $n_i = O(n \log n)$, the algorithm computes V_i by clipping the lines of L to D_{i-1}, and computing the arrangement of the resulting segments. This takes $O(n \log n + n_i)$ expected time, as this the time for computing the arrangement of n segments with n_i intersections.

The algorithm uses Lemma 10 on V_i to either find a feasible point, or a polygon D'_1 that must contain the feasible region, has at most $O(\log n)$ edges, and contains no point of V_i. The algorithm next uses Lemma 9 to further reduce the polygon into a polygon $D_i \subseteq D'_i$ with constant number of edges.

(C) Otherwise, the algorithm samples $O(n)$ vertices from V_i, and let R_i be the resulting sample. This is done using the algorithm of Lemma 8 in $O(n \log n)$ time. Next, the algorithm applies, as above, Lemma 10 and Lemma 9 to get a constant complexity polygon $D_i \subseteq D_{i-1}$ that does not contain any of the points of R_i.

The algorithm now scans the lines of L_{i-1}, computes the set of all lines $L_i \subseteq L_{i-1}$ that intersect D_i, and continues to the next iteration.

▶ **Lemma 11** (proof in full version [2]). *The above algorithm solves two dimensional undecided LP in expected $O(n \log n)$ time, using $O(\log n)$ separation oracle queries.*

▶ Remark 12. Combining the algorithm of Lemma 5 together with the algorithm of Lemma 11, when the dimension is two, results in an algorithm that solves ULP in d dimensions, with $O(\log^{d-1} n)$ separation queries, and running time $O(n \log n)$.

2.3.3 A linear time algorithm (using labeling oracle)

In the ith iteration, the algorithm computes a polygon D_i, with at most $k = 10$ boundary edges, that might contain a feasible point, and a list of lines L_i that intersect it in the ith iteration. Initially, D_0 is the whole plane, and $L_i = H$.

In the beginning of the ith iteration, the algorithm computes a random sample $R_i \subseteq L_i$ of size $O(\varepsilon^{-1} k \log k \log \varepsilon^{-1}) = O(1)$, where $\varepsilon = 1/2$. The sample R_i is, with probability close to one, an ε-net of L_i for polygons that have at most k boundary edges. The algorithm now calls the oracle for each undecided constraint of R_i to expose its true constraint. Let F_i be the face of the arrangement of the revealed constraints that is still feasible (if no such face exists, then we are done). The algorithm next considers the polygon $D'_i = F_i \cap D_{i-1}$. If this polygon is empty, then the algorithm is done. If D'_i has at most k edges, then the algorithm sets $D_i = D'_i$, computes $L_i = L_{i-1} \cap D_i$, and continues to the next iteration.

The only bad case here, is that D'_i has too many edges. Clearly, in the worst case, it can have at most $|R_i| + k$ edges. The algorithm now computes a centerpoint for the vertices of this polygon, and sends it to the oracle. If the point is feasible, then the algorithm is done. Otherwise, the oracle returned a violated constraint, and the polygon is split into two polygons, each with at most two thirds of its original vertices. Namely, in the new feasible polygon the number of edges is decreased by a constant factor. After a constant number of such iterations, the feasible polygon has at most k edges. The algorithm then sets this polygon to be D_i, and continues to the next iteration, as described above.

if L_i is empty, then the algorithm asks the oracle if some point in the interior of the polygon is feasible, and if not the given ULP is not feasible.

Analysis. Observe that D_i intersects no lines of R_i, and it has at most k edges. As such, by the ε-net theorem, with constant probability (say ≥ 0.99), the interior of D_i intersects at most $\varepsilon |L_{i-1}|$ lines. Namely, $|L_i| \leq |L_{i-1}| / 2$. This implies that the algorithm performs, in expectation (and also with high probability), at most $O(\log n)$ iterations. Each iteration requires $O(1) + O(|R_i|) = O(1)$ oracle queries, which readily implies a bound of $O(\log n)$ oracle queries.

The correctness of the algorithm itself follows as the feasible region if the LP is always contained inside D_i.

As for running time, each iteration requires $O(1 + |L_i|)$ amount of work. As such, the total expected running time is $\sum_i O(1 + |L_i|) = O(n)$.

▶ **Lemma 13** (proof in full version [2]). *Undecided LP with n constraints in $\mathbb{R}^2$, can be solved in $O(n)$ expected time, using $O(\log n)$ separation and labeling oracle queries.*

Combining the above algorithm with the cutting based algorithm of Lemma 5, results in the following slightly improved algorithm.

▶ **Theorem 14.** *Undecided LP with n constraints in $\mathbb{R}^d$, for $d \geq 2$, can be solved in $O_d(n)$ time, using $O(\log^{d-1} n)$ separation and labeling oracle queries.*

2.4 A query efficient algorithm in three dimensions

2.4.1 Emulating the two dimensional algorithm

In three dimensions, one can still get $O(\log n)$ queries, albeit getting running time $\tilde{O}(n^{3/2})$.

The input is a set H of n planes in three dimensions. The algorithm randomly samples a set V_1 of $O(n^{3/2}\log^3 n)$ vertices of $\mathcal{A}(H)$. Each vertex is generated by randomly choosing three constraints (planes) from H, and computing their intersection. Using the algorithm of Lemma 10, one computes a convex polytope $\mathcal{K}_1$ with $O(\log n)$ faces, which does not contain any vertex of V_1 (since Lemma 10 returns the $O(\log n)$ halfspaces whose intersection form the desired polytope, the polytope itself can be computed in $O(\log n \log\log n)$ time).

Next, the algorithm computes all the vertices of $\mathcal{A}(H)$ inside $\mathcal{K}_1$ – this can be done in an output sensitive fashion. To this end, one "walks" around the arrangement of $\mathcal{A}(H)$, starting (say) with the bottom vertex of $\mathcal{K}_1$. Specifically, one walks on edges of the arrangement (inside $\mathcal{K}_1$) using the data-structure of Chan [4], which provide dynamic maintenance of convex-hull in three dimensions, and extreme point queries. Each operation takes $O(\log^4 n)$ amortized time. The exact details of this exploration are somewhat delicate, but straightforward, and we omit them as they are similar in nature to the 2d algorithms (see [6] and references therein). Let V_2 be the resulting set of vertices. As we show below, with high probability $|V_2| = O(n^{3/2})$. As such, computing this set takes $O(n^{3/2}\log^4 n)$ time.

Deploying the algorithm of Lemma 10, one computes a convex polytope $\mathcal{K}_2$ with $O(\log n)$ faces, which does not contain any vertex of V_2. Consider the intersection polytope $\mathcal{K} = \mathcal{K}_1 \cap \mathcal{K}_2$. It avoids the vertices of V_1 and V_2 and thus avoids all the vertices of the arrangement of $\mathcal{A}(H)$. Furthermore, $\mathcal{K}$ is formed by the intersection of halfspaces, all induces by planes of H. It follows that $\mathcal{K}$ is a 3d face of the arrangement. As such, perform a single query on a point in the interior of $\mathcal{K}$, and return it as the output of the algorithm.

The algorithm again invokes Lemma 10 to compute a polytope $\mathcal{K}_2$ that contains no vertex of V_2. The algorithm computes the polytope $\mathcal{K}_1 \cap \mathcal{K}_2$ (in polylogarithmic time). If the intersection is empty then the given instance is infeasible. Otherwise, the algorithm query an point inside this intersection using the separation oracle. If the point is feasible then the algorithm is done, and otherwise, again, the instance is infeasible.

▶ **Lemma 15** (proof in full version [2]). *The above algorithm solves Undecided LP with n constraints, in $\mathbb{R}^3$, in $O\left(n^{3/2}\log^3 n\sqrt{\log\log n}\right)$ time, using $O(\log n)$ separation oracle queries.*

2.4.2 A faster algorithm

In the following, we assume that the set of planes of H is in general position – no three planes passes through a common line, and no four planes have a common intersection point.

Idea. A natural approach for getting a faster algorithm is to maintain a polytope $\mathcal{K}_i$, sample an ε-net for the vertices of the arrangement inside $\mathcal{K}_i$ (for an ε to yet be specified), and use the algorithm of Lemma 9 to find a low complexity polytope that avoids all the points in this ε-net. Intersecting this polytope with the previous active polytope, results in a shrunken feasible region $\mathcal{K}_i$. Furthermore, $\mathcal{K}_i$ contains an ε-fraction of the vertices of the arrangement inside it compared to $\mathcal{K}_{i-1}$. The algorithm then continues to the next iteration, till the polytope contains no vertices of $\mathcal{K}_i$, and then a single query in its interior settles the feasibility of the given ULP.

The challenge is that despite $\mathcal{K}_i$ being simple (i.e., having few faces), we do not know how to sample uniformly and efficiently from $\mathcal{V} \cap \mathcal{K}_i$, where $\mathcal{V} = \mathcal{V}(H)$ is the set of vertices of $\mathcal{A}(H)$. We leave this an open problem for further research. Instead, we offer the following over-sampling approach.

▶ **Lemma 16** (proof in full version [2]). *Let $\mathcal{K}$ be a polytope in three dimensions with k faces, and let H be a set of n planes, where all the faces of $\mathcal{K}$ lie on planes of H. One can sample a non-empty set X of at most $n-1$ vertices, such that (i) $X \subseteq \mathcal{V} \cap \mathcal{K}$, where $\mathcal{V} = \mathcal{V}(H)$, and the probability of any vertex of $\mathcal{V} \cap \mathcal{K}$ to be included in the sample is the same.*

The preprocessing time of the algorithm is $O(nk \log n)$, and a sampled set can be computed in $O(n)$ time.

▶ **Lemma 17** (proof in full version [2]). *Let $\delta \in (0,1)$ be a fixed constant, let $\mathcal{K}$ be a polytope in three dimensions with $t = O(\log n)$ faces, and let H be a set of n planes, where all the faces of $\mathcal{K}$ lie on planes of H. Let $\mathcal{V} = \mathcal{V}(H) \cap \mathcal{K}$ be the set of vertices of $\mathcal{A}(H)$ that lie inside $\mathcal{K}$. One can compute a polytope $\mathcal{K}'$, that is the intersection of $\mathcal{K}$ with $O(\log n)$ halfspaces, such that $\mathcal{K}'$ contains at most $|\mathcal{V}|/n^{\delta}$ vertices of $\mathcal{A}(H)$. The algorithm runs in $O(n^{1+\delta} \log^3 n \log\log n)$ time. The algorithm uses $O(\log n)$ separation oracle queries.*

Starting with $\mathbb{R}^3$ as the initial polytope, the algorithm repeatedly uses Lemma 17 to reduce the number of vertices of the arrangement inside the current polytope by a factor of $1/n^{\delta}$. In the ith iteration, the current polytope has $O(i \log n)$ faces, and as such the final polytope has at most $O((3/\delta)\log n)$ faces, as the algorithm has no vertices in it after $\lceil 3/\delta \rceil$ iterations. We conclude the following.

▶ **Theorem 18.** *For any $\delta \in (0,1)$, an instance of undecided LP in three dimensions with n constraints, one can solved using $O((\log n)/\delta)$ separation oracle queries, in*

$$O(n^{1+\delta} \log^4 n \log\log n)$$

time.

▶ Remark 19. Combining the algorithm of Lemma 5, together with the algorithm of Theorem 18, when the dimension is three, results in an algorithm that solves ULP in $d > 3$ dimensions, with $O(\delta^{-1} \log^{d-2} n)$ separation queries, and running time $\widetilde{O}(n^{1+\delta})$.

3 Covering points by monochromatic balls using proximity queries

3.1 Learning a single monochromatic ball using NN/FN queries

Problem statement. The input is a set $P = \{p_1, ..., p_n\}$ of n points in $\mathbb{R}^d$. The points are either blue or red, but their color is not initially provided. We have an access to a nearest-neighbor (NN) oracle, such that given a query point, and a color, it returns the closest point of this color to the query point. Similarly, we are given access to a furthest-neighbor (FN) oracle, that returns the furthest point of this color in P.

The task at hand is to correctly classify all the given points as either red or blue, minimizing the number of queries used.

Assumption: Single ball. Here we assume that all the red points in P can be covered by a single ball, while all the blue points are outside this ball. Our purpose here is to develop an efficient algorithm that performs as few oracle queries as possible, and exposes the color of all the points of P.

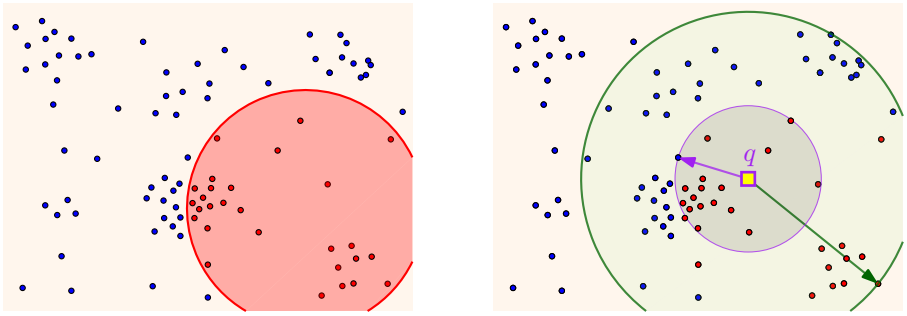


Figure 4 A set of red and blue points in $\mathbb{R}^2$ where a single disk suffices to separate the labels. A red furthest-neighbor (FN) query and a blue nearest-neighbor (NN) query at q confirms that a monochromatic disk centered at that point cannot contain every red point of the input.

The algorithm is described in the plane, but it also works in higher dimensions with minor modifications.

3.1.1 Lifting to three dimensions

Let $\text{disk}(p, r)$ denote the disk of radius r centered at p. Consider the mapping of such a disk to the point

$$\mathcal{d} = \vartheta(\text{disk}(p, r)) = \left(2p_x, 2p_y, r^2 - p_x^2 - p_y^2\right).$$

This can be interpreted as a somewhat bizarre encoding of disks as points in three dimensions. We also map a point $q \in \mathbb{R}^2$ in the plane, to the plane

$$\varphi(q) = \left(z = -q_x x - q_y y + q_x^2 + q_y^2\right).$$

Note that if $\mathcal{d}$ is above $h = \varphi(q)$ then

$$\mathcal{d}_z \geq -q_x \mathcal{d}_x - q_y \mathcal{d}_y + q_x^2 + q_y^2. \iff r^2 - p_x^2 - p_y^2 \geq -2q_x p_x - 2q_y p_y + q_x^2 + q_y^2.$$
$$\iff r^2 \geq \left(p_x - q_x\right)^2 + \left(p_y - q_y\right)^2 \iff q \in \text{disk}(p, r).$$

3.1.2 The algorithm

The above lifting of disks to points (in three dimensions), and points to planes, has the property that a point is above a plane $\iff$ the original disk contains the original point. In particular, if a lifted disk $\mathcal{d} \in \mathbb{R}^3$ is strictly below a plane h, then the original disk does not contain q (i.e., the original point lifted to h).

In the lifted space, the input is a set of n planes in three dimensions. If a plane h is red, then the computed disk must contain the original point, which means that the encoded disk $\mathcal{d}$ must lie above h. Namely, every red point, corresponds to a commitment of the corresponding plane, to the halfspace lying (vertically) above it. Similarly, an original blue point corresponds to a commitment to the downward halfspace. We want to find a point in this space which is feasible (after all the constraints have been committed).

The ULP oracle queries. A labeling oracle query on a plane, corresponds to providing the color of the original point, which can be done using a single NN colored query. A feasibility oracle query, is a point $\mathcal{d}$ in three dimensions, which corresponds to a disk, which asks

whether it contains all the red points, and no blue points. The later can be answered by performing a blue NN query, and a red FN query, and then making a decision according to how the points interact with the query disk.

As such, we can plug the lifted instance into the algorithm of Theorem 14. This algorithm also works verbatim in higher dimensions. We thus get the following.

▶ **Theorem 20.** *Let P be a set of n points in $\mathbb{R}^d$. Assume that there is an underlying coloring of the point set by (say) red and blue, and there is an oracle that can answer nearest-neighbor and furthest-neighbor colored queries on P. The above algorithm computes a ball that contains only the red points, and no blue points, if such a ball exists, using $O(\log^{d-1} n)$ NN/FN oracle queries. The running time of the algorithm is $O_d(n)$.*

3.2 Learning a cover by k monochromatic balls using NN queries

Problem statement. The input is a set of n colored points P, and assume that the points of P can be covered by k balls, such that each ball covers only points of a single color. Here, we assume that we are given access to the points via a NN oracle queries (i.e., no FN queries). The task at hand is to classify the points correctly (i.e., decide their color) using a small number of oracle queries, by an efficient algorithm.

3.2.1 The algorithm

▶ **Lemma 21** (proof in full version [2]). *Let Q be a set of m points in $\mathbb{R}^d$, and let $\mathcal{R}$ be the (infinite) set of all balls in $\mathbb{R}^d$. One can compute, in $O(m^{d+2})$ time, the family of $O(m^{d+1})$ canonical sets induced by $\mathcal{R}$ on Q, that is $\mathcal{F}(Q) = \{\mathbb{b} \cap Q \mid \mathbb{b} \in \mathcal{R}\}$.*

For $i > 0$, let P_i denote the set of unlabeled points at the beginning of the ith iteration and let $n_i = |P_i|$ (i.e., $P_1 = P$, and $n_1 = n$). The algorithm computes a random sample $R_i \subseteq P_i$ of size $O(\mathsf{k} \log \mathsf{k})$, and determines the color of the points in R_i using NN queries. The algorithm then computes the set of canonical sets $\mathcal{F}_i = \mathcal{F}(R_i)$, using Lemma 21. For every range $\mathbf{r} \in \mathcal{F}_i$, such that $\overline{s}_{R_i}(\mathbf{r}) = \dfrac{|\mathbf{r} \cap R_i|}{|R_i|} \geq \dfrac{1}{2\mathsf{k}}$, and such that all the points in $\mathbf{r}$ are of the same color, the algorithm runs a subroutine, described below in Section 3.2.2, to decide if there is a monochromatic ball that contains all the points of $\mathbf{r}$. Formally, the subroutine decides if there is ball $\mathbb{b}$, such that all the points of $P \cap \mathbb{b}_i$ are colored by the same color, and $\mathbf{r} \subseteq \mathbb{b}$. If no such ball is found, the algorithm repeats this iteration until success.

The algorithm sets $\mathbb{b}_i$ to be the ball computed, such that $|\mathbb{b}_i \cap P_i|$ is maximized among all such balls. The algorithm then adds $\mathbb{b}_i$ to the computed cover, assigns all the points in $\mathbb{b}_i \cap P_i$ their color, and sets $P_{i+1} \leftarrow P_i \setminus \mathbb{b}_i$.

The algorithm stops once all points have been assigned their correct color (i.e., $P_i = \emptyset$).

3.2.2 Searching for a monochromatic ball containing a set r

The subroutine is given a set of points $\mathbf{r}$ that are all (say) red. The subroutine is searching for a ball $\mathbb{b}$ such that the point in $P \cap \mathbb{b}$ are all red points, and $\mathbf{r} \subseteq \mathbb{b}$. The subroutine is similar in spirit to the single ball case of Section 3.1, but the details are somewhat different.

Specifically, consider the set B of all blue points in P (this set is not explicitly known, as many of the points color is yet unknown), and consider the problem of computing a ball that contains all the points of $\mathbf{r}$ and none of the points of B. This is an implicit undecided optimization problem, which via the lifting of Section 3.1.1, reduces to implicit undecided LP.

A separation oracle here, in the original settings, is a query ball $\mathcal{q}$. If $\mathcal{q}$ contains a blue point, one can find it by performing a colored nearest-neighbor query on the blue points. Similarly, one can verify that $\mathcal{b}$ contains all the red points of $\mathbf{r}$. Thus, one can use the implicit undecided LP algorithm of Lemma 5 (see Remark 6).

3.2.3 Analysis

Informally, each successful iteration reveals the color of a $\Omega(1/\mathsf{k})$-fraction of the unlabeled points. Specifically, at the ith iteration, at least one of the k balls of the optimal solution must contain at least n_i/k points of P_i, which are all of the same color (and are yet unlabeled). As such, after $O(\mathsf{k} \log n)$ iterations, the algorithm correctly exposes the colors of the points in P.

▶ **Lemma 22** (proof in full version [2]). *Given a set $\mathbf{r} \subseteq P$ of (say) red points, such that there exists a ball $\mathcal{b}$ such that all the points of $\mathcal{b} \cap P$ are of the same color, and $\mathbf{r} \subseteq \mathcal{b}$, the subroutine of Section 3.2.2 returns a monochromatic ball that covers the points of $\mathbf{r}$.*

▶ **Lemma 23** (proof in full version [2]). *Under the assumption that the input can be covered by k monochromatic balls, an iteration of the above algorithm succeeds with probability close to one, and computes a monochromatic ball that covers at least $2/(5\mathsf{k})$ fraction of the uncolored points.*

▶ **Theorem 24** (proof in full version [2]). *Let P be a set of n points $\mathbb{R}^d$, such that there are (unknown) k monochromatic balls that cover all the points of P. Furthermore, assume we are given an oracle that can answer NN colored queries on P. Then, one can compute the color of all the points of P using $O\Big(\mathsf{k}^{d+2} \log^{2d+3} n\Big)$ queries (this bound holds in expectation). The expected running time of the algorithm is $O\Big(\mathsf{k}^{d+2} n \log^{d+1} \mathsf{k}\Big)$.*

References

1. Dana Angluin. Queries and concept learning. *Mach. Learn.*, 2(4):319–342, 1987. doi:10.1007/BF00116828.
2. Stav Ashur and Sariel Har-Peled. On undecided LP, clustering and active learning. *CoRR*, abs/2103.09308, 2021. arXiv:2103.09308.
3. Timothy M. Chan. An optimal randomized algorithm for maximum Tukey depth. In J. Ian Munro, editor, *Proc. 15th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 430–436, Philadelphia, PA, USA, 2004. SIAM. URL: http://dl.acm.org/citation.cfm?id=982792.982853.
4. Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. In *Proc. 35th Int. Annu. Sympos. Comput. Geom.* (SoCG), pages 24:1–24:13, 2019. doi:10.4230/LIPIcs.SoCG.2019.24.
5. Martin E. Dyer, Nimrod Megiddo, and Emo Welzl. Linear programming. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 999–1014. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.pt6.
6. S. Har-Peled. Taking a walk in a planar arrangement. *SIAM J. Comput.*, 30(4):1341–1367, 2000.
7. S. Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Math. Surveys & Monographs*. Amer. Math. Soc., Boston, MA, USA, 2011. doi:10.1090/surv/173.
8. Sariel Har-Peled and Mitchell Jones. Journey to the center of the point set. In Gill Barequet and Yusu Wang, editors, *Proc. 35th Int. Annu. Sympos. Comput. Geom.* (SoCG), volume 129 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.41.

9 Sariel Har-Peled and Mitchell Jones. On separating points by lines. *Discret. Comput. Geom.*, 63(3):705–730, 2020. doi:10.1007/s00454-019-00103-z.

10 Sariel Har-Peled, Mitchell Jones, and Saladi Rahul. Active learning a convex body in low dimensions. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. 47th Int. Colloq. Automata Lang. Prog.* (ICALP), volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 64:1–64:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.64.

11 Sariel Har-Peled, Nirman Kumar, David M. Mount, and Benjamin Raichel. Space exploration via proximity search. *Discrete Comput. Geom.*, 56(2):357–376, 2016. doi:10.1007/s00454-016-9801-7.

12 S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete Comput. Geom.*, 12:291–312, 1994.

13 Daniel M. Kane, Shachar Lovett, Shay Moran, and Jiapeng Zhang. Active classification with comparison queries. In *Proc. 58th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 355–366, 2017. doi:10.1109/FOCS.2017.40.

14 Wolfgang Maass and György Turán. On the complexity of learning from counterexamples and membership queries. In *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 203–210. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89539.

15 Wolfgang Maass and György Turán. Algorithms and lower bounds for on-line learning of geometrical concepts. *Machine Learning*, 14(3):251–269, 1994. doi:10.1007/BF00993976.

16 Wolfgang Maass and György Turán. How fast can a threshold gate learn? In *Proc. Work. Comput. Learning Theory and Nat. Learn. Systems*, pages 381—414, Cambridge, MA, USA, 1994. MIT Press.

17 Burr Settles. Active learning literature survey. Technical Report #1648, Computer Science, Univ. Wisconsin, Madison, 2009. URL: https://minds.wisconsin.edu/bitstream/handle/1793/60660/TR1648.pdf?sequence=1&isAllowed=y.