Learning to Classify with Missing and Corrupted Features

Ofer Dekel OFERD@MICROSOFT.COM
Microsoft Research, 1 Microsoft Way, Redmond, WA 98052 USA

Ohad Shamir OHADSH@CS.HUJI.AC.IL
The Hebrew University, Jerusalem 91904, Israel

Abstract

After a classifier is trained using a machine learning algorithm and put to use in a real world system, it often faces noise which did not appear in the training data. Particularly, some subset of features may be missing or may become corrupted. We present two novel machine learning techniques that are robust to this type of classification-time noise. First, we solve an approximation to the learning problem using linear programming. We analyze the tightness of our approximation and prove statistical risk bounds for this approach. Second, we define the online-learning variant of our problem, address this variant using a modified Perceptron, and obtain a statistical learning algorithm using an online-to-batch technique. We conclude with a set of experiments that demonstrate the effectiveness of our algorithms.

1. Introduction

Supervised machine learning techniques often play a central role in solving complex real-world classification problems. First, we collect a training set of labeled examples and present this set to a machine learning algorithm. Then, the learning algorithm constructs a classifier, which can be put to use as a component in a working system. The process of collecting the training set and constructing the classifier is called the *training phase*, whereas everything that occurs after the hypothesis has been determined is called the *classification phase*. In many cases, the training phase can be performed under sterile and controlled conditions, and 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, and some of the features that were available during the training phase may be missing or corrupted. In this paper, we explore the possibility of anticipating and preparing for this type of classification-time noise.

The problem of corrupted and missing features occurs in a variety of different classification settings. 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, replacing the features that correspond to blood tests with random noise, while having no effect on other features. We would still like our diagnosis system to make accurate predictions. Alternatively, our goal may be to train a fingerprint recognition system that controls the lock on a door. After a few days of flawless operation, a user with greasy fingers comes along and leaves an oily smudge on the fingerprint scanner panel. From then on, all of the features measured from the area under the smudge are either distorted or cannot be extracted altogether. Ideally, the fingerprint recognition system should continue operating.

We take a worst-case approach to our problem, and assume that the set of affected features is chosen by an adversary individually per instance. More specifically, we assume that each feature is assigned an a-priori importance value and the adversary may remove or corrupt any feature subset whose total value is upper-bounded by a predefined parameter. In many natural settings, missing and damaged features are not actually chosen adversarially, but we find it beneficial to have our algorithm as robust as possible.

Appearing in *Proceedings of the 25^{th} International Conference on Machine Learning*, Helsinki, Finland, 2008.

We present two different learning algorithms for our problem, each with pros and cons. 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 (Vapnik, 1998). 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. Despite the fact that the distribution of training examples is different from the distribution of examples observed during the classification phase, we prove a statistical generalization bound for this approach.

Letting m denote the size of our training set and n the number of features, our polynomial LP formulation uses $O(mn)$ variables and $O(mn)$ sparse constraints. Depending on the dataset, this can still be rather large for off-the-shelf LP solvers. We see this as a shortcoming of our first approach, which brings us to our second algorithmic approach. We define an online learning problem, which is closely related to the original statistical learning problem. We devise a modified version of the Perceptron algorithm (Rosenblatt, 1958) for this online problem, and convert this Perceptron into a statistical learning algorithm using an online-to-batch conversion technique (Cesa-Bianchi et al., 2004). This approach benefits from the computational efficiency of the online 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 accuracy.

Choosing an adequate regularization scheme is one of the keys to solving this problem successfully. Many existing machine learning algorithms, such as the Support Vector Machine, 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. For example, in the degenerate case where one of the features actually equals the 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. 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 as much 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 the choice of the L_∞ norm emerges as a natural one in the theoretical analysis of our first, LP-based learning approach.

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 (Dietterich & Bakiri, 1995) and (Gamble et al., 2007) try to be robust to general additive noise that appears at classification time, but not necessarily to feature deletion or corruption. (Dalvi et al., 2004) 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 (Dalvi et al., 2004) but is otherwise significantly different. In the related field of online learning, where the training and classification phases are interlaced and cannot be distinguished, (Littlestone, 1991) proves that the Winnow algorithm can tolerate various types of noise, both adversarial and random.

Our work is most closely related to the work in (Globerson & Roweis, 2006), and its more recent enhancement in (Teo et al., 2008). Our motivation is the same as theirs, and the approaches share some similarities. Our experiments, presented in Sec. 4, suggest that our algorithms achieve considerably better performance, but we would also like to emphasize more fundamental differences between the two approaches: We allow features to have different a-priori importance levels, and we take this information into account in our algorithm and analysis. Our approach uses L_∞ regularization to promote a dense solution, where (Globerson & Roweis, 2006) uses L_2 regularization. Our second approach, which uses online-to-batch conversion techniques, is entirely novel. Finally, we prove statistical generalization bounds for our algorithms despite the change in distribution at classification time.

2. A Linear Programming Formulation

In this section, and throughout the paper, we use lowercase 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 $v_j \geq 0$. Infor-

mally, we think of v_j as the a-priori *informativeness* of feature j, or as the importance of feature j to the classification task. It can also represent the cost of obtaining the feature (such as the price of a medical test). 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

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

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 (Vapnik, 1998). We choose a margin parameter $\gamma > 0$ and a regularization parameter $C > 0$, and solve the problem

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

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 γ times 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 $\xi_i \geq \gamma V(J)/P$. The assumption $V([n] \setminus J) \leq N$ now implies that $V(J) \geq P$, and therefore $\xi_i \geq \gamma$. 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. We also restrict $\mathbf{w}$ to a hyper-box of radius C, which controls the complexity of the learned classifier and promotes 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 (Carr & Lancia, 2000), 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

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

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. *If $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\alpha}_1^\star, \ldots, \boldsymbol{\alpha}_m^\star)$ is an optimal solution to Eq. (4), then $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star)$ is a feasible point of Eq. (3), and therefore the value of Eq. (4) upper-bounds the value of Eq. (3). Moreover, if $v_j \in \{0,1\}$ for all $j \in [n]$, then $(\mathbf{w}^\star, b^\star, \boldsymbol{\xi}^\star)$ is an optimal solution to Eq. (3). Finally, 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/γ.*

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

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

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

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

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 $y_i(b + \sum_{j=1}^n \tau'_j w_j x_{i,j}) < (\gamma \sum_{j=1}^n \tau'_j v_j)/P - \xi_i$. 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:

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

To analyze this relaxation we require 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) 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. (c) There exists a minimizer of Eq. (7) with at most one coordinate in $(0,1)$.*

The proof of the lemma is straightforward but technical, and is omitted due to lack of space. Lemma 1 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 $j \in [n]$. We now proceed with proving the first part of Thm. 1 using claim (a) in Lemma 1. The remaining parts of the theorem follow similarly from claims (b) and (c) in the lemma.

Proof of 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 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

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

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

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

is at least $-\xi_i^\star$. The strong duality principle of linear programming (Boyd & Vandenberghe, 2004) states that the value of Eq. (8) equals the value of its dual optimization problem, which is:

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

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

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

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). □

2.3. Generalization Bounds

We now prove a generalization bound on the risk of the classifier learned in our framework, using PAC-Bayesian techniques (McAllester, 2003). 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 $\mathbf{w}$, let $\ell_\gamma(\mathbf{w}, \mathbf{x}, y)$ denote the γ-loss attained on the example $(\mathbf{x}, y)$, defined as

$$\left[\!\!\left[\min_{J\,:\,V([n]\setminus J)\leq N} y\sum_{j\in J} w_j x_j \;<\; \frac{\gamma V(J)}{P} \right]\!\!\right] , \qquad (10)$$

where $[\![\cdot]\!]$ again denotes the indicator function. Note that $\mathbb{E}[\ell_0(\mathbf{w}, \mathbf{x}, y)] = \mathcal{R}(\mathbf{w}, 0)$, where $\mathcal{R}$ is defined in Eq. (1).

Theorem 2. *Let S be a sample of size m drawn i.i.d from $\mathcal{D}$. 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 the risk associated with $\mathbf{w}$ is at most*

$$\sup\left\{\epsilon \;:\; \mathrm{KL}\left(\tfrac{1}{m}\textstyle\sum_{i=1}^m \ell_\gamma(\mathbf{w}, \mathbf{x}_i, y_i) \;\middle\|\; \epsilon\right) \leq \tfrac{\beta(m,\delta,\gamma)}{m-1}\right\},$$

where $\beta(m, \delta, \gamma) = \ln(m/\delta) + \sum_{j=1}^n \ln(4PC/(\gamma v_j))$ and KL *is the Kullback-Leibler divergence. The above is upper-bounded by the empirical γ-loss (which equals $\frac{1}{m}\sum_{i=1}^m \ell_\gamma(\mathbf{w}, \mathbf{x}_i, y_i)$), plus the additional term*

$$\sqrt{\frac{2}{m}\textstyle\sum_{i=1}^m \ell_\gamma(\mathbf{w}, \mathbf{x}_i, y_i)\frac{\beta(m,\delta,\gamma)}{m-1}} + \frac{2\beta(m,\delta,\gamma)}{m-1} \;\; .$$

Proof sketch. The proof follows along similar lines to the PAC-Bayesian bound for linear classifiers in (McAllester, 2003). First, define the axis-aligned box $B = \prod_{j=1}^n [w_j - \frac{\gamma v_j}{2P}, w_j + \frac{\gamma v_j}{2P}] \cap [-C, C]$. We use the margin concept to upper bound $\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell_0(\mathbf{w}, \mathbf{x}, y)]$ by the expected $\ell_{\gamma/2}$ loss over $\mathcal{D}$ of a classifier sampled uniformly from $B \cap [-C, C]^n$. We can upper bound this expected loss using the PAC-Bayesian theorem (McAllester, 2003), where the uniform distribution over $B \cap [-C, C]^n$ is the posterior classifier distribution, and the uniform distribution over $[-C, C]^n$ is the prior. The bound we get is defined in terms of the average empirical $\ell_{\gamma/2}$ loss of a random classifier from B, plus a complexity term dependent on the volume ratio between B and $[-C, C]^n$. Finally, this average loss can be upper bounded by the empirical ℓ_γ loss of $\mathbf{w}$ by repeating the technique of the first stage. The weaker bound stated in the theorem follows from a lower bound on the KL divergence, presented in (McAllester, 2003). □

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 $\ell_0, \ell_{\gamma/2}, \ell_\gamma$ loss functions as described above. This lends theoretical support to our choice of the L_∞ norm in our algorithms.

2.4. 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 the margin parameter γ in Eq. (4) is sufficiently large. 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 the 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 in the feature deletion setting, where the latter can be bounded with Thm. 2.

Theorem 3. *Let ϵ, C, and N be arbitrary positives, and let γ be at least $C\sqrt{N\ln(1/\epsilon)/2}$. Assume that we solve Eq. (4) with parameters γ, 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. (10), 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 which 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, $y\langle\mathbf{w}, f(\mathbf{x})\rangle$ is equal to:

$$y\sum_{j\in J} w_j x_j + y\sum_{j\notin J} w_j f_j(\mathbf{x}) \;>\; y\sum_{j\in J} w_j x_j - \gamma \;\; . \qquad (11)$$

Thus, with probability at least $1-\epsilon$, $\Pr(y\langle\mathbf{w}, f(\mathbf{x})\rangle < 0)$ is upper bounded by $\mathbb{E}[\ell_\gamma(\mathbf{w}, \mathbf{x}, y)]$. Otherwise, with probability at most ϵ, $\Pr(y\langle\mathbf{w}, f(\mathbf{x})\rangle < 0) \leq 1$. □

We conclude with an interesting 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 r arbitrarily close to 1 and a large enough n, the first sum in Eq. (11) dominates the second. Namely, by setting $\gamma = \Omega(\sqrt{N})$ in Eq. (4), our ability to withstand feature corruption matches our ability to withstand feature deletion.

3. Solving the Problem with the Perceptron

We now turn to our second learning algorithm, taking a 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 (Rosenblatt, 1958) 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 (Rosenblatt, 1958), 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 $\xi(\mathbf{w}, b, \mathbf{x}, y)$, defined as

$$\left[\max_{J\,:\,V([n]\setminus J)\leq N} \frac{\gamma V(J)}{P} \ - \ y\big(b + \textstyle\sum_{j\in J} w_j x_j\big)\right]_+ \ , \quad (12)$$

where $P = V([n]) - N$ and $[\alpha]_+$ denotes the hinge function, $\max\{\alpha, 0\}$. Note that $\xi(\mathbf{w}_i, b_i, \mathbf{x}_i, y_i)$ upper-bounds γ times 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

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

and $b_{i+1} = [b_i + y_i \tau]_{\pm C}$, where $\tau = \frac{\sqrt{n+1}C}{\sqrt{2m}}$ and $[\alpha]_{\pm C}$ abbreviates the function $\max\big\{\min\{\alpha, C\}, -C\big\}$. 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 $\big((\mathbf{x}_i, y_i)\big)_{i=1}^m$ 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. (12). Then it holds that $\frac{1}{\gamma m}\sum_{i=1}^m \xi(\mathbf{w}_i, b_i, \mathbf{x}_i, y_i)$ is upper-bounded by*

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

The next step is to convert our online algorithm into a statistical learning algorithm.

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 (Cesa-Bianchi et al., 2004) 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 (Cesa-Bianchi et al., 2004) to prove that the average classifier provides an adequate solution to our original problem.

Note that the loss function we use, defined in Eq. (12), is bounded and convex in its first two arguments. This allows us to apply (Cesa-Bianchi et al., 2004, Corollary 2) to

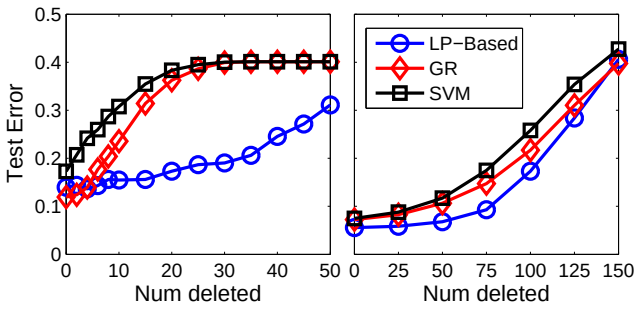

Figure 1. A comparison of our LP-based approach with the algorithm of (Globerson & Roweis, 2006) (GR) and with SVM on SPAM (left) and MNIST (right), with random noise.

relate the risk of $(\bar{\mathbf{w}}, \bar{b})$ with the cumulative online loss suffered by the Perceptron. It also allows us to apply Hoeffding's bound to relate the expected loss of any fixed classifier $(\mathbf{w}^\star, b^\star)$ with its empirical loss on the training set. Combining both bounds results in the following corollary.

Corollary 1. *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 $\phi = \gamma \max_{J:V([n]\setminus J)\leq N} \left(V(J)/P\right)$, 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 γ times 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 $\gamma\mathcal{R}(\bar{\mathbf{w}}, \bar{b})$.

4. Experiments and Conclusions

We compare the performance of our two algorithms (LP-based and online-to-batch) with that of a linear L_2 SVM (Joachims, 1998) and with the results reported in (Globerson & Roweis, 2006). We used the GLPK package (`http://www.gnu.org/software/glpk`) to solve the LP formulation of our LP-based algorithm.

We begin with a highly illustrative sanity check. 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 technique 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.

Following the lead of (Globerson & Roweis, 2006), we conducted experiments using the SPAM and MNIST datasets. The SPAM dataset, taken from the UCI repository, is a collection of spam and non-spam e-mails. Spam can be detected by different word combinations, so we expect considerable feature redundancy in this dataset. The MNIST dataset is a collection of pixel-maps of handwritten digits. Again, following (Globerson & Roweis, 2006), we focused on the binary problem of distinguishing the digit 4 from the digit 7. Adjacent pixels often contain redundant information, making MNIST well-suited for our needs.

On each dataset, we performed 2 types of experiments. The first type follows exactly the protocol used in (Globerson & Roweis, 2006). Namely, the algorithm is trained with a small training set of 50 instances, and its performance is tested in the face of *random* feature-deleting noise, which uniformly deletes N non-zero features from each test instance, for various choices of N. Notice that this setting deviates from the adversarial setting considered so far, and the reason for conducting this experiment is to compare our results to those reported in (Globerson & Roweis, 2006). A validation set is used for parameter tuning. We did not test our online-to-batch algorithm within this setting, since it has little advantage with such a small training set. The results are presented in Fig. 1, and show test error as a function of the number of deleted features. Compared to its competitors, our algorithm has a clear and substantial advantage.

The second type of experiment simulates more closely the adversarial setting discussed throughout the paper. Using 10-fold cross-validation, we corrupted each test instance using a greedy adversary, which deletes the most valuable features of each instance until either the limit N is reached or all useful features are deleted. $1/9$ of the training set was used for parameter tuning. Due to computational considerations when running our LP-based algorithm, we performed a variant of bagging by randomly splitting the training set into chunks, training on each chunk individually, and finally averaging the resulting weight vectors. In contrast, our online-to-batch algorithm trained on the entire training set at once, and so did the SVM algorithm. We

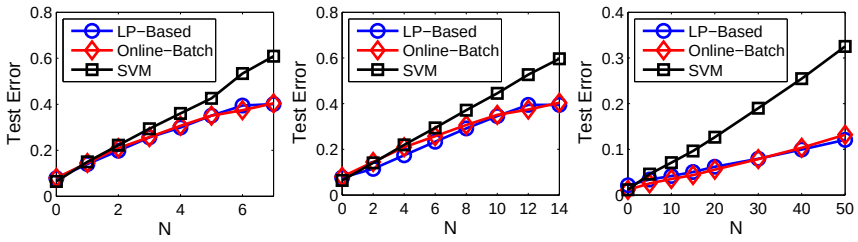

Figure 2. Experiments on SPAM with $\forall j \in J, v_j = 1$ (left) and with v_j set with a mutual information heuristic (center). Experiments on MNIST with v_j set with a mutual information heuristic (right).

repeated this process for different values of N. For the SPAM dataset, we repeated this entire experiment twice, once with features values v_j set uniformly to 1, and once with v_j set using a mutual information heuristic. Formally, we set

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

where Z is such that $\sum v_j = n$, and where $I([\![x_j > c]\!]; y)$ is the mutual information between the predicate $[\![x_j > c]\!]$ and the label y, over all examples in the training set. Intuitively, we are calculating the amount of information contained in each individual feature on the label, provided that we are looking only at linear threshold functions. When experimenting with the MNIST dataset, we only used the values of v_j set by our heuristic. This is a natural choice since the features of MNIST are of markedly different importance levels. For example, the corner pixels, which are always zero, are completely uninformative, while other pixels may be very informative. The results are presented in Fig. 2, and show test error as a function of N. Clearly, our algorithms have the advantage. SVM repeatedly puts all of its eggs in a small number of baskets, and is severely punished for this, while our technique anticipates the actions of the adversary and hedges its bets accordingly.

Moreover, the results in Fig. 2 demonstrate the tradeoffs between our LP-based and online-to-batch algorithms. Although we have handicapped the LP-based algorithm by chunking the training set, its performance is comparable and sometimes superior to that of the online-to-batch algorithm. With less or without chunking, we expect its performance to be even better.

We conclude that our proposed algorithms successfully withstand feature corruption at classification time, and considerably improve upon the current state of the art. On a more general note, this work has interesting connections to 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 here. The trade-off between robustness and sparsity provides fertile ground for future research.

References

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Carr, R. D., & Lancia, G. (2000). Compact vs. exponential-size LP relaxations SANDIA Report 2000-2170.

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2004). On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, *50*, 2050–2057.

Dietterich, T. G., & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, *2*, 263–286.

Dalvi, N., Domingos, P., Mausam, Sanghai, S., and Verma, S. (2004) Adversarial classification. *Proceedings of KDD 2004* (pp. 99–108).

Gamble, E., Macskassy, S., & Minton, S. (2007). Classification with pedigree and its applicability to record linkage. *Workshop on Text-Mining & Link-Analysis*.

Globerson, A., & Roweis, S. (2006). Nightmare at test time: robust learning by feature deletion. *Proceedings of ICML 23* (pp. 353–360).

Joachims, T. (1998). Making large-scale support vector machine learning practical. In *Advances in kernel methods - support vector learning*. MIT Press.

Littlestone, N. (1991). Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. *Proceedings of the COLT 4* (pp. 147–156).

McAllester, D. A. (2003). Simplified PAC-bayesian margin bounds. *Proceedings of COLT 16* (pp. 203–215).

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386–407.

Teo, C.-H., Globerson, A., Roweis, S., & Smola, A. (2008). Convex learning with invariances. *Advances in NIPS 21*.

Vapnik, V. N. (1998). *Statistical learning theory*. Wiley.