Noname manuscript No.
(will be inserted by the editor)

Learning to Classify with Missing and Corrupted Features

Ofer Dekel · Ohad Shamir · Lin Xiao

Received: March 31, 2008

Abstract A common assumption in supervised machine learning is that the training examples provided to the learning algorithm are statistically identical to the instances encountered later on, during the classification phase. This assumption is unrealistic in many real-world situations where machine learning techniques are used. We focus on the case where features of a binary classification problem, which were available during the training phase, are either deleted or become corrupted during the classification phase. We prepare for the worst by assuming that the subset of deleted and corrupted features is controlled by an adversary, and may vary from instance to instance. We design and analyze two novel learning algorithms that anticipate the actions of the adversary and account for them when training a classifier. Our first technique formulates the learning problem as a linear program. We discuss how the particular structure of this program can be exploited for computational efficiency and we prove statistical bounds on the risk of the resulting classifier. Our second technique addresses the robust learning problem by combining a modified version of the Perceptron algorithm with an online-to-batch conversion technique, and also comes with statistical generalization guarantees. We demonstrate the effectiveness of our approach with a set of experiments.

Keywords adversarial environment · binary classification · deleted features

1 Introduction

Supervised machine learning techniques are often used to train classifiers that are put to work in complex real-world systems. A training set of labeled examples is collected and

O. Dekel
Microsoft Research, One Microsoft Way, Redmond 98052 WA, USA
Tel.: +1 (425) 722-2182
E-mail: oferd@microsoft.com

O. Shamir
The Hebrew University, Jerusalem 91904, Israel
E-mail: ohadsh@cs.huji.ac.il

L. Xiao
Microsoft Research, One Microsoft Way, Redmond 98052 WA, USA
E-mail: lin.xiao@microsoft.com

presented to a machine learning algorithm, and the learning algorithm outputs a classifier. The process of collecting the training set and constructing the classifier is called the *training phase*, while everything that occurs after the classifier is constructed is called the *classification phase*. In many cases, the training phase can be performed under sterile and controlled conditions, and specifically, care can be taken to collect a high quality training set. In contrast, the classification phase often takes place in the noisy and uncertain conditions of the real world. Specifically, features that were available during the training phase may become missing or corrupted in the classification phase. In the worst case, the set of missing and corrupted features may be controlled by an adversary, who may even be familiar with the inner-workings of the classifier. In this paper, we explore the possibility of anticipating this scenario and preparing for it in advance.

The problem of missing and corrupted features that are controlled by an adversary occurs in a variety of classification problems. For example, consider the task of learning an email spam filter. Once the training phase is complete, adversaries attempt to infiltrate the learned filter by constructing emails with feature representations that appear to be benign. A reasonable approach to email spam filtering should prepare for the actions of these adversaries.

Our setting also encompasses learning problems where features are deleted and corrupted due to other, less malicious, circumstances. For example, say that our goal is to learn an automatic medical diagnosis system. Each instance represents a patient, each feature contains the result of a medical test performed on that patient, and the purpose of the system is to detect a certain disease. When constructing the training set, we go to the trouble of carefully performing every possible test on each patient. However, when the learned classifier is eventually deployed as part of a diagnosis system, and applied to new patients, it is highly unlikely that all of the test results will be available. Technical difficulties may prevent certain tests from being performed. Different patients may have different insurance policies, each covering a different set of tests. A patient's blood sample may become contaminated, essentially replacing the corresponding features with random noise, while having no effect on other features. We would still like our diagnosis system to make accurate predictions. In this example, the classification-time feature corruption is not adversarial, but it is not purely stochastic either. If a classifier is trained to tolerate adversarial noise, it will certainly be able to handle less deliberate forms of noise.

If we do not limit the adversary's ability to remove and modify features, our classifier obviously stands no chance of making correct predictions. We overcome this problem by assigning each feature with an a-priori importance value and assuming that the adversary may remove or corrupt any feature subset whose total value is upper-bounded by a predefined constant.

In this paper, we present two new learning algorithms for learning with missing and corrupted features. Both approaches attempt to learn a noise-tolerant linear threshold classifier. The first approach formulates the learning problem as a linear program (LP), in a way that closely resembles the quadratic programming formulation of the Support Vector Machine [20]. However, the number of constraints in this LP grows exponentially with the number of features. Using tricks from convex analysis, we derive a related polynomial-size LP, and give conditions under which it is an exact reformulation of the original exponential-size LP. When these conditions do not hold, the polynomial-size LP still approximates the exponential-size LP, and we prove an upper bound on the approximation difference. The polynomial-size LP can be solved efficiently by exploiting certain properties of its structure. Despite the fact that the distribution of training examples is effectively different from

the distribution of examples observed during the classification phase, we prove a statistical generalization bound for this approach.

We show that the time complexity of our LP-based approach scales linearly with the number of training examples. However, the running time of this approach grows quadratically with the number of features and this poses a problem when the approach is applied to large datasets. This brings us to our second algorithm: We define an online learning problem that is closely related to the original statistical learning problem. We address this online problem with a modified version of the online Perceptron algorithm [17], and then convert the online algorithm into a statistical learning algorithm using an online-to-batch conversion technique [5]. This approach benefits from the computational efficiency of the Perceptron, and from the generalization properties and theoretical guarantees provided by the online-to-batch technique. Experimentally, we observe that the efficiency of our second approach seems to come at the price of a small accuracy penalty.

Choosing an adequate regularization scheme is one of the keys to successfully learning a linear classifier in our setting. Existing learning algorithms for linear classifiers, such as the Support Vector Machine, often use L_2 regularization to promote statistical generalization. When L_2 regularization is used, the learning algorithm may put a large weight on one feature and compensate by putting a small weight on another feature. This promotes classifiers that focus their weight on the features that contribute the most during training. For example, in the degenerate case where one of the features actually equals the correct label, an L_2 regularized learning algorithm is likely to put most of its weight on that one feature. Some algorithms use L_1 regularization to further promote sparse solutions [2]. In the context of our work, sparsity actually makes a classifier more susceptible to adversarial feature-corrupting noise. Here, we prefer dense classifiers, which hedge their bets across as many features as possible. Both of the algorithms presented in this paper achieve this density by using a L_∞ regularization scheme. It is interesting to note that our L_∞ regularization scheme emerges as a natural choice in the statistical analysis of our LP-based learning approach.

This paper is organized as follows. We conclude this section by referencing related work. In Sec. 2 we present our LP-based learning algorithm. Section 2.1 casts the problem of learning with feature deletion as an exponential-size LP, Sec. 2.2 presents a polynomial approximation to this program, and Sec. 2.3 describes an efficient customized LP solver that takes advantage of the special structure of our problem. We prove statistical generalization bounds in Sec. 2.4 and extend our discussion from the feature deletion scenario to the feature corruption scenario in Sec. 2.5. Next, in Sec. 3, we move on to our second algorithm, which combines a modified Perceptron algorithm with an online-to-batch conversion technique. The modified Perceptron is presented in Sec. 3.1 and the online-to-batch technique is discussed in Sec. 3.2. We conclude the paper with experimental results in Sec. 4 and closing remarks in Sec. 5.

1.1 Related Work

Previous papers on "noise-robust learning" mainly deal with the problem of learning with a noisy training set, a research topic which is entirely orthogonal to ours. The learning algorithms presented in [8] and [9] try to be robust to general additive noise that appears at classification time, but not specifically to adversarial feature deletion or corruption. [6] presents adversarial learning as a one-shot two-player game between the classifier and an adversary, and designs a robust learning algorithm from a Bayesian-learning perspective. Our approach shares the motivation of [6] but is otherwise significantly different. The topic

of email spam filtering, and its wider implications on learning in the face of an adversary, has recently received special attention. Notable contributions on the intersection of spam filtering and machine learning are [15] and [21]. In the related field of online learning, where the training and classification phases are interlaced and cannot be distinguished, [14] proves that the Winnow algorithm can tolerate various types of noise, both adversarial and random.

Our work is most similar to the work in [10], and its more recent enhancement in [18]. Our experiments, presented in Sec. 4, suggest that our algorithms achieve significantly better performance, but we can also emphasize more fundamental differences between the two approaches: Our approach uses L_∞ regularization to promote a dense solution, where [10] uses L_2 regularization. We allow features to have different a-priori importance levels, and take this information into account in our algorithm and analysis, whereas [10] assume uniform feature values. Finally, we prove statistical generalization bounds for our algorithms despite the change in distribution at classification time, while [10] do not discuss this topic.

This paper is a long version of the preliminary work published in [7]. In this paper, we present a more complete and elaborate theoretical analysis of our algorithms, as well as a significantly improved empirical study. Specifically, this paper includes complete proofs of all theorems and new experiments using larger and more diverse datasets. The extended scope of our experiments now includes empirical evidence that our algorithms outperform the current state-of-the-art results of [18], and new empirical results in the feature corruption scenario. Moreover, the novel linear programming algorithm presented in Sec. 2.3 addresses important computational problems that were ignored in [7].

2 A Linear Programming Formulation

In this section, and throughout the paper, we use lower-case bold-face letters to denote vectors, and their plain-face counterparts to denote each vector's components. We also use the notation $[n]$ as shorthand for $\{1, \ldots, n\}$.

2.1 Feature Deleting Noise

We first examine the case where features are missing at classification time. Let $\mathcal{X} \subseteq \mathbb{R}^n$ be an instance space and let $\mathcal{D}$ be a probability distribution on the product space $\mathcal{X} \times \{\pm 1\}$. We receive a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ sampled i.i.d from $\mathcal{D}$, which we use to learn our classifier. We assign each feature $j \in [n]$ a value $v_j \geq 0$. Informally, we think of v_j as the a-priori *informativeness* of feature j, or as the importance of feature j to the classification task. Next, we define the value of a subset J of features as the sum of values of the features in that subset, and we denote $V(J) = \sum_{j \in J} v_j$. For instance, we frequently use $V([n])$ when referring to $\sum_{j=1}^n v_j$ and $V([n] \setminus J)$ when referring to $\sum_{j \notin J} v_j$. Next, we fix a noise-tolerance parameter N in $[0, V([n])]$ and define $P = V([n]) - N$. During the classification phase, instances are generated in the following way: First, a pair $(\mathbf{x}, y)$ is sampled from $\mathcal{D}$. Then, an adversary selects a subset of features $J \subset [n]$ such that $V([n] \setminus J) \leq N$, and replaces x_j with 0 for all $j \notin J$. The adversary selects J for each instance individually, and with full knowledge of the inner workings of our classifier. The noise-tolerance parameter N essentially acts as an upper bound on the amount of damage the adversary is allowed to inflict. We would like to use the training set S (which does not have missing features) to learn a binary classifier that is robust to this specific type of classification-time noise.

We focus on learning linear margin-based classifiers. A linear classifier is defined by a weight vector $\mathbf{w} \in \mathbb{R}^n$ and a bias term $b \in \mathbb{R}$. Given an instance $\mathbf{x}$, which is sampled from $\mathcal{D}$, and a set of coordinates J left intact by the adversary, the linear classifier outputs $b + \sum_{j \in J} w_j x_j$. The sign of $b + \sum_{j \in J} w_j x_j$ constitutes the actual binary prediction, while $|b + \sum_{j \in J} w_j x_j|$ is understood as the degree of confidence in that prediction. A classification mistake occurs if and only if $y(b + \sum_{j \in J} w_j x_j) \leq 0$, so we define the *risk* of the linear classifier $(\mathbf{w}, b)$ as

$$\mathcal{R}(\mathbf{w}, b) = \Pr_{(\mathbf{x},y)\sim\mathcal{D}} \Big(\exists J \text{ with } V([n] \setminus J) < N \text{ s.t. } y\big(b + \textstyle\sum_{j \in J} w_j x_j\big) \leq 0 \Big) \ . \quad (1)$$

Since $\mathcal{D}$ is unknown, we cannot explicitly minimize Eq. (1). Thus, we turn to the empirical estimate of Eq. (1), the *empirical risk*, defined as

$$\frac{1}{m} \sum_{i=1}^{m} \Big[\!\Big[\min_{J \,:\, V([n]\setminus J) \leq N} y_i\big(b + \textstyle\sum_{j \in J} w_j x_{i,j}\big) \leq 0 \Big]\!\Big], \quad (2)$$

where $[\![\pi]\!]$ denotes the indicator function of the predicate π. Minimizing the empirical risk directly constitutes a difficult combinatorial optimization problem. Instead, we formulate a linear program that closely resembles the formulation of the Support Vector Machine [20]. We choose a regularization parameter $C > 0$, and solve the problem

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \tfrac{1}{m} \textstyle\sum_{i=1}^{m} \xi_i & (3) \\ \text{s.t.} \quad & \forall\, i \in [m] \quad \forall J \,:\, V([n] \setminus J) \leq N \quad y_i\big(b + \textstyle\sum_{j \in J} w_j x_{i,j}\big) \geq \tfrac{V(J)}{P} - \xi_i \ , \\ & \forall\, i \in [m] \quad \xi_i \geq 0 \, , \\ & \|\mathbf{w}\|_\infty \leq C \ . \end{aligned}$$

The objective function of Eq. (3) is called the *empirical hinge-loss* obtained on the sample S. Since ξ_i is constrained to be non-negative, each training example contributes a non-negative amount to the total loss. Moreover, the objective function of Eq. (3) upper bounds the empirical risk of $(\mathbf{w}, b)$. More specifically, for any feasible point $(\mathbf{w}, b, \boldsymbol{\xi})$ of Eq. (3), ξ_i upper bounds the indicator function of the event

$$\min_{J \,:\, V([n]\setminus J) \leq N} y_i\big(b + \textstyle\sum_{j \in J} w_j x_{i,j}\big) \leq 0 \ .$$

To see this, note that for a given example $(\mathbf{x}_i, y_i)$, if there exists a feature subset J such that $V([n] \setminus J) \leq N$ and $y_i(b + \sum_{j \in J} w_j x_j) \leq 0$ then the first constraint in Eq. (3) enforces $\xi_i \geq V(J)/P$. The assumption $V([n] \setminus J) \leq N$ now implies that $V(J) \geq P$, and therefore $\xi_i \geq 1$. If such a set J does not exist, then the second constraint in Eq. (3) enforces $\xi_i \geq 0$.

The optimization problem above actually does more than minimize an upper bound on the empirical risk. It also requires the margin attained by the feature subset J to grow with proportion to $V(J)$. While a true adversary would always inflict the maximal possible damage, our optimization problem also prepares for the case where less damage is inflicted, requiring the confidence of our classifier to increase as less noise is introduced. Also, assuming that the margin scales with the number of features is a natural assumption to make when we have feature redundancy, a necessary prerequisite for our approach to work in the first place. We also restrict $\mathbf{w}$ to a hyper-box of radius C, which controls the complexity of the learned classifier and promotes robust dense solutions. Moreover, this constraint is easy to compute and makes our algorithms more efficient. Although Eq. (3) is a linear program,

it is immediately noticeable that the size of its constraint set may grow exponentially with the number of features n. For example, if $v_j = 1$ for all $j \in [n]$ and if N is a positive integer, then the linear program contains over $\binom{n}{N}$ constrains per example. We deal with this problem below.

2.2 A Polynomial Approximation

Taking inspiration from [4], we find an efficient approximate formulation of Eq. (3), which turns out to be an exact reformulation of Eq. (3) when $v_j \in \{0,1\}$ for all $j \in [n]$. Specifically, we replace Eq. (3) with

$$\begin{aligned} \min \ & \tfrac{1}{m}\textstyle\sum_{i=1}^m \xi_i \\ \text{s.t. } \forall\, i \in [m] \quad & P\lambda_i \ - \ \textstyle\sum_{j=1}^n \alpha_{i,j} + y_i b \ \geq -\xi_i \\ \forall\, i \in [m]\, \forall\, j \in [n] \quad & y_i w_j x_{i,j} - \tfrac{v_j}{P} \ \geq \ \lambda_i v_j - \alpha_{i,j} \ , \\ \forall\, i \in [m]\, \forall\, j \in [n] \quad & \alpha_{i,j} \geq 0 \ , \\ \forall\, i \in [m] \quad & \lambda_i \geq 0 \ \text{ and } \ \xi_i \geq 0 \ , \\ & \|\mathbf{w}\|_\infty \leq \ C \ \ , \end{aligned} \tag{4}$$

where the minimization is over $\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, $\boldsymbol{\xi} \in \mathbb{R}^m$, $\boldsymbol{\lambda} \in \mathbb{R}^m$, and $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_m$, each in $\mathbb{R}^n$. The number of variables and the number of constraints in this problem are both $O(mn)$. The following theorem explicitly relates the optimization problem in Eq. (4) with the one in Eq. (3).

Theorem 1 *Let $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}_1^\star, \ldots, \boldsymbol{\alpha}_m^\star)$ be an optimal solution to Eq. (4).*

(a) $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star)$ is a feasible point of Eq. (3), and therefore the value of Eq. (4) upper-bounds the value of Eq. (3).
(b) If $v_j \in \{0,1\}$ for all $j \in [n]$, then $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star)$ is also an optimal solution to Eq. (3).
(c) If it does not hold that $v_j \in \{0,1\}$ for all $j \in [n]$, and assuming $\|\mathbf{x}_i\| \leq 1$ for all i, then the difference between the value of Eq. (4) and the value of Eq. (3) is at most C.

As a first step towards proving Thm. 1, we momentarily forget about the optimization problem at hand and focus on another question: given a specific triplet $(\mathbf{w}, b, \boldsymbol{\xi})$, is it a feasible point of Eq. (3) or not? More concretely, for each training example $(\mathbf{x}_i, y_i)$, we would like to determine if for all J with $V([n] \setminus J) \leq N$ it holds that

$$y_i\big(b + \textstyle\sum_{j \in J} w_j x_{i,j}\big) \ \geq \ \frac{V(J)}{P} - \xi_i \ \ . \tag{5}$$

We can answer this question by comparing $-\xi_i$ with the value of the following integer program:

$$\begin{aligned} \min_{\boldsymbol{\tau} \in \{0,1\}^n} \ & y_i b + \textstyle\sum_{j=1}^n \tau_j \left(y_i w_j x_{i,j} - \frac{v_j}{P}\right) \\ \text{s.t.} \quad & P \ \leq \ \textstyle\sum_{j=1}^n \tau_j v_j \ \ . \end{aligned} \tag{6}$$

For example, if the value of this integer program is less than $-\xi_i$, then let $\boldsymbol{\tau}'$ be an optimal solution and we have that $y_i(b + \sum_{j=1}^n \tau_j' w_j x_{i,j}) < (\sum_{j=1}^n \tau_j' v_j)/P - \xi_i$. Namely, the set $J = \{j \in [n] \ : \ \tau_j' = 1\}$ violates Eq. (5). On the other hand, if there exists some J with $V([n] \setminus J) \leq N$ that violates Eq. (5) then its indicator vector is a feasible point of Eq. (6) whose objective value is less than $-\xi_i$.

Directly solving the integer program in Eq. (6) may be difficult, so instead we examine the properties of the following linear relaxation:

$$\begin{aligned} \min_{\boldsymbol{\tau}} \; & y_i b + \textstyle\sum_{j=1}^{n} \tau_j \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) \\ \text{s.t.} \; & \forall j \in [n] \;\; 0 \leq \tau_j \leq 1 \;\; \text{and} \;\; P \leq \textstyle\sum_{j=1}^{n} \tau_j v_j \;\; . \end{aligned} \tag{7}$$

The key result needed to analyze this relaxation is the following lemma.

Lemma 1 *Fix an example $(\mathbf{x}_i, y_i)$, a linear classifier $(\mathbf{w}, b)$, and a scalar $\xi_i > 0$, and let θ be the value of Eq. (7) with respect to these choices.*

(a) If $\theta \geq -\xi_i$ then Eq. (5) holds.
(b) There exists a minimizer of Eq. (7) with at most one non-integer element.
(c) In the special case where $v_j \in \{0, 1\}$ for all $j \in [n]$ and where N is an integer, $\theta \geq -\xi_i$ if and only if Eq. (5) holds.

The proof of Lemma 1 is rather technical and monotonous, and is therefore differed to Appendix A. The lemma tells us that comparing the value of the linear program in Eq. (7) with $-\xi_i$ provides a sufficient condition for Eq. (5) to hold for the example $(\mathbf{x}_i, y_i)$. Moreover, this condition becomes both sufficient and necessary in the special case where $v_j \in \{0, 1\}$ for all $j \in [n]$. This equivalence enables us to prove Thm. 1.

Proof of Theorem 1 We begin with the proof of claim (a). Let $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}_1^\star, \ldots, \boldsymbol{\alpha}_m^\star)$ be an optimal solution to the linear program in Eq. (4). Specifically, it holds for all $i \in [m]$ that $\boldsymbol{\alpha}_i^\star$ and $\lambda_i^\star$ are non-negative, that $P\lambda_i^\star - \sum_{j=1}^{n} \alpha_{i,j}^\star + y_i b^\star \geq -\xi_i^\star$, and that

$$\forall \, j \in [n] \;\; y_i w_j^\star x_{i,j} - \frac{v_j}{P} \;\geq\; \lambda_i^\star v_j - \alpha_{i,j}^\star \;\; .$$

Therefore, it also holds that the value of the following optimization problem

$$\begin{aligned} \max_{\boldsymbol{\alpha}_i, \lambda_i} \; & P\lambda_i \; - \; \textstyle\sum_{j=1}^{n} \alpha_{i,j} + y_i b^\star \\ \text{s.t.} \; & \forall \, j \in [n] \;\; y_i w_j^\star x_{i,j} - \tfrac{v_j}{P} \geq \lambda_i v_j - \alpha_{i,j} \; , \\ & \forall \, j \in [n] \;\; \alpha_{i,j} \geq 0 \;\text{ and }\; \lambda_i \geq 0 \;\; , \end{aligned} \tag{8}$$

is at least $-\xi_i^\star$. The strong duality principle of linear programming [3] states that the value of Eq. (8) equals the value of its dual optimization problem, which is:

$$\begin{aligned} \min_{\boldsymbol{\tau}} \; & y_i b^\star + \textstyle\sum_{j=1}^{n} \tau_j \left(y_i w_j^\star x_{i,j} - \frac{v_j}{P} \right) \\ \text{s.t.} \; & \forall \, j \in [n] \;\; 0 \leq \tau_j \leq 1 \quad \text{and} \quad P \;\leq\; \textstyle\sum_{j=1}^{n} \tau_j v_j \;\; . \end{aligned} \tag{9}$$

In other words, the value of Eq. (9) is also at least $-\xi_i^\star$. Using claim (a) of Lemma 1, we have that

$$y_i \big(b^\star + \textstyle\sum_{j \in J} w_j^\star x_{i,j} \big) \;\geq\; \frac{V(J)}{P} - \xi_i^\star \;\; ,$$

holds for all J with $V([n] \setminus J) \leq N$. The optimization problem in Eq. (4) also constrains $\|\mathbf{w}\|_\infty \leq C$ and $\xi_i \geq 0$ for all $i \in [m]$, thus, $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star)$ satisfies the constraints in Eq. (3). Since Eq. (3) and Eq. (4) have the same objective function, the value of Eq. (3) is upper bounded by the value of Eq. (4).

Claim (b) of the theorem states that if $v_j \in \{0, 1\}$ for all $j \in [n]$ then $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star)$ is an optimum of Eq. (3). Assume the contrary, namely, assume that there exist $\mathbf{w}'$, b', and $\boldsymbol{\xi}'$ in

the feasible region of Eq. (3) for which $\sum_{i=1}^{m} \xi_i' < \sum_{i=1}^{m} \xi_i^\star$. Using claim (c) of Lemma 1, we know that the value of Eq. (9) (with $w_j^\star$ replaced by w_j') is at least $-\xi_i'$ for all $i \in [m]$. Once again using strong duality, we have that the value of Eq. (8) (with $w_j^\star$ replaced by w_j') is at least $-\xi_i'$ for all $i \in [m]$. Moreover, let $\boldsymbol{\alpha}_i'$ and λ_i' denote the optimizers of Eq. (8) for all $i \in [m]$. We conclude that $(\mathbf{w}', b', \boldsymbol{\xi}', \boldsymbol{\lambda}', \boldsymbol{\alpha}_1', \ldots, \boldsymbol{\alpha}_m')$ is a feasible point of Eq. (4). This contradicts our assumption that $\sum_{i=1}^{m} \xi_i^\star$ is minimal over the feasible set of Eq. (4).

Finally, we prove claim (c) of the theorem. Let $(\mathbf{w}', b', \boldsymbol{\xi}')$ be an optimal solution to the exponential optimization problem in Eq. (3) and recall that $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}_1^\star, \ldots, \boldsymbol{\alpha}_m^\star)$ denotes the optimal solution to Eq. (4). We have already proven that $\sum_{i=1}^{m} \xi_i' \leq \sum_{i=1}^{m} \xi_i^\star$, and our current focus is on bounding the difference between these two sums.

Let $(\bar{\boldsymbol{\xi}}, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\alpha}}_1, \ldots, \bar{\boldsymbol{\alpha}}_m)$ be the optimal solution to Eq. (4) with the additional constraints $\mathbf{w} = \mathbf{w}'$ and $b = b'$. Note that Eq. (4) with these additional constraints still has a non-empty feasible set, for instance by setting each $\bar{\lambda}_j$ to zero, setting $\bar{\alpha}_{i,j}$ so as to satisfy the second constraint in Eq. (4), and finally setting $\bar{\boldsymbol{\xi}}$ to satisfy the first constraint in Eq. (4). These additional constraints decrease the feasible region of Eq. (4), and therefore $\sum_{i=1}^{m} \xi_i^\star \leq \sum_{i=1}^{m} \bar{\xi}_i$. It now suffices to prove an upper bound on $\sum_{i=1}^{m} \bar{\xi}_i - \sum_{i=1}^{m} \xi_i'$.

We define I to be the set $\{i \in [m] : \bar{\xi}_i > \xi_i'\}$ and note that

$$\sum_{i=1}^{m} \bar{\xi}_i - \sum_{i=1}^{m} \xi_i' \;\leq\; \sum_{i \in I} (\bar{\xi}_i - \xi_i') \;\; .$$

For every $i \in I$, we have that $0 \leq \xi_i' < \bar{\xi}_i$, namely, $\bar{\xi}_i$ is strictly greater than zero. Therefore, we know that the first constraint in Eq. (4) is binding, and

$$-\bar{\xi}_i \;=\; P\bar{\lambda}_i - \sum_{j=1}^{n} \bar{\alpha}_{i,j} + y_i b' \;\; .$$

Recall that the objective of Eq. (4) is to minimize $\bar{\xi}_i$, which is equivalent to maximizing $P\bar{\lambda}_i - \sum_{j=1}^{n} \bar{\alpha}_{i,j} + y_i b'$ subject to the other constraints of Eq. (4). In other words, $-\bar{\xi}_i$ equals the value of Eq. (8) (with $(\mathbf{w}^\star, b^\star)$ replaced by $(\mathbf{w}', b')$). Using strong duality, this value equals the value of Eq. (9) (with $(\mathbf{w}^\star, b^\star)$ replaced by $(\mathbf{w}', b')$). Letting $\boldsymbol{\tau}'$ denote the optimal solution to Eq. (9) (with $(\mathbf{w}^\star, b^\star)$ replaced by $(\mathbf{w}', b')$), we have that

$$-\bar{\xi}_i \;=\; y_i b' + \sum_{j=1}^{n} \tau_j' \left(y_i w_j' x_{i,j} - \frac{v_j}{P} \right) \;\; .$$

Using claim (b) of Lemma 1, we may assume, without loss of generality, that $\tau_2', \ldots, \tau_n'$ are all integers. We can therefore write

$$-\bar{\xi}_i \;=\; y_i b' + \sum_{j=1}^{n} \lceil \tau_j' \rceil \left(y_i w_j' x_{i,j} - \frac{v_j}{P} \right) \;-\; (1 - \tau_1') \left(y_i w_1' x_{i,1} - \frac{v_1}{P} \right) \;\; .$$

As previously discussed, $-\xi_i'$ is the value of the integer program in Eq. (6), with $\mathbf{w}$ replaced by $\mathbf{w}'$ and b replaced by b'. Since $\lceil \tau_1' \rceil, \ldots, \lceil \tau_n' \rceil$ is contained in the feasible set of Eq. (6), we conclude that

$$-\bar{\xi}_i \;\geq\; -\xi_i' \;-\; (1 - \tau_1') \left(y_i w_1' x_{i,1} - \frac{v_1}{P} \right) \;\; .$$

Rearranging terms above, we get

$$\bar{\xi}_i - \xi_i' \;\leq\; (1 - \tau_1') \left(y_i w_1' x_{i,1} - \frac{v_1}{P} \right) \;\; .$$

Upper-bounding $(1-\tau_1') \leq 1$, $y_i w_1' x_{i,1} \leq C$, and $-\frac{v_1}{P} \leq 0$ gives $\bar{\xi}_i - \xi_i' \leq C$.

Overall, we have shown that $\sum_{i \in I} \bar{\xi}_i - \xi_i' \leq mC$. Recalling the beginning of our proof, we have that $\sum_{i=1}^m \xi_i^\star - \sum_{i=1}^m \xi_i' \leq mC$. Dividing both sides of this inequality by m concludes our proof. □

We have established that the linear program in Eq. (4) is an adequate approximation, and sometimes even an exact reformulation, of the exponential linear program in Eq. (3). However, the linear program in Eq. (4) can still be very large for practical classification problems, as both the number of variables and number of constraints scales with mn. Nevertheless, this linear program has a very special structure that can be exploited to obtain an efficient solution. In the next section, we show that the *practical* complexity of solving this linear program, using a customized interior-point method, is $O(mn^2)$, with storage requirement $O(mn)$.

2.3 Solving the Linear Program Efficiently by Exploiting its Structure

The linear program in Eq. (4) has a special structure than should be exploited when calculating the optimal solution. We turn to primal-dual interior point methods to solve our problem. A detailed description of interior point optimization algorithms exceeds the scope of our paper, and we refer the interested reader to [22] and [3]. In this subsection, we assume a general familiarity with interior point methods, and discuss only the details that are specific to our problem. We note that a customized LP solver is required because generic LP solvers, even ones that exploit sparsity, do not solve our problem efficiently.

First, we put the linear program in Eq. (4) in standard form. The primal and dual formulations of a standard linear program are

$$\begin{array}{lll} \text{Primal:} & \text{minimize } \mathbf{c}^T\mathbf{p} & \text{Dual:} \quad \text{maximize } \mathbf{b}^T\mathbf{q} \\ & \text{subject to } \mathbf{A}\mathbf{p} = \mathbf{b},\ \mathbf{p} \geq \mathbf{0} & \text{subject to } \mathbf{A}^T\mathbf{q} \leq \mathbf{c} \end{array} \tag{10}$$

where $\mathbf{p}$ is the vector of primal variables and $\mathbf{q}$ is the vector of dual variables. Our linear program can be conveniently put into the dual standard form with variables

$$\mathbf{q} = [\mathbf{q}_1^T \ \cdots \ \mathbf{q}_m^T \ \mathbf{w}^T \ b]^T \quad \text{where} \quad \forall i \in [m] \ \ \mathbf{q}_i = [\alpha_{i1} \ \cdots \ \alpha_{in} \ \xi_i \ \lambda_i]^T \ .$$

Correspondingly, the coefficient matrix $\mathbf{A}$ is given by

$$\mathbf{A} = \left[\begin{array}{ccc|ccc|c} \mathbf{A}_1 & & & -\mathbf{I}_{n+2} & & & \\ & \ddots & & & \ddots & & \\ & & \mathbf{A}_m & & & -\mathbf{I}_{n+2} & \\ \hline \mathbf{B}_1 & \cdots & \mathbf{B}_m & & & & \mathbf{E} \ -\mathbf{E} \end{array}\right] , \tag{11}$$

where

$$\mathbf{A}_i = \begin{bmatrix} -\mathbf{I}_n & \mathbf{1}_n \\ & -1 \\ \mathbf{v}^T & -P \end{bmatrix}, \qquad \mathbf{B}_i = -y_i \begin{bmatrix} \text{diag}(\mathbf{x}_i) & \\ & 1 \end{bmatrix}, \qquad \mathbf{E} = \begin{bmatrix} \mathbf{I}_n \\ \mathbf{0}_n^T \end{bmatrix} \ , \tag{12}$$

$\mathbf{I}_{n+2}$ is the $(n+2)$-dimensional identity matrix, $\mathbf{1}_n$ is the n-dimensional all-ones column vector, $\mathbf{0}_n$ is the n-dimensional all-zeros column vector, and $\text{diag}(\mathbf{x}_i)$ is a diagonal matrix with the vector $\mathbf{x}_i$ on its diagonal. We note that the $\mathbf{A}_i$'s are identical and have an arrow

structure, namely, the only non-zero elements in A_i are on its main diagonal, its last row, and its last column.

The vectors $\mathbf{b}$ and $\mathbf{c}$ are given by

$$\mathbf{b} = \left[\mathbf{b}_1^T \ \cdots \ \mathbf{b}_m^T \ \mathbf{0}_{n+1}^T\right]^T \qquad \text{where} \quad \forall i \in [m] \ \ \mathbf{b}_i = \left[\mathbf{0}_n^T \ \tfrac{-1}{m} \ 0\right]^T \ ,$$
$$\mathbf{c} = \left[\mathbf{c}_1^T \ \cdots \ \mathbf{c}_m^T \ \mathbf{0}_{m(n+2)}^T \ C\mathbf{1}_{2n}^T\right]^T \qquad \text{where} \quad \forall i \in [m] \ \ \mathbf{c}_i = \left[\tfrac{-1}{P}\mathbf{v}^T \ 0\right]^T .$$

In the dual standard form, the number of variables and the number of inequality constraints are, respectively,

$$N_{\text{var}} = m(n+2) + n + 1, \qquad N_{\text{con}} = m(2n+3) + 2n.$$

The matrix A is of size $N_{\text{var}} \times N_{\text{con}}$, and the number of non-zero elements in the matrix $\mathbf{A}$ is $m(5n+5) + 2n$. Therefore, $\mathbf{A}$ is an extremely sparse matrix. Moreover, the non-zero elements in $\mathbf{A}$ form a special block structure, with many of the blocks identical.

Primal-dual interior-point methods iterate simultaneously over the variables $(\mathbf{p}, \mathbf{q}, \mathbf{s})$, where $\mathbf{s}$ is the dual slack variable defined as $\mathbf{s} = \mathbf{c} - \mathbf{A}^T\mathbf{q}$. They follow the simple outline:

given starting point $(\mathbf{p}, \mathbf{q}, \mathbf{s})$, which may be infeasible, but must satisfy $(\mathbf{p}, \mathbf{s}) > \mathbf{0}$.
repeat:
1. Compute the primal-dual search direction $(\Delta\mathbf{p}, \Delta\mathbf{q}, \Delta\mathbf{s})$.
2. Choose step length η and update: $(\mathbf{p}, \mathbf{q}, \mathbf{s}) := (\mathbf{p}, \mathbf{q}, \mathbf{s}) + \eta(\Delta\mathbf{p}, \Delta\mathbf{q}, \Delta\mathbf{s})$.

until solution precision is reached.

In step 1, we compute the search direction by solving the *normal equation*

$$\mathbf{A}\mathbf{D}\mathbf{A}^T\Delta\mathbf{q} \ = \ \mathbf{r} \tag{13}$$

for $\Delta\mathbf{q}$ and then obtaining $(\Delta\mathbf{p}, \Delta\mathbf{s})$ by simple substitutions. Here, $\mathbf{D}$ is a diagonal matrix with p_i/s_i as its i'th diagonal element. The right-hand side vector $\mathbf{r}$ is generated based on the current values of $(\mathbf{p}, \mathbf{q}, \mathbf{s})$. The specific definition of $\mathbf{r}$ varies between different interior-point variants. Our method of exploiting the problem structure in solving the normal equation (explained below) is independent of the right-hand side vector $\mathbf{r}$, thus it applies to all variants of primal-dual interior-point methods.

In order to solve a linear program to sufficient accuracy, the best theoretical bound on the number of iterations required for primal-dual interior-point methods is $O(\sqrt{N_{\text{con}}})$ (see for example [22]). In practice, however, the number of iterations required hardly grows with the problem size. It is fairly safe to say that a reasonable constant, say 100, would be enough to bound the number of iterations required for most problems we are able to solve today. Therefore, the *practical* complexity of solving our linear program is on the same order as the number of floating-point operations (flops) performed on each iteration, which in our case is dominated by the flop count for solving the normal equation (13).

The matrix $\mathbf{A}\mathbf{D}\mathbf{A}^T$ in Eq. (13) has a *block-arrow* structure. To see this, we partition the diagonal matrix $\mathbf{D}$ into smaller diagonal matrices $\mathbf{D}_1, \ldots, \mathbf{D}_{2m+2}$, corresponding to the blocks of columns in $\mathbf{A}$ (see Eq. (11)), and we have

$$\mathbf{A}\mathbf{D}\mathbf{A}^T = \left[\begin{array}{ccc|c} \mathbf{A}_1\mathbf{D}_1\mathbf{A}_1^T + \mathbf{D}_{m+1} & & & \mathbf{A}_1\mathbf{D}_1\mathbf{B}_1 \\ & \ddots & & \vdots \\ & & \mathbf{A}_m\mathbf{D}_m\mathbf{A}_m^T + \mathbf{D}_{2m} & \mathbf{A}_m\mathbf{D}_m\mathbf{B}_m \\ \hline \mathbf{B}_1\mathbf{D}_1\mathbf{A}_1^T & \cdots & \mathbf{B}_m\mathbf{D}_m\mathbf{A}_m^T & \mathbf{D}_S \end{array}\right] \tag{14}$$

where

$$\mathbf{D}_S = \sum_{i=1}^{m} \mathbf{B}_i \mathbf{D}_i \mathbf{B}_i + \mathbf{E}(\mathbf{D}_{2m+1} + \mathbf{D}_{2m+2})\mathbf{E}^T.$$

We solve the normal equation (13) via block elimination (see [3, Chapter 4]). We first partition the vector $\mathbf{r}$ into $\mathbf{r}_1, \ldots, \mathbf{r}_m, \mathbf{r}_S$, and $\Delta\mathbf{q}$ into $\Delta\mathbf{q}_1, \ldots, \Delta\mathbf{q}_m, \Delta\mathbf{q}_S$, corresponding to the block structure of $\mathbf{A}\mathbf{D}\mathbf{A}^T$. There are four steps in the block elimination approach:

1. Solve the following m sets of linear equations for $\mathbf{z}_1, \ldots, \mathbf{z}_m$

$$\forall i \in [m] \qquad (\mathbf{A}_i \mathbf{D}_i \mathbf{A}_i^T + \mathbf{D}_{m+i})\mathbf{z}_i \;=\; \mathbf{r}_i. \tag{15}$$

2. Form the Schur complement matrix $\mathbf{S}$

$$\mathbf{S} \;=\; \mathbf{D}_S - \sum_{i=1}^{m} \mathbf{B}_i \mathbf{D}_i \mathbf{A}_i^T (\mathbf{A}_i \mathbf{D}_i \mathbf{A}_i^T + \mathbf{D}_{m+i})^{-1} \mathbf{A}_i \mathbf{D}_i \mathbf{B}_i. \tag{16}$$

3. Solve the following linear system for $\Delta\mathbf{q}_S$

$$\mathbf{S}\Delta\mathbf{q}_S \;=\; \mathbf{r}_S - \sum_{i=1}^{m} \mathbf{B}_i \mathbf{D}_i \mathbf{A}_i^T \mathbf{z}_i \tag{17}$$

4. Obtain $\Delta\mathbf{q}_1, \ldots, \Delta\mathbf{q}_m$ through substitutions

$$\forall i \in [m] \qquad \Delta\mathbf{q}_i \;=\; \mathbf{z}_i - (\mathbf{A}_i \mathbf{D}_i \mathbf{A}_i^T + \mathbf{D}_{m+i})^{-1} \Delta\mathbf{q}_S. \tag{18}$$

Without exploiting further structure, the overall flop count for the block elimination approach is $O(mn^3)$, with corresponding storage requirement $O(mn^2)$ (see appendix B for details). Luckily, there is additional structure in the problem that allows further reduction in both time and space complexities.

Recall that the matrices $\mathbf{A}_i$ have an arrow structure. In particular, each $\mathbf{A}_i$ has only one dense column. Because of this, the matrices $\mathbf{A}_i \mathbf{D}_i \mathbf{A}_i^T + \mathbf{D}_{m+i}$, although dense, have an *arrow-plus-rank-one* structure. More precisely,

$$\mathbf{A}_i \mathbf{D}_i \mathbf{A}_i^T \;+\; \mathbf{D}_{m+i} \;=\; \mathbf{W}_i + \mathbf{u}_i \mathbf{u}_i^T \;, \tag{19}$$

where $\mathbf{W}_i$ is a sparse arrow matrix (similar in structure to $\mathbf{A}_i$), and $\mathbf{u}_i$ is a scaled version of the last column in $\mathbf{A}_i$. With this structure, each of the linear systems in (15) can be written as

$$(\mathbf{W}_i + \mathbf{u}_i \mathbf{u}_i^T)\mathbf{z}_i \;=\; \mathbf{r}_i \;.$$

The efficient way for solving such linear systems is to first solve two sparse linear systems

$$\mathbf{W}_i \bar{\mathbf{z}}_i \;=\; \mathbf{r}_i \,, \qquad \mathbf{W}_i \bar{\mathbf{u}}_i \;=\; \mathbf{u}_i \;, \tag{20}$$

and then apply the Sherman-Woodbury-Morrison formula (see, e.g., [3, Appendix C.4.3]):

$$\mathbf{z}_i \;=\; \bar{\mathbf{z}}_i \;-\; \frac{\mathbf{u}_i^T \bar{\mathbf{z}}_i}{1 + \mathbf{u}_i^T \bar{\mathbf{u}}_i} \bar{\mathbf{u}}_i \;. \tag{21}$$

By exploiting the arrow-plus-rank-one structure as above in solving the linear systems in Eq. (15), Eq. (16) and Eq. (18), the overall flop count for solving the normal equation is reduced to $O(mn^2)$, with storage requirement $O(mn)$. The details of the complexity analysis are given in appendix B.

2.4 Generalization Bounds

Next, we formally analyze the generalization performance of the classifier learned in our framework. Our analysis builds on the PAC-Bayesian theorem, given in [16]. Throughout, we assume that $\|\mathbf{x}\|_\infty \leq 1$ with probability 1 over $\mathcal{D}$. For simplicity, we assume that the bias term b is 0, and that $v_j > 0$ for all j. These assumptions can be relaxed at the cost of a somewhat more complicated analysis. Given a classifier $\mathbf{w}$, define the γ-loss attained on the example $(\mathbf{x}, y)$ as

$$\ell_\gamma(\mathbf{w}; \mathbf{x}, y) \;=\; \left[\!\!\left[\min_{J\,:\,V([n]\setminus J) \leq N} y \sum_{j \in J} w_j x_j \;<\; \frac{\gamma V(J)}{P} \right]\!\!\right] \;, \tag{22}$$

where $[\![\cdot]\!]$ denotes the indicator function. Note that $\ell_0(\mathbf{w}; \mathbf{x}, y)$ simply indicates the occurrence of a classification mistake on example $(\mathbf{x}, y)$ in our adversarial feature deletion setting. Therefore, $\mathbb{E}[\ell_0(\mathbf{w}; \mathbf{x}, y)] = \mathcal{R}(\mathbf{w}, 0)$, where $\mathcal{R}$ is the risk defined in Eq. (1). Overloading our notation, we define the *empirical γ-loss* attained on a sample S as

$$\ell_\gamma(\mathbf{w}; S) \;=\; \frac{1}{m} \textstyle\sum_{i=1}^m \ell_\gamma(\mathbf{w}; \mathbf{x}_i, y_i)$$

We now state the main technical theorem of this section.

Theorem 2 *Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be a sample of size m drawn i.i.d from $\mathcal{D}$. For any $\gamma \geq 0, \kappa > 0$ and for any $\delta > 0$, with probability at least $1 - \delta$, it holds for all $\mathbf{w} \in \mathbb{R}^n$ with $\|\mathbf{w}\|_\infty \leq C$ that*

$$\mathbb{E}[\ell_\gamma(\mathbf{w}; \mathbf{x}, y)] \;\leq\; \sup\left\{ \epsilon \;:\; \mathrm{KL}\left(\ell_{\gamma+\kappa}(\mathbf{w}; S) \;\middle\|\; \epsilon \right) \leq \tfrac{\beta(m,\delta,\kappa)}{m-1} \right\},$$

where

$$\beta(m, \delta, \kappa) \;=\; \ln\left(\frac{m}{\delta}\right) + \sum_{j=1}^{n} \ln\left(\max\left\{ \frac{4PC}{\kappa v_j}, 1 \right\} \right)$$

and KL *is the Kullback-Leibler divergence. The above implies the weaker bound*

$$\mathbb{E}[\ell_\gamma(\mathbf{w}; \mathbf{x}, y)] \;\leq\; \ell_{\gamma+\kappa}(\mathbf{w}; S) \;+\; \sqrt{\frac{2\ell_{\gamma+\kappa}(\mathbf{w}; S)\beta(m, \delta, \kappa)}{m-1}} \;+\; \frac{2\beta(m, \delta, \kappa)}{m-1} \;.$$

Plugging in $\gamma = 0$, and using the weaker bound for simplicity, we get the following corollary:

Corollary 1 *Under the conditions of Thm. 2, for any $\kappa > 0$, it holds with probability at least $1 - \delta$ that the expected risk of any $\mathbf{w} \in \mathbb{R}^n$ (with $\|\mathbf{w}\|_\infty \leq C$) is at most*

$$\ell_\kappa(\mathbf{w}; S) \;+\; \sqrt{\frac{2\ell_\kappa(\mathbf{w}; S)\beta(m, \delta, \kappa)}{m-1}} \;+\; \frac{2\beta(m, \delta, \kappa)}{m-1} \;.$$

The proof of the theorem follows along similar lines to the PAC-Bayesian bound for linear classifiers in [16], while carefully working around the problems that arise from our non-standard definition of the γ-loss in Eq. (22). Our proof relies on the following lemma.

Lemma 2 *Let $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{x} \in [-1, 1]^n$, $y \in \{\pm 1\}$ and $\kappa > 0$ be such that $\ell_\kappa(\mathbf{w}; \mathbf{x}, y) = 0$. Let $\mathbf{w}' \in \mathbb{R}^n$ and $\kappa' \in [0, \kappa]$ be such that for all $j \in [n]$ it holds that $|w_j - w'_j| \leq \frac{\kappa' v_j}{P}$. Then it holds that $\ell_{\kappa-\kappa'}(\mathbf{w}'; \mathbf{x}, y) = 0$.*

Proof If $\ell_\kappa(\mathbf{w};\mathbf{x},y)=0$, it holds that

$$\forall J: V([n]\setminus J)\le N \qquad y\sum_{j\in J} w_j x_j \ge \frac{\kappa V(J)}{P}. \tag{23}$$

The conditions on $\mathbf{x}$, y, $\mathbf{w}'$, and κ' imply that $|yw_jx_j - yw'_jx_j| \le \frac{\kappa' v_j}{P}$, and particularly

$$\forall J: V([n]\setminus J)\le N \qquad y\sum_{j\in J} w_j x_j - y\sum_{j\in J} w'_j x_j \;\le\; \frac{\kappa' V(J)}{P}. \tag{24}$$

Subtracting both sides of the inequality in Eq. (24) from the respective sides of Eq. (23) proves the lemma. □

Proof of Thm. 2 To facilitate the proof, we introduce some additional notation. Given a distribution $\mathcal{Q}$ over the space of linear classifiers $[-C,C]^n$, define

$$\ell_\gamma(\mathcal{Q};S) = \mathbb{E}_{\mathbf{w}\sim\mathcal{Q}}[\ell_\gamma(\mathbf{w};S)] \ .$$

Furthermore, denote

$$\ell_\gamma(\mathcal{Q};\mathcal{D}) = \mathbb{E}_{\mathbf{w}\sim\mathcal{Q},\,(\mathbf{x},y)\sim\mathcal{D}}[\ell_\gamma(\mathbf{w};\mathbf{x},y)] \ .$$

Let $B\subseteq\mathbb{R}^n$ be an axis-aligned box, defined as

$$B=\prod_{j=1}^{n}\left[\max\left\{w_j-\frac{\kappa v_j}{2P},-C\right\}\ ,\ \min\left\{w_j+\frac{\kappa v_j}{2P},C\right\}\right] \ ,$$

and let $\mathcal{Q}$ be the uniform distribution over B. For any $\mathbf{w}'\in B$ and for all $j\in[n]$ it holds that

$$|w_j - w'_j| \;\le\; \frac{\kappa v_j}{2P} \ .$$

Combining the above inequality with Lemma 2, for any example $(\mathbf{x}_i,y_i)$ in our sample S, we have that $\ell_{\gamma+\kappa}(\mathbf{w};\mathbf{x}_i,y_i)=0$ implies $\ell_{\gamma+\kappa/2}(\mathbf{w}';\mathbf{x}_i,y_i)=0$, and that in turn implies $\ell_\gamma(\mathbf{w};\mathbf{x}_i,y_i)=0$. Overall, we have $\ell_\gamma(\mathbf{w};S)\le\ell_{\gamma+\kappa/2}(\mathbf{w}';S)\le\ell_{\gamma+\kappa}(\mathbf{w};S)$. These inequalities also hold if we take the expectation over $\mathbf{w}'$ sampled from $\mathcal{Q}$, namely,

$$\ell_\gamma(\mathbf{w};S)\ \le\ \ell_{\gamma+\kappa/2}(\mathcal{Q};S)\ \le\ \ell_{\gamma+\kappa}(\mathbf{w};S)\ . \tag{25}$$

The inequalities above continue to hold if we take expectation over S sampled i.i.d according to $\mathcal{D}$, giving

$$\ell_\gamma(\mathbf{w};\mathcal{D})\ \le\ \ell_{\gamma+\kappa/2}(\mathcal{Q};\mathcal{D})\ \le \ell_{\gamma+\kappa}(\mathbf{w};\mathcal{D})\ . \tag{26}$$

Now let $\mathcal{P}$ be the uniform distribution over the box $[-C,C]^n$, which defines the set of all possible classifiers. Using the PAC-Bayesian theorem [16], we have that, with probability at least $1-\delta$,

$$\ell_{\gamma+\kappa/2}(\mathcal{Q};\mathcal{D})\ \le\ \sup\left\{\epsilon\ :\ \mathrm{KL}\left(\ell_{\gamma+\kappa/2}(\mathcal{Q};S)\middle\|\epsilon\right)\le\frac{\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{m}{\delta}}{m-1}\right\},$$

From this, it follows that

$$\ell_\gamma(\mathbf{w};\mathcal{D})\ \le\ \sup\left\{\epsilon\ :\ \mathrm{KL}\left(\ell_{\gamma+\kappa}(\mathbf{w};S)\middle\|\epsilon\right)\le\frac{\mathrm{KL}(\mathcal{Q}\|\mathcal{P})+\ln\frac{m}{\delta}}{m-1}\right\}. \tag{27}$$

This is a straightforward consequence of Eq. (25),Eq. (26) and the convexity of the KL function.

Since $\mathcal{Q}$ and $\mathcal{P}$ are uniform, $\mathrm{KL}(\mathcal{Q}\|\mathcal{P})$ is simply the logarithm of the volume ratio between $[-C,C]^n$ and B, which is upper-bounded by

$$\mathrm{KL}(\mathcal{Q}\|\mathcal{P}) \leq \sum_{j=1}^{n} \ln\left(\max\left\{\frac{4PC}{\kappa v_j}, 1\right\}\right) .$$

□

It is interesting to note that L_∞ regularization emerges as the most natural one in this setting, since it induces the most convenient type of margin for relating the $\ell_\gamma, \ell_{\gamma+\kappa/2}, \ell_{\gamma+\kappa}$ loss functions as described above. This lends theoretical support to our choice of the L_∞ norm in our algorithms.

2.5 Feature Corrupting Noise

We now shift our attention to the case where a subset of the features is corrupted with random noise, and show that the the same LP approach used to handle missing features can also deal with corrupted features if one can attain a reasonably large margin. For simplicity, we shall assume that all features are supported on $[-1,1]$ with zero mean. Unlike the feature deleting noise, we now assume that each feature selected by the adversary is replaced with noise sampled from some distribution, also supported on $[-1,1]$ and having zero mean. The following theorem relates the risk of a classifier in the above setting, to its expected γ-loss (defined in Eq. (22)) in the feature deletion setting. The expected γ-loss, $\mathbb{E}[\ell_\gamma(\mathbf{w};\mathbf{x},y)]$, can then be bounded using Thm. 2.

Theorem 3 *Let ϵ, C, and N be arbitrary positives, and let γ be at least $C\sqrt{2N\ln(1/\epsilon)}$. Assume that we solve Eq. (4) with parameters C, N and with $v_j = 1$ for all $j \in [n]$. Let $\mathbf{w}$ be the resulting linear classifier, and assume for simplicity that the bias term b is zero. Let f be a random vector-valued function on $\mathcal{X}$, such that for every $\mathbf{x} \in \mathcal{X}$, $f(\mathbf{x})$ is the instance $\mathbf{x}$ after the feature corruption scheme described above. Then, using ℓ_γ as defined in Eq. (22), for $(\mathbf{x},y)$ drawn randomly from $\mathcal{D}$, we have:*

$$\Pr\left(y\langle\mathbf{w}, f(\mathbf{x})\rangle \leq 0\right) \;\leq\; \mathbb{E}\left[\ell_\gamma(\mathbf{w};\mathbf{x},y)\right] + \epsilon \;\; .$$

Proof Let $(\mathbf{x},y)$ be an example and let J denote the feature subset that remains uncorrupted by the adversary. Using Hoeffding's bound and our assumption on γ, we have that $\Pr\left(y\sum_{j\notin J} w_j f_j(\mathbf{x}) \leq -\gamma)\right)$ is upper bounded by ϵ. Therefore, with probability at least $1-\epsilon$ over the randomness of f, it holds that

$$y\langle\mathbf{w}, f(\mathbf{x})\rangle \;=\; y\sum_{j\in J} w_j x_j + y\sum_{j\notin J} w_j f_j(\mathbf{x}) \;>\; y\sum_{j\in J} w_j x_j - \gamma \;\; . \tag{28}$$

Let A denote the event that Hoeffding's bound holds (note that this event depends just on the randomness of the noise, not on $(\mathbf{x},y)$ or the features selected by the adversary). Thus, with probability at least $1-\epsilon$ over the randomness of f,

$$\Pr\left(y\langle\mathbf{w}, f(\mathbf{x})\rangle < 0 \middle| A\right) \;\leq\; \Pr\left(y\textstyle\sum_{j\in J} w_j x_j \leq \gamma \middle| A\right) \;\leq\; \mathbb{E}[\ell_\gamma(\mathbf{w};\mathbf{x},y)|A] \;\; .$$

With probability at most ϵ, A does not hold, and we have just the trivial bound $\Pr(y\langle\mathbf{w}, f(\mathbf{x})\rangle < 0|\neg A) \leq 1$. Using the law of total probability, the theorem follows. □

We conclude with an important observation. In the feature corruption setting, making a correct prediction boils down to achieving a sufficiently large margin on the uncorrupted features. Let $r \in (0,1)$ be a fixed ratio between N and n, and let n grow to infinity. Assuming a reasonable degree of feature redundancy, the term $y \sum_{j \in J} w_j x_j$ grows as $\Theta(n)$. On the other hand, Hoeffding's bound tells us that $y \sum_{j \notin J} w_j x_j$ grows only as $O(\sqrt{N})$. Therefore, for large enough n, the first sum in Eq. (28) dominates the second one. This holds for r arbitrarily close to 1. Namely, for problems with enough features and a reasonable feature redundancy assumption, our approach's ability to withstand feature corruption matches its ability to withstand feature deletion.

3 Solving the Problem with the Perceptron

We now turn to our second learning algorithm, taking a radically different angle on the problem. We momentarily forget about the original statistical learning problem and instead define a related online prediction problem. In online learning there is no distinction between the training phase and the classification phase, so we cannot perfectly replicate the classification-time noise scenario discussed above. Instead, we assume that an adversary removes features from every instance that is presented to the algorithm. We address this online problem with a modified version of the Perceptron algorithm [17] and use an online-to-batch conversion technique to convert the online algorithm back into a statistical learning algorithm. The detour through online learning gives us efficiency while the online-to-batch technique provides us with the statistical generalization properties we are interested in.

3.1 Perceptron with Projections onto the Cube

We start with a modified version of the well-known Perceptron algorithm [17], which observes a sequence of examples $\big((\mathbf{x}_i, y_i)\big)_{i=1}^m$, one example at a time, and incrementally builds a sequence $\big((\mathbf{w}_i, b_i)\big)_{i=1}^m$ of linear margin-based classifiers, while constraining them to a hyper-cube. Before processing example i, the algorithm has the vector $\mathbf{w}_i$ and the bias term b_i stored in its memory. An adversary takes the instance $\mathbf{x}_i$ and reveals only a subset J_i of its features to the algorithm, attempting to cause the online algorithm to make a prediction mistake. In choosing J_i, the adversary is restricted by the constraint $V([n] \setminus J) \leq N$. Next, the algorithm predicts the label associated with $\mathbf{x}_i$ to be

$$\text{sign}\left(b_i + \textstyle\sum_{j \in J_i} w_{i,j} x_{i,j}\right) \ .$$

After the prediction is made, the correct label y_i is revealed and the algorithms suffers a hinge-loss

$$\xi(\mathbf{w}, b; \mathbf{x}, y) \ = \ \left[\max_{J \,:\, V([n] \setminus J) \leq N} \frac{V(J)}{P} \ - \ y\big(b + \textstyle\sum_{j \in J} w_j x_j\big)\right]_+ \ , \tag{29}$$

where $P = V([n]) - N$ and $[\alpha]_+$ denotes the hinge function, $\max\{\alpha, 0\}$. Note that the hinge loss $\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i)$ upper-bounds the indicator of a prediction mistake on the current example, for any choice of J_i made by the adversary. We choose to denote the loss by ξ to emphasize the close relation between $\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i)$ and ξ_i in Eq. (3). Due to our choice of loss function, we can assume that the adversary chooses the subset J_i that inflicts the greatest loss.

The algorithm now uses the correct label y_i to construct the pair $(\mathbf{w}_{i+1}, b_{i+1})$, which is used to make the next prediction. If $\xi(\mathbf{w}, b; \mathbf{x}, y) = 0$, the algorithm defines $\mathbf{w}_{i+1} = \mathbf{w}_i$ and $b_{i+1} = b_i$. Otherwise, the algorithm defines $\mathbf{w}_{i+1}$ using the following coordinate-wise update

$$j \in [n] \quad w_{i+1,j} \;=\; \begin{cases} \left[w_{i,j} + y_i \tau x_{i,j}\right]_{\pm C} & \text{if } j \in J_i \\ w_{i,j} & \text{otherwise} \end{cases} \;,$$

and $b_{i+1} = [b_i + y_i \tau]_{\pm C}$, where $\tau = C\sqrt{n + 1/2m}$ and $[\alpha]_{\pm C}$ abbreviates the function $\max\left\{\min\{\alpha, C\}, -C\right\}$. This update is nothing more than the standard Perceptron update with constant learning rate τ, with an added projection step onto the hyper-cube of radius C. The specific value of τ used above is the value that optimizes the cumulative loss bound below. As in the previous section, restricting the online classifier to the hyper-cube helps us control its complexity, while promoting dense classifiers. It also comes in handy in the next stage, when we convert the online algorithm into a statistical learning algorithm.

Using a rather straightforward adaptation of standard Perceptron loss bounds, to the case where the hypothesis is confined to the hyper-cube, leads us to the following theorem, which compares the cumulative loss suffered by the algorithm with the cumulative loss suffered by any fixed hypothesis in the hyper-cube of radius C.

Theorem 4 *Choose any $C > 0$ and let $\mathbf{w}^\star \in \mathbb{R}^n$ and $b^\star \in \mathbb{R}$ be such that $\|\mathbf{w}^\star\|_\infty \leq C$ and $|b^\star| \leq C$. Let $\left((\mathbf{x}_i, y_i)\right)_{i=1}^m$ be an arbitrary sequence of examples, with $\|\mathbf{x}_i\|_1 \leq 1$ for all i. Assume that this sequence is presented to our modified Perceptron, and let $\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i)$ be as defined in Eq. (29). Then it holds that $\frac{1}{m}\sum_{i=1}^m \xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i)$ is upper-bounded by*

$$\frac{1}{m}\sum_{i=1}^m \xi(\mathbf{w}^\star, b^\star; \mathbf{x}_i, y_i) \;+\; C\sqrt{\frac{2(n+1)}{m}} \;.$$

Proof Define $\varDelta_i = \|\mathbf{w}_i - \mathbf{w}^\star\|_2^2 + (b_i - b^\star)^2 - \|\mathbf{w}_{i+1} - \mathbf{w}^\star\|_2^2 - (b_{i+1} - b^\star)^2$. We prove the theorem by bounding $\sum_{i=1}^m \varDelta_i$ from above and from below. First, we note that $\sum_{i=1}^m \varDelta_i$ is a telescopic sum that collapses to

$$\sum_{i=1}^m \varDelta_i \;=\; \|\mathbf{w}_1 - \mathbf{w}^\star\|_2^2 + (b_1 - b^\star)^2 - \|\mathbf{w}_{m+1} - \mathbf{w}^\star\|_2^2 - (b_{m+1} - b^\star)^2 \;.$$

Using the facts that $\mathbf{w}_1$ is the zero vector, $b_1 = 0$, and $\|\mathbf{w}_{m+1} - \mathbf{w}^\star\|_2^2 + (b_{m+1} - b^\star)^2 \geq 0$, we obtain the upper bound

$$\sum_{i=1}^m \varDelta_i \;\leq\; \|\mathbf{w}^\star\|_2^2 + (b^\star)^2 \;\leq\; (n+1)C^2 \;. \tag{30}$$

Next, we lower bound each $\varDelta_i$ individually. Let i be the index of a round on which a positive loss is incurred, namely, $\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) > 0$. Let $\mathbf{x}'$ be the vector defined by

$$\forall j \in \{1, \ldots, n\} \quad x'_j \;=\; \begin{cases} x_{i,j} & \text{if } j \in J_i \\ 0 & \text{otherwise} \end{cases} \;,$$

and define $\mathbf{w}' = \mathbf{w}_i + y_i \tau \mathbf{x}'$ and $b' = y_i \tau$. Note that $w_{i+1,j} = [w'_j]_{\pm C}$ for all j, and that $b_{i+1} = [b']_{\pm C}$. We can rewrite $\varDelta_i$ as

$$\begin{aligned} \varDelta_i = &\left(\|\mathbf{w}_i - \mathbf{w}^\star\|^2 + (b_i - b^\star)^2 - \|\mathbf{w}' - \mathbf{w}^\star\|^2 - (b' - b^\star)^2\right) \\ &+ \left(\|\mathbf{w}' - \mathbf{w}^\star\|^2 + (b' - b^\star)^2 - \|\mathbf{w}_{i+1} - \mathbf{w}^\star\|^2 - (b_{i+1} - b^\star)^2\right) \;, \end{aligned} \tag{31}$$

denoting the first term on the right-hand side above by α and the second term by β. Using the definitions of $\mathbf{w}'$ and b', α can be rewritten as

$$\|\mathbf{w}_i - \mathbf{w}^\star\|^2 + (b_i - b^\star)^2 - \|\mathbf{w}_i + y_i\tau\mathbf{x}' - \mathbf{w}^\star\|^2 - (b_i + y_i\tau - b^\star)^2 \ .$$

Using the facts that $\|\mathbf{w}_i - \mathbf{w}^\star + y_i\tau\mathbf{x}'\|^2 = \|\mathbf{w}_i - \mathbf{w}^\star\|^2 + 2y_i\tau\langle\mathbf{x}', \mathbf{w}_i - \mathbf{w}^\star\rangle + \tau^2\|\mathbf{x}'\|^2$ and $(b_i - b^\star + y_i\tau)^2 = (b_i - b^\star)^2 + 2y_i\tau(b_i - b^\star) + \tau^2$, we can rewrite α as

$$-2y_i\tau\langle\mathbf{x}', \mathbf{w}_i - \mathbf{w}'\rangle - 2y_i\tau(b_i - b^\star) - \tau^2\left(\|\mathbf{x}'\|^2 + 1\right) \ .$$

By definition, $\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) = \frac{V(J_i)}{P} - y_i b_i - y_i\langle\mathbf{w}_i, \mathbf{x}_i\rangle$ and $\xi(\mathbf{w}^\star, b^\star; \mathbf{x}_i, y_i) \geq \frac{V(J_i)}{P} - y_i b^\star - y_i\langle\mathbf{w}^\star, \mathbf{x}_i\rangle$. We also know that $\|\mathbf{x}_i\|_2^2 \leq \|\mathbf{x}_i\|_1^2 \leq 1$. We use these facts to obtain the following lower bound,

$$\alpha \ \geq \ 2\tau\big(\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) - \xi(\mathbf{w}^\star, b^\star; \mathbf{x}_i, y_i)\big) - 2\tau^2 \ .$$

Moving onto the second term on the right-hand side of Eq. (31), note that if $|b'| \leq C$ then $(b' - b^\star)^2 - (b_{i+1} - b^\star)^2 = 0$. Otherwise, assuming w.l.o.g. that $b' \geq 0$, we have

$$(b_{i+1} - b^\star)^2 = (C - b^\star)^2 < (C - b^\star + |b' - C|)^2 = (b' - b^\star)^2 \ .$$

Therefore, $(b' - b^\star)^2 - (b_{i+1} - b^\star)^2$ is always non-negative. The same argument applies to $(w'_j - w^\star_j)^2 - (w_{i+1,j} - w^\star_j)^2$ for all j. Overall, we have that $\beta \geq 0$, and that

$$\Delta_i \ \geq \ \alpha \ \geq \ 2\tau\big(\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) - \xi(\mathbf{w}^\star, b^\star; \mathbf{x}_i, y_i)\big) - 2\tau^2 \ . \tag{32}$$

Recall that the above holds for all rounds on which $\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) > 0$. On rounds on which $\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) = 0$, the above holds trivially, since the left hand side equals zero while the right hand side is non-positive. We conclude that Eq. (32) holds for all i. Summing Eq. (32) over all i in $1, \ldots, m$, we get

$$\sum_{i=1}^m \Delta_i \ \geq \ 2\tau\sum_{i=1}^m \big(\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) - \xi(\mathbf{w}^\star, b^\star; \mathbf{x}_i, y_i)\big) - 2m\tau^2 \ .$$

Comparing the above to the upper bound in Eq. (30) and rearranging terms, we get

$$\frac{1}{m}\sum_{i=1}^m \xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) \ \leq \ \frac{1}{m}\sum_{i=1}^m \xi(\mathbf{w}^\star, b^\star; \mathbf{x}_i, y_i) \ \leq \ \frac{(n+1)C^2}{2\tau m} + \tau \ .$$

Plugging in the definition of τ proves the bound. □

We now have an online learning algorithm for our problem, and the next step is to convert it into a statistical learning algorithm, with a risk bound.

3.2 Converting Online to Batch

To obtain a statistical learning algorithm, with risk guarantees, we assume that the sequence of examples presented to the modified Perceptron algorithm is a training set sampled i.i.d from the underlying distribution $\mathcal{D}$. We turn to the simple averaging technique presented in [5] and define $\bar{\mathbf{w}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{w}_{i-1}$ and $\bar{b} = \frac{1}{m}\sum_{i=1}^{m} b_{i-1}$. $(\bar{\mathbf{w}}, \bar{b})$ is called the *average hypothesis*, and defines our robust classifier. We use the derivation in [5] to prove that the average classifier provides an adequate solution to our original problem.

Note that the loss function we use, defined in Eq. (29), is bounded and convex in its first two arguments. Using [5, Corollary 2], we have that for any $\delta > 0$, with probability at least $1 - \frac{\delta}{2}$ over the random sampling of S, the average hypothesis $(\bar{\mathbf{w}}, \bar{b})$ satisfies

$$\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\xi(\bar{\mathbf{w}}, \bar{b}, \mathbf{x}, y)\right] \;\leq\; \frac{1}{m}\sum_{i=1}^{m}\xi(\mathbf{w}_i, b_i; \mathbf{x}_i, y_i) \;+\; (2C+\phi)\sqrt{\frac{\ln(\frac{2}{\delta})}{2m}} \;. \tag{33}$$

Setting

$$(\mathbf{w}^\star, b^\star) \;=\; \arg\min_{(\mathbf{w},b)} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\xi(\mathbf{w}, b; \mathbf{x}, y)\right] \quad \text{s.t.} \quad \|\mathbf{w}\|_\infty \leq C \quad \text{and} \quad |b| \leq C \;,$$

we use Hoeffding's bound to get, for any $\delta > 0$, with probability at least $1 - \frac{\delta}{2}$ over the random sampling of S, that

$$\frac{1}{m}\sum_{i=1}^{m}\xi(\mathbf{w}^\star, b^\star; \mathbf{x}_i, y_i) \;\leq\; \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\xi(\mathbf{w}^\star, b^\star; \mathbf{x}, y)\right] \;+\; (2C+\phi)\sqrt{\frac{\ln(\frac{2}{\delta})}{2m}} \;. \tag{34}$$

Finally, using the union bound, Eq. (33) and Eq. (34) hold simultaneously with probability at least $1 - \delta$. Combining Eq. (33) and Eq. (34) with the inequality in Thm. 4 proves the following corollary.

Corollary 2 *For any $\delta > 0$, with probability at least $1 - \delta$ over the random sampling of S, our algorithm constructs $(\bar{\mathbf{w}}, \bar{b})$ such that $\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\xi(\bar{\mathbf{w}}, \bar{b}, \mathbf{x}, y)\right]$ is at most*

$$\min_{(\mathbf{w},b)\in\mathcal{H}} \mathbb{E}\left[\xi(\mathbf{w}, b; \mathbf{x}, y)\right] \;+\; (3C+\phi)\sqrt{\frac{2(n+1+\ln(\frac{2}{\delta}))}{m}} \;,$$

where $\phi = \max_{J:V([n]\setminus J)\leq N}\big(V(J)/P\big)$, and $\mathcal{H}$ is the set of all pairs $(\mathbf{w}, b)$ such that $\|\mathbf{w}\|_\infty \leq C$ and $|b| \leq C$.

Using the fact that the hinge loss upper-bounds the indicator function of a prediction mistake, regardless of the adversary's choice of the feature set, we have that the expected hinge loss upper-bounds $\mathcal{R}(\bar{\mathbf{w}}, \bar{b})$.

4 Experiments

In this section, we experimentally investigate the efficacy of our two proposed algorithms in the face of feature-deleting and feature-corrupting adversaries. We use LP when referring to our linear programming based approach and O2B when referring to our online-to-batch based approach. We compare the performance of these algorithms with the performances of the following two algorithms:

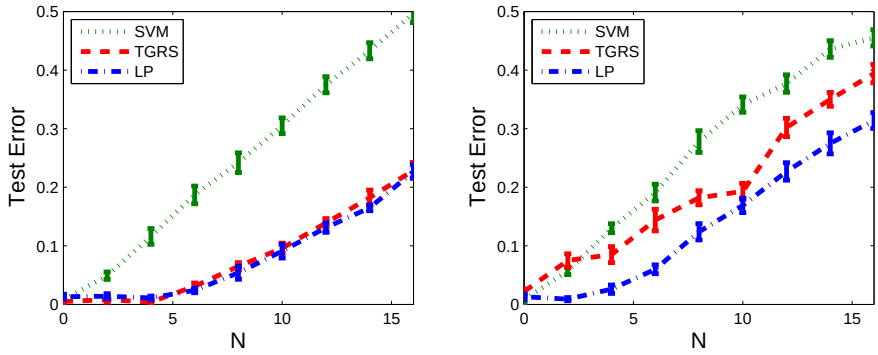

Fig. 1 Results of the synthetic experiment based on feature redundancy. For each value of N, we report average results over the 10-fold cross validation, as well as standard deviation. The left figure displays the results for the feature deletion scenario, whereas the right figure displays the results for the feature corruption scenario.

SVM - A linear L_2 support vector machine (using SVM$^{\text{light}}$ [12]), which is trained without regard to feature deletion/corruption in the test set. This algorithm allows us to study the effect an adversary might have on a generic learning algorithm that is not tailored to this setting.

TGRS - The robust learning algorithm presented in [10]. Concretely, we implemented the efficient version of this algorithm, using a stochastic gradient-descent algorithm, as described in [18]. As far as we know, this algorithm represents the current state-of-the-art for the setting considered in this paper.

In all of our experiments, we simulated the adversary by greedily choosing the most valuable features for each example, until the limit of N is reached. Specifically, the adversary sorts the features in descending order by yw_jx_j/v_j, and considers them one by one. He chooses to remove/corrupt feature j if $yw_jx_j > 0$, and the noise limit N is still respected after the removal. These chosen features are then either replaced with zeros, or replaced with random Gaussian noise with the same mean and variance as the original feature.

4.1 Illustrative Synthetic Experiments

We begin with two illustrative synthetic experiments, which are meant to cleanly demonstrate the importance of robust classification when one is faced with missing and corrupted features. The first experiment is as follows: We generated a synthetic dataset of 1000 linearly separable instances in $\mathbb{R}^{20}$ and added label noise by flipping each label with probability 0.2. Then, we added two copies of the actual label as additional features to each instance, for a total of 22 features. We randomly split the data into equally sized training and test sets, and trained an SVM classifier on the training set. We set $v_j = 1$ for $j \in [20]$ and $v_{21} = v_{22} = 10$, expressing our prior knowledge that the last two features are more valuable. Using these feature values, we applied our LP-based algorithm with different values of the parameter N. We removed one or both of the high-value features from the test set and evaluated the classifiers. With only one feature removed both SVM and our approach attained a test error of zero. With two features removed, the test error of the SVM classifier jumped to 0.477 ± 0.004 (over 100 random repetitions of the experiment), indicating that it essentially put all of its

weight on the two perfect features. With the noise parameter set to $N = 20$, our approach attained a test error of only 0.22 ± 0.002. This is only marginally above the best possible error rate for this setting.

Our second synthetic experiment focused on the way our approach utilizes feature redundancy. The datasets for this experiment was created as follows: We started by creating a linearly separable sample $\{(\mathbf{u}_i, y_i)\}_{i=1}^{200}$, where each $\mathbf{u}_i$ is a column vector in $\mathbb{R}^3$. This was done by sampling points from a standard Gaussian distribution in $\mathbb{R}^n$, choosing a random hyperplane in $\mathbb{R}^n$ and using it to label the points, and finally removing points whose distance from the hyperplane was less than 1. Next, we generated a random matrix A of size 40×3 and set $\mathbf{x}_i = A\mathbf{u}_i$ for all i. The result is a set of instances in $\mathbb{R}^{40}$ with a large amount of feature redundancy. Formally, any 3 features out of the 40 suffice to linearly separate the sample, with probability 1. We then added random Gaussian noise to each feature, where the noise distribution used for feature j was $\mathcal{N}(0, 0.15j)$. In other words, the magnitude of noise increased with the feature index, making the quality of the various features less homogeneous.

We trained classifiers using the LP algorithm, the TGRS algorithm, and the SVM algorithm, on 10 random train-test splits, with different values of the noise parameter N. Parameter tuning, using logarithmic grid search, was done based on a held-out validation set taken from the training data. We simulated the adversary with the appropriate level of N for each classifier, and the results of this experiment are displayed in Fig. 1.

It is readily seen that our LP algorithm produces a classifier considerably more robust than SVM. SVM put a significant portion of its weight on a small number of highly informative features, and did not take full advantage of the feature redundancy of the data. Compared to the TGRS algorithm, we achieve similar results in the feature deletion scenario, and superior results in the feature corruption scenario.

4.2 Main Experimental Results

Our main set of experiments were conducted using the following publicly available datasets:

`breast` : The Breast Cancer Wisconsin (Diagnostic) Dataset from the UCI repository [1]. This dataset specifies characteristics of cell nuclei, and the goal is to characterize a tumor as either malignant or benign. The datasets contains 569 instances, of which 357 are benign and 212 are malignant, with 10 features each.

`spam` : The Spambase Dataset from the UCI repository [1], which contain e-mails (described mostly by word counts) classified as either spam or non-spam. The dataset contains 4601 instances, with 57 features each.

`usps` : The training set of the USPS dataset of handwritten digits [11], which contains 9298 images, each assigned one of ten possible labels. Each image is represented by a 16×16 gray-scale pixel-map, for a total of 256 features. Since our algorithms are designed to deal with binary classification problems, we constructed a binary dataset from each pair of labels, for a total of $\binom{10}{2} = 45$ different problems.

`mnist` : The MNIST dataset of handwritten digits [13], which contains 70,000 images. Each image is represented by a 26×26 gray-scale pixel-map, for a total of 784 features. As with the previous dataset, we used this dataset to generate 45 different binary classification problems.

We deliberately chose datasets of various sizes and with different levels of feature redundancy. As a rule of thumb, it should be expected that high-dimensional datasets, those containing many features, will have more feature redundancy. Thus, even if the same fraction of features is deleted or corrupted, better results are expected on high-dimensional datasets. This intuition is substantiated in the results reported below. We tested both feature deletion and feature corruption scenarios with all datasets.

The summary of our empirical results is as follows. Our algorithms significantly outperform SVM in all but one experiment, which involved feature corruption with the `breast` dataset. In this one case, all of the tested algorithms performed equally well. On the `breast` and `spam` datasets, where feature redundancy is not especially high, the performance of our algorithms is indistinguishable from the state-of-the-art TGRS algorithm. In other words, the moderate level of feature redundancy in these datasets leaves little room to improve over the TGRS classifier. However, on the `usps` and `mnist` datasets, where feature redundancy is higher, our algorithms significantly outperform TGRS in both feature deletion and feature corruption scenarios. In the remainder of this section, we present these results in detail.

An important decision we had to make when conducting these experiments is how to choose the value v_i associated with each feature. Recall that these values represent the importance of the respective features to our classification problem, and that the adversary uses these values to determine how much damage he is allowed to inflict. The simplistic choice of setting all of these values to 1 is unsuitable for some of the datasets considered here. For example, when our features represent pixels in an image, the corner pixels are much less informative than the features in the center of the image. We used a heuristic, based on mutual information, to set these values. Formally, we set v_j to be

$$v_j \;=\; \tfrac{1}{Z} \max_{c \in \mathbb{R}} I\big([\![X_j > c]\!]; Y\big) \;\;,$$

where (X_j, Y) are random variables jointly distributed according to the uniform distribution over the set $\{(x_{i,j}, y_i)\}_{i=1}^m$, and where Z is set such that $\sum v_j = n$. Roughly speaking, our heuristic calculates the information contained in the optimal linear threshold function applied to each individual feature.

On some datasets, such as `spam` , we observe that most of the features are equally important, and setting v_j using this heuristic is not different than setting $v_j = 1$ for all j. On other datasets, such as `mnist` , setting $v_j = 1$ for all j enables the adversary to completely devastate our classifiers, as well as the classifiers trained using SVM and TGRS, even with small values of N. It is reasonable to assume that prior knowledge on the importance of each feature could be used to make important features less susceptible to malicious corruption. In the image recognition example given above, we could conceivably use a more fault tolerant sensor on the important pixels. In our formulation of the learning problem, the varying importance of different features is precisely captured by our non-uniform choice of v_j.

We tested robustness to both feature deletion and feature corruption on ten different train-test splits. We performed parameter tuning over a logarithmic grid of candidate parameters, using a held-out validation set taken from the training data. The first dataset tested was the relatively small `breast` dataset, with the results displayed in Fig. 2. In the feature deletion scenario, all noise-robust algorithms perform approximately the same, and better than the standard SVM. Once more, this shows the importance of robustness to feature deletion at test time. In the feature corruption scenario, however, the results are noisy, without a significant difference in the performance of the algorithms. This should not come as too much of a surprise, due to the low dimension of this dataset. In this setting, it is very difficult to overcome random Gaussian noise even when it is applied to a small number of features.

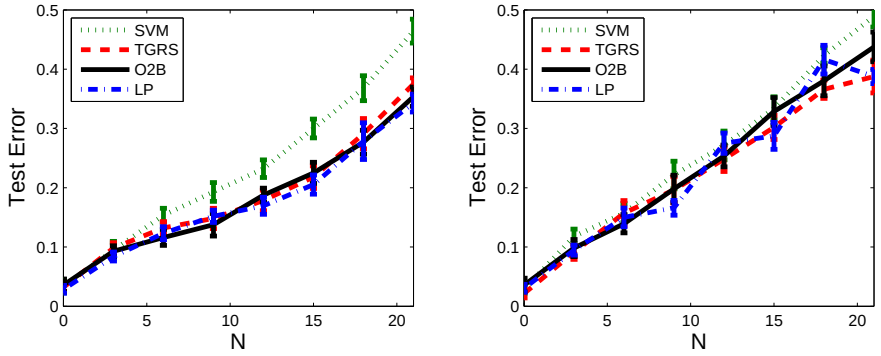

Fig. 2 Results of the `breast` dataset, for different values of N, with standard error. The left figure displays the results for the feature deletion scenario, whereas the right figure displays the results for the feature corruption scenario.

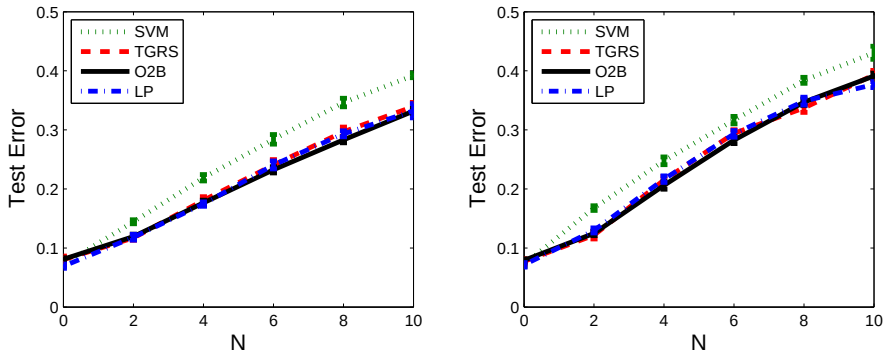

Fig. 3 Averaged results over 10 train-test splits of the `spam` dataset, for different values of N, with standard error. The left figure displays the results for the feature deletion scenario, whereas the right figure displays the results for the feature corruption scenario.

In the larger `spam` dataset, the results are better than the `breast` dataset (see Fig. 3). In both the feature deletion and feature corruption scenario, our algorithm outperforms SVM, but still achieve approximately the same accuracy as TGRS.

For the digit datasets, `usps` and `mnist` , we performed an all-pairs experiment, namely, we tested the performance of the algorithms on the binary classification problem defined by every possible digit pair (45 pairs in all). We present the average results over all digit pairs for both feature deletion and feature corruption in Fig. 4 and Fig. 6. We also present the results for each individual digit pair in the feature deletion scenario in Fig. 5 and Fig. 7. In all cases considered, our proposed algorithm clearly achieved better results than both SVM and TGRS.

5 Discussion

We presented two learning algorithms that anticipate adversarial feature deletion and feature corruption at classification time. A common idea behind both algorithms is that they simu-

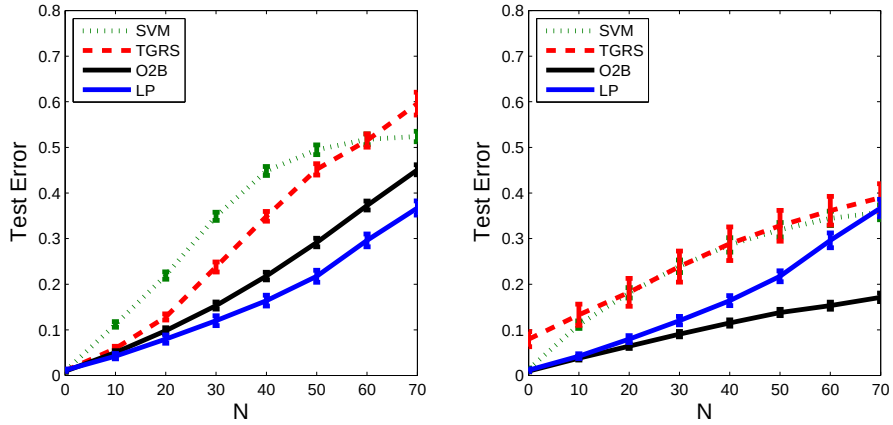

Fig. 4 Averaged results over 10 train-test splits on the usps dataset for different values of N, with standard error. The results displayed here are averaged over all 45 digit pairs. The left figure displays the results for the feature deletion scenario, whereas the right figure displays the results for the feature corruption scenario.

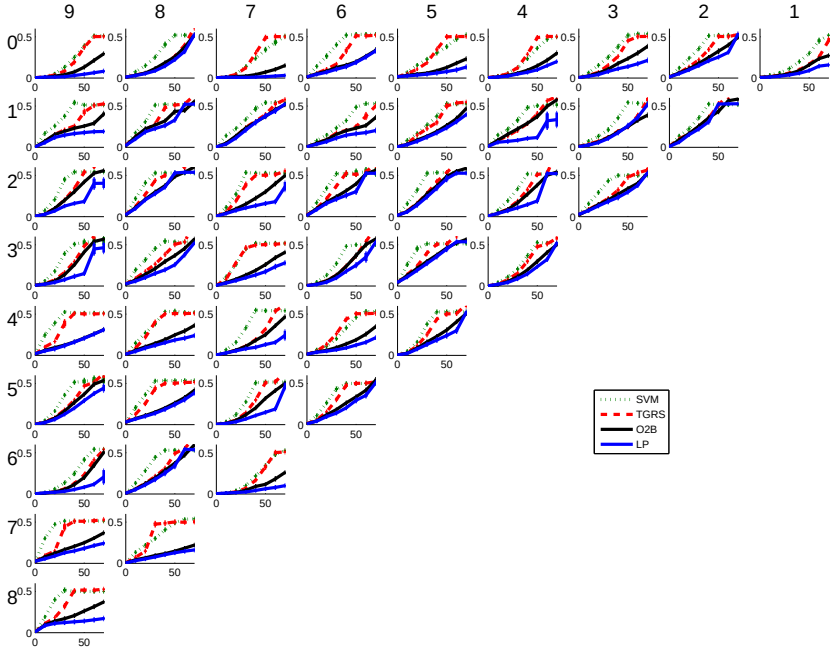

Fig. 5 Averaged results over 10 train-test splits on the usps dataset in the feature deletion scenario, for different values of N, with standard error. Plot in row i and column j represent classification results on a dataset composed of digits i and j.

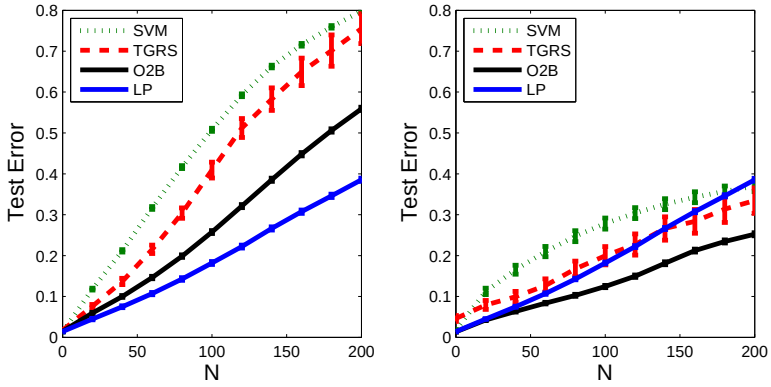

Fig. 6 Averaged results over 10 train-test splits on the `mnist` dataset for different values of N, with standard error. The results displayed here are averaged over all 45 digit pairs. The left figure displays the results for the feature deletion scenario, whereas the right figure displays the results for the feature corruption scenario.

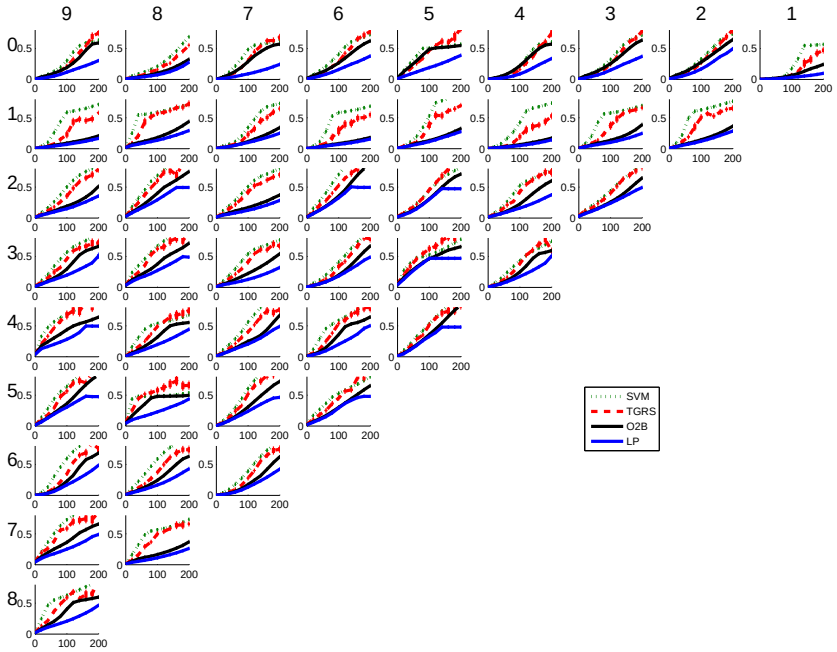

Fig. 7 Averaged results over 10 train-test splits on the `mnist` dataset in the feature deletion scenario, for different values of N, with standard error. Plot in row i and column j represent classification results on a dataset composed of digits i and j.

late the actions of the adversary on the training data, and use L_∞ regularization to promote classifier density. Both algorithms come with statistical risk bounds, despite the fact that the algorithms encounter different distributions at training time and at classification time. Our experiments demonstrate a significant improvement over SVM across the board, and a significant improvement over the current state-of-the-art technique on problems with sufficient feature redundancy.

Our two algorithms come with similar theoretical guarantees and perform comparably well in practice. The LP approach seems to have better accuracy when features are deleted, while the O2B algorithm performs better in the feature corruption scenario. A main technical difference between the two algorithms is their use of memory: our interior point solution keeps the entire linear program in memory while the online-to-batch algorithm streams through the data and has a constant-size memory footprint. For example, applying our implementation of the interior point LP solver to the MNIST dataset required a server with a 16GB memory. Additionally, the online-to-batch solution is simpler and easier to implement. These advantages make the online-to-batch approach a more practical solution. On the other hand, in the feature deletion scenario, the LP approach seems to be more accurate.

This work focuses on static adversaries, which do not evolve and improve with time. An interesting extension of this work would be to deal with adaptive adversaries, which corrupt features one by one over time. Our online-to-batch approach could serve as a useful starting-point for this research direction, as it uses an online learning algorithm as its main building block. Although time is not an explicit component in our model, our algorithms can still be useful when the adversary adapts. Concretely, consider the spam filtering example described in the introduction, and assume that the spammer corrupts features one by one. After enough time goes by and enough features become permanently damaged, our only option is to design new features and to retrain a new classifier. This is inevitable in any "arms-race" with an adversary. However, a robust classifier is able to survive for a longer period of time before it must be replaced. By deliberately hedging our bets across many features, we are able to slow down the arms-race cycle and to give ourselves more time to respond to new attacks.

On a more general note, our work seems to have an interesting duality with a recent trend in machine learning research, which is to develop sparse classifiers supported on a small subset of the features. In our setting, we are interested in the exact opposite, and the efficacy of using the L_∞ norm is clearly demonstrated in our theory and in our empirical evaluation. The trade-off between robustness and sparsity provides fertile ground for future research.

Acknowledgements We thank the anonymous reviewers of this paper for helpful comments and suggestions.

References

1. A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.
2. K.P. Bennett. Combining support vector and mathematical programming methods for classification. In *Advances in kernel methods: support vector learning*, pages 307–326. MIT Press, 1999.
3. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
4. R.D. Carr and G. Lancia. Compact vs. exponential-size LP relaxations. Technical Report SAND2000-2170, SANDIA Report, September 2000.

5. N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of online learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, September 2004.
6. N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 99–108, New York, NY, 2004. ACM Press.
7. Ofer Dekel and Ohad Shamir. Learning to classify with missing and corrupted features. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, 2008.
8. T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.
9. E.S. Gamble, S.A. Macskassy, and S. Minton. Classification with pedigree and its applicability to record linkage. In *Workshop on Text-Mining & Link-Analysis*, 2007.
10. A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 353–360, 2006.
11. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.
12. T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.
13. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
14. N. Littlestone. Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 147–156, 1991.
15. D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of The Second Conference on Email and Anti-Spam (CEAS)*, 2005.
16. D.A. McAllester. Simplified PAC-bayesian margin bounds. In *Proceedings of the Sixteenth Annual Conference on Computational Learning Theory*, pages 203–215, 2003.
17. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.
18. C-H. Teo, A. Globerson, S. Roweis, and A.J. Smola. Convex learning with invariances. In *Advances in Neural Information Processing Systems 21*, 2008.
19. L.N. Trefethen and D. Bau, III. *Numerical Linear Algebra*. SIAM, 1997.
20. V.N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.
21. G. Wittel and S. Wu. On attacking statistical spam filters. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004.
22. S.J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.

A Proof of Lemma 1

We prove claim (a) by proving its counter-positive, namely, we assume that there exists a feature subset J with $V([n] \setminus J) \leq N$ for which Eq. (5) does not hold, and we prove that $\theta < -\xi_i$. Set $\tau'_j = 1$ for all $j \in J$ and $\tau'_j = 0$ for all $j \in ([n] \setminus J)$. We now have that,

$$\theta \;\leq\; y_i b + \sum_{j=1}^{n} \tau'_j \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) < -\xi_i \;\;,$$

where the first inequality follows from the fact that the vector $(\tau'_1, \ldots, \tau'_n)$ is a feasible point of Eq. (7), and the second inequality follows from the assumption that J violates Eq. (5) and from $\sum_{j=1}^{n} \tau'_j v_j = V(J)$.

Moving on to claim (b), let $\boldsymbol{\tau}$ be a minimizer of Eq. (7) and let s be the number of elements of $\boldsymbol{\tau}$ in $(0, 1)$. If $s \leq 1$ then there is nothing to prove, so we assume that $s \geq 2$. We prove claim (b) by showing we can find another minimizer of Eq. (7), which we denote by $\boldsymbol{\tau}'$, with at most $s - 1$ elements in $(0, 1)$.

First, we deal with some very simple cases. If $\tau_j \in (0, 1)$ and $w_j x_{i,j} = 0$, then set $\tau'_j = 1$, and set the remaining elements of $\boldsymbol{\tau}'$ equal to the respective elements in $\boldsymbol{\tau}$. The new vector $\boldsymbol{\tau}'$ clearly satisfies the constraints of Eq. (7), while obtaining an objective function less than or equal to that of $\boldsymbol{\tau}$. We conclude that $\boldsymbol{\tau}'$ is a minimizer of Eq. (7) with at most $s - 1$ non-integer elements. If $w_j x_{i,j} \neq 0$ but $v_j = 0$ then define

$$\tau'_j \;=\; \begin{cases} 1 \text{ if } y_i w_j x_{i,j} < 0 \\ 0 \text{ otherwise} \end{cases} \;\;,$$

and set the remaining elements of $\boldsymbol{\tau}'$ equal to the respective elements in $\boldsymbol{\tau}$. Again, we have found a minimizer of Eq. (7) with at most $s - 1$ non-integer elements. Having dealt with these simple cases first, we can now assume that $w_j x_{i,j} \neq 0$ and that but $v_j > 0$.

If $s \geq 2$, assume without loss of generality that $0 < \tau_1 < 1$ and $0 < \tau_2 < 1$. Using our assumption that $v_1 > 0$ and $v_2 > 0$, we assume without loss of generality that

$$\frac{y_i w_1 x_{i,1}}{v_1} \;\leq\; \frac{y_i w_2 x_{i,2}}{v_2} \;\;. \tag{35}$$

We deal with two separate cases. First, if $\tau_1 + (\tau_2 v_2 / v_1) \leq 1$ then define $\tau'_1 = \tau_1 + (\tau_2 v_2 / v_1)$, $\tau'_2 = 0$, and $\tau'_j = \tau_j$ for all $j \in \{3, \ldots, n\}$. We now have that $\sum_{j=1}^{n} \tau'_j v_j = \sum_{j=1}^{n} \tau_j v_j$, so $\boldsymbol{\tau}'$ is a feasible point of Eq. (7). On the other hand, using the assumption in Eq. (35), we also have that $(\tau_2 v_2 / v_1) y_i w_1 x_{i,1} \leq \tau_2 y_i w_2 x_{i,2}$, and therefore

$$y_i \sum_{j=1}^{n} \tau'_j w_j x_{i,j} \;\leq\; y_i \sum_{j=1}^{n} \tau_j w_j x_{i,j} \;\;.$$

Once again, using the fact that $\sum_{j=1}^{n} \tau'_j v_j = \sum_{j=1}^{n} \tau_j v_j$, we conclude that

$$y_i b + \textstyle\sum_{j=1}^{n} \tau'_j \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) \;\leq\; y_i b + \sum_{j=1}^{n} \tau'_j \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) \;\;,$$

and therefore $\boldsymbol{\tau}'$ is a minimizer of Eq. (7) with at most $s - 1$ elements in $(0, 1)$.

The second case is where $\tau_1 + (\tau_2 v_2 / v_1) > 1$. In this case, set $\tau'_1 = 1$, $\tau'_2 = \tau_2 - (1 - \tau_1) v_1 / v_2$, and once again, $\tau'_j = \tau_j$ for all $j \in \{3, \ldots, n\}$. Our assumptions imply that $\tau_2 > \tau'_2 > 0$, and by definition we have $\sum_{j=1}^{n} \tau'_j v_j = \sum_{j=1}^{n} \tau_j v_j$. Therefore, $\boldsymbol{\tau}'$ is a feasible point of Eq. (7). We can rewrite

$$y_i \sum_{j=1}^{n} \tau'_j w_j x_{i,j} \;=\; y_i \sum_{j=1}^{n} \tau_j w_j x_{i,j} \;+\; \left((1 - \tau_1) y_i w_1 x_{i,1} - \tfrac{v_1}{v_2} (1 - \tau_1) y_i w_2 x_{i,2} \right) \;\;.$$

Using Eq. (35), we know that the term in brackets above is non-positive. We conclude that the value of the objective function obtained by $\boldsymbol{\tau}'$ is smaller or equal to the value obtained by $\boldsymbol{\tau}$. Again, we conclude that $\boldsymbol{\tau}'$ is a minimizer of Eq. (7) with only $s - 1$ elements in $(0, 1)$. This concludes the proof of claim (b).

Finally, we turn to proving claim (c). We assume that $v_j \in \{0, 1\}$ for all $j \in [n]$ and that N is an integer. One direction of the claim follows from claim (a), so we focus on the other direction, namely, we assume that Eq. (5) holds for all J with $V([n] \setminus J) \leq N$, and we prove $\theta \geq -\xi_i$. As we have just shown, there exists a minimizer of Eq. (7), which we now denote by $\boldsymbol{\tau}$, that has at most one element in $(0, 1)$. If all of the elements of $\boldsymbol{\tau}$ are integers then $\boldsymbol{\tau}$ is also a solution to Eq. (6). As we have previously seen,

checking that the value of Eq. (6) upper bounds $-\xi_i$ is equivalent to verifying that Eq. (5) holds for all J with $V([n] \setminus J) \leq N$. Therefore, we assume without loss of generality that $0 < \tau_1 < 1$, and that $\tau_j \in \{0, 1\}$ for all $j \in \{2, \ldots, n\}$. It must be the case that

$$y_i w_1 x_{i,1} - \frac{v_1}{P} \;\geq\; 0 \;, \tag{36}$$

since otherwise we could decrease the objective function by increasing τ_1, without violating any of the constraints in Eq. (7). That would be in contradiction to our assumption that $\boldsymbol{\tau}$ attains a global minimum. Now define $\tau'_1 = 0$ and $\tau'_j = \tau_j$ for all $j \in \{2, \ldots, n\}$. We know that $\sum_{j=1}^n \tau_j v_j \geq P$ and we assumed that P is an integer. Therefore,

$$\left\lfloor \sum_{j=1}^{n} \tau_j v_j \right\rfloor \;\geq\; P \;.$$

Now note that the left-hand side above equals $\sum_{j=1}^n \tau'_j v_j$, and therefore $\boldsymbol{\tau}'$ is a feasible point of Eq. (7). Using Eq. (36), we have that

$$y_i b + \textstyle\sum_{j=1}^n \tau'_j \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) \;\leq\; y_i b + \sum_{j=1}^n \tau'_j \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) \;,$$

and $\boldsymbol{\tau}'$ is a minimizer of Eq. (7) whose elements all have values in $\{0, 1\}$. If $\theta < -y_i b - \xi_i$ then define $J = \{j \,:\, \tau'_j = 1\}$, and

$$y_i b + \textstyle\sum_{j \in J} \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) \;=\; y_i b + \sum_{j=1}^n \tau'_j \left(y_i w_j x_{i,j} - \frac{v_j}{P} \right) \;=\; \theta \;<\; -\xi_i \;.$$

Rearranging terms above gives

$$y_i \big(b + \textstyle\sum_{j \in J} w_j x_{i,j} \big) \;<\; \frac{V(J)}{P} - \xi_i \;.$$

This concludes our proof.

B Flop Counts for Solving the Normal Equation

In this appendix we give detailed analysis of the flop counts for solving the normal equation (13). We first go through the four steps of the block-elimination approach:

1. We solve the linear equations (15) using the *factor-solve* approach (e.g., [3, Appendix C.2]). First we conduct the Cholesky factorization

   $$\mathbf{A}_i \mathbf{D}_i \mathbf{A}_i^T + \mathbf{D}_{m+i} = \mathbf{L}_i \mathbf{L}_i^T$$

   where $\mathbf{L}_i$ is a lower-triangular matrix. Then we do forward and backward substitutions (the solve step):

   $$\mathbf{L}_i \tilde{\mathbf{z}}_i = \mathbf{r}_i, \qquad \mathbf{L}_i^T \mathbf{z}_i = \tilde{\mathbf{z}}_i.$$

   Let f be the flop count for the factorization and s be the flop count for the solve step (two substitutions). The total flop count for this step is $m(f + s)$.
2. To form the Schur complement in (16), we first compute the matrices $(\mathbf{A}_i \mathbf{D}_i \mathbf{A}_i^T + \mathbf{D}_{m+i})^{-1} \mathbf{A}_i \mathbf{D}_i \mathbf{B}_i$ by solving a linear system like (15) for each column of $\mathbf{A}_i \mathbf{D}_i \mathbf{B}_i$. Since the factors $\mathbf{L}_i$ have been pre-computed in step 1, we only need $n + 1$ solve steps, which lead to a flop count of $(n + 1)s$ for each i. Multiplication by the sparse matrix $\mathbf{B}_i \mathbf{D}_i \mathbf{A}_i^T$ (which has the same sparsity as $\mathbf{A}_i^T$) takes $5(n + 1)^2$ flops. Together with the m matrix additions, each with a cost $(n + 1)^2$, the total flop count for forming the matrix $\mathbf{S}$ is $m((n + 1)s + 6(n + 1)^2)$.
3. Forming the right-hand side vector in (17) takes $6m(n+1)$ flops. The Cholesky factorization of $\mathbf{S}$ takes $(1/3)(n+1)^3$ flops, and the two triangular solves take $2(n+1)^2$ flops. So the dominant flop count for this step is $6m(n + 1) + (1/3)(n + 1)^3$.
4. In the substitution step, we first compute the second term on the right-hand side of the equation (18). For each i, this takes s flops using a solve step with pre-computed factors $\mathbf{L}_i$. The vector subtraction takes $(n + 2)$ flops for each i. The total flop count for this step is $m(s + n + 2)$.

The overall flop count for the block elimination approach is dominated by

$$m\left(f+(n+3)s+6(n+1)^2\right)+(1/3)(n+1)^3. \tag{37}$$

Now let's take a closer look at the flop counts f and s for solving the linear systems in (15). The matrices $\mathbf{A}_i\mathbf{D}_i\mathbf{A}_i^T+\mathbf{D}_{m+i}$ are all dense with size $n+2$ by $n+2$. Without exploiting further structure, we have the factorization cost $f=(1/3)(n+2)^3$, and the solve step costs $s=2(n+2)^2$ (see, e.g., [19]). Thus the overall flop count in (37) is dominated by $(7/3)m(n+2)^3$, or $O(mn^3)$. The corresponding storage requirement would be $O(mn^2)$.

By further exploiting the arrow-plus-rank-one structure, instead of solving the dense linear systems in (15) directly, we solve the two sparse linear systems in (20) and then use the rank-one update formula (21). To solve the two sparse linear systems, we need one factorization step and two solve steps. Cholesky factor of $\mathbf{W}_i$ has the same sparsity as the lower triangular part of $\mathbf{W}_i$ (no fill-in), and it only costs $3(n+2)$ flops to compute. Each solve step costs $6(n+2)$ flops. The rank-one update in (21) also costs $6(n+2)$ flops. The corresponding values for f and s in (37) are $f=3(n+2)$ and $s=18(n+2)$. Therefore, the dominating terms in the complexity analysis is $24m(n+2)^2+(1/3)(n+1)^3$, or simply $O(mn^2)$ (under the assumption $m>n$). It is easy to check that, by only storing the sparse matrices $\mathbf{W}_i$ and vectors $\mathbf{u}_i$ instead of the dense matrices $\mathbf{A}_i\mathbf{D}_i\mathbf{A}_i^T+\mathbf{D}_{m+i}$, the storage cost is reduced to $O(mn)$.