

---

# Adversarial Examples for $k$ -Nearest Neighbor Classifiers Based on Higher-Order Voronoi Diagrams

---

**Chawin Sitawarin**  
UC Berkeley  
chawins@eecs.berkeley.edu

**Evgenios M. Kornaropoulos**  
George Mason University  
evgenios@gmu.edu

**Dawn Song**  
UC Berkeley  
dawnsong@cs.berkeley.edu

**David Wagner**  
UC Berkeley  
daw@cs.berkeley.edu

## Abstract

Adversarial examples are a widely studied phenomenon in machine learning models. While most of the attention has been focused on neural networks, other practical models also suffer from this issue. In this work, we propose an algorithm for evaluating the adversarial robustness of  $k$ -nearest neighbor classification, i.e., finding a minimum-norm adversarial example. Diverging from previous proposals, we propose the first geometric approach by performing a search that expands outwards from a given input point. On a high level, the search radius expands to the nearby higher-order Voronoi cells until we find a cell that classifies differently from the input point. To scale the algorithm to a large  $k$ , we introduce approximation steps that find perturbation with smaller norm, compared to the baselines, in a variety of datasets. Furthermore, we analyze the structural properties of a dataset where our approach outperforms the competition.

## 1 Introduction

It is well-known that machine learning models can easily be misled by maliciously crafted inputs, called adversarial examples, which are generated by adding a tiny perturbation to test samples [Biggio et al., 2013, Szegedy et al., 2013, Goodfellow et al., 2015]. Adversarial examples are often studied in the context of neural networks, leaving the problem largely unexplored for other classifiers. The  $k$ -nearest neighbor, or simply  $k$ -NN, classifier is a simple yet widely used model in various applications such as data mining, recommendation, and anomaly detection systems where interpretability and simplicity are preferred [Wu et al., 2008]. This non-parametric classifier does not require a training phase and has a well-understood and elegant geometric foundation [Okabe et al., 1992].  $k$ -NN is also an active area of research with lots of developments in popular libraries like Google’s ScaNN [Guo et al., 2020] and Facebook’s FAISS [Johnson et al., 2017]. In recent works [Papernot and McDaniel, 2018, Dubey et al., 2019, Sitawarin and Wagner, 2019b],  $k$ -NN is combined with neural networks to enhance the robustness and the interpretability. Despite the importance of  $k$ -NN, there are only a handful of works that study its robustness [Wang et al., 2018, Yang et al., 2020, Wang et al., 2019, Sitawarin and Wagner, 2019a, 2020].

The first step towards evaluating the robustness of a classifier is to generate an adversarial example that is close to a given test point, under some definition of closeness. In the case of  $k$ -NN where  $k > 1$ , this problem is challenging as the computation involves high-dimensional polytopes that satisfy a set of geometric properties, the so-called Voronoi cells.

In this work, we propose the GeoAdEx algorithm for finding adversarial examples for  $k$ -NN classifiers. GeoAdEx is the first algorithm that exploits the underpinning geometry of the  $k$ -NN classifier. Specifically, our approach performs a principled search on the underlying high-order Voronoi diagram by expanding the search radius around the test point until it finds an adversarial, or simply incorrect, classification. The geometric foundation of GeoAdEx allows us to locate nearby adversarial cells that the other attacks typically miss. GeoAdEx follows the footsteps of the work by Jordan et al. [2019] who exploited the geometry of neural networks for verifying the outputs. Our algorithmic approach stands in sharp contrast to previous attacks for  $k$ -NN [Yang et al., 2020, Wang et al., 2019] which refine the exhaustive approach by heuristic-based filtering.

Furthermore, we introduce optimizations and approximations to the main algorithm, and as a result, the experiments show that GeoAdEx discovers the smallest adversarial distance compared to all of the baselines in the vast majority of our experiments for  $k \in \{3, 5, 7\}$ . GeoAdEx finds up to 25% smaller mean adversarial distance compared to the second best attack. We note that one inherent shortcoming of our geometric approach is the increased computation time, an aspect that can be further improved.

Finally, we present experiments that demonstrate that GeoAdEx performs significantly better when points from different classes are “mixed” together, i.e., there is no clear separation between classes. On a high level, such a setup generates a more intricate spatial tessellation with nearby cells that alternate classes. In this challenging case, GeoAdEx performs up to  $5\times$  better than the baselines.

## 2 Background and Related Work

### 2.1 Background

Let  $X = (x_1, \dots, x_n)$  be a set of points from  $\mathbb{R}^d$ , also called *generators*, and let  $Y = (y_1, \dots, y_n)$  be the class of each point from a set of possible classes  $\{1, \dots, c\}$ . Let  $d(x, x')$  denote a metric between  $x, x' \in \mathbb{R}^d$ . Let  $Z^{(k)}(X)$  be the set of all possible subsets consisting of  $k$  points from  $X$ , i.e.,  $Z^{(k)}(X) = \{L_1^{(k)}, \dots, L_i^{(k)}, \dots, L_l^{(k)}\}$ , where  $L_i^{(k)} = \{x_{i1}, \dots, x_{ik}\}$ ,  $x_{ij} \in X$  and  $l = \binom{n}{k}$ . We define as *order- $k$  Voronoi cell* associated with  $L_i^{(k)}$  the set  $V(L_i^{(k)})$  that includes the points of  $\mathbb{R}^d$  that are closer to  $L_i^{(k)}$  than any other  $k$  points of  $X$ . More formally:

$$V(L_i^{(k)}) = \left\{ p \mid \max_x \{d(p, x) \mid x \in L_i^{(k)}\} \leq \min_{x'} \{d(p, x') \mid x' \in X \setminus L_i^{(k)}\} \right\}$$

With the term *Voronoi facet*, or simply *facet*, we refer to a boundary of an order- $k$  Voronoi cell. We define as bisector  $B\{x_a, x_b\}$  the set of points from  $\mathbb{R}^d$  that are equidistant from  $x_a \in X$  and  $x_b \in X$ . There is a tight connection between bisectors and facets. Every Voronoi facet is part of a bisector, but not every bisector,  $\binom{n}{2}$  in total, includes a facet. We call the bisectors that do include a facet *active* bisectors and those that do not *inactive* bisectors. We also note that an order- $k$  Voronoi cell has at most  $k(n - k)$  facets. The collection of order- $k$  Voronoi cells for all  $L_i^{(k)}$  is called *order- $k$  Voronoi diagram*. In this work, we assume the standard non-cocircularity assumption<sup>1</sup> to avoid degenerate cases. We only consider the Euclidean distance for the  $k$ -NN, i.e.,  $d(x_a, x_b) = \|x_a - x_b\|_2$ .

Up to this point, we have not used the classes of the  $X$ . In this work, we focus on  $k$ -NN classifiers that classify based on the majority (class) vote of the  $k$ -nearest points. We define as *adversarial cell* with respect to  $(x, y)$  any order- $k$  Voronoi cell that has different majority than label  $y$ . We denote with  $A(x)$  the set of all adversarial cells with respect to  $(x, y)$ . With the term *adversarial facet* with respect to  $(x, y)$  we refer to a facet that belongs to a cell from  $A(x)$ .

### 2.2 Related Work

**Adversarial Robustness of  $k$ -NN Classifiers.** Wang et al. [2018] study the robustness property of  $k$ -NN classifiers from a theoretical perspective. The authors show that a  $k$ -NN classifier can be as robust as the optimal Bayes classifier given a sufficiently large number of generators and  $k$ . Wang et al. [2018] and Yang et al. [2020] also propose potential methods for improving the robustness of  $k$ -NN by pruning away some of the “ambiguous” generators. Other works consider different alternatives for

<sup>1</sup>In practice this can be achieved by adding a small random noise on each generator to break potential cocircularity between them.

enhancing the robustness of neural networks by combining them with  $k$ -NN. [Papernot and McDaniel, 2018, Sitawarin and Wagner, 2019b, Dubey et al., 2019]

**Attacks on  $k$ -NN Classifiers.** Sitawarin and Wagner [2019a, 2020] propose a method for finding minimum-norm adversarial examples on  $k$ -NN and deep  $k$ -NN classifiers [Papernot and McDaniel, 2018]. Their method approximates a  $k$ -NN classifier with a soft differentiable function so adversarial examples can be found via a gradient-based optimization algorithm. While the approach is efficient and scalable to large  $k$ , it provides no guarantee for finding the nearest adversarial example and overlooks the geometry of  $k$ -NN.

Yang et al. [2020] and Wang et al. [2019] take a similar approach for finding the adversarial example closest to a given test  $x$ . Their method follows an exhaustive search by computing the distance between  $x$  and all the Voronoi cells that have a different class from  $x$ . Both of them show that this can be done exactly when  $k = 1$  but does not scale well for  $k > 1$ . Consequently, they introduce heuristics to improve the efficiency by choosing only a subset of the Voronoi cells. However, in the process, they lose the optimality guarantee on the resulted adversarial example. More importantly, the heuristics will miss an exponential number of cells as  $k$  increases.

**Attacks Based on Geometric Insights.** The problem of verifying or finding the nearest adversarial examples in neural networks is an active research direction where solutions still do not scale to a satisfactory degree [Huang et al., 2017, Katz et al., 2017, Tjeng et al., 2017, Wong and Kolter, 2018, Weng et al., 2018, Cohen et al., 2019, Jordan et al., 2019]. Among these works, Jordan et al. [2019] proposed an alternative for a ReLU network, using a geometric approach. Specifically, the authors notice that piecewise linear networks partition the input space into numerous polytopes where points of the same polytope are classified with the same label. Armed with this observation, they propose an algorithm that iteratively searches through these polytopes that are further away from the test sample until they find a polytope with a different label.

### 3 The GeoAdEx Algorithm

**Objective.** The goal of the algorithm is to find the smallest perturbation  $\delta^*$  that moves a test point  $(x, y)$  to an adversarial cell. This objective can be expressed as the following optimization problem:

$$\begin{aligned} \delta^* = \arg \min_{\delta} \quad & \|\delta\|_2^2 \\ \text{s.t.} \quad & x + \delta \in A(x) \end{aligned} \quad (1)$$

We define as  $\epsilon^*$  *optimal adversarial distance* where  $\epsilon^* := \|\delta^*\|_2$ . The norm of any other (non-optimal) perturbation  $\delta$  that misclassifies  $(x, y)$ , i.e.,  $x + \delta \in A(x)$ , is simply called *adversarial distance*.

**A First Approach.** The constraint of the above formulation implies that  $x + \delta$  must be a member of an adversarial cell from  $A(x)$ . A simple approach to solve this minimization is to build a series of optimization problems, one for each of the cells in  $A(x)$ , and pick the solution with the minimum adversarial distance. Unfortunately, this would require solving  $O(\binom{n}{k})$  quadratic programs each of which has  $k(n - k)$  constraints. While this complexity may be manageable when  $k = 1$  and  $n$  is small, it does not scale with  $k$ . On a high level, Yang et al. [2020] and Wang et al. [2019] take this approach and additionally develop heuristics to scale to cases with  $k > 1$ .

**The GeoAdEx Approach.** The core idea of GeoAdEx is to perform a principled *geometric exploration* around the test point  $x$ . GeoAdEx processes order- $k$  Voronoi cells à la breadth-first search until it discovers an adversarial cell. Algorithm 1 provides a pseudocode of the main steps of GeoAdEx, and Figure 1 illustrates some steps of GeoAdEx on a 3-NN classifier with two classes. In the following subsections, we explain these steps in detail and describe performance speedups as well as approximation steps to scale the algorithm to  $k > 1$ .

#### 3.1 Data Structures

We maintain a priority queue  $PQ$  as an auxiliary data structure that is defined for a given test point  $(x, y)$ .  $PQ$  tracks the progress made so far towards locating the optimal adversarial distance.

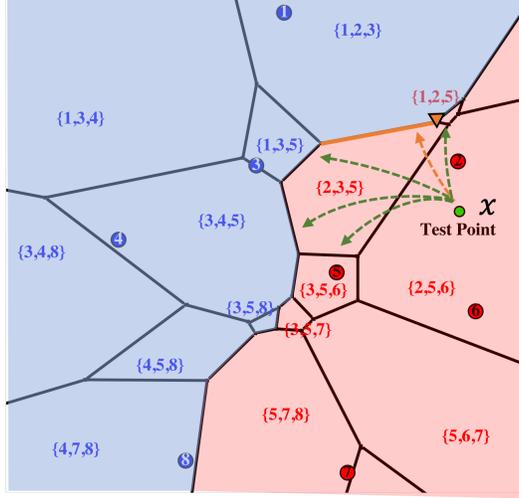


Figure 1: GeoAdEx on a Voronoi diagram for  $k = 3$  in  $\mathbb{R}^2$  with two classes. The color of each numbered generator point shows its class, and the color of each cell indicates its classification outcome. The illustration presents the step where GeoAdEx has processed the facets of  $V(\{2, 5, 6\})$  and is transitioning to the next closest cell  $V(\{2, 3, 5\})$ . The arrows indicate the facets that are inserted to the priority queue  $PQ$ . When GeoAdEx terminates, it outputs the closest adversarial facet (orange line); this transition is indicated with the orange arrow, and the orange triangle is the optimal adversarial example for  $x$ .

---

### Algorithm 1: GeoAdEx Algorithm

---

**Data:** Test point  $(x, y)$

**Result:** Adversarial distance  $\epsilon$

- 1 Initialize the smallest adversarial distance  $\epsilon$  found so far as  $\epsilon \leftarrow \infty$ ;
  - 2 Initialize  $PQ$  by inserting the facets of the order- $k$  Voronoi cell that  $x$  falls into. Mark this cell as *visited*;
  - 3 Call *deleteMin* from  $PQ$  until the returned facet is part of an unvisited order- $k$  Voronoi cell. Initialize  $\Psi$  to this unvisited Voronoi cell;
  - 4 **while**  $\Psi$  is not an adversarial cell **do**
  - 5     Find the set of facets  $\Phi$  that comprise unvisited cell  $\Psi$  (Section 3.2);
  - 6     Compute the distance between query point  $x$  and each of the facets in  $\Phi$  (Section 3.3);
  - 7     If (i) a facet from  $\Phi$  is an adversarial facet, and (ii) the distance to this adversarial facet implies a smaller norm than  $\epsilon$ , then update  $\epsilon$ ;
  - 8     Insert to  $PQ$  the facets in  $\Phi$  with their distance if smaller than  $\epsilon$ ;
  - 9     Call *deleteMin* from  $PQ$  until the returned facet is part of an unvisited order- $k$  Voronoi cell. Update  $\Psi$  to the new unvisited Voronoi cell;
  - 10 **end**
  - 11 If an adversarial facet is removed from  $PQ$  via *deleteMin*, the algorithm return the optimal adversarial distance (see Lemma 1);
  - 12 **return**  $\epsilon$ ;
- 

Specifically,  $PQ$  performs the operations *insert* (resp. *deleteMin*) where the input (resp. output) is a facet of the order- $k$  Voronoi diagram of  $X$ . Every facet in  $PQ$  is accompanied by the distance between the test point  $x$  and the facet. Priority is given to the facet with the *minimum* distance to  $x$ . We also need to mark which Voronoi cells are processed. For that, we use a hash table (Python’s native dictionary object) that has an entry for every order- $k$  Voronoi cell that the algorithm has visited. We refer to Voronoi cells that are part of the dictionary as *visited* and those that are not as *unvisited*.

### 3.2 Find Neighbors of a Voronoi Cell

Line 3 & 9 of Algorithm 1, GeoAdEx discovers an *unvisited* order- $k$  Voronoi cell  $\Psi$ . In Line 5, GeoAdEx needs to find the facets that comprise  $\Psi$ , which is equivalent to finding the Voronoi cells that are *neighboring* with cell  $\Psi$ . We present three options for implementing the discovery of neighbors; the choosing among them depends on the application, the size of the dataset, and the value of  $k$ .

**(A) Neighbors via Order- $k$  Voronoi Diagram.** The exact computation of an order- $k$  Voronoi diagram in high dimensions constitutes a computationally challenging task. The cost of this option amortizes when there is a large number of test points to be processed, and the dataset is fairly small in size and dimension. However, it is highly unlikely to scale when  $k > 1$  which is the goal of our work.

**(B) Neighbors via Enumerating Bisectors.** In this approach, one has to process a quadratic number of bisectors that are potentially *active* with respect to the Voronoi cell  $\Psi$  (see the definition from Section 2). Interestingly, we can filter out this set by only considering the bisectors whose generator, exactly one of the two, also generates cell  $\Psi$ . Consequently, the task of finding the facets of the current cell  $\Psi$  is reduced to testing the *activeness* of  $k(n - k)$  bisectors. Jumping ahead, this step can be accomplished simultaneously with computing distance from  $x$  to each bisector so we defer its analysis for Section 3.3. For comparison, the complexity of this step is  $O(\text{poly}(n, d, k))$  without any speedups, and in the worst case, which is extremely unlikely in practice, it can be called up to  $N$  times where  $N$  is the number of all cells ( $N$  is  $O(\binom{n}{k})$ ).

**(C) Neighbors via Order-1 Voronoi Diagram.** This approach is the middle ground between the first two. That is, it still tests whether a bisector is active, much like (B), but it uses the Voronoi diagram, much like (A), to reduce the number of bisectors to test. Given a Voronoi cell with  $k$  generators, e.g.,  $V(\{x_1, \dots, x_{k-1}, x_k\})$ , we know that its neighboring cells must have a set of generators that differs by exactly one point, e.g.,  $V(\{x_1, \dots, x_{k-1}, x_l\})$ . We draw a connection between the Voronoi diagrams of order 1 and  $k$  and show that the new point  $x_l$  *must be neighboring with at least one of the order-1 cells*  $V(\{x_1\}), V(\{x_2\}), \dots, V(\{x_k\})$ . We formalize this in Theorem 1, and its proof can be found in Appendix D.

**Theorem 1.** *Let  $S = \{x_1, \dots, x_{k-1}\} \subset X$  be a set of  $k - 1$  generators. Let  $x_k, x_l \in X$  be two generators such that  $x_k, x_l \notin S$ . If  $V(S \cup \{x_k\})$  and  $V(S \cup \{x_l\})$  are two neighboring order- $k$  Voronoi cells, then the order-1 Voronoi cell  $V(\{x_l\})$  is neighboring with at least one of the  $V(\{x_1\}), \dots, V(\{x_{k-1}\}), V(\{x_k\})$ .*

With this insight, we can *narrow down significantly* the number of bisectors considered in approach (B). More formally, let  $nb(x')$  denote the set of generators of order-1 Voronoi cells that neighbor with  $V\{x'\}$ . Let  $\Psi$  be the Voronoi cell  $V(\{x_1, \dots, x_k\})$ . By Theorem 1, it is enough to consider the bisectors between  $z \in \{x_1, \dots, x_k\}$  and  $z' \in \left(\bigcup_{i=1}^k nb(x_i)\right) - \{x_1, \dots, x_k\}$ . Hence, the total number of bisectors we need to test is at most  $k \cdot |\bigcup_{i=1}^k nb(x_i)|$ . Assuming that  $\ell := \max_i |nb(x_i)|$ , the number of bisectors cannot exceed  $k^2 \ell$ . Hence, approach (C) is much faster than approach (B) when  $k^2 \ell \ll n$ . In this work, we use approach (B) for the  $k = 1$  case. However, in order to scale to large datasets and  $k > 1$ , we propose some approximations steps that are rooted in approach (C) later in Section 3.5.2.

### 3.3 Computing Distance & Testing Activeness

Now we describe Line 6 of Algorithm 1, i.e. how to compute the distance between a point  $x$  and a facet. Then, expanding on this process, we detail how to test the activeness of bisectors.

**Distance computation.** A Voronoi cell is a polytope that can also be described as an intersection of halfspaces:  $\{z \mid Az \leq b\}$ . A bisector can be described as a hyperplane:  $\{z \mid \langle z, \hat{a} \rangle = \hat{b}\}$ . The corresponding facet needs to both be part of the polytope *and* satisfy the equation of the hyperplane. Thus, the shortest distance from  $x$  to a facet is given by the square root of the optimum of the following quadratic program:

$$\begin{aligned} \min_z \quad & \|z - x\|_2^2 \\ \text{s.t.} \quad & Az \leq b \text{ and } \langle z, \hat{a} \rangle = \hat{b} \end{aligned} \tag{2}$$

This problem can be solved by any off-the-shelf solver, but in Appendix F, we describe a faster method also used by Wang et al. [2019].

**Testing Bisector Activeness.** Assuming that  $\{z \mid Az \leq b\} \neq \emptyset$ , the bisector  $\langle z, \hat{a} \rangle = \hat{b}$  is active *if and only if* Eqn. (2) is feasible since their intersection has to be non-empty. In case this problem

is infeasible, the bisector is inactive. From duality theory, we know that Eqn. (2), i.e. the primal, is feasible if and only if the dual objective is unbounded since the dual problem is always feasible in this case. Checking the unboundedness can be accomplished as we are solving Eqn. (2) in its dual form so we can combine the steps of “distance computation” and “testing activeness” into a single function that either returns the distance or indicates that the bisector is inactive.

### 3.4 Optimality of GeoAdEx

Here, we formally state two lemmas that together prove the output optimality of GeoAdEx. The next lemma is straightforward as it is a direct consequence of Lemma C.2 from Jordan et al. [2019]. For completion, we prove it in Appendix D.

**Lemma 1 (Correctness of GeoAdEx).** *Provided no time limit, GeoAdEx terminates when it finds the optimal adversarial examples or equivalently, one of the solutions of Eqn. (1).*

The next lemma states that if GeoAdEx terminates early, i.e. in case we enforce a time limit for performance purposes (see Section 3.5.2), it can still guarantee a lower bound to  $\epsilon^*$ . Specifically, the distance to  $x$  from the most recent facet deleted from  $PQ$  serves as a lower bound to  $\epsilon^*$ .

**Lemma 2 (Lower bound guarantee).** *If GeoAdEx terminates early, the distance from test point  $x$  to the last deleted facet from  $PQ$  is a lower bound to the optimal adversarial distance  $\epsilon^*$ .*

We note that if GeoAdEx terminates early, then  $\epsilon$  serves as an upper bound of  $\epsilon^*$ . None of the previous approaches on adversarial examples for  $k$ -NN provide both an upper and lower bound guarantee.

### 3.5 Towards Scaling GeoAdEx

For simplicity, in Algorithm 1, we present the main algorithmic steps that need to be performed to compute the optimal adversarial distance using the geometric structure of the problem. In this section we delve into details on how to accelerate the performance of GeoAdEx by *performance optimization* and *approximation* steps.

#### 3.5.1 Optimizing Computation Time

We introduce four performance optimizations that significantly speed up GeoAdEx. We present the two most important ones here and the rest in Appendix E.

**(I) Pruning Distant Facets.** Given *any* adversarial distance, e.g., the intermediate result  $\epsilon$  from Algorithm 1, we know that any facet that is more than  $\epsilon$  afar from  $x$  is not the facet associated with the optimal adversarial distance, i.e.,  $\epsilon$  acts as an upper bound that is refined during the execution. Hence, we can safely filter out these facets, and it is unnecessary to compute their distance to  $x$ , test activeness, or insert them to  $PQ$ . This technique can be used both before and during the facet distance computation (Section 3.3). A benefit of our principled geometric approach is that we can apply geometric arguments to eliminate redundant computation on Voronoi cells that are far away.

**(II) Rethinking the Initialization of  $\epsilon$ .** Recall that Line 1 of Algorithm 1 initializes  $\epsilon$  to  $\infty$ . Given the upgraded role of  $\epsilon$  in the previous paragraph, it is clear that a non-simplistic initialization would filter out more unnecessary computation early on and scale the overall performance. For our experiments, we run Sitawarin and Wagner [2020] to initialize  $\epsilon$  since it yields a reasonable  $\epsilon$  and is faster than the other attacks.

#### 3.5.2 Acceleration via Approximations

As mentioned in Section 3.2, approach (C) for finding the neighbors of an order- $k$  Voronoi cell still requires the knowledge of the first-order cells. This can be obtained by either computing the entire order-1 Voronoi diagram or enumerating all possible facets of order-1 cells. Nonetheless, first-order cells in a high-dimension dataset may have a large number of neighbors, and building a Voronoi diagram can easily become a bottleneck in high-dimensions. In this case, approach (C) is no better than approach (B) of Section 3.2. To scale to large and high-dimension datasets, we introduce some approximations. Here, we propose the *approximate version* of GeoAdEx built upon the relationship between order-1 and order- $k$  neighbors from Theorem 1.

**Description.** To circumvent the expensive (in high dimensions) computation of the order-1 neighbors, we approximate approach (C) of Section 3.2. Specifically, instead of operating on the neighbors of the order-1 cell  $V\{x_i\}$ , we operate on a subset of  $m$  points chosen from the entire set of  $X$  according to a fast heuristic. In other words,  $nb(x_i)$  is (roughly) approximated by a new subset of fixed size  $m$  denoted as  $\widetilde{nb}_m(x_i)$ . For cell  $V\{x_1, \dots, x_k\}$  and a given generator  $x_i$  of this cell, we select from  $X$  the  $m$  closest points to  $x_i$  that are not in  $\{x_1, \dots, x_k\}$ . Mathematically, we define  $\widetilde{nb}_m(x_i)$  as:

$$\widetilde{nb}_m(x_i) = \{x_j \in X \mid d(x_i, x_j) \leq d(x_i, x_{\pi(m)})\} \setminus \{x_1, \dots, x_k\}$$

where  $x_{\pi(m)}$  is the  $m$ -th nearest neighbor of  $x_i$ . With the above heuristic we guarantee that for each cell, we only have to compute the distance (and test activeness) for at most  $k^2m$  facets and, thus, sidestep the computation of the first order Voronoi diagram in high dimensions. Each subsequent optimization problem has at most  $k^2m$  constraints. Finding  $m$  nearest points to a single point  $x_i$  is a well studied problem and can be *approximately* solved very fast [Johnson et al., 2017, Aumüller et al., 2017, Andoni et al., 2018].

**Limitations due to Approximation.** Due to the above approximation, it is possible that some of the true order-1 neighbors are not included in  $\bigcup_{i=1}^k \widetilde{nb}_m(x_i)$ . Hence, some active facets and cells may be missed completely<sup>2</sup>. This leads to two limitations. The first limitation is that we can no longer guarantee the optimality of GeoAdEx through Lemmas 1 and 2. In other words, we cannot conclude whether the adversarial distance returned by the approximate version of GeoAdEx is the optimal adversarial distance (or its lower bound).

The second limitation is that the approximation may affect the correctness of the distance computation. Recall that in the exact version of GeoAdEx, we have shown that it is possible to replace the bisectors in the constraint of the optimization problem in Eqn. (2) with bisectors between  $\{x_1, \dots, x_k\}$  and  $\bigcup_{i=1}^k nb(x_i)$ . But with the approximation, the new feasible set in Eqn. (2) becomes a *superset* of the one in the exact version. Consequently, the steps in Section 3.3 may falsely label an inactive bisector as active or return smaller distance than the true value.

**Addressing Limitations.** The first limitation is inherent to the deployment of heuristics to boost performance; a similar issue appears in Sitawarin and Wagner [2020] as well as the approximate version of Yang et al. [2020] and Wang et al. [2019]. It is difficult to avoid without increasing the runtime significantly. However, the second limitation can be addressed, and we do so by using the full set of bisectors when computing the distance and testing the activeness of adversarial facets like in the exact version. This incurs only a small computational cost. We do not need to do the same for non-adversarial cells since their distance and activeness do not affect the correctness of GeoAdEx.

## 4 Experiments

We compare GeoAdEx to all of the previously proposed attacks on  $k$ -NN classifiers, namely Sitawarin and Wagner [2020], Yang et al. [2020], and Wang et al. [2019]. The attacks are evaluated on seven datasets most of which were used in the experiments by Yang et al. [2020]. Namely, we evaluate the attacks in that following datasets: Australian, Covtype, Diabetes, Fourclass, and fmnist06 (a two-class subset, ‘0’ and ‘6’, of Fashion-MNIST). Additionally, we also evaluate on Gaussian and Letters datasets which have more classes and data points. More details on the datasets and the computational setup are included in Appendix A.

### 4.1 Main Findings

Table 1 compares the proposed GeoAdEx attack against the three baselines with respect to the mean perturbation norm. Interestingly, GeoAdEx *outperforms the other baselines* on all of the seven datasets for  $k = 3, 5$  and on four of them for  $k = 7$ . In the three remaining datasets, Sitawarin and

<sup>2</sup>Depending on the dataset and  $m$ , the chance of this happening may not be high for two reasons: (i) For a large  $k$ ,  $\bigcup_{i=1}^k \widetilde{nb}_m(x_i)$  becomes a large set and is likely to cover most, if not entire,  $\bigcup_{i=1}^k nb(x_i)$ . (ii) Even if we miss some cells as neighbors of a particular order- $k$  cell, they may still be picked up by the other cells.

Table 1: Mean norm of the adversarial perturbations on 100 random test points across datasets (lower is better). We report the numbers averaged over 10 runs with random training and test splits. The error, highlighted in gray, is the 95%-confidence interval. The numbers in parentheses is the ratio of the mean perturbation norm found by GeoAdEx over that of the best baseline. The smallest mean perturbation norm among the attacks for each dataset and each  $k$  is bolded.

$k$	Attacks	Australian	Covtype	Diabetes	Fourclass	Gaussian	Letters	fmnist06
3	S&W	.4242	.1931	.1055	.1079	.0442	.1128	.1554
		$\pm .0201$	$\pm .0148$	$\pm .0068$	$\pm .0039$	$\pm .0028$	$\pm .0049$	$\pm .0121$
	Yang et al.	.4658	.2385	.1392	.1209	.1138	.1370	.1773
		$\pm .0258$	$\pm .0101$	$\pm .0077$	$\pm .0055$	$\pm .0023$	$\pm .0046$	$\pm .0043$
	Wang et al.	.4466	.2134	.1164	.1117	.0825	.1218	.1643
		$\pm .0205$	$\pm .0170$	$\pm .0052$	$\pm .0039$	$\pm .0030$	$\pm .0050$	$\pm .0047$
	<b>GeoAdEx</b>	<b>.3646</b>	<b>.1448</b>	<b>.0786</b>	<b>.1073</b>	<b>.0426</b>	<b>.1091</b>	<b>.1509</b>
		$\pm .0250$	$\pm .0116$	$\pm .0042$	$\pm .0045$	$\pm .0015$	$\pm .0066$	$\pm .0076$
		(.8595)	(.7499)	(.7450)	(.9944)	(.9638)	(.9672)	(.9710)
5	S&W	.4748	.2281	.1215	.1087	.0463	.1134	.1648
		$\pm .0199$	$\pm .0121$	$\pm .0060$	$\pm .0052$	$\pm .0030$	$\pm .0064$	$\pm .0093$
	Yang et al.	.5524	.3047	.1824	.1309	.1776	.1503	.2087
		$\pm .0141$	$\pm .0161$	$\pm .0068$	$\pm .0043$	$\pm .0031$	$\pm .0048$	$\pm .0059$
	Wang et al.	.5110	.2613	.1382	.1127	.1195	.1298	.1877
		$\pm .0147$	$\pm .0110$	$\pm .0056$	$\pm .0054$	$\pm .0047$	$\pm .0055$	$\pm .0088$
	<b>GeoAdEx</b>	<b>.4608</b>	<b>.1856</b>	<b>.1021</b>	<b>.1066</b>	<b>.0401</b>	<b>.1130</b>	<b>.1632</b>
		$\pm .0175$	$\pm .0218$	$\pm .0065$	$\pm .0048$	$\pm .0023$	$\pm .0021$	$\pm .0075$
		(.9705)	(.8137)	(.8403)	(.9981)	(.8661)	(.9965)	(.9903)
7	S&W	<b>.5110</b>	.2528	.1259	.1129	.0463	<b>.1127</b>	<b>.1662</b>
		$\pm .0145$	$\pm .0211$	$\pm .0089$	$\pm .0043$	$\pm .0022$	$\pm .0049$	$\pm .0051$
	Yang et al.	.6010	.3335	.2005	.1403	.2204	.1637	.2328
		$\pm .0140$	$\pm .0245$	$\pm .0097$	$\pm .0053$	$\pm .0021$	$\pm .0035$	$\pm .0057$
	Wang et al.	.5410	.3078	.1631	.1182	.1570	.1395	.2010
		$\pm .0181$	$\pm .0145$	$\pm .0083$	$\pm .0051$	$\pm .0017$	$\pm .0034$	$\pm .0056$
	<b>GeoAdEx</b>	<b>.5134</b>	<b>.2153</b>	<b>.1202</b>	<b>.1109</b>	<b>.0425</b>	<b>.1151</b>	<b>.1692</b>
		$\pm .0149$	$\pm .0174$	$\pm .0049$	$\pm .0019$	$\pm .0033$	$\pm .0065$	$\pm .0055$
		(1.0047)	(.8517)	(.9547)	(.9822)	(.9081)	(1.0213)	(1.0181)

Wagner [2020] performs best and is narrowly better than GeoAdEx. GeoAdEx performs notably well on datasets Gaussian and Covtype with up to 25% smaller perturbation norm than the second best attack. On average, GeoAdEx reduces the perturbation norm by 11%, 8%, and 4% for  $k = 3, 5, 7$ , respectively. Since the results of the exact attacks for  $k = 1$  are not the main focus of this work, they appear in Appendix B.1.

We report that while our attack finds an adversarial distance that is closer to the optimal compared to the baselines, its main limitation is the runtime. In some cases, the runtime of GeoAdEx can be an order of magnitude larger than the second best attack. This is discussed in more detail in Section 4.3.

It is also important to note that each of the proposed attacks (including GeoAdEx) performs better when the underlying data present certain structural properties. As a result, there is no single attack, so far, that performs universally better across all datasets. In Section 4.2, we explore the property of datasets that make our attack superior.

## 4.2 Advantages of a Geometric Search

Intuitively, GeoAdEx is based on a search that expands outwards from the original test point. As a result, it performs significantly better than the baselines when there exists an adversarial cell in relatively close proximity to the given test point. In other words, GeoAdEx performs really well on datasets when the class-conditioned distributions are closer or present significant ‘‘overlap.’’ In this case, GeoAdEx is less likely to miss small adversarial cells that are close to the test point as confirmed by the main experiment. We verify this hypothesis in the following experiment where we control the closeness of classes and study the performance of GeoAdEx compared to the baselines.

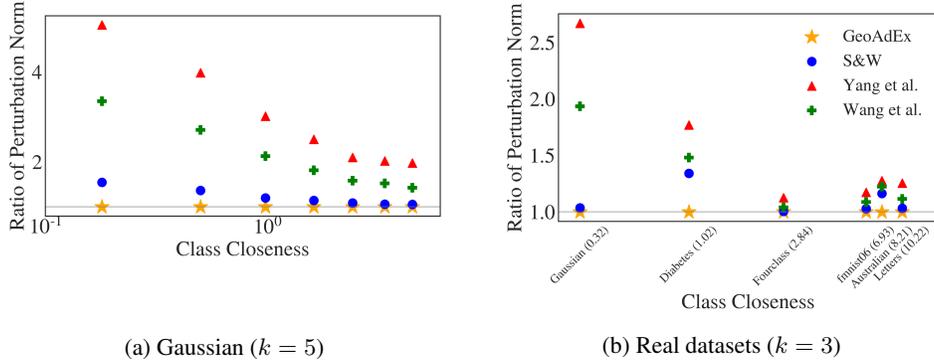


Figure 2: Mean perturbation norm found by the attacks as a function of the class closeness (lower is better).

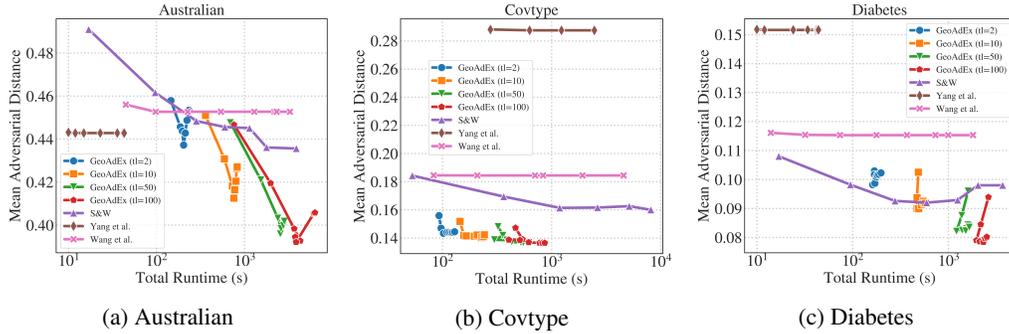


Figure 3: Mean adversarial distance vs. total runtime for GeoAdEx and all the baselines with different choices of hyperparameters on (a) Australian, (b) Covtype, and (c) Diabetes. Each point represents a single run with a unique set of hyperparameters. For a given runtime, an attack with the lowest mean adversarial distance is the best. GeoAdEx utilizes the extra runtime better and outperforms the baselines.

**Experiments with Class Closeness.** Given a dataset with  $c$  class-conditioned data distributions  $D_1, \dots, D_c$ , we can compute the minimum KL-divergence between a pair of distributions  $D_i$  and  $D_j$ :  $\text{KLD}(D_i || D_j)$ . Now we define as *class closeness* of a given data distribution as the average of the above set of minima, i.e.,  $\frac{1}{c} \sum_{i=1}^c \min_j \text{KLD}(D_i || D_j)$ . The smaller the class closeness, the closer their distributions are and, consequently the larger the degree of “mixing” between Voronoi cells of different classes. For each dataset, we record the associated class closeness and the ratio of the mean perturbation norm of each baseline over that of GeoAdEx.

First, we confirm this hypothesis on synthetic datasets where each of the two classes is generated by a Gaussian distributions, i.e.,  $D_1, D_2$ , in  $\mathbb{R}^{20}$ . Note that for Gaussian, KL-divergence can be analytically computed as the closed form exists. In this case, Figure 2a shows that GeoAdEx *significantly outperforms* all the baselines by a larger margin when the class closeness is small. That is, for datasets that are challenging to classify, GeoAdEx outperforms the competition.

Furthermore, we revisit the real datasets from Table 1 and re-interpret the results under the lens of their class closeness. Since the true data distributions of these datasets are unknown, we approximate the KL-divergence via data samples using the estimator in Equation (5) from Wang et al. [2009]. Indeed, Figure 2b shows that the datasets on which GeoAdEx clearly outperforms Yang et al. [2020] and Wang et al. [2019], are the ones with small class closeness (e.g., Diabetes, Gaussian).

### 4.3 Runtime Comparisons

Runtimes of the experiments in Table 1 are reported in Table 4. Through the choice of hyperparameters, we attempt to control the runtime of each algorithm to roughly be within the same order of magnitude, but this has proven difficult given that we do not wish to fine-tune the hyperparameters specifically per dataset. To better compare the algorithms under a similar set of runtimes, we repeat the experiments with different sets of hyperparameters chosen at certain intervals and plot the mean adversarial distance vs. the total runtime for each experiment in Figure 3.

For each algorithm, we started with its default hyperparameters and adjusted them (either linearly or exponentially) in the direction that should find smaller adversarial perturbations with a longer runtime. Each point on the curves represents an experiment with one set of hyperparameters. One way to read the plot is to fix a particular runtime and compare the mean adversarial distance from each line. This experiment is expensive so we only run it for the first three datasets from Table 1.

For Australian and Diabetes, in a regime with very short runtime ( $\sim 100$ s total or  $\sim 1$ s per sample), there is at least one baseline that is both faster and finds smaller adversarial perturbation than GeoAdEx. However, with longer runtimes, our GeoAdEx outperforms all the baselines by a large margin (10%, 15% and 20% improvement over the second best for Australian, Covtype, and Diabetes, respectively). In most settings, the baselines do not benefit much, if at all, from the increased runtime. Conversely, GeoAdEx always finds smaller adversarial distances given a longer time limit. More details and discussions of this experiment are in Appendix B.2.

**Effect of the initialization attack.** The initialization of  $\epsilon$  with Sitawarin and Wagner [2020] attack, as described in the performance optimization (II), also improves the total runtime of GeoAdEx. Depending on the dataset, the runtime reduction range from 2% to 75%. In the interest of space, ablation study and analysis of the initialization attack is included in Appendix B.3.

## 5 Discussion & Open Problems

GeoAdEx has been shown to be successful in discovering adversarial examples for  $k$ -NN classifiers with  $k \geq 1$ . Compared to the baselines, it finds adversarial examples with a considerably smaller perturbation norm on most of the commonly tested datasets.

The main limitation of GeoAdEx is its runtime, particularly in the short-runtime regime. Sitawarin and Wagner [2020] is always the fastest, generally followed by Wang et al. [2019], Yang et al. [2020], and then GeoAdEx. However, in the long-runtime regime, our GeoAdEx outperforms the baselines by a large margin given the same runtime. GeoAdEx utilizes a longer runtime by expanding the search radius while the baselines search more cells without a particular order or prioritization. As a result, the adversarial distance found by Sitawarin and Wagner [2020], Yang et al. [2020], and Wang et al. [2019] typically does not improve much further with more computation.

We note that compared to the geometric approach by Jordan et al. [2019], our experiments were run on a significantly larger scale. Specifically, Jordan et al. [2019] was tested on neural networks with only up to 70 ReLUs, which are much smaller than typical networks used in practice. This is equivalent to Voronoi cells with only 70 neighbors/facets in our setting. GeoAdEx was tested on datasets with over ten thousand generator points and  $k$  up to 7, which results in a substantially larger number of polytopes to search through.

GeoAdEx has shown convincing results with our new geometric take on the problem, but there is room for improvement in terms of efficiency. Additional speedups can be introduced via parallelization, GPU utilization, and a faster optimization technique. A more sophisticated heuristic for determining the order-1 neighbors can also be used. Furthermore, geometric properties of a high-order Voronoi diagram may be better exploited to save unnecessary computation on non-adversarial cells. For instance, multiple neighboring non-adversarial cells may be combined and approximated as a single large cell if we can guarantee that there is no adversarial cell inside this new polytope. This would remove a large number of distance computations surrounding the given test point, which is the main bottleneck of GeoAdEx. Additionally, GeoAdEx can also be extended to other space-partitioning classifiers such as decision trees and random forests.

## 6 Conclusion

We propose GeoAdEx, an algorithm based on geometric insights for finding adversarial examples on  $k$ -NN classifiers. We leverage the structural properties of higher-order Voronoi diagrams to propose efficient approximations and speedup the final algorithm. While GeoAdEx typically requires a higher computational cost, it significantly outperforms the baselines in most of the datasets. Finally, for the case where there is no clear separation between the classes (a typical characteristic of real data) GeoAdEx significantly outperforms the competition.

## Acknowledgement

The first author of this paper was supported by the Hewlett Foundation through the Center for Long-Term Cybersecurity (CLTC) and by generous gifts from Open Philanthropy and Google Cloud Research Credits program with the award GCP19980904.

The second and third authors were supported by the Center for Long-Term Cybersecurity (CLTC), the Berkeley Deep Drive project, NSF grant TWC-1518899, and Open Philanthropy.

## References

- A. Andoni, P. Indyk, and I. Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 7, 2018.
- M. Aumüller, E. Bernhardsson, and A. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *International Conference on Similarity Search and Applications*, pages 34–49. Springer, 2017.
- B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrđić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 387–402, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40994-3.
- S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, USA, 2004. ISBN 0521833787.
- C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.
- A. Dubey, L. van der Maaten, Z. Yalniz, Y. Li, and D. K. Mahajan. Defense against adversarial images using web-scale nearest-neighbor search. *CoRR*, abs/1903.01612, 2019.
- I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.
- R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020.
- X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In R. Majumdar and V. Kunčák, editors, *Computer Aided Verification*, pages 3–29, Cham, 2017. Springer International Publishing.
- J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.
- M. Jordan, J. Lewis, and A. G. Dimakis. Provable certificates for adversarial examples: Fitting a ball in the union of polytopes. In *Advances in Neural Information Processing Systems*, pages 14082–14092, 2019.
- G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017. URL <http://arxiv.org/abs/1702.01135>.
- A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Inc., USA, 1992. ISBN 0471934305.
- N. Papernot and P. D. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *CoRR*, abs/1803.04765, 2018. URL <http://arxiv.org/abs/1803.04765>.
- C. Sitawarin and D. Wagner. On the robustness of deep k-nearest neighbors. volume abs/1903.08333, 2019a. URL <http://arxiv.org/abs/1903.08333>.
- C. Sitawarin and D. Wagner. Defending against adversarial examples with k-nearest neighbor. *arXiv preprint arXiv:1906.09525*, 2019b.
- C. Sitawarin and D. Wagner. Minimum-norm adversarial examples on knn and knn-based models. *ArXiv*, abs/2003.06559, 2020.

- C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL <http://arxiv.org/abs/1312.6199>.
- V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.
- L. Wang, X. Liu, J. Yi, Z.-H. Zhou, and C.-J. Hsieh. Evaluating the robustness of nearest neighbor classifiers: A primal-dual perspective. *arXiv preprint arXiv:1906.03972*, 2019.
- Q. Wang, S. R. Kulkarni, and S. Verdu. Divergence estimation for multidimensional densities via  $k$ -nearest-neighbor distances. *IEEE Transactions on Information Theory*, 55(5):2392–2405, 2009.
- Y. Wang, S. Jha, and K. Chaudhuri. Analyzing the robustness of nearest neighbors to adversarial examples. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5133–5142, Stockholm, Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/wang18c.html>.
- T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.
- E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. volume 80 of *Proceedings of Machine Learning Research*, pages 5286–5295, Stockholm, Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/wong18a.html>.
- X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, Jan. 2008. ISSN 0219-3116. doi: 10.1007/s10115-007-0114-2.
- Y.-Y. Yang, C. Rashtchian, Y. Wang, and K. Chaudhuri. Robustness for non-parametric classification: A generic attack and defense. In *International Conference on Artificial Intelligence and Statistics*, pages 941–951. PMLR, 2020.

# Appendix

Appendix A contains details on the experiments conducted throughout this paper. In Appendix B, we include additional results from the experiments on GeoAdEx, e.g., attacks on  $k$ -NN with  $k = 1$ , runtime comparisons, an ablation study, and different hyperparameter choices. In Appendix C, more details on the class closeness metric are provided. In Appendix D, we provide the proofs of the theory and lemmas stated in the paper. Appendix E explains all the performance optimization used in GeoAdEx, and lastly, in Appendix F, we describe the optimization algorithm, greedy coordinate ascent, used for the distance computation.

## A Details of the Experiments

**Datasets.** Details regarding the datasets used in the experiments are included in Table 2. It also includes the accuracy of  $k$ -NN classifiers at  $k = 1, 3, 5, 7$ . Australian, Covtype, Diabetes, Fourclass, and fmnist06 are taken directly from Yang et al. [2020]’s implementation. The dataset fmnist06 is a two-class subset of Fashion-MNIST with a dimension reduction to 25 via PCA. The Letters dataset, together with the others, is taken from LIBSVM [Chang and Lin, 2011]<sup>3</sup>. Gaussian is a dataset we create by sampling from two isotropic Gaussian distributions of 20 dimension and variance of 1. The distance between the means of the two distributions is 1 by default and is varied only in Section 4.2 to get different values of class closeness.

**Environment and implementation.** All of the attacks are run on an Ubuntu (16.04) cluster with 128 AMD EPYC 7551 CPU cores (2.5GHz each) and 252 GB of memory. No GPU is used in any of the experiments. Using GPU could further speed up the attacks, but the official implementation of the baselines is not compatible with GPUs so we stick to CPUs to present a fair comparison. Yang et al. [2020] uses explicit parallelization and Cython C-extensions, whereas the rest of the attacks use pure Python code without explicit parallelization. The code for the baselines are taken directly from their respective public repository.<sup>4</sup> Yang et al. [2020] uses Gurobi as the solver.

**Hyperparameters.** We evaluate all the baselines using their publicly available code and default hyperparameters. For fairness, we also attempt to tune the hyperparameters for each baseline to keep the total runtime comparable across attacks (see Appendix 4.3). In general, the baselines are fairly insensitive to changes in their hyperparameters. For example, increasing the number of Voronoi cells searched by Yang et al. [2020] and Wang et al. [2019] almost never reduces the mean perturbation norm beyond one obtained with the default value. For GeoAdEx, we choose to compute distance to cell and set  $m$  to 20 and applied a time limit of 100 seconds per test point.

Table 2: Details of the datasets used in the experiments.

Datasets	# points	# features	# classes	$k = 1$ acc	$k = 3$ acc	$k = 5$ acc	$k = 7$ acc
Australian	490	14	2	0.805	0.805	0.830	0.845
Covtype	2000	54	7	0.755	0.715	0.730	0.705
Diabetes	568	8	2	0.695	0.755	0.695	0.685
Fourclass	662	2	2	1.000	0.995	1.000	1.000
Gaussian	10000	20	2	0.550	0.660	0.640	0.635
Letters	15000	16	26	0.925	0.940	0.940	0.930
fmnist06	12000	25	2	0.800	0.795	0.810	0.810

## B Additional Results

### B.1 Exact Attacks for $k = 1$

For completeness, we also compare the exact version of the attacks on 1-NN where the results are presented in Table 3. Note that Sitawarin and Wagner [2020] is excluded since it does not have an

<sup>3</sup><https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html>

<sup>4</sup>Sitawarin and Wagner [2020]: <https://github.com/chawins/knn-defense>, Yang et al. [2020]: <https://github.com/yangarbiter/adversarial-nonparametrics>, Wang et al. [2019]: [https://github.com/wangwllu/knn\\_robustness](https://github.com/wangwllu/knn_robustness)

Table 3: Runtime for the exact version of the attacks on all of the datasets with  $k = 1$ . Sitawarin and Wagner [2020] is not included because it does not offer an exact solution and provide no guarantee on the adversarial distance.

Attacks	Australian	Covtype	Diabetes	Fourclass	Gaussian	Letters	fmnist06
Yang et al. [2020]	72	7186	66	59	9109	34997	18461
Wang et al. [2019]	2	10	2	25	159	78	333
<b>GeoAdEx</b>	17	38	15	39	476	7450	19307

Table 4: Runtime (in seconds) for each of the attacks on all of the seven datasets. The mean adversarial distance corresponding to these runtimes are shown in Table 1. The numbers in the gray rows are 95%-confidence interval from 10 runs with random splits between training and testing samples.

$k$	Attacks	Australian	Covtype	Diabetes	Fourclass	Gaussian	Letters	fmnist06
3	S&W [2020]	654	1225	465	336	811	3372	972
		$\pm 1$	$\pm 250$	$\pm 121$	$\pm 5$	$\pm 14$	$\pm 13$	$\pm 179$
	Yang et al. [2020]	8	925	9	7	599	1555	2151
		$\pm 1$	$\pm 97$	$\pm 1$	$\pm 1$	$\pm 6$	$\pm 38$	$\pm 62$
	Wang et al. [2019]	361	363	128	129	226	259	272
		$\pm 21$	$\pm 20$	$\pm 9$	$\pm 5$	$\pm 19$	$\pm 19$	$\pm 21$
	<b>GeoAdEx</b>	3351	727	1987	1424	2525	4030	6427
		$\pm 447$	$\pm 100$	$\pm 152$	$\pm 106$	$\pm 146$	$\pm 354$	$\pm 338$
5	S&W [2020]	654	1225	465	336	811	3372	972
		$\pm 1$	$\pm 250$	$\pm 121$	$\pm 5$	$\pm 14$	$\pm 13$	$\pm 179$
	Yang et al. [2020]	8	925	9	7	599	1555	2151
		$\pm 1$	$\pm 97$	$\pm 1$	$\pm 1$	$\pm 6$	$\pm 38$	$\pm 62$
	Wang et al. [2019]	361	363	128	129	226	259	272
		$\pm 21$	$\pm 20$	$\pm 9$	$\pm 5$	$\pm 19$	$\pm 19$	$\pm 21$
	<b>GeoAdEx</b>	3351	727	1987	1424	2525	4030	6427
		$\pm 447$	$\pm 100$	$\pm 152$	$\pm 106$	$\pm 146$	$\pm 354$	$\pm 338$
7	S&W [2020]	654	1225	465	336	811	3372	972
		$\pm 1$	$\pm 250$	$\pm 121$	$\pm 5$	$\pm 14$	$\pm 13$	$\pm 179$
	Yang et al. [2020]	8	925	9	7	599	1555	2151
		$\pm 1$	$\pm 97$	$\pm 1$	$\pm 1$	$\pm 6$	$\pm 38$	$\pm 62$
	Wang et al. [2019]	361	363	128	129	226	259	272
		$\pm 21$	$\pm 20$	$\pm 9$	$\pm 5$	$\pm 19$	$\pm 19$	$\pm 21$
	<b>GeoAdEx</b>	3351	727	1987	1424	2525	4030	6427
		$\pm 447$	$\pm 100$	$\pm 152$	$\pm 106$	$\pm 146$	$\pm 354$	$\pm 338$

exact version. Wang et al. [2019] is generally the fastest, and GeoAdEx is faster than Yang et al. [2020], which does not seem to scale well with the number of generators and dimension. This is a direct effect of the solvers of the quadratic programs. Greedy coordinate ascent, used by Wang et al. [2019] and our attack, is much more efficient than a general-purpose commercial solver.

## B.2 Runtime Comparisons and Attack Hyperparameters

Table 4 includes the runtime of all the attacks for  $k = 3, 5, 7$ . As we have mentioned, GeoAdEx with  $m = 20$  and the time limit of 100 seconds takes longer to run compared to the other attacks for most cases. Sitawarin and Wagner [2020] is the fastest, followed by Wang et al. [2019] and Yang et al. [2020]. For the other cases, Yang et al. [2020] has the longest runtime. The reported runtime of GeoAdEx also includes the time used to initialize the adversarial distance  $\epsilon$  found by Sitawarin and Wagner [2020].

As mentioned in Section 4.3, we conduct more thorough sets of experiments on the first three datasets (Australian, Covtype, and Diabetes) to fairly compare the attacks under a similar runtime. To this end, we plot the runtime vs. mean adversarial distance curves for GeoAdEx and all the baselines by varying their hyperparameters in Figure 3.

### B.2.1 Runtime Experiment Setup

For GeoAdEx, we vary  $m$  with four different values of time limit per sample which result in four different curves. For Yang et al. [2020], we progressively doubled the number of regions searched which is the only adjustable hyperparameter. For Wang et al. [2019], we progressively doubled the number of trials (both min and max) and the number of neighbors to consider (until it exceeds the number of all generators). There are four hyperparameters for Sitawarin and Wagner [2020]: `binary_search_steps`, `max_iterations`, `thres_steps`, and `check_adv_steps`. We progressively increased the first two and decreased the last two linearly. For GeoAdEx, we tested a more detailed breakdown by varying both (from 5 to 120) and time limit.

Note that the runtime can be slightly different from what reported in Table 4 since we use a different machine. To make this figure, we run all the experiments (all attacks, datasets, and choices of hyperparameters) on a server with 40 cores of Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz.

### B.2.2 Discussion on Figure 3

The major trends have already been covered in Section 4.3. Here, we discuss other observations and minor trends.

**Increasing  $m$  in GeoAdEx can either increase or decrease the mean adversarial distance.** Each curve for GeoAdEx is generated by increasing  $m$  but fixing the time limit. Increasing  $m$  reduces the chance of missing the adversarial facets (hence the downward trend in the adversarial distance), but it also increases the computation time for each cell which means that there are fewer cells it can search given a fixed time limit (hence the upward trend). This implies that there is an optimal value of  $m$  for a given time limit.

**Increasing  $m$  in GeoAdEx can either increase or decrease the total runtime.** This outcome is seemingly perplexing than the previous one. We explain it for different values of  $m$ , namely the small- $m$  and the large- $m$  regions.

*Small- $m$  region.* When a smaller  $m$  is used with GeoAdEx, fewer first-order neighbors are considered, and thus, the search has a higher chance of missing facets and nearby adversarial cells completely. As a result, it has to expand the search radius which, in turn, would discover adversarial examples that are further away and use a longer runtime. Conversely, if we increase  $m$ , we may find these previously missed cells and terminate earlier, resulting in both lower adversarial distance and runtime. We call the first scenario from the above exposition, the “small- $m$  region.”

*Large- $m$  region.* On the other hand, when  $m$  is sufficiently large and no adversarial cell is missed, increasing  $m$  could have a reversed effect. In particular, when  $m$  increases, more first-order neighbors have to be considered, and hence more *nearby* cells will have to be searched. For each of the test samples, this could lead to (i) an increased runtime and/or (ii) the previously found adversarial cells that are further away may now be missed instead. When (ii) happens, GeoAdEx will timeout and just return the initialized upper bound. Therefore, in the “large- $m$  region,” both the adversarial distance and the total runtime may increase with  $m$ .

We verify this hypothesis by inspecting the number of samples that are timed out by GeoAdEx. If our hypothesis holds, when we test different values of  $m$ , we expect to see a decreasing trend on the number of timeouts in the small- $m$  region and an increasing trend in the large- $m$  region. Specifically, when varying  $m \in \{5, 10, 20, 40, 60, 80, 100, 120\}$  on the Diabetes dataset, we observe the following number of timeouts over 100 samples: 41, 21, 14, 16, 10, 15, 16, and 17. The first three experiments ( $m = 5, 10, 20$ ) which correspond to the small- $m$  region show a decreasing number of timeouts from 41 to 14. The last three experiments ( $m = 80, 100, 120$ ) correspond to the large- $m$  region where both runtime and distance increase with  $m$ . The same phenomenon also happens on Covtype given the same hyperparameters and the range of values of  $m$ . In this case, the numbers of timeouts are 24, 9, 3, 3, 4, 4, and 5, respectively.

Note that whether a value of  $m$  is considered “small” or “large” varies by datasets and the time limit. Increasing the time limit reduces the number of timeouts and hence delays the large- $m$  region (i.e., occurs at a larger  $m$ ). Additionally, this observation leads to two practical suggestions: (1) regardless of  $m$ , increasing the time limit is always beneficial in terms of the adversarial distance, but (2) for a fixed time limit, there is an optimal value of  $m$ .

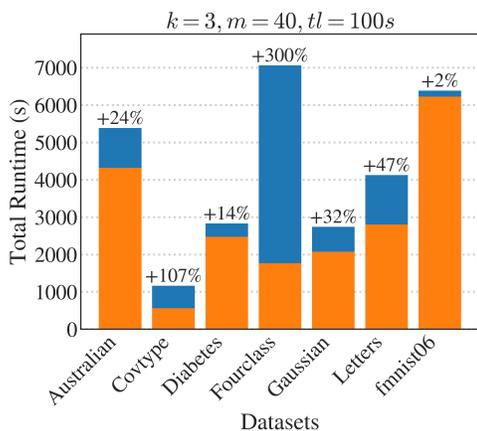


Figure 4: Improvement in the total runtime of GeoAdEx with (orange) and without (blue) Sitawarin and Wagner [2020] initialization.

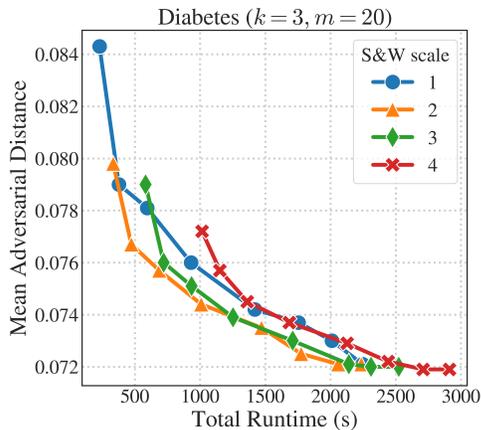


Figure 5: Mean adversarial distance vs. total runtimes on GeoAdEx using S&W initialization with four different hyperparameter scaling (corresponding to the four curves). Each point on the curve represents a unique choice of the time limit per sample in GeoAdEx (from 5 to 200 seconds).

**Curves for Yang et al. [2020] are shorter than the others.** For Australian and Diabetes, the lines associated with Yang et al. [2020] are shorter than the rest because we cannot increase the total runtime by adjusting the hyperparameter any further. This is a fundamental flaw of the heuristic used by Yang et al. [2020] which only searches the cells that contain any generator from a wrong class. So the total number of cells searched is upper bound by the number of generators from a wrong class which is very limited compared to the total number of cells.

### B.3 Ablation Study

#### B.3.1 Importance of S&W Initialization

We want to compare GeoAdEx with and without the S&W initialization. Figure 4 compares the total runtime of GeoAdEx on all of the datasets with  $k = 3$  and shows that the S&W initialization speeds up the attack in all cases. While effects of the initialization are generally minor, we found two cases, namely Covtype and Fourclass, where the initialization leads to a large improvement in the runtime. Without the initialization, the runtime increases by 107% and 300% in these two datasets respectively. We give our explanation for these two datasets below.

For Covtype, GeoAdEx’s runtimes are 561s with the initialization step and 1163s without. Without an initial upper bound on  $\epsilon$ , there is a small chance that GeoAdEx misses a nearby adversarial cell and keeps running until the time limit is met. While only several test samples are affected, it ends up raising the total runtime by a relatively large margin (100% from 500s) because the time limit is set to 100s per sample. The initialization has the same kind of effect on Fourclass as on Covtype, but it affects a much larger number of samples. The number of timeouts goes from 4 to 43 as the initialization is removed, explaining the significant increase in the total runtime. One hypothesis is that this phenomenon takes place because S&W attack is particularly effective on Fourclass (note its competitive performance compared to the other attacks), and thus removing it as an initialization results in an equally large degradation on the performance of GeoAdEx.

#### B.3.2 Effects of S&W Initialization’s Hyperparameters

To test the effect of the initialization’s hyperparameters on GeoAdEx, we ran a simple experiment on the Diabetes dataset by varying the runtime of S&W initialization and the time limit of GeoAdEx. The trade-off curves between total runtime and the adversarial distance are shown in Figure 5. Each curve uses a different scaling factor of the S&W attack as described in our previous experiment to vary its runtime. Each dot in the curve is generated by varying the time limit of GeoAdEx from 5 to 200 seconds. For a wide range of total runtimes, S&W initialization with a scaling factor between 2 and 3 (or anywhere between 4% and 35% of the total runtime) seems like an optimal spot. This

suggests that performance of GeoAdEx is fairly insensitive to the hyperparameters of the S&W initialization. Based on this observation alone, one may aim to run the initialization for  $\sim 10\text{--}20\%$  of the total runtime as the simplest baseline and avoid any additional overhead.

We simply choose S&W attack because it finds an adversarial example quickly (even though the upper bound is relatively loose) with little impact on the total runtime. The choice is somewhat arbitrary and can be replaced with the attacks by Yang et al. [2020] or Wang et al. [2019]. Deciding how long to run S&W initialization is a good question from a practical standpoint. It also depends on many factors including the dataset, the desired time limit, and hyperparameters of GeoAdEx. So it should be considered on a case-by-case basis, perhaps by a hyperparameter optimization on a small subset of the data. A cheaper strategy that works across all datasets is to terminate the initialization attack (per-sample) when the adversarial distance starts to plateau (e.g, less than  $A\%$  improvement in the last  $B$  iterations).

## C Class Closeness

Intuitively, the *class closeness* measures distance between one distribution to another that has a different class and is closest to it. We only consider the closest class because when generating an adversarial example, one only has to perturb the test point towards the nearest distribution with a different class and the other classes are almost irrelevant. More formally, we can write the class closeness as

$$\text{class closeness} := \frac{1}{c} \sum_{i=1}^c \min_{j \in \{1, \dots, c\}} \text{KLD}(D_i || D_j)$$

where  $c$  is the number of classes, and  $D_i$  is the distribution conditioned on class  $i$ .

We first experiment with the Gaussian dataset because its KL-divergence has a analytical form. Specifically, the KL-divergence between two multivariate Gaussian distributions in  $\mathbb{R}^d$ ,  $D_1 = \mathcal{N}(\mu_1, \Sigma_1)$  and  $D_2 = \mathcal{N}(\mu_2, \Sigma_2)$ , is given by

$$\text{KLD}(D_1 || D_2) = \frac{1}{2} \left[ \log \frac{|\Sigma_2|}{|\Sigma_1|} - d + \text{tr}(\Sigma_2^{-1} \Sigma_1) + (\mu_2 - \mu_1)^\top \Sigma_2^{-1} (\mu_2 - \mu_1) \right]$$

In particular, we use isotropic Gaussian distributions so the means and the covariance matrices can be simplified even further.

$$\mu_1 = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mu_2 = \begin{bmatrix} -\alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \Sigma_1 = \Sigma_2 = I_d$$

Note that we pick  $d = 20$ , and without loss of generality, we can simply assign different values of  $\alpha$  and  $-\alpha$  to the first coordinate of  $\mu_1$  and  $\mu_2$  to vary the distance between the two means. We pick  $\alpha$  among  $\{0.3, 0.5, 0.7, 0.9, 1.1, 1.3, 1.5\}$ . This specific case yields a very simple form of KL-divergence:

$$\text{KLD}(D_1 || D_2) = 2\alpha^2$$

For the second part, since the distributions of the other datasets are unknown, we use a non-parametric method from Wang et al. [2009] to approximate the KL-divergence. This method only requires samples from the distributions and is coincidentally based on  $k$ -NN. We pick  $k = 5$  for this approximation method which has nothing to do with the value of  $k$  in  $k$ -NN classifiers we experiment with.

## D Proofs

### D.1 Theorem 1

Now we restate Theorem 1 and then the proof.

**Theorem 1.** Let  $S = \{x_1, \dots, x_{k-1}\} \subset X$  be a set of  $k - 1$  generators. Let  $x_k, x_l \in X$  be two generators such that  $x_k, x_l \notin S$ . If  $V(S \cup \{x_k\})$  and  $V(S \cup \{x_l\})$  are two neighboring order- $k$  Voronoi cells, then the order-1 Voronoi cell  $V(\{x_l\})$  is neighboring with at least one of the  $V(\{x_1\}), \dots, V(\{x_{k-1}\}), V(\{x_k\})$ .

*Proof.* Let  $V(x'|G)$  denote the order-1 Voronoi cell for  $x'$  on the set of generators  $G$ . From property OK1 in Section 3.2.1 of Okabe et al. [1992] we know that the order- $k$  Voronoi cell  $V(S \cup \{x_i\})$  can be expressed as:

$$V(S \cup \{x_i\}) = \left( \bigcap_{l=1}^{k-1} V(x_l | (X \setminus (S \cup \{x_i\})) \cup \{x_l\}) \right) \cap V(x_i | X \setminus S)$$

From the fact that  $V(S \cup \{x_i\})$  is a order- $k$  Voronoi cell, we know that  $V(S \cup \{x_i\})$  is nonempty. Let us assume for the sake of contradiction that  $V(\{x_i\})$  is not neighboring with any of the  $V(\{x_1\}), \dots, V(\{x_{k-1}\})$ . Then we know that:

$$V(\{x_i\}) = V(x_i | X \setminus S)$$

This is because the removal of  $S$  from the set of generators of the Voronoi diagram did not affect  $V(\{x_i\})$  since  $V(\{x_i\})$  is not neighboring with any of  $V(\{x_1\}), \dots, V(\{x_{k-1}\})$ . Additionally, the removal of  $x_i$  from the set of generators in the term  $V(x_l | x_l \cup (X \setminus S \cup \{x_i\}))$  is not affecting the corresponding cell, again, because  $V(\{x_i\})$  is not neighboring with  $V(\{x_1\}), \dots, V(\{x_{k-1}\})$ . This implies that:

$$V(x_l | (X \setminus (S \cup \{x_i\})) \cup \{x_l\}) = V(x_l | (X \setminus S) \cup \{x_l\})$$

Using the above observations we can rewrite the first relation as:

$$V(S \cup \{x_i\}) = \left( \bigcap_{l=1}^{k-1} V(x_l | (X \setminus S) \cup \{x_l\}) \right) \cap V(\{x_i\})$$

For the last part of the proof we will show that the above intersection is empty which contradicts the fact that  $V(S \cup \{x_i\})$  is a nonempty Voronoi cell. Notice that the changes in the set of generators that take place in the term  $V(x_l | (X \setminus S) \cup \{x_l\})$  for  $l = [1, k - 1]$  do not affect the Voronoi cell of  $x_i$ . This means that even after the changes in the set of generators, the cell of  $x_i$  is a superset of  $V(\{x_i\})$ , or to put it differently, the polytope  $V(x_l | (X \setminus S) \cup \{x_l\})$  never enters the area of  $V(\{x_i\})$ . As a result, none of these terms intersects with  $V(\{x_i\})$ . But this contradict the fact that  $V(S \cup \{x_i\})$  is nonempty. □

## D.2 Lemma 1: Correctness of GeoAdEx

**Lemma 1 (Correctness of GeoAdEx).** *Provided no time limit, GeoAdEx terminates when it finds the optimal adversarial examples or equivalently, one of the solutions of Eqn. (1).*

Lemma 1 can be obtained directly from a similar theorem in Jordan et al. [2019]. We first restate this theorem and the definition of *polyhedral complex*. Then, we provide a short proof.

**Theorem 2 (Correctness of GeoCert).** *(Restate from Theorem C.2 in Jordan et al. [2019]) For a fixed polyhedral complex  $\mathcal{P}$ , a fixed input point  $x_0$  and a potential function  $\phi$  that is ray-monotonic, GeoCert returns a boundary facet with minimal potential  $\Phi$ .*

**Definition 1 (Polyhedral Complex).** *(Restate from Definition 2 in Jordan et al. [2019]) A nonconvex polytope, described as the union of elements of the set  $\mathcal{P} = \mathcal{P}_1, \dots, \mathcal{P}_k$  forms a polyhedral complex if, for every  $\mathcal{P}_i, \mathcal{P}_j \in \mathcal{P}$  with nonempty intersection,  $\mathcal{P}_i \cap \mathcal{P}_j$  is a face of both  $\mathcal{P}_i$  and  $\mathcal{P}_j$ .*

*Proof.* In order to apply Theorem 2 to GeoAdEx, we need to show two things: (i) the test point lies in a polyhedral complex, and (ii) Euclidean distance is a ray-monotonic potential function. First, it is trivial to see that the set of non-adversarial Voronoi cells connected to and including the cell the

test input  $x$  falls into forms a polyhedral complex. Since Voronoi cells are polytopes and any pair of them intersect at most at the shared facet, any set of Voronoi cells forms a polyhedral complex. This (informally) proves part (i).

For part (ii), we refer the readers to Corollary C.3 of Jordan et al. [2019] which shows that Euclidean distance is a ray-monotonic potential function. With these two conditions in mind, GeoAdEx behaves in the same way as GeoCert algorithmically in their respective settings, and so Theorem 2 directly applies to GeoAdEx as well.  $\square$

### D.3 Lemma 2: Lower Bound of the Optimal Adversarial Distance

**Lemma 2 (Lower bound guarantee).** *If GeoAdEx terminates early, the distance from test point  $x$  to the last deleted facet from  $PQ$  is a lower bound to the optimal adversarial distance  $\epsilon^*$ .*

**Theorem 3.** *(Restate from Lemma C.1 in Jordan et al. [2019]) For any polyhedral complex  $\mathcal{P}$  point  $x_0$ , and ray-monotonic potential  $\phi$ , let  $\mathcal{F}_i$  be the facet popped at the  $i$ -th iteration of GeoCert. Then for all  $i < j$ ,  $\Phi(\mathcal{F}_i) < \Phi(\mathcal{F}_j)$ .*

*Proof.* From Lemma 1, the first adversarial facet deleted from  $PQ$  is the nearest one to  $x$ , and if that happens, GeoAdEx terminates. It is implied by Theorem 3 that the facets are always deleted from  $PQ$  in an ascending order of their distance to  $x$ . Combining these two facts, we can conclude that the distance of any facets deleted before the adversarial one is always smaller than  $\epsilon^*$ .  $\square$

## E GeoAdEx Performance Optimization

We introduce a total of four performance optimizations to speed up the computation of GeoAdEx. We have explained the first two in Section 3.5 and will describe all of them here in more details.

### E.1 Pruning Distant Facets

This was described in the main text. So here, we only provide examples of where the pruning can occur to remove unnecessary facet computation. First, before computing the distance between  $x$  and a facet, as proposed in Section 3.3, we can first use  $\epsilon$  to filter unnecessary computation. Specifically, we can compute the orthogonal projection of  $x$  onto the bisector implied by the facet. If the distance to the bisector, which is a lower bound on the distance to the facet, is larger than  $\epsilon$ , then this facet can be safely discarded.

Even if we proceed with the distance computation, we can still use  $\epsilon$  to terminate the optimization in Eqn. (2) early. Specifically, if the dual objective of Eqn. (2) surpasses  $\epsilon^2$ , we can terminate the solver and discard this facet since, from strong duality, the dual objective is a lower bound of the primal objective, i.e. the (squared) distance between  $x$  and this facet.

### E.2 Rethinking the Initialization of $\epsilon$

Recall that Line 1 of Algorithm 1 initializes  $\epsilon$  to  $\infty$ . Given the upgraded role of  $\epsilon$  in the previous paragraph, it is clear that a non-simplistic initialization would filter out more unnecessary computation early on and, thus, scale the overall performance. A natural choice is to pick one of the baseline attacks for this purpose. The closer this adversarial distance is to the optimal one, the more computation we are likely to save by the first performance optimization, (I) Pruning Distant Facets.

However, there is a trade-off between the tightness of the estimates and its computation time. Using an expensive attack to initialize  $\epsilon$  can be a huge overhead that increases the total runtime rather than reduces it. For our experiments, we run Sitawarin and Wagner [2020] to initialize  $\epsilon$  since it yields a reasonable and is significantly faster than Yang et al. [2020] and Wang et al. [2019].

### E.3 Exploiting the Sparsity of Solutions

Solving a typical quadratic program has a complexity of  $O(\text{poly}(n, d, k))$ , but fortunately, this problem can be solved very efficiently in its dual form. Wang et al. [2019] show that solving Eqn. (2) via greedy coordinate ascent (GCA) is much faster than using a standard off-the-shelf solver as

it is able to exploit the sparsity in the solution. More details about this speedup can be found in Appendix F.

#### E.4 Setting a Time Limit

To ensure that GeoAdEx terminates in a reasonable time even when no adversarial facet has been deleted from  $PQ$ , we set a time limit as a termination criterion. In this case, lower and upper bounds of  $\epsilon^*$  are returned instead. Note that the approximate version of Yang et al. [2020] and Wang et al. [2019] also terminates early by setting the maximum number of adversarial cells to search through instead of a limit on the runtime.

## F Distance Computation with Greedy Coordinate Ascent

We first restate the distance computation from Eqn. (2):

$$\begin{aligned} \min_z \quad & \|z - x\|_2^2 \\ \text{s.t.} \quad & Az \leq b \end{aligned}$$

Note that without loss of generality, the equality constraint  $\langle z, \hat{a} \rangle = \hat{b}$  can be subsumed by the inequality. Now we provide the dual form of Eqn. (2):

$$\begin{aligned} \max_{\lambda} \quad & g(\lambda) := -\frac{1}{2}\lambda^\top AA^\top \lambda + \lambda^\top (Ax - b) \\ \text{s.t.} \quad & \lambda \geq 0 \end{aligned}$$

In the case that the primal and the dual problems are feasible, we know that strong duality holds because the objective is convex quadratic, and the constraints are affine [Boyd and Vandenberghe, 2004]. Thus, by setting the derivative of the Lagrangian to zero, we have that  $z^* = x + A^\top \lambda^*$ .

According to the complementary slackness from the KKT conditions, we know that  $\lambda_i^* \neq 0$  if and only if  $\langle a_i, z^* \rangle = b_i$ , which geometrically corresponds to  $z^*$  lying on the  $i$ -th bisector associated with  $a_i$  and  $b_i$ . Intuitively, it is unlikely that  $z^*$  lies on an intersection of many bisectors. Hence, there should be very few indices  $i$  such that  $\lambda_i^* \neq 0$ . This is the condition that makes solving the dual problem with greedy coordinate ascent very fast [Wang et al., 2019].

Greedy coordinate ascent (or descent) optimizes the variable only one coordinate per iteration, and there are multiple rules for choosing the coordinate at each iteration. Here, we follow Wang et al. [2019] and simply pick the  $i$ -th coordinate of  $\lambda$  such that its projected gradient is the largest. We describe greedy coordinate ascent in Algorithm 2. To avoid the full gradient computation at every iteration, we keep track and update it given that  $\lambda$  only changes by one coordinate.

---

#### Algorithm 2: Greedy Coordinate Ascent

---

**Data:** Test point  $(x, y)$ , Voronoi cell described by  $Az \leq b$

**Result:** Projection of  $x$  onto the Voronoi cell

```

1 Initialize  $\lambda \leftarrow \mathbf{0}$ 
2 for  $t \in \{1, \dots, T\}$  do
3    $\nabla g(\lambda) \leftarrow -AA^\top \lambda + Ax - b$ 
4    $j \leftarrow \arg \max_i |(\max\{\lambda + \nabla g(\lambda), 0\} - \lambda)_i|$ 
5    $\lambda_j \leftarrow \max \left\{ \lambda_j + \frac{\nabla g(\lambda)_j}{\|a_j\|_2^2}, 0 \right\}$ 
6 end
7 return  $z = x + A^\top \lambda;$ 

```

---

#### F.1 Details on Bisector Activeness Testing

We do a total of three checks to determine the feasibility: (i) check if the dual objective converges fast. When unbounded, the dual objective diverges or keeps increasing with a constant rate or faster.

Additionally, we test whether the KKT conditions hold at the end of the optimization. Namely, (ii) check if the primal residual is zero, and (iii) check if the complementary slackness is satisfied. When all three checks pass, we conclude that the bisector is active. Otherwise, it is considered inactive, and there is no need to finish the distance computation or insert it to  $PQ$ .