8 Near-Optimal Evasion of Classifiers

In this chapter, we explore a theoretical model for quantifying the difficulty of *Exploratory* attacks against a trained classifier. Unlike the previous work, since the classifier has already been trained, the adversary can no longer exploit vulnerabilities in the learning algorithm to mistrain the classifier as we demonstrated in the first part of this book. Instead, the adversary must exploit vulnerabilities that the classifier accidentally acquired from training on benign data (or at least data not controlled by the adversary in question). Most nontrivial classification tasks will lead to some form of vulnerability in the classifier. All known detection techniques are susceptible to blind spots (i.e., classes of miscreant activity that fail to be detected), but simply knowing that they exist is insufficient. The principal question is how difficult it is for an adversary to discover a blind spot that is most advantageous for the adversary. In this chapter, we explore a framework for quantifying how difficult it is for the adversary to search for this type of vulnerability in a classifier.

At first, it may appear that the ultimate goal of these *Exploratory* attacks is to reverse engineer the learned parameters, internal state, or the entire boundary of a classifier to discover its blind spots. However, in this work, we adopt a more refined strategy; we demonstrate successful *Exploratory* attacks that only *partially* reverse engineer the classifier. Our techniques find blind spots using only a *small* number of queries and yield near-optimal strategies for the adversary. They discover data points that the classifier will classify as benign and that are close to the adversary's desired attack instance.

While learning algorithms allow the detection algorithm to adapt over time, real-world constraints on the learning algorithm typically allow an adversary to programmatically find blind spots in the classifier. We consider how an adversary can systematically discover blind spots by querying the filter to find a low-cost (for some cost function) instance that evades the filter. Consider, for example, a spammer who wishes to minimally modify a spam message so it is not classified as spam (here cost is a measure of how much the spam must be modified). By observing the responses of the spam detector,[1] the spammer can search for a modification while using few queries. The design of an exploit that must avoid intrusion detection systems can also be cast into this setting (here cost may be a measure of the exploit's severity).

[1] There are a variety of domain-specific mechanisms an adversary can use to observe the classifier's response to a query; e.g., the spam filter of a public email system can be observed by creating a *test* account on that system and sending the queries to that account. In this chapter, we assume the filter is able to be queried.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

The problem of near-optimal evasion (i.e., finding a low-cost negative instance with few queries) was introduced by Lowd & Meek (2005*a*). We continue studying this problem by generalizing it to the family of convex-inducing classifiers—classifiers that partition their feature space into two sets, one of which is convex. The family of convex-inducing classifiers is a particularly important and natural set of classifiers to examine that includes the family of linear classifiers studied by Lowd & Meek, as well as anomaly detection classifiers using bounded PCA (Lakhina et al. 2004*b*), anomaly detection algorithms that use hypersphere boundaries (Bishop 2006), one-class classifiers that predict anomalies by thresholding the log-likelihood of a log-concave (or unimodal) density function, and quadratic classifiers with a decision function of the form $\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c \geq 0$ if $\mathbf{A}$ is semidefinite (cf. Boyd & Vandenberghe 2004, Chapter 3), to name a few. The family of convex-inducing classifiers also includes more complicated bodies such as the countable intersection of halfspaces, cones, or balls.

We further show that near-optimal evasion does not require complete reverse engineering of the classifier's internal state or decision boundary, but instead only partial knowledge about its general structure. The algorithm of Lowd & Meek (2005*a*) for evading linear classifiers in a continuous domain reverse engineers the decision boundary by estimating the parameters of their separating hyperplane. The algorithms we present for evading convex-inducing classifiers do not require fully estimating the classifier's boundary (which is hard in the case of general convex bodies; see Rademacher & Goyal 2009) or its parameters (internal state). Instead, these algorithms directly search for a minimal cost-evading instance. These search algorithms require only polynomially many queries, with one algorithm solving the linear case with better query complexity than the previously published reverse-engineering technique. Finally, we also extend near-optimal evasion to general ℓ_p costs. We show that the algorithms for ℓ_1 costs can also be extended to near-optimal evasion on ℓ_p costs, but are generally not efficient. However, in the cases when these algorithms are not efficient, we show that there is no efficient query-based algorithm.

The rest of this chapter is organized as follows. We first present an overview of the prior work most closely related to the near-optimal evasion problem in the remainder of this section (see Chapter 3 for additional related work). In Section 8.1, we formalize the near-optimal evasion problem and review Lowd & Meek (2005*a*) definitions and results. We present algorithms for evasion that are near-optimal under weighted ℓ_1 costs in Section 8.2, and we provide results for minimizing general ℓ_p costs in Section 8.3.

This chapter builds on (Nelson, Rubinstein, Huang, Joseph, Lau, Lee, Rao, Tran, & Tygar 2010; Nelson, Rubinstein, Huang, Joseph, & Tygar 2010; and Nelson, Rubinstein, Huang, Joseph, Lee, Rao, & Tygar 2012).

Related Work

Lowd & Meek (2005*a*) first explored near-optimal evasion and developed a method that reverse engineered linear classifiers in a continuous domain, as is discussed in Sections 3.4.2.4 and 3.4.4. The theory we present here generalizes that result and provides three significant improvements:

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

- This analysis considers a more general family of classifiers: the family of convex-inducing classifiers that partition the space of instances into two sets, one of which is convex. This family subsumes the family of linear classifiers considered by Lowd & Meek.
- The approach we present does not fully estimate the classifier's decision boundary, which is generally hard for arbitrary convex bodies (Rademacher & Goyal 2009) or reverse engineer the classifier's state. Instead, the algorithms search directly for an instance that the classifier labels as negative and is close to the desired attack instance; i.e., an evading instance of near-minimal cost. Lowd & Meek previously demonstrated a direct search technique for linear classifiers in Boolean spaces, but it is not applicable to the classifiers we consider.
- Despite being able to evade a more general family of classifiers, these algorithms still only use a limited number of queries: they require only polynomially many queries in the dimension of the feature space and the desired accuracy of the approximation. Moreover, the K-step MultiLineSearch (Algorithm 8.3) solves the linear case with asymptotically fewer queries than the previously published reverse-engineering technique for this case.

Further, as summarized in Section 3.4.2.4, Dalvi et al., Brückner & Scheffer, and Kantarcioglu et al. studied cost-sensitive game-theoretic approaches to preemptively patch a classifier's blind spots and developed techniques for computing an equilibrium for their games. This prior work is complementary to query-based evasion problems; the near-optimal evasion problem studies how an adversary can use queries to find blind spots of a classifier that is unknown but is able to be queried, whereas their game-theoretic approaches assume the adversary knows the classifier and can optimize their evasion accordingly at each step of an iterated game. Thus, the near-optimal evasion setting studies how difficult it is for an adversary to optimize its evasion strategy only by querying, and cost-sensitive game-theoretic learning studies how the adversary and learner can optimally play and adapt in the evasion game given knowledge of each other: These are two separate aspects of evasion.

A number of authors also have studied evading sequence-based IDSs as discussed in Section 3.4.2.2 (cf. Tan et al. 2002, 2003; Wagner & Soto 2002). In exploring mimicry attacks, these authors used offline analysis of the (IDS) to construct their modifications; by contrast, the adversary in near-optimal evasion constructs optimized modifications designed by querying the classifier.

The field of active learning also studies a form of query-based optimization (e.g., see Schohn & Cohn 2000). As summarized by Settles (2009), the three primary approaches to active learning are membership query synthesis, stream-based selective sampling, and pool-based sampling. Our work is most closely related to the membership query synthesis subfield introduced by Angluin (1988), in which the learner can request the label for any instance in feature space, rather than for unlabeled instances drawn from a distribution. However, while active learning and near-optimal evasion are similar in their exploration of query strategies, the objectives for these two settings are quite different—evasion approaches search for low-cost negative instances within a factor

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

$1 + \epsilon$ of the optimal cost, whereas active learning algorithms seek to obtain hypotheses with low-generalization error, often in a PAC setting (see Section 8.1.2 for a discussion on reverse-engineering approaches to evasion and active learning). It is interesting to note, nonetheless, that results in active learning settings (e.g., Dasgupta, Kalai, & Monteleoni 2009; Feldman 2009) have also achieved polynomial query complexities in specific settings. However, the focus of this chapter is solely on the evasion objective, and we leave the exploration of relationships between our results and those in active learning to future work.

Another class of related techniques that use query-based optimization are nongradient global optimization methods often referred to as direct search. Simple examples of these techniques include bisection and golden-section search methods for finding roots and extrema of univariate functions, as well as derivative approximation approaches such as the secant method and interpolation methods (e.g., Burden & Faires 2000). Combinations of these approaches include Brent's (1973) algorithms, which exhibit superlinear convergence under certain conditions on the query function; i.e., the number of queries is inversely quadratic in the desired error tolerance. However, while these approaches can be adapted to multiple dimensions, their query complexity grows exponentially with the dimension. Other approaches include the simplex method of Nelder & Mead (1965) and the DIRECT search algorithm introduced by Jones, Perttunen, & Stuckman (1993) (refer to Jones 2001, and Kolda, Lewis & Torczon 2003, for surveys of direct search methods); however, we are unaware of query bounds for these methods. While any direct search methods can be adapted for near-optimal evasion, these methods were designed to optimize an irregular function in a regular domain with few dimensions, whereas the near-optimal evasion problem involves optimizing regular known functions (the cost function) over an unknown, possibly irregular, and high-dimensional domain (the points labeled as negative by the classifier). The methods we present specifically exploit the regular structure of ℓ_p costs and of the convex-inducing classifiers to achieve near-optimality with only polynomially many queries.

8.1 Characterizing Near-Optimal Evasion

We begin by introducing the assumptions made for this problem. First, we assume that feature space $\mathcal{X}$ for the learner is a real-valued D-dimensional Euclidean space; i.e., $\mathcal{X} = \Re^D$ such as for some intrusion detection systems (e.g., Wang & Stolfo 2004). (Lowd & Meek also consider integer- and Boolean-valued feature spaces and provide interesting results for several classes of Boolean-valued learners, but these spaces are not compatible with the family of convex-inducing classifiers we study in this chapter.) We assume the feature space representation is known to the adversary and there are no restrictions on the adversary's queries; i.e., any point $\mathbf{x}$ in feature space $\mathcal{X}$ can be queried by the adversary to learn the classifier's prediction $f(\mathbf{x})$ at that point. These assumptions may not be true in every real-world setting (for instance, spam detectors are often defined with discrete features, and designers often attempt to hide or

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

randomize their feature set; e.g., see Wang et al. 2006), but allow us to consider a worst-case adversary.

As in Section 2.2.4, we assume the target classifier f is a member of a family of classifiers $\mathcal{F}$—the adversary does not know f but knows the family $\mathcal{F}$. (This knowledge is congruous with the security assumption that the adversary knows the learning algorithm, but not the training set or parameters used to tune the learner.) We also restrict our attention to binary classifiers and use $\mathcal{Y} = \{"-", "+"\}$. We assume the adversary's attack will be against a fixed f so the learning method and the training data used to select f are irrelevant for this problem. Further, we assume $f \in \mathcal{F}$ is deterministic, and so it partitions $\mathcal{X}$ into two sets—the positive class $\mathcal{X}_f^+ = \{\mathbf{x} \in \mathcal{X} \mid f(\mathbf{x}) = "+"\}$ and the negative class $\mathcal{X}_f^- = \{\mathbf{x} \in \mathcal{X} \mid f(\mathbf{x}) = "-"\}$. As before, we take the negative set to be *normal* instances where the sought-after blind spots reside. We assume that the adversary is aware of at least one instance in each class, $\mathbf{x}^- \in \mathcal{X}_f^-$ and $\mathbf{x}^A \in \mathcal{X}_f^+$, and can observe the class for any $\mathbf{x}$ by issuing a membership query: $f(\mathbf{x})$.

8.1.1 Adversarial Cost

We assume the adversary has a notion of utility over the feature space, which we quantify with a cost function $A : \mathcal{X} \mapsto \Re_{0+}$. The adversary wishes to optimize A over the negative class, $\mathcal{X}_f^-$; e.g., a spammer wants to send spam that will be classified as normal email ("−") rather than as spam ("+"). We assume this cost function is a distance to some target instance $\mathbf{x}^A \in \mathcal{X}_f^+$ that is most desirable to the adversary; e.g., for a spammer, this could be a string edit distance required to change $\mathbf{x}^A$ to a different message. We focus on the general class of weighted ℓ_p ($0 < p \leq \infty$) cost functions relative to $\mathbf{x}^A$ defined in terms of the ℓ_p norm $\|\cdot\|_p$ as

$$A_p^{(\mathbf{c})}\left(\mathbf{x} - \mathbf{x}^A\right) = \left\|\mathbf{c} \odot \left(\mathbf{x} - \mathbf{x}^A\right)\right\|_p = \left(\sum_{d=1}^{D} c_d^p \left|x_d - x_d^A\right|^p\right)^{1/p}, \tag{8.1}$$

where $0 < c_d < \infty$ is the relative cost the adversary associates with altering the d^{th} feature. When the relative costs are uniform, $c_d = 1$ for all d, we use the simplified notation A_p to refer to the cost function. Similarly, when referring to a generic weighted cost function with weights $\mathbf{c}$, we use the notation $A^{(\mathbf{c})}$. In Section 8.2.1.3, we also consider the special cases when some features have $c_d = 0$ (the adversary does not care about the d^{th} feature) or $c_d = \infty$ (the adversary requires the d^{th} feature to match x_d^A), but otherwise, the weights are on the interval $(0, \infty)$. We use $\mathbb{B}^C(A; \mathbf{y})$ to denote the C-cost ball (or sublevel set) centered at $\mathbf{y}$ with cost no more than the threshold, C; i.e., $\mathbb{B}^C(A; \mathbf{y}) = \{\mathbf{x} \in \mathcal{X} \mid A(\mathbf{x} - \mathbf{y}) \leq C\}$. For instance, $\mathbb{B}^C\left(A_1; \mathbf{x}^A\right)$ is the set of instances that do not exceed an ℓ_1 cost of C from the target $\mathbf{x}^A$. For convenience, we also use $\mathbb{B}^C(A) \triangleq \mathbb{B}^C\left(A; \mathbf{x}^A\right)$ to denote the C-cost-ball of A recentered at the adversary's target, $\mathbf{x}^A$, since we focus on costs relative to this instance.

Unfortunately, ℓ_p costs do not include many interesting costs such as string edit distances for spam, and in other real-world settings, such as the intrusion detection example given earlier, there may be no natural notion of distance between points. Nevertheless,

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

the objective of this chapter is not to provide practical evasion algorithms, but rather to understand the theoretic capabilities of an adversary on the analytically tractable, albeit practically restrictive, family of ℓ_p costs. Weighted ℓ_1 costs are, however, particularly appropriate for adversarial problems in which the adversary is interested in some features more than others and its cost is assessed based on the degree to which a feature is altered. The ℓ_1-norm is a natural measure of edit distance for email spam, while larger weights can model tokens that are more costly to remove (e.g., a payload URL). We focus first on the weighted ℓ_1 costs studied by Lowd & Meek in Section 8.2 and then explore general ℓ_p costs in Section 8.3. In the latter case, our discussion will focus on uniform weights for ease of exposition, but the results easily extend to the cost-sensitive case as presented for weighted ℓ_1 costs.

Lowd & Meek (2005*a*) define minimal adversarial cost (*MAC*) of a classifier f to be the value

$$MAC\left(f, A\right) \triangleq \inf_{\mathbf{x} \in \mathcal{X}_f^-} \left[A\left(\mathbf{x} - \mathbf{x}^A\right)\right]; \tag{8.2}$$

i.e., the greatest lower bound on the cost obtained by any negative instance. They further define a data point to be an ϵ-approximate *instance of minimal adversarial cost (ϵ-IMAC)* if it is a negative instance with a cost no more than a factor $(1+\epsilon)$ of the *MAC*; i.e., every ϵ-*IMAC* is a member of the set[2]

$$\epsilon\text{-}IMAC\left(f, A\right) \triangleq \left\{\mathbf{x} \in \mathcal{X}_f^- \;\middle|\; A\left(\mathbf{x} - \mathbf{x}^A\right) \leq (1+\epsilon) \cdot \mathrm{MAC}\left(f, A\right)\right\}. \tag{8.3}$$

Alternatively, this set can be characterized as the intersection of the negative class and the ball of A of costs within a factor $(1+\epsilon)$ of $MAC\left(f, A\right)$ (i.e., $\epsilon\text{-}IMAC\left(f, A\right) = \mathcal{X}_f^- \cap \mathbb{B}^{(1+\epsilon)\cdot MAC}\left(A\right)$); a fact we exploit in Section 8.2.2. The adversary's goal is to find an ϵ-*IMAC* efficiently while issuing as few queries as possible. In the next section, we introduce formal notions to quantify how effectively an adversary can achieve this objective.

8.1.2 Near-Optimal Evasion

Lowd & Meek (2005*a*) introduce the concept of *adversarial classifier reverse engineering (ACRE) learnability* to quantify the difficulty of finding an ϵ-*IMAC* instance for a particular family of classifiers, $\mathcal{F}$, and a family of adversarial costs, $\mathcal{A}$.

Using our notation, their definition of *ACRE* ϵ-learnable is as follows: A set of classifiers $\mathcal{F}$ is *ACRE* ϵ-learnable under a set of cost functions $\mathcal{A}$ if an algorithm exists such that for all $f \in \mathcal{F}$ and $A \in \mathcal{A}$, it can find an $\mathbf{x} \in \epsilon\text{-}IMAC\left(f, A\right)$ using only polynomially many membership queries in terms of the dimensionality D, the encoded size of f, and the encoded size of $\mathbf{x}^+$ and $\mathbf{x}^-$.

In this definition, Lowd & Meek use encoded size to refer to the length of the string of digits used to encode f, $\mathbf{x}^+$, and $\mathbf{x}^-$. In generalizing their result, we use a slightly altered definition of query complexity. First, to quantify query complexity, we only use

[2] We use the term ϵ-*IMAC* to refer both to this set and members of it. The usage will be clear from the context.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

the dimension, D, and the number of steps, L_ϵ, required by a unidirectional binary search to narrow the gap to within a factor $1 + \epsilon$, the desired accuracy. By including $L_\epsilon^{(*)}$ in our definition of query complexity, we do not require the encoded size of $\mathbf{x}^+$ and $\mathbf{x}^-$ since L_ϵ implicitly captures the size of the distance between these points as discussed earlier.

Using the encoded sizes of f, $\mathbf{x}^+$, and $\mathbf{x}^-$ in defining ϵ-$IMAC$ searchable is problematic. For our purposes, it is clear that the encoded size of both $\mathbf{x}^+$ and $\mathbf{x}^-$ is D so it is unnecessary to include additional terms for their size. Further we allow for families of nonparametric classifiers for which the notion of *encoding size* is ill defined, but is also unnecessary for the algorithms we present. In extending beyond linear and parametric family of classifiers, it is not straightforward to define the encoding size of a classifier f. One could use notions such as the VC-dimension of $\mathcal{F}$ or its covering number, but it is unclear why size of the classifier is important in quantifying the complexity of ϵ-$IMAC$ search. Moreover, as we demonstrate in this chapter, there are families of classifiers for which ϵ-$IMAC$ search is polynomial in D and L_ϵ alone.

Second, we assume the adversary only has two initial points $\mathbf{x}^- \in \mathcal{X}_f^-$ and $\mathbf{x}^A \in \mathcal{X}_f^+$ (the original setting used a third $\mathbf{x}^+ \in \mathcal{X}_f^+$); this yields simpler search procedures. As is apparent in the algorithms we demonstrate, using $\mathbf{x}^+ = \mathbf{x}^A$ makes the attacker less covert since it is significantly easier to infer the attacker's intentions based on its queries. Covertness is not an explicit goal in ϵ-$IMAC$ search, but it would be a requirement of many real-world attackers. However, since the goal of the near-optimal evasion problem is not to design real attacks but rather to analyze the best possible attack so as to understand a classifier's vulnerabilities, we exclude any covertness requirement but return to the issue in Section 8.4.2.1.

Finally, our algorithms do not reverse engineer so ACRE would be a misnomer. Instead, we call the overall problem near-optimal evasion and replace *ACRE* ϵ-learnable with the following definition of ϵ-$IMAC$ searchable: a family of classifiers $\mathcal{F}$ is ϵ-$IMAC$ *searchable* under a family of cost functions $\mathcal{A}$ if for all $f \in \mathcal{F}$ and $A \in \mathcal{A}$, there is an algorithm that finds some $\mathbf{x} \in \epsilon\text{-}IMAC\,(f, A)$ using polynomially many membership queries in D and L_ϵ. We will refer to such an algorithm as *efficient*.

Our definition does not include the encoded size of the classifier, f, because our approach to near-optimal evasion does not reverse engineer the classifier's parameters as we now discuss in detail.

Near-optimal evasion is only a *partial* reverse-engineering strategy. Unlike Lowd & Meek's approach for continuous spaces, we introduce algorithms that construct queries to provably find an ϵ-$IMAC$ without *fully* reverse engineering the classifier; i.e., estimating the decision surface of f or estimating the parameters that specify it. Efficient query-based reverse engineering for $f \in \mathcal{F}$ is sufficient for minimizing A over the estimated negative space. However, generally reverse engineering is an expensive approach for near-optimal evasion, requiring query complexity that is exponential in the feature space dimension D for general convex classes (Rademacher & Goyal 2009), while finding an ϵ-$IMAC$ need not be as we demonstrate in this chapter.[3] In fact, the requirements

[3] Lowd & Meek (2005*a*) also previously showed that the reverse-engineering technique of finding a feature's sign witness is NP-complete for linear classifiers with Boolean features, but also that this family was nonetheless 2-$IMAC$ searchable.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

for finding an ϵ-$IMAC$ differ significantly from the objectives of reverse-engineering approaches such as active learning. Both approaches use queries to reduce the size of version space $\hat{\mathcal{F}} \subseteq \mathcal{F}$; i.e., the set of classifiers consistent with the adversary's membership queries. Reverse-engineering approaches minimize the expected number of disagreements between members of $\hat{\mathcal{F}}$. In contrast, to find an ϵ-$IMAC$, the adversary only needs to provide a single instance, $\mathbf{x}^\dagger \in \epsilon\text{-}IMAC\,(f, A)$, for all $f \in \hat{\mathcal{F}}$, while leaving the classifier largely unspecified; i.e., we need to show that

$$\bigcap_{f \in \hat{\mathcal{F}}} \epsilon\text{-}IMAC\,(f, A) \neq \emptyset.$$

This objective allows the classifier to be unspecified over much of $\mathcal{X}$. We present algorithms for ϵ-$IMAC$ search on a family of classifiers that generally cannot be efficiently reverse engineered—the queries necessarily only elicit an ϵ-$IMAC$; the classifier itself will be underspecified in large regions of $\mathcal{X}$ so these techniques do not reverse engineer the classifier's parameters or decision boundary except in a shrinking region near an ϵ-$IMAC$. Similarly, for linear classifiers in Boolean spaces, Lowd & Meek demonstrated an efficient algorithm for near-optimal evasion that does not reverse engineer the classifier—it too searches directly for an ϵ-$IMAC$, and it shows that this family is 2-$IMAC$ searchable for ℓ_1 costs with uniform feature weights, $\mathbf{c}$.

8.1.3 Search Terminology

The notion of near-optimality introduced in Equation (8.3) and of the overall near-optimal evasion problem in the previous section is that of multiplicative optimality; i.e., an ϵ-$IMAC$ must have a cost within a factor of $(1 + \epsilon)$ of the MAC. However, the results of this chapter can also be immediately adopted for additive optimality in which the adversary seeks instances with cost no more than $\eta > 0$ *greater* than the MAC. To differentiate between these notions of optimality, we use the notation ϵ-$IMAC^{(*)}$ to refer to the set in Equation (8.3) and define an analogous set η-$IMAC^{(+)}$ for additive optimality as

$$\eta\text{-}IMAC^{(+)}\,(f, A) \triangleq \left\{ \mathbf{x} \in \mathcal{X}_f^- \;\middle|\; A\left(\mathbf{x} - \mathbf{x}^A\right) \leq \eta + \mathrm{MAC}\,(f, A) \right\}. \tag{8.4}$$

We use the terms ϵ-$IMAC^{(*)}$ and η-$IMAC^{(+)}$ to refer both to the sets defined in Equation (8.3) and (8.4) as well as the members of them—the usage will be clear from the context.

We consider algorithms that achieve either additive or multiplicative optimality of the family of convex-inducing classifiers. For either notion of optimality one can efficiently use bounds on the MAC to find an ϵ-$IMAC^{(*)}$ or an η-$IMAC^{(+)}$. Suppose there is a negative instance, $\mathbf{x}^-$, with cost C^-, and there is a $C^+ > 0$ such that all instances with cost no more than C^+ are positive; i.e., C^- is an upper bound and C^+ is a lower bound on the MAC; i.e., $C^+ \leq MAC\,(f, A) \leq C^-$. Under that supposition, then the negative instance $\mathbf{x}^-$ is ϵ-multiplicatively optimal if $C^-/C^+ \leq (1 + \epsilon)$, whereas it is η-additively optimal

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

if $C^- - C^+ \leq \eta$. We consider algorithms that can achieve either additive or multiplicative optimality via binary search. Namely, if the adversary can determine whether an intermediate cost establishes a new upper or lower bound on the *MAC*, then binary search strategies can iteratively reduce the t^{th} gap between any bounds C_t^- and C_t^+ with the fewest steps. We now provide common terminology for the binary search, and in Section 8.2 we use convexity to establish a new bound at the t^{th} iteration.

Remark 8.1 If an algorithm can provide bounds $0 < C^+ \leq MAC(f, A) \leq C^-$, then this algorithm has achieved $(C^- - C^+)$-additive optimality and $(\frac{C^-}{C^+} - 1)$-multiplicative optimality.

In the t^{th} iteration of an additive binary search, the additive gap between the t^{th} bounds, C_t^- and C_t^+, is given by $G_t^{(+)} = C_t^- - C_t^+$ with $G_0^{(+)}$ defined accordingly by the initial bounds $C_0^- = C^-$ and $C_0^+ = C^+$. The search uses a proposal step of $C_t = \frac{C_t^- + C_t^+}{2}$, a stopping criterion of $G_t^{(+)} \leq \eta$, and achieves η-additive optimality in

$$L_\eta^{(+)} = \left\lceil \log_2 \left(\frac{G_0^{(+)}}{\eta} \right) \right\rceil \tag{8.5}$$

steps. In fact, binary search has the least worst-case query complexity for achieving the η-additive stopping criterion for a unidirectional search (e.g., search along a ray).

Binary search can also be used for multiplicative optimality by searching in exponential space. Assuming that $C^- \geq C^+ > 0$, we can rewrite the upper and lower bounds as $C^- = 2^a$ and $C^+ = 2^b$, and thus the multiplicative optimality condition becomes $a - b \leq \log_2(1 + \epsilon)$; i.e., an additive optimality condition. Thus, binary search on the exponent achieves ϵ-multiplicative optimality and does so with the best worst-case query complexity (again in a unidirectional search). The multiplicative gap of the t^{th} iteration is $G_t^{(*)} = C_t^- / C_t^+$ with $G_0^{(*)}$ defined accordingly by the initial bounds C_0^- and C_0^+. The t^{th} query is $C_t = \sqrt{C_t^- \cdot C_t^+}$, the stopping criterion is $G_t^{(*)} \leq 1 + \epsilon$, and it achieves ϵ-multiplicative optimality in

$$L_\epsilon^{(*)} = \left\lceil \log_2 \left(\frac{\log_2 \left(G_0^{(*)} \right)}{\log_2 (1 + \epsilon)} \right) \right\rceil \tag{8.6}$$

steps. Notice that multiplicative optimality only makes sense when both C_0^- and C_0^+ are strictly positive.

It is also worth noting that both $L_\epsilon^{(+)}$ and $L_\epsilon^{(*)}$ can be instead replaced by $\log\left(\frac{1}{\epsilon}\right)$ for asymptotic analysis. As pointed out by Rubinstein (2010), the near-optimal evasion problem is concerned with the difficulty of making accurate estimates of the *MAC*, and this difficulty increases as $\epsilon \downarrow 0$. In this sense, clearly $L_\epsilon^{(+)}$ and $\log\left(\frac{1}{\epsilon}\right)$ are asymptotically equivalent. Similarly, comparing $L_\epsilon^{(*)}$ and $\log\left(\frac{1}{\epsilon}\right)$ as $\epsilon \downarrow 0$, the limit of their ratio (by application of L'Hôpital's rule) is

$$\lim_{\epsilon \downarrow 0} \frac{L_\epsilon^{(*)}}{\log\left(\frac{1}{\epsilon}\right)} = 1;$$

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

i.e., they are also asymptotically equivalent. Thus, in the following asymptotic results, $L_\epsilon^{(*)}$ can be replaced by $\log\left(\frac{1}{\epsilon}\right)$.

Binary searches for additive and multiplicative optimality differ in their proposal step and their stopping criterion. For additive optimality, the proposal is the arithmetic mean $C_t = \frac{C_t^- + C_t^+}{2}$ and search stops when $G_t^{(+)} \leq \eta$, whereas for multiplicative optimality, the proposal is the geometric mean $C_t = \sqrt{C_t^- \cdot C_t^+}$ and search stops when $G_t^{(*)} \leq 1 + \epsilon$. In the remainder of this chapter, we will use the fact that binary search is optimal for unidirectional search to search the cost space. At each step in the search, we use several probes in the feature space $\mathcal{X}$ to determine if the proposed cost is a new upper or lower bound and then continue the binary search accordingly.

8.1.4 Multiplicative vs. Additive Optimality

Additive and multiplicative optimality are intrinsically related by the fact that the optimality condition for multiplicative optimality $C_t^-/C_t^+ \leq 1 + \epsilon$ can be rewritten as additive optimality condition $\log_2\left(C_t^-\right) - \log_2\left(C_t^+\right) \leq \log_2(1+\epsilon)$. From this equivalence one can take $\eta = \log_2(1+\epsilon)$ and utilize the additive optimality criterion on the logarithm of the cost. However, this equivalence also highlights two differences between these notions of optimality.

First, multiplicative optimality only makes sense when C_0^+ is strictly positive, whereas additive optimality can still be achieved if $C_0^+ = 0$. Taking $C_0^+ > 0$ is equivalent to assuming that $\mathbf{x}^A$ is in the interior of $\mathcal{X}_f^+$ (a requirement for our algorithms to achieve multiplicative optimality). Otherwise, when $\mathbf{x}^A$ is on the boundary of $\mathcal{X}_f^+$, there is no ϵ-$IMAC^{(*)}$ for any $\epsilon > 0$ unless there is some point $\mathbf{x}^\star \in \mathcal{X}_f^-$ that has 0 cost. Practically though, the need for a lower bound is a minor hindrance—as we demonstrate in Section 8.2.1.3, there is an algorithm that can efficiently establish a lower bound C_0^+ for any ℓ_p cost if such a lower bound exists.

Second, the additive optimality criterion is not scale invariant (i.e., any instance $\mathbf{x}^\dagger$ that satisfies the optimality criterion for cost A also satisfies it for $A'(\mathbf{x}) = s \cdot A(\mathbf{x})$ for any $s > 0$), whereas multiplicative optimality is scale invariant. Additive optimality is, however, shift invariant (i.e., any instance $\mathbf{x}^\dagger$ that satisfies the optimality criterion for cost A also satisfies it for $A'(\mathbf{x}) = s + A(\mathbf{x})$ for any $s \geq 0$), whereas multiplicative optimality is not. Scale invariance is more salient in near-optimal evasion because if the cost function is also scale invariant (all proper norms are), then the optimality condition is invariant to a rescaling of the underlying feature space; e.g., a change in units for all features. Thus, multiplicative optimality is a unit-less notion of optimality whereas additive optimality is not. The following result is a consequence of additive optimality's lack of scale invariance.

PROPOSITION 8.2 *Consider any hypothesis space $\mathcal{F}$, target instance $\mathbf{x}^A$, and cost function A. If there exists some $\bar{\epsilon} > 0$ such that no efficient query-based algorithm can find an ϵ-$IMAC^{(*)}$ for any $0 < \epsilon \leq \bar{\epsilon}$, then there is no efficient query-based algorithm that can find an η-$IMAC^{(+)}$ for any $0 < \eta \leq \bar{\epsilon} \cdot MAC(f, A)$. In particular consider a sequence of classifiers f_n admitting unbounded MACs, and a sequence $\epsilon_n > 0$ such that*

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

$1/\epsilon_n = o(MAC(f_n, A))$. *Then if no general algorithm can efficiently find an* ϵ_n-$IMAC^{(*)}$ *on each* f_n, *no general algorithm can efficiently find an* η_n-$IMAC^{(+)}$ *for* $\eta_n \to \infty$.

Proof Consider any classifier $f \in \mathcal{F}$ such that $MAC(f, A) > 0$. Suppose there exists some $\mathbf{x} \in \eta$-$IMAC^{(+)}$ for some $\eta > 0$. Let $\epsilon = \eta / MAC(f, A)$; then by definition

$$A\left(\mathbf{x} - \mathbf{x}^A\right) \leq \eta + MAC(f, A) = (1 + \epsilon) MAC(f, A), \tag{8.7}$$

implying that $\mathbf{x} \in \epsilon$-$IMAC^{(*)}$. Then by the contrapositive, if no ϵ-$IMAC^{(*)}$ can be efficiently found for any $0 < \epsilon \leq \bar{\epsilon}$, no η-$IMAC^{(+)}$ can be efficiently found for any $0 < \eta \leq \bar{\epsilon} \cdot MAC(f, A)$. The last result is an immediate corollary. □

The last statement is, in fact, applicable to many common settings. For instance, for any of the weighted ℓ_p costs (with $0 < p \leq \infty$ and $0 < c_d < \infty$ for all d) the family of linear classifiers and the family of hypersphere classifiers are both sufficiently diverse to yield such a sequence of classifiers that admit unbounded MACs as required by the last statement. Thus, the family of convex-inducing classifiers can also yield such a sequence. Moreover, as we show in Section 8.3, there are indeed ℓ_p costs for which there exists $\bar{\epsilon} > 0$ such that no efficient query-based algorithm can find an ϵ-$IMAC^{(*)}$ for any $0 < \epsilon \leq \bar{\epsilon}$. The consequence of this is that there is no general algorithm capable of achieving additive optimality for any fixed η with respect to the convex-inducing classifiers for these ℓ_p costs, as is shown by the following theorem:

THEOREM 8.3 *If for some hypothesis space* $\mathcal{F}$, *cost function* A, *and any initial bounds* $0 < C_0^+ < C_0^-$ *on the* $MAC(f, A)$ *for some* $f \in \mathcal{F}$, *there exists some* $\bar{\epsilon} > 0$ *such that no efficient query-based algorithm can find an* ϵ-$IMAC^{(*)}$ *for any* $0 < \epsilon \leq \bar{\epsilon}$, *then there is no efficient query-based algorithm that can find an* η-$IMAC^{(+)}$ *for any* $0 < \eta \leq \bar{\epsilon} \cdot C_0^-$. *As a consequence, if there is* $\bar{\epsilon} > 0$ *as stated above, then there is generally no efficient query-based algorithm that can find an* η-$IMAC^{(+)}$ *for any* $\eta \geq 0$ *since* C_0^- *could be arbitrarily large.*

Proof By contraposition. If there is an efficient query-based algorithm that can find an $\mathbf{x} \in \eta$-$IMAC^{(+)}$ for some $0 < \eta \leq \bar{\epsilon} \cdot C_0^-$, then, by definition of η-$IMAC^{(+)}$, $A\left(\mathbf{x} - \mathbf{x}^A\right) \leq \eta + MAC(f, A)$. Equivalently, by taking $\eta = \epsilon \cdot MAC(f, A)$ for some $\epsilon > 0$, this algorithm achieved $A\left(\mathbf{x} - \mathbf{x}^A\right) \leq (1 + \epsilon) MAC(f, A)$; i.e., $\mathbf{x} \in \epsilon$-$IMAC^{(*)}$. Moreover, since $MAC(f, A) \leq C_0^-$, this efficient algorithm is able to find an ϵ-$IMAC^{(*)}$ for some $\epsilon \leq \bar{\epsilon}$. The last remark follows directly from the fact that there is no efficient query-based algorithm for any $0 < \eta \leq \bar{\epsilon} \cdot C_0^-$ and C_0^- could generally be arbitrarily large. □

This result further suggests that additive optimality in near-optimal evasion is an awkward notion. If there is a cost function A for which some family of classifiers $\mathcal{F}$ cannot be efficiently evaded within any accuracy $0 < \epsilon \leq \bar{\epsilon}$, then the question of whether efficient additive optimality can be achieved for some $\eta > 0$ depends on the scale of the cost function. That is, if η-additive optimality can be efficiently achieved for A, the feature space could be rescaled to make η-additive optimality no longer generally efficiently since the rescaling could be chosen to make C_0^- large. This highlights the

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

limitation of the lack of scale invariance in additive optimality: *the units of the cost determine whether a particular level of additive accuracy can be achieved, whereas multiplicative optimality is unit-less*. For (weighted) ℓ_1 costs, this is not an issue since, as Section 8.2 shows, there is an efficient algorithm for ϵ-multiplicative optimality for any $\epsilon > 0$. However, as we demonstrate in Section 8.3, there are ℓ_p costs where this becomes problematic.

For the remainder of this chapter, we primarily only address ϵ-multiplicative optimality for an ϵ-*IMAC* (except where explicitly noted) and define $G_t = G_t^{(*)}$, $C_t = \sqrt{C_t^- \cdot C_t^+}$, and $L_\epsilon = L_\epsilon^{(*)}$. Nonetheless, the algorithms we present can be immediately adapted to additive optimality by simply changing the proposal step, stopping condition, and the definitions of $L_\epsilon^{(*)}$ and G_t, although they may not be generally efficient as discussed earlier.

8.1.5 The Family of Convex-Inducing Classifiers

We introduce the family of convex-inducing classifiers, $\mathcal{F}^{\text{convex}}$; i.e., the set of classifiers that partition the feature space $\mathcal{X}$ into a positive and negative class, one of which is convex. The convex-inducing classifiers include the linear classifiers studied by Lowd & Meek, as well as anomaly detection classifiers using boundeds PCA (Lakhina et al. 2004*b*), anomaly detection algorithms that use hypersphere boundaries (Bishop 2006), one-class classifiers that predict anomalies by thresholding the log-likelihood of a log-concave (or unimodal) density function, and quadratic classifiers of the form $\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c \geq 0$ if $\mathbf{A}$ is semidefinite (cf. Boyd & Vandenberghe 2004, Chapter 3). The convex-inducing classifiers also include complicated bodies such as any intersections of a countable number of halfspaces, cones, or balls.

There is a correspondence between the family of convex-inducing classifiers and the set of all convex sets; i.e., $\mathbb{C} = \{\mathbb{X} \mid \textit{convex}(\mathbb{X})\}$. By definition of the convex-inducing classifiers, every classifier $f \in \mathcal{F}^{\text{convex}}$ corresponds to some convex set in $\mathbb{C}$. Further, for any convex set $\mathbb{X} \in \mathbb{C}$, there are at least two trivial classifier that create that set; namely the classifiers $f_{\mathbb{X}}^{"+"}(\mathbf{x}) = \mathrm{I}[\mathbf{x} \in \mathbb{X}]$ and $f_{\mathbb{X}}^{"-"}(\mathbf{x}) = \mathrm{I}[\mathbf{x} \notin \mathbb{X}]$. Thus, in the remainder of this chapter, we use the existence of particular convex sets to prove results about the convex-inducing classifiers since there is always a corresponding classifier.

It is also worth mentioning the following alternative characterization of the near-optimal evasion problem on the convex-inducing classifiers. For any convex set $\mathbb{C}$ with a non-empty interior, let $\mathbf{x}^{(c)}$ be a point in its interior and define the Minkowski metric (recentered at $\mathbf{x}^{(c)}$) as $m_{\mathbb{C}}(\mathbf{x}) = \inf\left\{\lambda \;\middle|\; (\mathbf{x} - \mathbf{x}^{(c)}) \in \lambda(\mathbb{C} - \mathbf{x}^{(c)})\right\}$. This function is convex and non-negative, and it satisfies $m_{\mathbb{C}}(\mathbf{x}) \leq 1$ if and only if $\mathbf{x} \in \mathbb{C}$. Thus, we can rewrite the definition of the *MAC* of a classifier in terms of the Minkowski metric—if $\mathcal{X}_f^+$ is convex we require $m_{\mathcal{X}_f^+}(\mathbf{x}) > 1$, and if $\mathcal{X}_f^-$ is convex we require $m_{\mathcal{X}_f^-}(\mathbf{x}) \leq 1$. In this way, the near-optimal evasion problem (for $\mathcal{X}_f^-$ convex) can be rewritten as

$$\begin{aligned} &\operatorname{argmin}_{\mathbf{x} \in \mathcal{X}} \left[A\left(\mathbf{x} - \mathbf{x}^A\right)\right] \\ &\text{s.t.} \qquad\qquad m_{\mathcal{X}_f^-}(\mathbf{x}) \leq 1 \end{aligned} \tag{8.8}$$

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

If A is convex, the fact that $m_{\mathbb{C}}(\cdot)$ is convex makes this a convex program that can be solved by optimizing its Lagrangian

$$\underset{\mathbf{x}\in\mathcal{X},\gamma\in\Re_{0+}}{\operatorname{argmin}} \left[A\left(\mathbf{x}-\mathbf{x}^A\right)+\gamma\left(1-m_{\mathcal{X}_f^-}(\mathbf{x})\right)\right].$$

In cases where $m_{\mathcal{X}_f^-}(\cdot)$ has a closed form, this optimization may have a closed-form solution, but generally this approach seems difficult. Instead, we use the special structure of the ℓ_1 cost function to construct efficient search over the family of convex-inducing classifiers.

8.2 Evasion of Convex Classes for ℓ_1 Costs

We generalize ϵ-*IMAC* searchability to the family of convex-inducing classifiers. Restricting $\mathcal{F}$ to be the family of convex-inducing classifiers simplifies ϵ-*IMAC* search. In our approach to this problem, we divide $\mathcal{F}^{\text{convex}}$, the family of convex-inducing classifiers, into $\mathcal{F}^{\text{convex,"}-\text{"}}$ and $\mathcal{F}^{\text{convex,"}+\text{"}}$ corresponding to the classifiers that induce a convex set $\mathcal{X}_f^-$ or $\mathcal{X}_f^+$, respectively (of course, linear classifiers belong to both). When the negative class $\mathcal{X}_f^-$ is convex (i.e., $f\in\mathcal{F}^{\text{convex,"}-\text{"}}$), the problem reduces to minimizing a (convex) function A constrained to a convex set—if $\mathcal{X}_f^-$ were known to the adversary, this problem reduces simply to solving a convex optimization program (cf. Boyd & Vandenberghe 2004, Chapter 4). When the positive class $\mathcal{X}_f^+$ is convex (i.e., $f\in\mathcal{F}^{\text{convex,"}+\text{"}}$), however, the problem becomes minimizing a (convex) function A outside of a convex set; this is generally a difficult problem (see Section 8.3.1.4 where we show that minimizing an ℓ_2 cost can require exponential query complexity). Nonetheless for certain cost functions A, it is easy to determine whether a particular cost ball $\mathbb{B}^C(A)$ is completely contained within a convex set. This leads to efficient approximation algorithms that we present in this section.

We construct efficient algorithms for query-based optimization of the (weighted) ℓ_1 cost $A_1^{(\mathbf{c})}$ of Equation (8.1) for the family of convex-inducing classifiers. There is, however, an asymmetry in this problem depending on whether the positive or negative class is convex as illustrated in Figure 8.1. When the positive set is convex, determining whether the ℓ_1 ball $\mathbb{B}^C\left(A_1^{(\mathbf{c})}\right)$ is a subset of $\mathcal{X}_f^+$ only requires querying the vertexes of the ball as depicted in Figure 8.1(a). When the negative set is convex, determining whether or not $\mathbb{B}^C\left(A_1^{(\mathbf{c})}\right)\cap\mathcal{X}_f^-=\emptyset$ is nontrivial since the intersection need not occur at a vertex as depicted in Figure 8.1(b). We present an efficient algorithm for optimizing (weighted) ℓ_1 costs when $\mathcal{X}_f^+$ is convex and a polynomial random algorithm for optimizing any convex cost when $\mathcal{X}_f^-$ is convex, although in both cases, we only consider convex sets with non-empty interiors. The algorithms we present achieve multiplicative optimality via the binary search strategies discussed in the previous section. In the sequel, we use Equation (8.6) to define L_ϵ as the number of phases required by binary

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

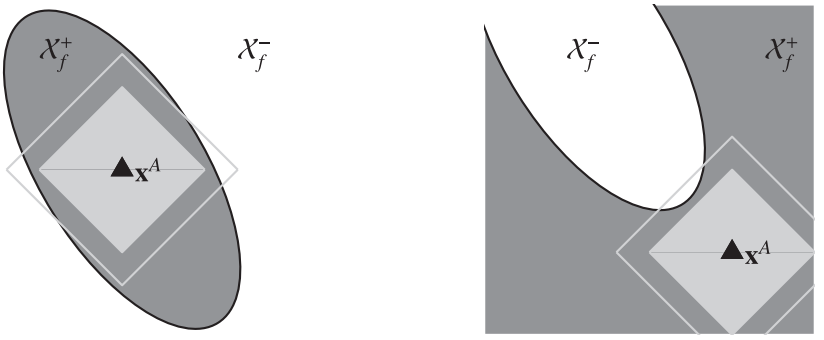

Figure 8.1 Geometry of convex sets and ℓ_1 balls. **(a)** If the positive set $\mathcal{X}_f^+$ is convex, finding an ℓ_1 ball contained within $\mathcal{X}_f^+$ establishes a lower bound on the cost; otherwise at least one of the ℓ_1 ball's corners witnesses an upper bound. **(b)** If the negative set $\mathcal{X}_f^-$ is convex, the adversary can establish upper and lower bounds on the cost by determining whether or not an ℓ_1 ball intersects with $\mathcal{X}_f^-$, but this intersection need not include any corner of the ball.

search[4] and $C_0^- = A_1^{(\mathbf{c})}\left(\mathbf{x}^- - \mathbf{x}^A\right)$ as an initial upper bound on the *MAC*. We also assume there is some $C_0^+ > 0$ that lower bounds the *MAC* (i.e., $\mathbf{x}^A \in int\left(\mathcal{X}_f^+\right)$).

8.2.1 ϵ-*IMAC* Search for a Convex $\mathcal{X}_f^+$

Solving the ϵ-*IMAC* search problem when $f \in \mathcal{F}^{\text{convex,"+"}}$ is difficult for the general case of optimizing a convex cost A. We demonstrate algorithms for the (weighted) ℓ_1 cost of Equation (8.1) that solve the problem as a binary search. Namely, given initial costs C_0^+ and C_0^- that bound the *MAC*, we introduce an algorithm that efficiently determines whether $\mathbb{B}^{C_t}(A_1) \subseteq \mathcal{X}_f^+$ for any intermediate cost $C_t^+ < C_t < C_t^-$. If the ℓ_1 ball is contained in $\mathcal{X}_f^+$, then C_t becomes the new lower bound C_{t+1}^+. Otherwise C_t becomes the new upper bound C_{t+1}^-. Since the objective given in Equation (8.3) is to obtain multiplicative optimality, the steps will be $C_t = \sqrt{C_t^+ \cdot C_t^-}$ (for additive optimality, see Section 8.1.3).

The existence of an efficient query-based algorithm relies on three facts: (1) $\mathbf{x}^A \in \mathcal{X}_f^+$; (2) every weighted ℓ_1 cost C-ball centered at $\mathbf{x}^A$ intersects with $\mathcal{X}_f^-$ only if at least one of its vertexes is in $\mathcal{X}_f^-$; and (3) C-balls of weighted ℓ_1 costs only have $2 \cdot D$ vertexes. The vertexes of the weighted ℓ_1 ball $\mathbb{B}^C(A_1)$ are axis-aligned instances differing from $\mathbf{x}^A$ in exactly one feature (e.g., the d^{th} feature) and can be expressed in the form

$$\mathbf{x}^A \pm \frac{C}{c_d} \cdot \mathbf{e}^{(d)} \tag{8.9}$$

[4] As noted in Section 8.1.3, the results of this section can be replicated for additive optimality by using Equation (8.5) for L_ϵ and by using the regular binary search proposal and stopping criterion.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

which belongs to the C-ball of the weighted ℓ_1 cost (the coefficient $\frac{C}{c_d}$ normalizes for the weight c_d on the d^{th} feature). The second fact is formalized as the following lemma:

LEMMA 8.4 *For all $C > 0$, if there exists some $\mathbf{x} \in \mathcal{X}_f^-$ that achieves a cost of $C = A_1^{(\mathbf{c})}\left(\mathbf{x} - \mathbf{x}^A\right)$, then there is some feature d such that a vertex of the form of Equation* (8.9) *is in $\mathcal{X}_f^-$ (and also achieves cost C by Equation 8.1).*

Proof Suppose not; then there is some $\mathbf{x} \in \mathcal{X}_f^-$ such that $A_1^{(\mathbf{c})}\left(\mathbf{x} - \mathbf{x}^A\right) = C$ and $\mathbf{x}$ has $M \geq 2$ features that differ from $\mathbf{x}^A$ (if $\mathbf{x}$ differs in one or no features it would be of the form of Equation 8.9). Let $\{d_1, \ldots, d_M\}$ be the differing features, and let $b_{d_i} = \text{sign}\left(x_{d_i} - x_{d_i}^A\right)$ be the sign of the difference between $\mathbf{x}$ and $\mathbf{x}^A$ along the d_i^{th} feature. For each d_i, let $\mathbf{w}_{d_i} = \mathbf{x}^A + \frac{C}{c_{d_i}} \cdot b_{d_i} \cdot \mathbf{e}^{(d_i)}$ be a vertex of the form of Equation (8.9) that has a cost C (from Equation 8.1). The M vertexes $\mathbf{w}_{d_i}$ form an M-dimensional equi-cost simplex of cost C on which $\mathbf{x}$ lies; i.e., $\mathbf{x} = \sum_{i=1}^{M} \alpha_i \cdot \mathbf{w}_{d_i}$ for some $0 \leq \alpha_i \leq 1$. If all $\mathbf{w}_{d_i} \in \mathcal{X}_f^+$, then the convexity of $\mathcal{X}_f^+$ implies that all points in their simplex are in $\mathcal{X}_f^+$ and so $\mathbf{x} \in \mathcal{X}_f^+$, which violates the premise. Thus, if any instance in $\mathcal{X}_f^-$ achieves cost C, there is always at least one vertex of the form Equation (8.9) in $\mathcal{X}_f^-$ that also achieves cost C. □

As a consequence, if all such vertexes of any C ball $\mathbb{B}^C(A_1)$ are positive, then all $\mathbf{x}$ with $A_1^{(\mathbf{c})}(\mathbf{x}) \leq C$ are positive, thus establishing C as a lower bound on the MAC. Conversely, if any of the vertexes of $\mathbb{B}^C(A_1)$ are negative, then C is an upper bound on MAC. Thus, by simultaneously querying all $2 \cdot D$ equi-cost vertexes of $\mathbb{B}^C(A_1)$, the adversary either establishes C as a new lower or upper bound on the MAC. By performing a binary search on C the adversary iteratively halves the multiplicative gap until it is within a factor of $1 + \epsilon$. This yields an ϵ-$IMAC$ of the form of Equation (8.9).

A general form of this multi-line search procedure is presented as Algorithm 8.1 and depicted in Figure 8.3. MULTILINESEARCH simultaneously searches along all

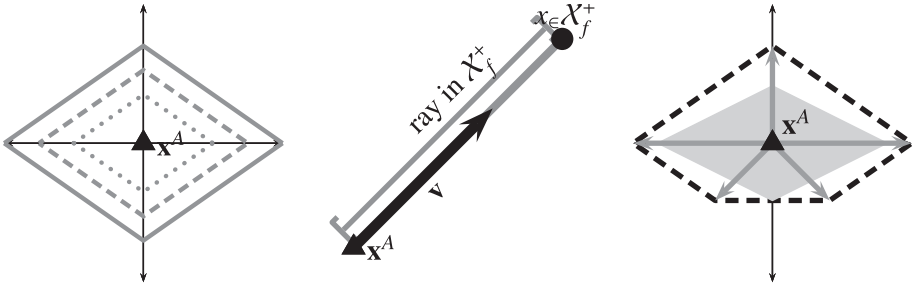

Figure 8.2 The geometry of multi-line search. **(a)** Weighted ℓ_1 balls are centered around the target $\mathbf{x}^A$ and have $2 \cdot D$ vertexes. **(b)** Search directions in multi-line search radiate from $\mathbf{x}^A$ to probe specific costs. **(c)** In general, the adversary leverages convexity of the cost function when searching to evade. By probing all search directions at a specific cost, the convex hull of the positive queries bounds the ℓ_1 cost ball contained within it.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

unit-cost search directions in the set $\mathbb{W}$, which contains search directions that radiate from their origin at $\mathbf{x}^A$ and are unit vectors for their cost; i.e., $A(\mathbf{w}) = 1$ for every $\mathbf{w} \in \mathbb{W}$. Of course, any set of non-normalized search vectors $\{\mathbf{v}\}$ can be transformed into unit search vectors simply by applying a normalization constant of $A(\mathbf{v})^{-1}$ to each. At each step, MultiLineSearch (Algorithm 8.1) issues at most $|\mathbb{W}|$ queries to construct a bounding shell (i.e., the convex hull of these queries will either form an upper or lower bound on the *MAC*) to determine whether $\mathbb{B}^C(A_1) \subseteq \mathcal{X}_f^+$. Once a negative instance is found at cost C, the adversary ceases further queries at cost C since a single negative instance is sufficient to establish a lower bound. We call this policy *lazy querying*[5]—a practice that will lead to better bounds for a malicious classifier. Further, when an upper bound is established for a cost C (i.e., a negative vertex is found), the algorithm prunes all directions that were positive at cost C. This pruning is sound; by the convexity assumption, these pruned directions are positive for all costs less than the new upper bound C on the *MAC* so no further queries will be required along such a direction. Finally, by performing a binary search on the cost, MultiLineSearch finds an ϵ-*IMAC* with no more than $|\mathbb{W}| \cdot L_\epsilon$ queries but at least $|\mathbb{W}| + L_\epsilon$ queries. Thus, this algorithm has a best-case query complexity of $\mathcal{O}(|\mathbb{W}| \cdot L_\epsilon)$ and a worst-case query complexity of $\mathcal{O}(|\mathbb{W}| \cdot L_\epsilon)$.

It is worth noting that, in its present form, MultiLineSearch has two implicit assumptions. First, we assume all search directions radiate from a common origin, $\mathbf{x}^A$, and $A(\mathbf{0}) = 0$. Without this assumption, the ray-constrained cost function $A(s \cdot \mathbf{w})$ is still convex in $s \geq 0$, but not necessarily monotonic as required for binary search. Second, we assume the cost function A is a positive homogeneous function along any ray from $\mathbf{x}^A$; i.e., $A(s \cdot \mathbf{w}) = |s| \cdot A(\mathbf{w})$. This assumption allows MultiLineSearch to scale its unit search vectors to achieve the same scaling of their cost. Although the algorithm could be adapted to eliminate these assumptions, the cost functions in Equation (8.1) satisfy both assumptions since they are norms recentered at $\mathbf{x}^A$.

Algorithm 8.2 uses MultiLineSearch for (weighted) ℓ_1 costs by making $\mathbb{W}$ be the vertexes of the unit-cost ℓ_1 ball centered at $\mathbf{x}^A$. In this case, the search issues at most $2 \cdot D$ queries to determine whether $\mathbb{B}^C(A_1)$ is a subset of $\mathcal{X}_f^+$ and thus is $\mathcal{O}(L_\epsilon \cdot D)$. However, MultiLineSearch does not rely on its directions being vertexes of the ℓ_1 ball, although those vertexes are sufficient to span the ℓ_1 ball. Generally, MultiLineSearch is agnostic to the configuration of its search directions and can be adapted for any set of directions that can provide a sufficiently tight bound on the cost using the convexity of $\mathcal{X}_f^+$ (see Section 8.3.1.1 for the bounding requirements that the search directions must satisfy). However, as we show in Section 8.3.1, the number of search directions required to adequately bound an ℓ_p-cost ball for $p > 1$ can be exponential in D.

[5] The search algorithm could continue to query at any distance B^- where there is a known negative instance as it may expedite the pruning of additional search directions early in the search. However, in analyzing the malicious classifier, these additional queries will not lead to further pruning, but instead will prevent improvements on the worst-case query complexity, as demonstrated in Section 8.2.1.1. Thus, the algorithms we present only use lazy querying and only queries at costs below the upper bound C_t^- on the *MAC*.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

ALGORITHM 8.1 MULTILINESEARCH

$MLS\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C_0^+, C_0^-, \epsilon\right)$
$\mathbf{x}^\star \leftarrow \mathbf{x}^-$
$t \leftarrow 0$
while $C_t^- / C_t^+ > 1 + \epsilon$ **do begin**
  $C_t \leftarrow \sqrt{C_t^+ \cdot C_t^-}$
  for all $\mathbf{w} \in \mathbb{W}$ **do begin**
    Query: $f_{\mathbf{w}}^t \leftarrow f\left(\mathbf{x}^A + C_t \cdot \mathbf{w}\right)$
    if $f_{\mathbf{w}}^t =$ "$-$" **then begin**
      $\mathbf{x}^\star \leftarrow \mathbf{x}^A + C_t \cdot \mathbf{w}$
      Prune $\mathbf{i}$ from $\mathbb{W}$ if $f_{\mathbf{i}}^t =$ "$+$"
      break for-loop
    end if
  end for
  $C_{t+1}^+ \leftarrow C_t^+$ and $C_{t+1}^- \leftarrow C_t^-$
  if $\forall \mathbf{w} \in \mathbb{W}\ f_{\mathbf{w}}^t =$ "$+$" **then** $C_{t+1}^+ \leftarrow C_t$
  else $C_{t+1}^- \leftarrow C_t$
  $t \leftarrow t + 1$
end while
return: $\mathbf{x}^\star$

ALGORITHM 8.2 CONVEX $\mathcal{X}_f^+$ SEARCH

$ConvexSearch\left(\mathbf{x}^A, \mathbf{x}^-, \mathbf{c}, \epsilon, C^+\right)$
$D \leftarrow dim\left(\mathbf{x}^A\right)$
$C^- \leftarrow A^{(\mathbf{c})}\left(\mathbf{x}^- - \mathbf{x}^A\right)$
$\mathbb{W} \leftarrow \emptyset$
for $i = 1$ to D **do begin**
  $\mathbf{w}^i \leftarrow \frac{1}{c_i} \cdot \mathbf{e}^{(i)}$
  $\mathbb{W} \leftarrow \mathbb{W} \cup \left\{\pm \mathbf{w}^i\right\}$
end for
return: $MLS\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C^+, C^-, \epsilon\right)$

Figure 8.3 Algorithms for multi-line search. Algorithm 8.1 is a generic procedure for performing simultaneous binary searches along multiple search directions emanating from $\mathbf{x}^A$; each direction, $\mathbf{w} \in \mathbb{W}$, must be a unit-cost direction. Algorithm 8.2 uses this MULTILINESEARCH procedure to minimize weighted ℓ_1 costs when the positive class of a classifier is convex. For this procedure, every weight, c_i, must be on the range $(0, \infty)$, although extensions are discussed in Section 8.2.1.3.

8.2.1.1 K-step Multi-line Search

Here we present a variant of the multi-line search algorithm that better exploits pruning to reduce the query complexity of Algorithm 8.1. The original MULTILINESEARCH algorithm is $2 \cdot |\mathbb{W}|$ simultaneous binary searches (i.e., a breadth-first search simultaneously along all search directions). This strategy prunes directions most effectively when the convex body is asymmetrically elongated relative to $\mathbf{x}^A$, but fails to prune for symmetrically round bodies. The algorithm could instead search sequentially (i.e., a depth-first search of L_ϵ steps along each direction sequentially). This alternative search strategy also obtains a best case of $\mathcal{O}\left(L_\epsilon + |\mathbb{W}|\right)$ queries (for a body that is symmetrically round about $\mathbf{x}^A$, it uses L_ϵ queries along the first direction to establish an upper and lower bound within a factor of $1 + \epsilon$, and then D queries to verify the lower bound) and a worst case of $\mathcal{O}\left(L_\epsilon \cdot |\mathbb{W}|\right)$ queries (for asymmetrically elongated bodies, in the worst case, the strategy would require L_ϵ queries along each of the D search directions). Surprisingly, these two alternatives have opposite best-case and worst-case convex bodies, which inspired a hybrid approach called K-STEP MULTILINESEARCH. This algorithm

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

mixes simultaneous and sequential strategies to achieve a better worst-case query complexity than either pure search strategy.[6]

At each phase, the K-step MultiLineSearch (Algorithm 8.3) chooses a single direction $\mathbf{w}$ and queries it for K steps to generate candidate bounds B^- and B^+ on the *MAC*. The algorithm makes substantial progress toward reducing G_t without querying other directions (a depth-first strategy). It then iteratively queries all remaining directions at the candidate lower bound B^+ (a breadth-first strategy). Again, we use lazy querying and stop as soon as a negative instance is found since B^+ is then no longer a viable lower bound. In this case, although the candidate bound is invalidated, the algorithm can still prune all directions that were positive at B^+ (there will always be at least one such direction). Thus, in every iteration, either the gap is substantially decreased or at least one search direction is pruned. We show that for $K = \lceil \sqrt{L_\epsilon} \rceil$, the algorithm achieves a delicate balance between the usual breadth-first and depth-first approaches to attain a better worst-case complexity than either.

THEOREM 8.5 *Algorithm 8.3 will find an ϵ-IMAC with at most $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon}\,|\mathbb{W}|\right)$ queries when $K = \lceil \sqrt{L_\epsilon} \rceil$.*

The proof of this theorem appears in Appendix D. As a consequence of Theorem 8.5, finding an ϵ-*IMAC* with Algorithm 8.3 for a (weighted) ℓ_1 cost requires $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon} D\right)$ queries. Further, Algorithm 8.2 can incorporate K-step MultiLineSearch directly by replacing its function call to MultiLineSearch with K-step MultiLineSearch and using $K = \lceil \sqrt{L_\epsilon} \rceil$.

8.2.1.2 Lower Bound

Here we find a lower bound on the number of queries required by any algorithm to find an ϵ-*IMAC* when $\mathcal{X}_f^+$ is convex for any convex cost function; e.g., Equation (8.1) for $p \geq 1$. Below, we present theorems for additive and multiplicative optimality. Notably, since an ϵ-*IMAC* uses multiplicative optimality, we incorporate a bound $C_0^+ > 0$ on the *MAC* into the theorem statement.

THEOREM 8.6 *For any $D > 0$, any positive convex function $A : \Re^D \mapsto \Re_+$, any initial bounds $0 \leq C_0^+ < C_0^-$ on the MAC, and $0 < \eta < C_0^- - C_0^+$, all algorithms must submit at least $\max\{D, L_\eta^{(+)}\}$ membership queries in the worst case to be η-additive optimal on $\mathcal{F}^{\text{convex},''+''}$.*

THEOREM 8.7 *For any $D > 0$, any positive convex function $A : \Re^D \mapsto \Re_+$, any initial bounds $0 < C_0^+ < C_0^-$ on the MAC, and $0 < \epsilon < \frac{C_0^-}{C_0^+} - 1$, all algorithms must submit at least $\max\{D, L_\epsilon^{(*)}\}$ membership queries in the worst case to be ϵ-multiplicatively optimal on $\mathcal{F}^{\text{convex},''+''}$.*

The proof of both of these theorems is in Appendix D. Note that these theorems only apply to $\eta \in \left(0, C_0^- - C_0^+\right)$ and $\epsilon \in \left(0, \frac{C_0^-}{C_0^+} - 1\right)$, respectively. In fact, outside of these intervals the query strategies are trivial. For either $\eta = 0$ or $\epsilon = 0$, no approximation

[6] K-step MultiLineSearch also has a best case of $\mathcal{O}\left(L_\epsilon + |\mathbb{W}|\right)$.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

ALGORITHM 8.3 K-STEP MULTILINESEARCH

$KMLS\left(\mathbb{W}, \mathbf{x}^A, \mathbf{x}^-, C_0^+, C_0^-, \epsilon, K\right)$
$\mathbf{x}^\star \leftarrow \mathbf{x}^-$
$t \leftarrow 0$
while $C_t^- / C_t^+ > 1 + \epsilon$ **do begin**
  Choose a direction $\mathbf{w} \in \mathbb{W}$
  $B^+ \leftarrow C_t^+$
  $B^- \leftarrow C_t^-$
  for K steps **do begin**
    $B \leftarrow \sqrt{B^+ \cdot B^-}$
    Query: $f_{\mathbf{w}} \leftarrow f\left(\mathbf{x}^A + B \cdot \mathbf{w}\right)$
    if $f_{\mathbf{w}} =$ "+" **then** $B^+ \leftarrow B$
    else $B^- \leftarrow B$ **and** $\mathbf{x}^\star \leftarrow \mathbf{x}^A + B \cdot \mathbf{w}$
  end for
  for all $\mathbf{i} \in \mathbb{W} \setminus \{\mathbf{w}\}$ **do begin**
    Query: $f_{\mathbf{i}}^t \leftarrow f\left(\mathbf{x}^A + (B^+) \cdot \mathbf{i}\right)$
    if $f_{\mathbf{i}}^t =$ "−" **then begin**
      $\mathbf{x}^\star \leftarrow \mathbf{x}^A + (B^+) \cdot \mathbf{i}$
      Prune $\mathbf{k}$ from $\mathbb{W}$ if $f_{\mathbf{k}}^t =$ "+"
      break for-loop
    end if
  end for
  $C_{t+1}^- \leftarrow B^-$
  if $\forall \mathbf{i} \in \mathbb{W}\ f_{\mathbf{i}}^t =$ "+" **then** $C_{t+1}^+ \leftarrow B^+$
  else $C_{t+1}^- \leftarrow B^+$
  $t \leftarrow t + 1$
end while
return: $\mathbf{x}^\star$

algorithm terminates. Similarly, for $\eta \geq C_0^- - C_0^+$ or $\epsilon \geq \frac{C_0^-}{C_0^+} - 1$, $\mathbf{x}^-$ is an $IMAC$ since it has a cost $A\left(\mathbf{x}^- - \mathbf{x}^A\right) = C_0^-$, so no queries are required.

Theorems 8.6 and 8.7 show that η-additive optimality and ϵ-multiplicative optimality require $\Omega\left(L_\eta^{(+)} + D\right)$ and $\Omega\left(L_\epsilon^{(*)} + D\right)$ queries, respectively. Thus, the K-STEP MULTILINESEARCH algorithm (Algorithm 8.3) has close to the optimal query complexity for weighted ℓ_1-costs with its $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon} D\right)$ queries. This lower bound also applies to any ℓ_p cost with $p > 1$, but in Section 8.3 we present tighter lower bounds for $p > 1$ that substantially exceed this result for some ranges of ϵ and any range of η.

8.2.1.3 Special Cases

Here we present a number of special cases that require minor modifications to Algorithms 8.1 and 8.3 by adding preprocessing steps.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

Revisiting Linear Classifiers

Lowd & Meek originally developed a method for reverse engineering linear classifiers for a (weighted) ℓ_1 cost. First their method isolates a sequence of points from $\mathbf{x}^-$ to $\mathbf{x}^A$ that cross the classifier's boundary, and then it estimates the hyperplane's parameters using D binary line searches. However, as a consequence of the ability to efficiently minimize our objective when $\mathcal{X}_f^+$ is convex, we immediately have an alternative method for linear classifiers (i.e., halfspaces). Because linear classifiers are a special case of convex-inducing classifiers, Algorithm 8.2 can be applied, and our K-STEP MULTILINESEARCH algorithm improves on complexity of their reverse-engineering technique's $\mathcal{O}(L_\epsilon \cdot D)$ queries and applies to a broader family of classifiers.

While Algorithm 8.2 has better complexity, it uses $2 \cdot D$ search directions rather than the D directions used in the approach of Lowd & Meek, which may require our technique to issue more queries in some practical settings. However, for some restrictive classifier families, it is also possible to eliminate search directions if they can be proven to be infeasible based on the current set of queries. For instance, given a set $\mathbb{W}$ of search directions, t queries $\left\{\mathbf{x}^{(i)}\right\}_{i=1}^t$ and their corresponding responses $\left\{y^{(i)}\right\}_{i=1}^t$, a search direction $\mathbf{e}$ can be eliminated from $\mathbb{W}$ if for all $C_t^+ \leq \alpha < C_t^-$ there does not exist any classifier $f \in \mathcal{F}$ consistent with all previous queries (i.e., $f\left(\mathbf{x}^-\right) = \text{"}-\text{"}$, $f\left(\mathbf{x}^A\right) = \text{"}+\text{"}$ and for all $i \in \{1, \ldots, t\}$, $f\left(\mathbf{x}^{(i)}\right) = y^{(i)}$) that also has $f(\alpha \cdot \mathbf{e}) = \text{"}-\text{"}$ and $f(\alpha \cdot \mathbf{i}) = \text{"}+\text{"}$ for every $\mathbf{i} \in \mathbb{W} \setminus \{\mathbf{e}\}$). That is, $\mathbf{e}$ is feasible if and only if it is the only search direction among the set of remaining search directions, $\mathbb{W}$, that would be classified as a negative for a cost α by some consistent classifier. Further, since subsequent queries only restrict the feasible space of α and the set of consistent classifiers $\hat{\mathcal{F}}$, pruning these infeasible directions is sound for the remainder of the search.

For restrictive families of convex-inducing classifiers, these feasibility conditions can be efficiently verified and may be used to prune search directions without issuing further queries. In fact, for the family of linear classifiers written as $f(\mathbf{x}) = \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right)$ for a normal vector $\mathbf{w}$ and displacement b, the above conditions become a set of linear inequalities along with quadratic inequalities corresponding to the constraint involving search directions. This can be cast as the following optimization program with respect to α, $\mathbf{w}$, and b:

$$
\begin{aligned}
\min_{\alpha,\mathbf{w},b} \quad & \alpha \cdot \mathbf{w}^\top \mathbf{e} + b \\
& \alpha \in [C_t^+, C_t^-) \\
& \mathbf{w}^\top \mathbf{x}^- + b \leq 0 \\
\text{s.t.} \quad & \mathbf{w}^\top \mathbf{x}^A + b \geq 0 \\
& y^i(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 0 \qquad \forall\, i \in \{1, \ldots, t\} \\
& \alpha \cdot \mathbf{w}^\top \mathbf{i} + b \geq 0 \qquad \forall\, \mathbf{i} \neq \mathbf{e} \in \mathbb{W}
\end{aligned}
$$

If the resulting minimum is less than zero, direction $\mathbf{e}$ is feasible; otherwise, it can be pruned. Such programs can be efficiently solved and may allow the adversary to rapidly eliminate infeasible search directions without issuing additional queries. However, refining these pruning procedures further is beyond the scope of this chapter.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

Extending MULTILINESEARCH *Algorithms to $c_d = \infty$ or $c_d = 0$ Weights*
In Algorithm 8.2, we reweighted the d^{th} axis-aligned directions by a factor $\frac{1}{c_d}$ to make unit cost vectors by implicitly assuming $c_d \in (0, \infty)$. The case where $c_d = \infty$ (e.g., immutable features) is dealt with by simply removing those features from the set of search directions $\mathbb{W}$ used in the MULTILINESEARCH. In the case when $c_d = 0$ (e.g., useless features), MULTILINESEARCH-like algorithms no longer ensure near-optimality because they implicitly assume that cost balls are bounded sets. If $c_d = 0$, $\mathbb{B}^0(A)$ is no longer a bounded set, and 0 cost can be achieved if $\mathcal{X}_f^-$ anywhere intersects the subspace spanned by the 0-cost features—this makes near-optimality unachievable unless a negative 0-cost instance can be found. In the worst case, such an instance could be arbitrarily far in any direction within the 0-cost subspace, making search for such an instance intractable. Nonetheless, one possible search strategy is to assign all 0-cost features a nonzero weight that decays quickly toward 0 (e.g., $c_d = 2^{-t}$ in the t^{th} iteration) as we repeatedly rerun MULTILINESEARCH on the altered objective for T iterations. The algorithm will either find a negative instance that only alters 0-cost features (and hence is a 0-*IMAC*), or it terminates with a nonzero cost instance, which is an ϵ-*IMAC* if no 0-cost negative instances exist. This algorithm does not ensure near-optimality, but may be suitable for practical settings using some fixed T runs.

Lack of an Initial Lower Bound
Thus far, to find an ϵ-*IMAC* the algorithms we presented searched between initial bounds C_0^+ and C_0^-, but in general, C_0^+ may not be known to a real-world adversary. We now present an algorithm called SPIRALSEARCH that efficiently establishes a lower bound on the *MAC* if one exists. This algorithm performs a halving search on the exponent along a single direction to find a positive example and then queries the remaining directions at this candidate bound. Either the lower bound is verified or directions that were positive can be pruned for the remainder of the search.

ALGORITHM 8.4 SPIRALSEARCH

spiral $\left(\mathbb{W}, \mathbf{x}^A, C_0^-\right)$
$t \leftarrow 0$ and $\mathbb{V} \leftarrow \emptyset$
repeat
  Choose a direction $\mathbf{w} \in \mathbb{W}$
  Remove $\mathbf{w}$ from $\mathbb{W}$ and $\mathbb{V} \leftarrow \mathbb{V} \cup \{\mathbf{w}\}$
  Query: $f_{\mathbf{w}} \leftarrow f\left(\mathbf{x}^A + C_0^- \cdot 2^{-2^t} \cdot \mathbf{w}\right)$
  if $f_{\mathbf{w}} =$ "$-$" **then begin**
    $\mathbb{W} \leftarrow \mathbb{W} \cup \{\mathbf{w}\}$ and $\mathbb{V} \leftarrow \emptyset$
    $t \leftarrow t + 1$
  end if
until $\mathbb{W} = \emptyset$
$C_0^+ \leftarrow C_0^- \cdot 2^{-2^t}$
if $t > 0$ **then** $C_0^- \leftarrow C_0^- \cdot 2^{-2^{t-1}}$
return: $(\mathbb{V}, C_0^+, C_0^-)$

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

At the t^{th} iteration of SpiralSearch, a direction is selected and queried at the candidate lower bound of $(C_0^-)2^{-2^t}$. If the query is positive, that direction is added to the set $\mathbb{V}$ of directions consistent with the lower bound. Otherwise, all positive directions in $\mathbb{V}$ are pruned, a new upper bound is established, and the candidate lower bound is lowered with an exponentially decreasing exponent. By definition of the MAC, this algorithm will terminate after $t = \left\lceil \log_2 \log_2 \frac{C_0^-}{MAC(f,A)} \right\rceil$ iterations. Further, in this algorithm, multiple directions are probed only during iterations with positive queries, and it makes at most one positive query for each direction. Thus, given that some lower bound $C_0^+ > 0$ does exist, SpiralSearch will establish a lower bound with $\mathcal{O}\left(L'_\epsilon + D\right)$ queries, where L'_ϵ is given by Equation (8.6) defined using $C_0^+ = MAC\,(f, A)$: the largest possible lower bound.

This algorithm can be used as a precursor to any of the previous searches[7] and can be adapted to additive optimality by halving the lower bound instead of the exponent (see Section 8.1.3). Upon completion, the upper and lower bounds it establishes have a multiplicative gap of $2^{2^{t-1}}$ for $t > 0$ or 2 for $t = 0$. From the definition of t provided above in terms of the MAC, MultiLineSearch can hence proceed using $L_\epsilon = L'_\epsilon$. Further, the search directions pruned by SpiralSearch are also invalid for the subsequent MultiLineSearch so the set $\mathbb{V}$ returned by SpiralSearch will be used as the initial set $\mathbb{W}$ for the subsequent search. Thus, the query complexity of the subsequent search is the same as if it had started with the best possible lower bound.

Lack of a Negative Example

The MultiLineSearch algorithms can also naturally be adapted to the case when the adversary has no negative example $\mathbf{x}^-$. This is accomplished by querying ℓ_1 balls of doubly exponentially increasing cost until a negative instance is found. During the t^{th} iteration, the adversary probes along every search direction at a cost $(C_0^+)2^{2^t}$; either all probes are positive (a new lower bound), or at least one is negative (a new upper bound) and search can terminate. Once a negative example is located (having probed for T iterations), we must have $(C_0^+)2^{2^{T-1}} < MAC\,(f, A) \leq (C_0^+)2^{2^T}$; thus, $T = \left\lceil \log_2 \log_2 \left(\frac{MAC(f,A)}{C_0^+} \right) \right\rceil$. After this preprocessing, the adversary can subsequently perform MultiLineSearch with $C_0^+ = 2^{2^{T-1}}$ and $C_0^- = 2^{2^T}$; i.e., $\log_2 (G_0) = 2^{T-1}$. This precursor step requires at most $|\mathbb{W}| \cdot T$ queries to initialize the MultiLineSearch algorithm with a gap such that $L_\epsilon = \left\lceil (T-1) + \log_2 \left(\frac{1}{\log_2(1+\epsilon)} \right) \right\rceil$ according to Equation (8.6).

If there is neither an initial upper bound or lower bound, the adversary can proceed by probing each search direction at unit cost using additional $|\mathbb{W}|$ queries. This will either establish an upper or lower bound, and the adversary can then proceed accordingly.

[7] If no lower bound on the cost exists, no algorithm can find an ϵ-$IMAC$. As presented, this algorithm would not terminate, but in practice, the search would be terminated after sufficiently many iterations.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

8.2.2 ϵ-*IMAC* Learning for a Convex $\mathcal{X}_f^-$

In this section, we minimize convex cost function A with bounded cost balls (we focus on weighted ℓ_1 costs in Equation 8.1) when the feasible set $\mathcal{X}_f^-$ is convex. Any convex function can be efficiently minimized within a known convex set (e.g., using an ellipsoid method or interior point methods; see Boyd & Vandenberghe 2004). However, in the near-optimal evasion problem, the convex set is only accessible via membership queries. We use a randomized polynomial algorithm of Bertsimas & Vempala (2004) to minimize the cost function A given an initial point $\mathbf{x}^- \in \mathcal{X}_f^-$. For any fixed cost, C^t, we use their algorithm to determine (with high probability) whether $\mathcal{X}_f^-$ intersects with $\mathbb{B}^{C^t}(A)$; i.e., whether C^t is a new lower or upper bound on the *MAC*. With high probability, this approach can find an ϵ-*IMAC* in no more than L_ϵ repetitions using binary search. The following theorem is the main result of this section.

THEOREM 8.8 *Let cost function A be convex and have bounded balls; i.e., bounded sublevel sets. Let the feasible set $\mathcal{X}_f^-$ be convex and assume there is some $r > 0$ and $\mathbf{y} \in \mathcal{X}_f^-$ such that $\mathcal{X}_f^-$ contains the cost ball $\mathbb{B}^r(A; \mathbf{y})$. Then given access to an oracle returning separating hyperplanes for the A cost balls, Algorithm 8.7 will find an ϵ-IMAC using $\mathcal{O}^*\left(D^5\right)$ queries with high probability.*[8]

The proof of this result is outlined in the remainder of this section and is based on Bertsimas & Vempala (2004, theorem 14). We first introduce their randomized ellipsoid algorithm, then we elaborate on their procedure for efficient sampling from a convex body, and finally we present our application to optimization. In this section, we focus only on weighted ℓ_1 costs (Equation 8.1) and return to more general cases in Section 8.3.2.

8.2.2.1 Intersection of Convex Sets

Bertsimas & Vempala present a query-based procedure for determining whether two convex sets (e.g., $\mathcal{X}_f^-$ and $\mathbb{B}^{C^t}(A_1)$) intersect. Their IntersectSearch procedure, which we present as Algorithm 8.5 (see Figure 8.4) is a randomized ellipsoid method for determining whether there is an intersection between two bounded convex sets: $\mathbb{P}$ is only accessible through membership queries, and $\mathbb{B}$ provides a separating hyperplane for any point not in $\mathbb{B}$. They use efficient query-based approaches to uniformly sample from $\mathbb{P}$ to obtain sufficiently many samples such that cutting $\mathbb{P}$ through the centroid of these samples with a separating hyperplane from $\mathbb{B}$ significantly reduces the volume of $\mathbb{P}$ with high probability. Their technique thus constructs a sequence of progressively smaller feasible sets $\mathbb{P}^{(s)} \subseteq \mathbb{P}^{(s-1)}$ until either the algorithm finds a point in $\mathbb{P} \cap \mathbb{B}$ or it is highly likely that the intersection is empty.

As noted earlier, the cost optimization problem reduces to finding the intersection between $\mathcal{X}_f^-$ and $\mathbb{B}^{C^t}(A_1)$. Though $\mathcal{X}_f^-$ may be unbounded, we are minimizing a cost with bounded cost balls, so we can instead use the set $\mathbb{P}^{(0)} = \mathcal{X}_f^- \cap \mathbb{B}^{2R}\left(A_1; \mathbf{x}^-; \mathbf{x}^-\right)$ (where $R = A\left(\mathbf{x}^- - \mathbf{x}^A\right) > C^t$), which is a (convex) subset of $\mathcal{X}_f^-$. Since, by the triangle

[8] $\mathcal{O}^*(\cdot)$ denotes the standard complexity notation $\mathcal{O}(\cdot)$ without logarithmic terms. The dependence on $\epsilon*$ is in these logarithmic terms; see Bertsimas & Vempala (2004) for details.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

ALGORITHM 8.5 INTERSECT SEARCH

IntersectSearch $\left(\mathbb{P}^{(0)}, \mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}^{(0)}\right\}, \mathbf{x}^A, C\right)$
for $s = 1$ to T **do begin**
(1) Generate $2N$ samples $\left\{\mathbf{x}^{(j)}\right\}_{j=1}^{2N}$
Choose $\mathbf{x}$ from $\mathbb{Q}$
$\mathbf{x}^{(j)} \leftarrow HitRun\left(\mathbb{P}^{(s-1)}, \mathbb{Q}, \mathbf{x}^{(j)}\right)$
(2) If any $\mathbf{x}^{(j)}$, $A\left(\mathbf{x}^{(j)} - \mathbf{x}^A\right) \leq C$ terminate the for-loop
(3) Put samples into 2 sets of size N
$\mathbb{R} \leftarrow \left\{\mathbf{x}^{(j)}\right\}_{j=1}^{N}$ and $\mathbb{S} \leftarrow \left\{\mathbf{x}^{(j)}\right\}_{j=N+1}^{2N}$
(4) $\mathbf{z}^{(s)} \leftarrow \frac{1}{N}\sum_{\mathbf{x}^{(j)} \in \mathbb{R}} \mathbf{x}^{(j)}$
(5) Compute $\mathbb{H}^{\left(\mathbf{h}\left(\mathbf{z}^{(s)}\right), \mathbf{z}^{(s)}\right)}$ using Equation (8.11)
(6) $\mathbb{P}^{(s)} \leftarrow \mathbb{P}^{(s-1)} \cap \mathbb{H}^{\left(\mathbf{h}(\mathbf{z}), \mathbf{z}^{(s)}\right)}$
(7) Keep samples in $\mathbb{P}^{(s)}$
$\mathbb{Q} \leftarrow \mathbb{S} \cap \mathbb{P}^{(s)}$
end for
Return: the found $[\mathbf{x}^{(j)}, \mathbb{P}^{(s)}, \mathbb{Q}]$; or No Intersect

ALGORITHM 8.6 HIT-AND-RUN SAMPLING

HitRun $\left(\mathbb{P}, \left\{\mathbf{y}^{(j)}\right\}, \mathbf{x}^{(0)}\right)$
for $i = 1$ to K **do begin**
(1) Choose a random direction:
$\nu_j \sim \mathrm{N}(0, 1)$
$\mathbf{v} \leftarrow \sum_j \nu_j \cdot \mathbf{y}^{(j)}$
(2) Sample uniformly along $\mathbf{v}$ using rejection sampling:
Choose $\hat{\omega}$ s.t. $\mathbf{x}^{(i-1)} + \hat{\omega} \cdot \mathbf{v} \notin \mathbb{P}$
repeat
$\omega \sim Unif\,(0, \hat{\omega})$
$\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i-1)} + \omega \cdot \mathbf{v}$
$\hat{\omega} \leftarrow \omega$
until $\mathbf{x}^{(i)} \in \mathbb{P}$
end for
Return: $\mathbf{x}^{(K)}$

Figure 8.4 Algorithms IntersectSearch and Hit-and-Run are used for the randomized ellipsoid algorithm of Bertsimas & Vempala (2004). IntersectSearch is used to find the intersection between a pair of convex sets: $\mathbb{P}^{(0)}$ is queryable and $\mathbb{B}$ provides a separating hyperplane from Equation (8.11). Note that the Rounding algorithm discussed in Section 8.2.2.2 can be used as a preprocessing step so that $\mathbb{P}^{(0)}$ is near-isotropic and to obtain the samples for $\mathbb{Q}$. The Hit-and-Run algorithm is used to efficiently obtain uniform samples from a bounded near-isotropic convex set, $\mathbb{P}$, based on a set of uniform samples from it, $\left\{\mathbf{y}^{(j)}\right\}$, and a starting point $\mathbf{x}^{(0)}$.

inequality, the ball $\mathbb{B}^{2R}\left(A_1; \mathbf{x}^-\right)$ centered at $\mathbf{x}^-$ envelops all of $\mathbb{B}^{C^t}\left(A_1; \mathbf{x}^A\right)$ centered at $\mathbf{x}^A$, the set $\mathbb{P}^{(0)}$ contains the entirety of the desired intersection, $\mathcal{X}_f^- \cap \mathbb{B}^{C^t}(A_1)$, if it exists. We also assume that there is some $r > 0$ such that there is an r-ball contained in the convex set $\mathcal{X}_f^-$; i.e., there exists $\mathbf{y} \in \mathcal{X}_f^-$ such that the r-ball centered at $\mathbf{y}$, $\mathbb{B}^r(A_1; \mathbf{y})$, is a subset of $\mathcal{X}_f^-$. This assumption both ensures that $\mathcal{X}_f^-$ has a non-empty interior (a requirement for the Hit-and-Run algorithm discussed later) and provides a stopping condition for the overall intersection search algorithm.

The foundation of Bertsimas & Vempala's search algorithm is the ability to sample uniformly from an unknown but bounded convex body by means of the Hit-and-Run random walk technique introduced by Smith (1996) (Algorithm 8.6). Given an instance $\mathbf{x}^{(j)} \in \mathbb{P}^{(s-1)}$, Hit-and-Run selects a random direction $\mathbf{v}$ through $\mathbf{x}^{(j)}$ (we revisit the selection of $\mathbf{v}$ in Section 8.2.2.2). Since $\mathbb{P}^{(s-1)}$ is a bounded convex set, the set $\mathbb{W} = \left\{\omega \geq 0 \;\middle|\; \mathbf{x}^{(j)} + \omega\mathbf{v} \in \mathbb{P}^{(s-1)}\right\}$ is a bounded interval (i.e., there is some $\hat{\omega} \geq 0$ such that $\mathbb{W} \subseteq [0, \hat{\omega}]$) that indexes all feasible points along direction $\mathbf{v}$ through $\mathbf{x}^{(j)}$. Sampling ω uniformly from $\mathbb{W}$ yields the next step of the random walk: $\mathbf{x}^{(j)} + \omega\mathbf{v}$. Even though $\hat{\omega}$ is generally unknown, it can be upper bounded, and ω can be sampled using rejection sampling along the interval as demonstrated in Algorithm 8.6. As noted earlier, this

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

random walk will not make progress if the interior of $\mathbb{P}^{(s-1)}$ is empty (which we preclude by assuming that $\mathcal{X}_f^-$ contains an r-ball), and efficient sampling also requires that $\mathbb{P}^{(s-1)}$ is sufficiently round. However, under the conditions discussed in Section 8.2.2.2, the Hit-and-Run random walk generates a sample uniformly from the convex body after $\mathcal{O}^*\left(D^3\right)$ steps (Lovász & Vempala 2004). We now detail the overall IntersectSearch procedure (Algorithm 8.5) and then discuss the mechanism used to maintain efficient sampling after each successive cut. It is worth noting that Algorithm 8.5 requires $\mathbb{P}^{(0)}$ to be in near-isotropic position and that $\mathbb{Q}$ is a set of samples from it; these requirements are met by using the Rounding algorithm of Lovász & Vempala discussed at the end of Section 8.2.2.2.

Randomized Ellipsoid Method

We use Hit-and-Run to obtain $2N$ samples $\left\{\mathbf{x}^{(j)}\right\}$ from $\mathbb{P}^{(s-1)} \subseteq \mathcal{X}_f^-$ for a single phase of the randomized ellipsoid method. If any satisfy the condition $A\left(\mathbf{x}^{(j)} - \mathbf{x}^A\right) \leq C^t$, then $\mathbf{x}^{(j)}$ is in the intersection of $\mathcal{X}_f^-$ and $\mathbb{B}^{C^t}(A_1)$, and the procedure is complete. Otherwise, the search algorithm must significantly reduce the size of $\mathbb{P}^{(s-1)}$ without excluding any of $\mathbb{B}^{C^t}(A_1)$ so that sampling concentrates toward the desired intersection (if it exists)—for this we need a separating hyperplane for $\mathbb{B}^{C^t}(A_1)$. For any point $\mathbf{y} \notin \mathbb{B}^{C^t}(A_1)$, the (sub)gradient denoted as $\mathbf{h}(\mathbf{y})$ of the weighted ℓ_1 cost is given by

$$[\mathbf{h}(\mathbf{y})]_f = c_f \cdot \operatorname{sign}\left(y_f - x_f^A\right). \tag{8.10}$$

and thus the hyperplane specified by $\left\{\mathbf{x} \;\middle|\; (\mathbf{x}-\mathbf{y})^\top \mathbf{h}(\mathbf{y})\right\}$ is a separating hyperplane for $\mathbf{y}$ and $\mathbb{B}^{C^t}(A_1)$.

To achieve sufficient progress, the algorithm chooses a point $\mathbf{z} \in \mathbb{P}^{(s-1)}$ so that cutting $\mathbb{P}^{(s-1)}$ through $\mathbf{z}$ with the hyperplane $\mathbf{h}(\mathbf{z})$ eliminates a significant fraction of $\mathbb{P}^{(s-1)}$. To do so, $\mathbf{z}$ must be centrally located within $\mathbb{P}^{(s-1)}$. We use the empirical centroid of half of the samples in $\mathbb{R}$: $\mathbf{z} = \frac{1}{N}\sum_{\mathbf{x}\in\mathbb{R}} \mathbf{x}$ (the other half will be used in Section 8.2.2.2). We cut $\mathbb{P}^{(s-1)}$ with the hyperplane $\mathbf{h}(\mathbf{z})$ through $\mathbf{z}$; i.e., $\mathbb{P}^{(s)} = \mathbb{P}^{(s-1)} \cap \mathbb{H}^{(\mathbf{h}(\mathbf{z}),\mathbf{z})}$ where $\mathbb{H}^{(\mathbf{h}(\mathbf{z}),\mathbf{z})}$ is the halfspace

$$\mathbb{H}^{(\mathbf{h}(\mathbf{z}),\mathbf{z})} = \left\{\mathbf{x} \;\middle|\; \mathbf{x}^\top \mathbf{h}(\mathbf{z}) < \mathbf{z}^\top \mathbf{h}(\mathbf{z})\right\}. \tag{8.11}$$

As shown by Bertsimas & Vempala, this cut achieves $vol\left(\mathbb{P}^{(s)}\right) \leq \frac{2}{3} vol\left(\mathbb{P}^{(s-1)}\right)$ with high probability if $N = \mathcal{O}^*(D)$ and $\mathbb{P}^{(s-1)}$ is near-isotropic (see Section 8.2.2.2). Since the ratio of volumes between the initial circumscribing and inscribed balls of the feasible set is $\left(\frac{R}{r}\right)^D$, the algorithm can terminate after $T = \mathcal{O}\left(D \log\left(\frac{R}{r}\right)\right)$ unsuccessful iterations with a high probability that the intersection is empty.

Because every iteration in Algorithm 8.5 requires $N = \mathcal{O}^*(D)$ samples, each of which needs $K = \mathcal{O}^*\left(D^3\right)$ random walk steps, and there are $T = \mathcal{O}^*(D)$ iterations, the total number of membership queries required by Algorithm 8.5 is $\mathcal{O}^*\left(D^5\right)$.

8.2.2.2 Sampling from a Queryable Convex Body

In the randomized ellipsoid method, random samples are used for two purposes: estimating the convex body's centroid and maintaining the conditions required for

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

the Hit-and-Run sampler to efficiently generate points uniformly from a sequence of shrinking convex bodies. Until now, we assumed the Hit-and-Run random walk efficiently produces uniformly random samples from any bounded convex body $\mathbb{P}$ using $K = \mathcal{O}^*\left(D^3\right)$ membership queries. However, if the body is asymmetrically elongated, randomly selected directions will rarely align with the long axis of the body, and the random walk will take small steps (relative to the long axis) and mix slowly in $\mathbb{P}$. For the sampler to mix effectively, the convex body $\mathbb{P}$ has to be sufficiently round, or more formally near-isotropic; i.e., for any unit vector $\mathbf{v}$,

$$\frac{1}{2}vol\left(\mathbb{P}\right) \;\leq\; \mathrm{E}_{\mathbf{x}\sim\mathbb{P}}\left[\left(\mathbf{v}^\top\left(\mathbf{x} - \mathrm{E}_{\mathbf{x}\sim\mathbb{P}}\left[\mathbf{x}\right]\right)\right)^2\right] \;\leq\; \frac{3}{2}vol\left(\mathbb{P}\right). \tag{8.12}$$

If the body is not near-isotropic, $\mathcal{X}$ can be rescaled with an appropriate affine transformation $\mathbf{T}$ so the resulting transformed body $\mathbb{P}' = \{\mathbf{T}\mathbf{x} \mid \mathbf{x} \in \mathbb{P}\}$ is near-isotropic. With sufficiently many samples from $\mathbb{P}$ we can estimate $\mathbf{T}$ as their empirical covariance matrix. Instead, we rescale $\mathcal{X}$ implicitly using a technique described by Bertsimas & Vempala (2004). We maintain a set $\mathbb{Q}$ of sufficiently many uniform samples from the body $\mathbb{P}^{(s)}$, and in the Hit-and-Run algorithm (Algorithm 8.6), we sample the direction $\mathbf{v}$ based on this set. Intuitively, because the samples in $\mathbb{Q}$ are distributed uniformly in $\mathbb{P}^{(s)}$, the directions we sample based on the points in $\mathbb{Q}$ implicitly reflect the covariance structure of $\mathbb{P}^{(s)}$. This is equivalent to sampling the direction $\mathbf{v}$ from a normal distribution with zero mean and covariance of $\mathbb{P}$.

Further, the set $\mathbb{Q}$ must retain sufficiently many samples from $\mathbb{P}^{(s)}$ after each cut: $\mathbb{P}^{(s)} \leftarrow \mathbb{P}^{(s-1)} \cap \mathbb{H}^{\left(\mathbf{h}\left(\mathbf{z}^{(s)}\right),\mathbf{z}^{(s)}\right)}$. To do so, we initially resample $2N$ points from $\mathbb{P}^{(s-1)}$ using Hit-and-Run—half of these, $\mathbb{R}$, are used to estimate the centroid $\mathbf{z}^{(s)}$ for the cut, and the other half, $\mathbb{S}$, are used to repopulate $\mathbb{Q}$ after the cut. Because $\mathbb{S}$ contains independent uniform samples from $\mathbb{P}^{(s-1)}$, those in $\mathbb{P}^{(s)}$ after the cut constitute independent uniform samples from $\mathbb{P}^{(s)}$ (i.e., rejection sampling). By choosing N sufficiently large, the cut will be sufficiently deep, and there will be sufficiently many points to resample $\mathbb{P}^{(s)}$ after the cut.

Finally, for this sampling approach to succeed, we need the initial set $\mathbb{P}^{(0)}$ to be transformed into near-isotropic position, and we also need an initial set $\mathbb{Q}$ of uniform samples from the transformed $\mathbb{P}^{(0)}$ as input to Algorithm 8.5. However, in the near-optimal evasion problem, we only have a single point $\mathbf{x}^- \in \mathcal{X}_f^-$ and our set, $\mathbb{P}^{(0)}$, need not be near-isotropic. Fortunately, there is an iterative procedure that uses the Hit-and-Run algorithm to simultaneously transform the initial convex set, $\mathbb{P}^{(0)}$, into a near-isotropic position and construct our initial set of samples, $\mathbb{Q}$. This algorithm, the Rounding algorithm as described by Lovász & Vempala (2003), uses $\mathcal{O}^*\left(D^4\right)$ membership queries to find a transformation that places $\mathbb{P}^0$ into a near-isotropic position and produces an initial set of samples from it. We use this as a preprocessing step for Algorithms 8.5 and 8.7; that is, given $\mathcal{X}_f^-$ and $\mathbf{x}^- \in \mathcal{X}_f^-$, we construct $\mathbb{P}^{(0)} = \mathcal{X}_f^- \cap \mathbb{B}^{2R}\left(A_1; \mathbf{x}^-\right)$ and then can use the Rounding algorithm to transform $\mathbb{P}^{(0)}$ and produce an initial uniform sample from it; i.e., $\mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}^{(0)}\right\}$. These sets are then the inputs to our search algorithms.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

8.2.2.3 Optimization over ℓ_1 Balls

We now revisit the outermost optimization loop (for searching the minimum feasible cost) of the algorithm to optimize the naive approach, which repeats the intersection search at each step of the binary search over cost balls. These improvements are reflected in our final procedure SetSearch in Algorithm 8.7 (as with previous binary search procedures, this algorithm can be trivially adapted for η-additive optimality simply by changing its stopping criterion and proposal step as explained in Section 8.1.3)—the total number of queries required is also $\mathcal{O}^*\left(D^5\right)$ since the algorithm only takes L_ϵ binary search steps (see Figure 8.5). Again, Algorithm 8.7 requires $\mathbb{P}$ to be near-isotropic and that $\mathbb{Q}$ is a set of samples from it, which is accomplished by the Rounding algorithm discussed at the end of Section 8.2.2.2. First, notice that $\mathbf{x}^A$ and $\mathbf{x}^-$ are the same for every iteration of the optimization procedure. Further, in each iteration of Algorithm 8.7, the new set, $\mathbb{P}$, remains near-isotropic, and the new $\mathbb{Q}$ is a set of samples from it since the sets returned by Algorithm 8.5 retain these properties. Thus, the set, $\mathbb{P}$, and the set of samples, $\mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}\right\}$, maintained by Algorithm 8.7 are sufficient to initialize IntersectSearch at each stage of its overall binary search over C^t, and we only need to execute the Rounding procedure once as a preprocessing step rather than re-invoking it before every invocation of IntersectSearch. Second, the separating hyperplane $\mathbf{h}(\mathbf{y})$ given by Equation (8.10) does not depend on the target cost C^t but only on $\mathbf{x}^A$, the common center of all the ℓ_1 balls used in this search. In fact, the separating hyperplane at point $\mathbf{y}$ is valid for all ℓ_1-balls of cost $C < A\left(\mathbf{y} - \mathbf{x}^A\right)$. Further, if $C < C^t$, then $\mathbb{B}^C(A_1) \subseteq \mathbb{B}^{C^t}(A_1)$. Thus, the final state from a successful call to IntersectSearch for the C^t-ball can be used as the starting state for any subsequent call to IntersectSearch for all $C < C^t$. Hence, in Algorithm 8.7, we update $\mathbb{P}$ and $\mathbb{Q}$ only when Algorithm 8.5 succeeds.

8.3 Evasion for General ℓ_p Costs

Here we further extend ϵ-*IMAC* searchability over the family of convex-inducing classifiers to the full family of ℓ_p costs for any $0 < p \leq \infty$. As we demonstrate in this section, many ℓ_p costs are not generally ϵ-*IMAC* searchable for all $\epsilon > 0$ over the family of convex-inducing classifiers (i.e., we show that finding an ϵ-*IMAC* for this family can require exponentially many queries in D and L_ϵ). In fact, only the weighted ℓ_1 costs have known (randomized) polynomial query strategies when either the positive or negative set is convex.

8.3.1 Convex Positive Set

We explore the ability of the MultiLineSearch and K-step MultiLineSearch algorithms presented in Section 8.2.1 to find solutions to the near-optimal evasion problem for ℓ_p cost functions with $p \neq 1$. Particularly for $p > 1$, we explore the consequences of using the MultiLineSearch algorithms using more search directions than just the $2 \cdot D$

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

ALGORITHM 8.7 CONVEX $\mathcal{X}_f^-$ SET SEARCH

SetSearch $\left(\mathbb{P}, \mathbb{Q} = \left\{\mathbf{x}^{(j)} \in \mathbb{P}\right\}, \mathbf{x}^A, \mathbf{x}^-, C_0^+, C_0^-, \epsilon\right)$
$\mathbf{x}^\star \leftarrow \mathbf{x}^-$ and $t \leftarrow 0$
while $C_t^- / C_t^+ > 1 + \epsilon$ **do begin**
  $C_t \leftarrow \sqrt{C_t^- \cdot C_t^+}$
  $[\mathbf{x}^\star, \mathbb{P}', \mathbb{Q}'] \leftarrow$ *IntersectSearch* $\left(\mathbb{P}, \mathbb{Q}, \mathbf{x}^A, C_t\right)$
  if intersection found **then begin**
    $C_{t+1}^- \leftarrow A\left(\mathbf{x}^\star - \mathbf{x}^A\right)$ and $C_{t+1}^+ \leftarrow C_t^+$
    $\mathbb{P} \leftarrow \mathbb{P}'$ and $\mathbb{Q} \leftarrow \mathbb{Q}'$
  else
    $C_{t+1}^- \leftarrow C_t^-$ and $C_{t+1}^+ \leftarrow C_t$
  end if
  $t \leftarrow t + 1$
end while
Return: $\mathbf{x}^\star$

Figure 8.5 Algorithm SETSEARCH that efficiently implements the randomized ellipsoid algorithm of Bertsimas & Vempala (2004). SETSEARCH performs a binary search for an ϵ-*IMAC* using the randomized INTERSECTSEARCH procedure to determine, with high probability, whether or not $\mathcal{X}_f^-$ contains any points less than a specified cost, C_t. Note that the ROUNDING algorithm discussed in Section 8.2.2.2 can be used as a preprocessing step so that $\mathbb{P}$ is near-isotropic and to obtain the samples for $\mathbb{Q}$.

axis-aligned directions. Figure 8.6 demonstrates how queries can be used to construct upper and lower bounds on general ℓ_p costs. The following lemma also summarizes well-known bounds on general ℓ_p costs using an ℓ_1 cost.

LEMMA 8.9 *The largest ℓ_p ($p > 1$) ball enclosed within a C-cost ℓ_1 ball has a cost of $C \cdot D^{\frac{1-p}{p}}$ and for $p = \infty$ the cost is $C \cdot D^{-1}$.*

Proof By symmetry, the point $\mathbf{x}^\star$ on the simplex $\left\{\mathbf{x} \in \Re^D \;\middle|\; \sum_{i=1}^D x_i = 1, x_i \geq 0 \forall i\right\}$ that minimizes the ℓ_p norm for any $p > 1$ is

$$\mathbf{x}^\star = \frac{1}{D}(1, 1, \ldots, 1).$$

The ℓ_p norm (cost) of the minimizer is

$$\begin{aligned}\left\|\mathbf{x}^\star\right\|_p &= \frac{1}{D}\left(\sum_{i=1}^D 1^p\right)^{1/p} \\ &= \frac{1}{D} D^{1/p} \\ &= D^{\frac{1-p}{p}}\end{aligned}$$

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

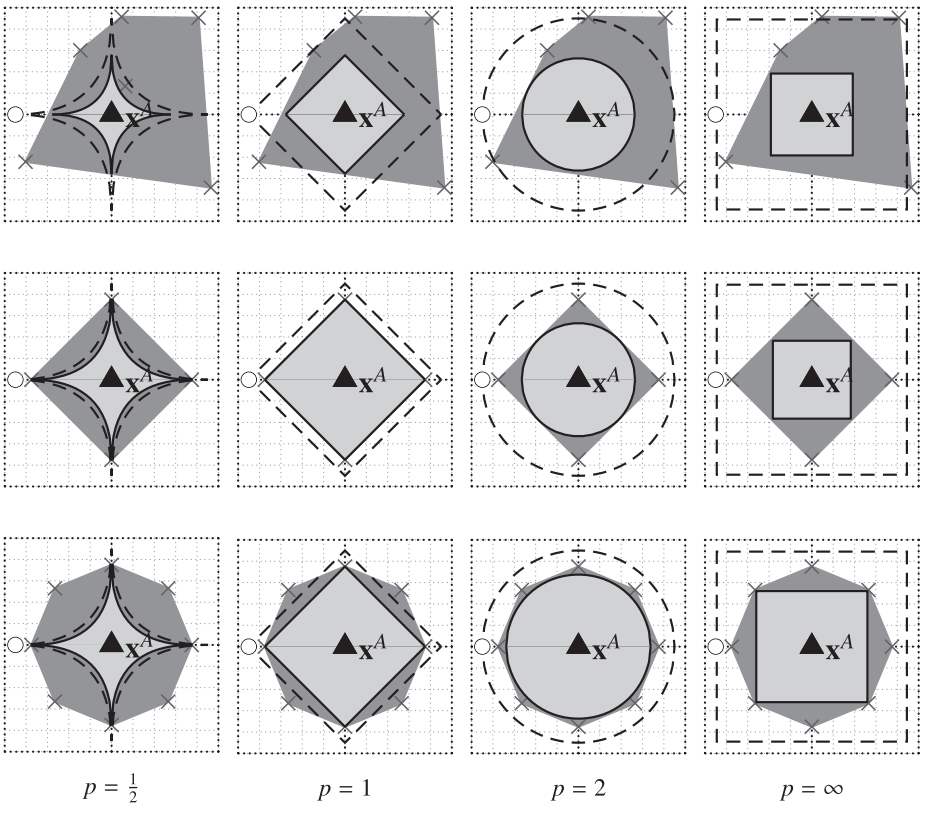

Figure 8.6 The convex hull for a set of queries and the resulting bounding balls for several ℓ_p costs. Each row represents a unique set of positive ("×" points) and negative (black "○" points) queries, and each column shows the implied upper bound (the black dashed line) and lower bound (the black solid line) for a different ℓ_p cost. In the first row, the body is defined by a random set of seven queries, in the second, the queries are along the coordinate axes, and in the third, the queries are around a circle.

for $p \in (1, \infty)$ and is otherwise

$$\begin{aligned} \left\| \mathbf{x}^\star \right\|_\infty &= \max \left[\frac{1}{D}, \frac{1}{D}, \ldots, \frac{1}{D} \right] \\ &= D^{-1}. \end{aligned}$$

□

8.3.1.1 Bounding ℓ_p Balls

In general, suppose one probes along some set of M unit directions, and eventually there is at least one negative point supporting an upper bound of C_0^- and M positive points supporting a lower bound of C_0^+. However, the lower bound provided by those M positive points is the cost of the largest ℓ_p cost ball that fits entirely within their convex hull; let's say this cost is $C^\dagger \leq C_0^+$. To achieve ϵ-multiplicative optimality, we

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

need $\frac{C_0^-}{C^\dagger} \leq 1+\epsilon$, which can be rewritten as

$$\left(\frac{C_0^-}{C_0^+}\right)\left(\frac{C_0^+}{C^\dagger}\right) \leq 1+\epsilon.$$

This divides the problem into two parts. The first ratio C_0^-/C_0^+ is controlled solely by the accuracy ϵ achieved by running the MultiLineSearch algorithm for L_ϵ steps, whereas the second ratio $C_0^+/C^\dagger$ depends only on how well the ℓ_p ball is approximated by the convex hull of the M search directions. These two ratios separate the search task into choosing M and L_ϵ sufficiently so that their product is less than $1+\epsilon$. First we select parameters $\alpha \geq 0$ and $\beta \geq 0$ such that $(1+\alpha)(1+\beta) \leq 1+\epsilon$. Then we choose M so that $\frac{C_0^+}{C^\dagger} = 1+\beta$ and use L_α steps so that MultiLineSearch with M directions will achieve $\frac{C_0^-}{C_0^+} = 1+\alpha$. This process describes a generalized MultiLineSearch that achieves ϵ-multiplicative optimality for costs whose cost balls are not spanned by the convex hull of equi-cost probes along the M search directions.

In the case of $p=1$, we demonstrated in Section 8.2.1 that choosing the $M = 2 \cdot D$ axis-aligned directions $\left\{\pm \mathbf{e}^{(d)}\right\}$ spans the ℓ_1 ball so that $C_0^+/C^\dagger = 1$ (i.e., $\beta = 0$). Thus, choosing $\alpha = \epsilon$ recovers the original multi-line search result.

We now address costs where $\beta > 0$. For a MultiLineSearch algorithm to be efficient, it is necessary that $\frac{C_0^+}{C^\dagger} = 1+\beta$ can be achieved with polynomially many search directions (in D and L_ϵ) for some $\beta \leq \epsilon$; otherwise, $(1+\alpha)(1+\beta) > 1+\epsilon$, and the MultiLineSearch approach cannot succeed for any $\alpha > 0$. Thus, we quantify how many search directions (or queries) are required to achieve $\frac{C_0^+}{C^\dagger} \leq 1+\epsilon$. Note that this ratio is independent of the relative size of these costs, so without loss of generality we only consider bounds for unit-cost balls. Thus, we compute the largest value of $C^\dagger$ that can be achieved for the unit-cost ℓ_p ball (i.e., let $C_0^+ = 1$) within the convex hull of M queries. In particular, we quantify how many queries are required to achieve

$$C^\dagger \geq \frac{1}{1+\epsilon}. \tag{8.13}$$

If this can be achieved with only polynomially many queries, then the generalized MultiLineSearch approach is efficient. More generally,

LEMMA 8.10 *If there exists a configuration of M unit search directions with a convex hull that yields a bound $C^\dagger$ for the cost function A, then* MultiLineSearch *algorithms can use those search directions to achieve ϵ-multiplicative optimality with a query complexity that is polynomial in M and $L_\epsilon^{(*)}$ for any*

$$\epsilon > \frac{1}{C^\dagger} - 1.$$

Moreover, if the M search directions yield $C^\dagger = 1$ for the cost function A, then MultiLineSearch *algorithms can achieve ϵ-multiplicative optimality with a query complexity that is polynomial in M and $L_\epsilon^{(*)}$ for any $\epsilon > 0$.*

Notice that this lemma also reaffirms that, for $p=1$, using the $M = 2 \cdot D$ axis-aligned directions allows MultiLineSearch algorithms to achieve ϵ-multiplicative

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

optimality for any $\epsilon > 0$ with a query complexity that is polynomial in M and $L_\epsilon^{(*)}$ since in this case $C^\dagger = 1$. Also recall that as a consequence of Theorem 8.3, if a particular multiplicative accuracy ϵ cannot be efficiently achieved, then additive optimality cannot be generally achieved for any additive accuracy $\eta > 0$.

8.3.1.2 Multi-Line Search for $0 < p < 1$

A simple result holds here. Namely, since the unit ℓ_1 ball bounds any unit ℓ_p balls with $0 < p < 1$, one can achieve $C_0^+/C^\dagger = 1$ using only the $2 \cdot D$ axis-aligned search directions. Thus, for any $0 < p < 1$, evasion is efficient for any value of $\epsilon > 0$. Whether or not any ℓ_p $(0 < p < 1)$ cost function can be efficiently searched with fewer search directions is an open question.

8.3.1.3 Multi-Line Search for $p > 1$

For this case, one can trivially use the ℓ_1 bound on ℓ_p balls as summarized by the following corollary.

COROLLARY 8.11 *For $1 < p < \infty$ and $\epsilon \in \left(D^{\frac{p-1}{p}} - 1, \infty\right)$ any multi-line search algorithm can achieve ϵ-multiplicative optimality on A_p using $M = 2 \cdot D$ search directions. Similarly for $\epsilon \in (D-1, \infty)$ any multi-line search algorithm can achieve ϵ-multiplicative optimality on A_∞ also using $M = 2 \cdot D$ directions.*

Proof From Lemma 8.9, the largest co-centered ℓ_p ball contained within the unit ℓ_1 ball has radius $D^{\frac{1-p}{p}}$ cost (or D for $p = \infty$). The bounds on ϵ then follow from Lemma 8.10. □

Unfortunately, this result only applies for a range of ϵ that grows with D, which is insufficient for ϵ-*IMAC* searchability. In fact, for some fixed values of ϵ, there is no query strategy that can bound ℓ_p costs using polynomially many queries in D as the following result shows.

THEOREM 8.12 *For $p > 1$, $D > 0$, any initial bounds $0 < C_0^+ < C_0^-$ on the MAC, and $\epsilon \in \left(0, 2^{\frac{p-1}{p}} - 1\right)$ (or $\epsilon \in (0,1)$ for $p = \infty$), all algorithms must submit at least $\alpha_{p,\epsilon}^D$ membership queries (for some constant $\alpha_{p,\epsilon} > 1$) in the worst case to be ϵ-multiplicatively optimal on $\mathcal{F}^{\text{convex,"+"}}$ for ℓ_p costs.*

The proof of this theorem is provided in Appendix D, and the definitions of $\alpha_{p,\epsilon}$ and $\alpha_{\infty,\epsilon}$ are provided by Equations (D.7) and (D.8), respectively. A consequence of this result is that there is no query-based algorithm that can efficiently find an ϵ-*IMAC* of any ℓ_p cost $(p > 1)$ for any *fixed* ϵ within the range $0 < \epsilon < 2^{\frac{p-1}{p}} - 1$ (or $0 < \epsilon < 1$ for $p = \infty$) on the family $\mathcal{F}^{\text{convex,"+"}}$. However, from Theorem 8.11 and Lemma 8.10, multi-line search type algorithms efficiently find the ϵ-*IMAC* of any ℓ_p cost $(p > 1)$ for any $\epsilon \in \left(D^{\frac{p-1}{p}} - 1, \infty\right)$ (or $D - 1 < \epsilon < \infty$ for $p = \infty$). It is generally unclear if efficient algorithms exist for any values of ϵ between these intervals, but in the following section, we derive a stronger bound for the case $p = 2$.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

8.3.1.4 Multi-Line Search for $p = 2$

THEOREM 8.13 *For any $D > 1$, any initial bounds $0 < C_0^+ < C_0^-$ on the MAC, and $0 < \epsilon < \frac{C_0^-}{C_0^+} - 1$, all algorithms must submit at least $\alpha_\epsilon^{\frac{D-2}{2}}$ membership queries (where $\alpha_\epsilon = \frac{(1+\epsilon)^2}{(1+\epsilon)^2-1} > 1$) in the worst case to be ϵ-multiplicatively optimal on $\mathcal{F}^{\text{convex},"+"}$ for ℓ_2 costs.*

The proof of this result is in Appendix D.2.

This result says that there is no algorithm that can generally achieve ϵ-multiplicative optimality for ℓ_2 costs for any *fixed* $\epsilon > 0$ using only polynomially many queries in D since the ratio $\frac{C_0^-}{C_0^+}$ could be arbitrarily large. It may appear that Theorem 8.13 contradicts Corollary 8.11. However, Corollary 8.11 only applies for an interval of ϵ that depends on D; i.e., $\epsilon > \sqrt{D} - 1$. Interestingly, by substituting this lower bound on ϵ into the bound given by Theorem 8.13, the number of required queries for $\epsilon > \sqrt{D} - 1$ need only be

$$M \quad \geq \quad \left(\frac{(1+\epsilon)^2}{(1+\epsilon)^2 - 1}\right)^{\frac{D-2}{2}} \quad = \quad \left(\frac{D}{D-1}\right)^{\frac{D-2}{2}},$$

which is a monotonically increasing function in D that asymptotes at $\sqrt{e} \approx 1.64$. Thus, Theorem 8.13 and Corollary 8.11 are in agreement since for $\epsilon > \sqrt{D} - 1$, Theorem 8.13 only requires at least two queries, which is a trivial bound for all D. Indeed, this occurs because the ϵ considered here is bounded below by a function that increases with D.

A Tighter Bound

The bound derived for Lemma A.1 was sufficient to demonstrate that there is no algorithm that can generally achieve ϵ-multiplicative optimality for ℓ_2 costs for any fixed $\epsilon > 0$. It is, however, possible to construct a tighter lower bound on the number of queries required for ℓ_2 costs, although it is not easy to express this result as an exponential in D. A straightforward way to construct a better lower bound is to make a tighter upper bound on the integral $\int_0^\phi \sin^D(t)\, dt$ as is suggested in Appendix A.2. Namely, the result given in Equation (A.4) upper bounds this integral by

$$\frac{\sin^{D+1}(\phi)}{(D+1)\cos(\phi)},$$

which is tighter for large D and $\phi < \frac{\pi}{2}$. Applying this bound to the covering number result of Theorem 8.13 achieves the following bound on the number of queries required to achieve multiplicative optimality:

$$M \geq \frac{\sqrt{\pi}}{1+\epsilon} \cdot \frac{D \cdot \Gamma\left(\frac{D+1}{2}\right)}{\Gamma\left(1 + \frac{D}{2}\right)} \left(\frac{(1+\epsilon)^2}{(1+\epsilon)^2 - 1}\right)^{\frac{D-1}{2}}. \tag{8.14}$$

While not as obvious as the result presented in Appendix D.2, this bound is also exponential in D for any ϵ. Also, as with the previous result, this bound does not contradict the polynomial result for $\epsilon \geq \sqrt{D} - 1$. For $D = 1$ Equation 8.14 requires exactly two

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

queries (in exact agreement with the number of queries required to bound an ℓ_2 ball in 1-dimension); for $D = 2$ it requires more than π queries (whereas at least four queries are actually required); and for $D > 2$ the bound asymptotes at $\sqrt{2e\pi} \approx 4.13$ queries. Again, this tighter bound does not contradict the efficient result achieved by bounding ℓ_2 balls with ℓ_1 balls.

8.3.2 Convex Negative Set

Algorithm 8.7 generalizes immediately to all weighted ℓ_p costs ($p \geq 1$) centered at $\mathbf{x}^A$ since these costs are convex. For these costs, an equivalent separating hyperplane for $\mathbf{y}$ can be used in place of Equation (8.10). They are given by the equivalent (sub)-gradients for ℓ_p cost balls:

$$h^{(\mathbf{y})}_{p,d} = c_d \cdot \operatorname{sign}\left(y_d - x^A_d\right) \cdot \left(\frac{|y_d - x^A_d|}{A^{(\mathbf{c})}_p\left(\mathbf{y} - \mathbf{x}^A\right)}\right)^{p-1},$$

$$h^{(\mathbf{y})}_{\infty,d} = c_d \cdot \operatorname{sign}\left(y_d - x^A_d\right) \cdot \mathrm{I}\left[|y_d - x^A_d| = A^{(\mathbf{c})}_\infty\left(\mathbf{y} - \mathbf{x}^A\right)\right].$$

By only changing the cost function A and the separating hyperplane $\mathbf{h}(\mathbf{y})$ used for the halfspace cut in Algorithms 8.5 and 8.7, the randomized ellipsoid method can also be applied for any weighted ℓ_p cost $A^{(\mathbf{c})}_p$ with $p > 1$.

For more general convex costs A, every C-cost ball is a convex set (i.e., the sublevel set of a convex function is a convex set; see Boyd & Vandenberghe 2004, Chapter 3) and thus has a separating hyperplane. Further, since for any $D > C$, $\mathbb{B}^C(A) \subseteq \mathbb{B}^D(A)$, the separating hyperplane of the D-cost ball is also a separating hyperplane of the C-cost ball and can be reused in Algorithm 8.7. Thus, this procedure is applicable for any convex cost function, A, so long as one can compute the separating hyperplanes of any cost ball of A for any point $\mathbf{y}$ not in the cost ball.

For nonconvex costs A such as weighted ℓ_p costs with $0 < p < 1$, minimization over a convex set $\mathcal{X}^-_f$ is generally hard. However, there may be special cases when minimizing such a cost can be accomplished efficiently.

8.4 Summary

In this chapter we primarily studied membership query algorithms that efficiently accomplish ϵ-*IMAC* search for convex-inducing classifiers with weighted ℓ_1 costs. When the positive class is convex, we demonstrated efficient techniques that outperform the previous reverse-engineering approaches for linear classifiers in a continuous space. When the negative class is convex, we appied the randomized ellipsoid method introduced by Bertsimas & Vempala to achieve efficient ϵ-*IMAC* search. If the adversary is unaware of which set is convex, it can trivially run both searches to discover an ϵ-*IMAC* with a combined polynomial query complexity; thus, for ℓ_1 costs, the family of convex-inducing classifiers can be efficiently evaded by an adversary; i.e., this family is ϵ-*IMAC* searchable.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

Further, we also extended the study of convex-inducing classifiers to the full family of ℓ_p costs. We showed that $\mathcal{F}^{convex}$ is only generally ϵ-$IMAC$ searchable for both positive and negative convexity for any $\epsilon > 0$ when $p = 1$. For $0 < p < 1$, the MultiLineSearch algorithms of Section 8.2.1 achieve identical results when the positive set is convex, but the nonconvexity of these ℓ_p costs precludes the use of the randomized ellipsoid method when the negative class is convex. The ellipsoid method does provide an efficient solution for convex negative sets when $p > 1$ (since these costs are convex). However, for convex positive sets, we show that for $p > 1$ there is no algorithm that can efficiently find an ϵ-$IMAC$ for all $\epsilon > 0$. Moreover, for $p = 2$, we prove that there is no efficient algorithm for finding an ϵ-$IMAC$ for any fixed value of ϵ.

8.4.1 Open Problems in Near-Optimal Evasion

By investigating near-optimal evasion for the convex-inducing classifiers and ℓ_1 costs, we have significantly expanded the extent of the framework established by Lowd & Meek , but there are still a number of interesting unanswered questions about the near-optimal evasion problem. We summarize the problems we believe are most important and suggest potential directions for pursuing them.

As we showed in this chapter, the current upper bound on the query complexity to achieve near-optimal evasion for the convex positive class is $\mathcal{O}\left(L_\epsilon + \sqrt{L_\epsilon} D\right)$ queries, but the tightest known lower bound is $\mathcal{O}\left(L_\epsilon + D\right)$. Similarly, for the case of convex negative class, the upper bound is given by the randomized ellipsoid method of Bertsimas & Vempala that finds a near-optimal instance with high probability using $\mathcal{O}^*\left(D^5\right)$ queries (ignoring logarithmic terms). In both cases, there is a gap between the upper and lower bound.

Question 8.1 Can we find matching upper and lower bounds for evasion algorithms? Is there a deterministic strategy with polynomial query complexity for all convex-inducing classifiers?

The algorithms we present in this chapter built on the machinery of convex optimization over convex sets, which relies on family of classifiers inducing a convex set. However, many interesting classifiers are not convex-inducing classifiers. Currently, the only known result for non-convex-inducing classifiers is due to Lowd & Meek; they found that linear classifiers on Boolean feature space are 2-$IMAC$ searchable for unweighted ℓ_1 costs. In this case, the classifiers are linear, but the integer-valued domains do not have a usual notion of convexity. This raises questions about the extent to which near-optimal evasion is efficient.

Question 8.2 Are there families larger than the convex-inducing classifiers that are ϵ-$IMAC$ searchable? Are there families outside of the convex-inducing classifiers for which near-optimal evasion is efficient?

A particularly interesting family of classifiers to investigate is the family of support vector machines (SVMs) defined by a particular nonlinear kernel. This popular learning technique can induce nonconvex positive and negative sets (depending on its kernel),

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

but it also has a great deal of structure. An SVM classifier can be nonconvex in its input space $\mathcal{X}$, but it is always linear in its kernel's reproducing kernel Hilbert space (RKHS; see Definition 7.6). However, optimization within the RKHS is complicated because mapping the cost balls into the RKHS destroys their structure and querying in the RKHS is nontrivial. However, SVMs also have additional structure that may facilitate near-optimal evasion. For instance, the usual SVM formulation encourages a sparse representation that could be exploited; i.e., in classifiers with few support vectors, the adversary would only need to find these instances to reconstruct the classifier.

Question 8.3 Is some family of SVMs (e.g., with a known kernel) ϵ-*IMAC* searchable for some ϵ? Can an adversary incorporate the structure of a nonconvex classifier into the ϵ-*IMAC* search?

In addition to studying particular families of classifiers, it is also of interest to further characterize general properties of a family that lead to efficient search algorithms or preclude their existence. As we showed in this chapter, convexity of the induced sets allows for efficient search for some ℓ_p-costs but not others. Aside from convexity, other properties that describe the shape of the induced sets $\mathcal{X}_f^+$ and $\mathcal{X}_f^-$ could be explored. For instance, one could investigate the family of contiguous-inducing classifiers (i.e., classifiers for which either $\mathcal{X}_f^+$ or $\mathcal{X}_f^-$ is a contiguous, or connected, set). However, it appears that this family is not generally ϵ-*IMAC* searchable since it includes induced sets with many locally minimal cost regions, which rule out global optimization procedures like the MultiLineSearch or the randomized ellipsoid method. More generally, for families of classifiers that can induce noncontiguous bodies, ϵ-*IMAC* searchability seems impossible to achieve (disconnected components could be arbitrarily close to $\mathbf{x}^A$) unless the classifiers' structure can be exploited. However, even if near-optimal evasion is generally not possible in these cases, perhaps there are subsets of these families that are ϵ-*IMAC* searchable; e.g., as we discussed for SVMs earlier. Hence, it is important to identify what characteristics make near-optimal evasion inefficient.

Question 8.4 Are there characteristics of nonconvex, contiguous bodies that are indicative of the hardness of the body for near-optimal evasion? Similarly, are there characteristics of noncontiguous bodies that describe their query complexity?

Finally, as discussed in Section 8.1.2, reverse engineering a classifier (i.e., using membership queries to estimate its decision boundary) is a strictly more difficult problem than the near-optimal evasion problem. Reverse engineering is sufficient for solving the evasion problem, but we show that it is not necessary. Lowd & Meek showed that reverse engineering linear classifiers is efficient, but here we show that reverse engineering is strictly more difficult than evasion for convex-inducing classifiers. It is unknown whether there exists a class in between linear and convex-inducing classifiers on which the two tasks are efficient.

Question 8.5 For what families of classifiers is reverse engineering as easy as evasion?

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

8.4.2 Alternative Evasion Criteria

We suggest variants of near-optimal evasion that generalize or reformulate the problem investigated in this chapter to capture additional aspects of the overall challenge.

8.4.2.1 Incorporating a Covertness Criterion

As mentioned in Section 8.1.2, the near-optimal evasion problem does not require the attacker to be covert in its actions. The primary concern for the adversary is that a defender may detect the probing attack and make it ineffectual. For instance, the MultiLineSearch algorithms we present in Section 8.2 are very overt about the attacker's true intention; i.e., because the queries are issued in ℓ_p shells about $\mathbf{x}^A$, it is trivial to infer $\mathbf{x}^A$. The queries issued by the randomized ellipsoid method in Section 8.2.2 are less overt due to the random walks, but still the queries occur in shrinking cost balls centered around $\mathbf{x}^A$. The reverse-engineering approach of Lowd & Meek (2005*a*), however, is quite covert. In their approach, all queries are based only on the features of $\mathbf{x}^-$ and a third $\mathbf{x}^+ \in \mathcal{X}_f^+$—$\mathbf{x}^A$ is not used until an ϵ-$IMAC$ is discovered.

Question 8.6 What covertness criteria are appropriate for a near-optimal evasion problem? Can a defender detect nondiscrete probing attacks against a classifier? Can the defender effectively mislead a probing attack by falsely answering suspected queries?

Misleading an adversary is an especially promising direction for future exploration. If probing attacks can be detected, a defender could frustrate the attacker by falsely responding to suspected queries. However, if too many benign points are incorrectly identified as queries, such a defense could degrade the classifier's performance. Thus, strategies to mislead could backfire if an adversary fooled the defender into misclassifying legitimate data—yet another security game between the adversary and defender.

8.4.2.2 Additional Information about Training Data Distribution

Consider an adversary that knows the training algorithm and obtains samples drawn from a natural distribution. A few interesting settings include scenarios where the adversary's samples are *i*) a subset of the training data, *ii*) from the same distribution P_Z as the training data, or *iii*) from a perturbation of the training distribution. With these forms of additional information, the adversary could estimate its own classifier $\tilde{f}$ and analyze it offline. Open questions about this variant include.

Question 8.7 What can be learned from $\tilde{f}$ about f? How can $\tilde{f}$ best be used to guide search? Can the sample data be directly incorporated into ϵ-$IMAC$-search without $\tilde{f}$?

Relationships between f and $\tilde{f}$ can build on existing results in learning theory. One possibility is to establish bounds on the difference between $MAC(f, A)$ and $MAC\left(\tilde{f}, A\right)$ in one of the above settings. If, with high probability, the difference is sufficiently small, then a search for an ϵ-$IMAC$ could use $MAC\left(\tilde{f}, A\right)$ to initially lower bound $MAC(f, A)$. This should reduce search complexity since lower bounds on the MAC are typically harder to obtain than upper bounds.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

8.4.2.3 Beyond the Membership Oracle

In this scenario, the adversary receives more from the classifier than just a "+"/"−" label. For instance, suppose the classifier is defined as $f(x) = \mathrm{I}[g(x) > 0]$ for some real-valued function g (as is the case for SVMs) and the adversary receives $g(x)$ for every query instead of $f(x)$. If g is linear, the adversary can use $D + 1$ queries and solve a linear regression problem to reverse engineer g. This additional information may also be useful for approximating the support of an SVM.

Question 8.8 What types of additional feedback may be available to the adversary, and how do they affect the query complexity of ϵ-*IMAC*-search?

8.4.2.4 Evading Randomized Classifiers

In this variant of near-optimal evasion, we consider randomized classifiers that generate random responses from a distribution conditioned on the query x. To analyze the query complexity of such a classifier, we first generalize the concept of the *MAC* to randomized classifiers. We propose the following generalization:

$$RMAC(f, A) = \inf_{x \in \mathcal{X}} \left\{ A\left(x - \mathbf{x}^A\right) + \lambda \mathrm{P}\left(f(x) = \text{"}-\text{"}\right) \right\}.$$

Instead of the unknown set $\mathcal{X}_f^-$ in the near-optimal evasion setting, the objective function here contains the term $\mathrm{P}\left(f(x) = \text{"}-\text{"}\right)$ that the adversary does not know and must approximate. If f is deterministic , $\mathrm{P}\left(f(x) = \text{"}-\text{"}\right) = \mathrm{I}\left[f(x) = \text{"}-\text{"}\right]$, this definition is equivalent to Equation (8.2) only if $\lambda \geq MAC(f, A)$ (e.g., $\lambda = A\left(\mathbf{x}^- - \mathbf{x}^A\right) + 1$ is sufficient); otherwise, a trivial minimizer is $\mathbf{x}^A$. For a randomized classifier, λ balances the cost of an instance with its probability of successful evasion.

Question 8.9 Given access to the membership oracle only, how difficult is near-optimal evasion of randomized classifiers? Are there families of randomized classifiers that are ϵ-*IMAC* searchable?

Potential randomized families include classifiers (*i*) with a fuzzy boundary of width δ around a deterministic boundary, and (*ii*) based on the class-conditional densities for a pair of Gaussians, a logistic regression model, or other members of the exponential family. Generally, evasion of randomized classifiers seems to be more difficult than for deterministic classifiers as each query provides limited information about the query probabilities. Based on this argument, Biggio et al. (2010) promote randomized classifiers as a defense against evasion. However, it is not known if randomized classifiers have provably worse query complexities.

8.4.2.5 Evading an Adaptive Classifier

Finally, we consider a classifier that periodically retrains on queries. This variant is a multi-fold game between the attacker and learner, with the adversary now able to issue queries that degrade the learner's performance. Techniques from game-theoretic online learning should be well suited to this setting (Cesa-Bianchi & Lugosi 2006).

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

Question 8.10 Given a set of adversarial queries (and possibly additional innocuous data) will the learning algorithm converge to the true boundary, or can the adversary deceive the learner and evade it simultaneously? If the algorithm does converge, then at what rate?

To properly analyze retraining, it is important to have an oracle that labels the points sent by the adversary. If all points sent by the adversary are labeled "+", the classifier may prevent effective evasion, but with large numbers of false positives due to the adversary queries in $\mathcal{X}_f^-$, this itself constitutes an attack against the learner (Barreno et al. 2010).

8.4.3 Real-World Evasion

While the cost-centric evasion framework presented by Lowd & Meek formalizes the near-optimal evasion problem, it fails to capture some aspects of reality. From the theory of near-optimal evasion, certain classes of learners have been shown to be easy to evade, whereas others require a practically infeasible number of queries for evasion to be successful. However, real-world adversaries often do not require near-optimal cost evasive instances to be successful; it would suffice if they could find any low-cost instance able to evade the detector. Real-world evasion differs from the near-optimal evasion problem in several ways. Understanding query strategies and the query complexity for a real-world adversary requires incorporating real-world constraints that were relaxed or ignored in the theoretical version of this problem. We summarize the challenges for real-world evasion.

To adapt to these challenges, we propose a *realistic evasion problem* that weakens several of the assumptions of the theoretical near-optimal evasion problem for studying real-world evasion techniques. We still assume the adversary does not know f and may not even know the family $\mathcal{F}$; we only assume that the classifier is a deterministic classifier that uniquely maps each instance in $\mathcal{X}$ to {"+", *negLbl*}. For a real-world adversary, we require that the adversary send queries that are representable as actual objects in Ω; e.g., emails cannot have 1.7 occurrences of the word "viagra" in a message and IP addresses must have four integers between $0 - -255$. However, we no longer assume that the adversary knows the feature space of the classifier or its feature mapping.

Real-world near-optimal evasion is also more difficult (i.e., requires more queries) than is suggested by the theory because the theory simplifies the problem faced by the adversary. Even assuming that a real-world adversary can obtain query responses from the classifier, it cannot directly query it in the feature space $\mathcal{X}$. Real-world adversaries must make their queries in the form of real-world objects like email that are subsequently mapped into $\mathcal{X}$ via a feature map. Even if this mapping is known by the adversary, designing an object that maps to a desired query in $\mathcal{X}$ is itself a difficult problem—there may be many objects that map to a single query (e.g., permuting the order of words in a message yields the same unigram representation), and certain portions of $\mathcal{X}$ may not correspond to any real-world object (e.g., for the mapping $x \mapsto \left(x, x^2\right)$ no point x can map to $(1, 7)$).

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press

Question 8.11 How can the feature mapping be inverted to design real-world instances to map to desired queries? How can query-based algorithms be adapted for approximate querying?

Real-world evasion also differs dramatically from the near-optimal evasion setting in defining an *efficient* classifier. For a real-world adversary, even polynomially many queries in the dimensionality of the feature space may not reasonable. For instance, if the dimensionality of the feature space is large (e.g., hundreds of thousands of words in unigram models) the adversary may require the number of queries to be sub-linear, $o(D)$, but in the near-optimal evasion problem this is not even possible for linear classifiers. However, real-world adversaries do not need to be provably near-optimal. Near-optimality is a surrogate for the adversary's true evasion objective: to use a small number of queries to find a negative instance with acceptably low cost; i.e., below some maximum cost threshold. This corresponds to an alternative cost function $A'(x) = \max[A(x), \delta]$ where δ is the maximum allowable cost. Clearly, if an ϵ-$IMAC$ is obtained, either it satisfies this condition or the adversary can cease searching. Thus, ϵ-$IMAC$ searchability is sufficient to achieve the adversary's goal, but the near-optimal evasion problem ignores the maximum cost threshold even though it may allow for the adversary to terminate its search using far fewer queries. To accurately capture real-world evasion with sub-linearly many queries, query-based algorithms must efficiently use every query to glean essential information about the classifier. Instead of quantifying the query complexity required for a family of classifiers, perhaps it is more important to quantify the query performance of an evasion algorithm for a fixed number of queries based on a target cost.

Question 8.12 In the real-world evasion setting, what is the worst-case or expected reduction in cost for a query algorithm after making M queries to a classifier $f \in \mathcal{F}$? What is the expected value of each query to the adversary, and what is the best query strategy for a fixed number of queries?

The final challenge for real-world evasion is to design algorithms that can thwart attempts to evade the classifier. Promising potential defensive techniques include randomizing the classifier, identifying queries, and sending misleading responses to the adversary. We discuss these and other defensive techniques in Section 9.1.2.

https://doi.org/10.1017/9781107338548.008 Published online by Cambridge University Press