JMLR: Workshop and Conference Proceedings vol 40:1–21, 2015

Learning and inference in the presence of corrupted inputs

Uriel Feige URIEL.FEIGE@WEIZMANN.AC.IL
Weizmann Institute of Science

Yishay Mansour MANSOUR.YISHAY@GMAIL.COM
Microsoft Research, Hertzelia and Tel Aviv University

Robert E. Schapire SCHAPIRE@MICROSOFT.COM
Microsoft Research, New York and Princeton University

Abstract

We consider a model where given an uncorrupted input an adversary can corrupt it to one out of m corrupted inputs. We model the classification and inference problems as a zero-sum game between a learner, minimizing the expected error, and an adversary, maximizing the expected error. The value of this game is the optimal error rate achievable.

For learning using a limited hypothesis class $\mathcal{H}$ over corrupted inputs, we give an efficient algorithm that given an uncorrupted sample returns a hypothesis $h \in \mathcal{H}$ whose error on adversarially corrupted inputs is near optimal. Our algorithm uses as a blackbox an oracle that solves the ERM problem for the hypothesis class $\mathcal{H}$. We provide a generalization bound for our setting, showing that for a sufficiently large sample, the performance on the sample and on future unseen corrupted inputs will be similar. This gives an efficient learning algorithm for our adversarial setting, based on an ERM oracle.

We also consider an inference related setting of the problem, where given a corrupted input, the learner queries the target function on various uncorrupted inputs and generates a prediction regarding the given corrupted input. There is no limitation on the prediction function the learner may generate, so implicitly the hypothesis class includes all possible hypotheses. In this setting we characterize the optimal learner policy as a minimum vertex cover in a given bipartite graph, and the optimal adversary policy as a maximum matching in the same bipartite graph. We design efficient local algorithms for approximating minimum vertex cover in bipartite graphs, which implies an efficient near optimal algorithm for the learner.

1. Introduction

Consider the following classification setting. Inputs, described by attributes, are to be classified as having one of two labels. For example, an input might represent an employee, the attributes might represent her performance under various measures, and the classification function maps the values of the attributes to a binary decision, such as "deserves a bonus" or not. Suppose now that an adversary (a manager in the company? a trade union?) can manipulate the attributes of inputs in some limited way. For example, the adversary might round up or down the number of work hours that the employee reports, or might over- or under-emphasize the employee's contribution to a completed project. We refer to such manipulations as corruption of the input. As a consequence of this corruption, some employees that deserve a bonus (according to the intended classification) might not get one, and vice versa. Our goal is to design a classification algorithm that is robust to

such corruption of inputs. Thus, even if the algorithm observes only the corrupted input, in most cases, the label that it predicts should match the label of the original (unknown) uncorrupted input.[1]

We focus on the case in which the corruption is of a worst-case, adversarial nature. This case is interesting for two reasons. One is that positive results in the adversarial model extend to all settings that are less adversarial. The other is that adversarial corruption can model situations in which corruption is due to malicious behavior (e.g., a manager wants to arrange bonuses for his friends and withhold bonuses from others), and not just a result of random mistakes.

We model situations such as the above as a game between a learner and an adversary. For notational convenience, we separate between the uncorrupted input space and the corrupted input space. The adversary maps an uncorrupted input to a corrupted one. We assume that for each uncorrupted input there are only a limited number of corrupted inputs to which it might be mapped, and the learner knows what this set is. Given a corrupted input, the learner needs to output a label that we call its prediction.

We consider two settings. In the *learning* setting, there is an unknown distribution over the uncorrupted inputs, the learner receives a sample of uncorrupted examples (inputs and labels) and selects a hypothesis (or a mixture of hypotheses) from some limited hypothesis class, mapping a corrupted input to a prediction (or to a distribution over predictions). The error is defined on future inputs which are corrupted by the adversary. For this setting we assume we are given an oracle which on a set of examples finds a minimizing hypothesis in the hypothesis class, i.e., an ERM (Empirical Risk Minimization) oracle. We design an algorithm based on the idea of Freund and Schapire (1999), and using a variation of a regret algorithm of Cesa-Bianchi et al. (2007) to find near optimal learner and adversary policies, for the specific sample. We complement this by deriving a generalization bound, relating the error on the sample (when it is corrupted) with the error of a random input corrupted by the adversary. The end result is an efficient reduction from learning uncorrupted inputs to learning corrupted inputs near optimally (using ERM).

The second setting is the *inference* setting. Here, the distribution over the uncorrupted inputs is known. This is very much in line with other inference work, where the generating distribution is known and one needs to infer the value of a given variable, in our case the label of the input. The learner receives a corrupted input and his goal is to predict the label of the original uncorrupted input. For this task the learner can query the target function on various uncorrupted inputs. We model the corruption as a directed bipartite graph, where one side is the uncorrupted inputs while the other side is the corrupted inputs and edges represent the adversary's ability to map a certain uncorrupted input to a corrupted one. From this graph we define another bipartite graph, whose one side are the inputs labeled positive by the target function and the other side are inputs labeled negative, with edges representing the fact that they both can be mapped to the same corrupted input. We characterize the optimal learner policy as a minimum vertex cover in this bipartite graph, and the optimal adversary policy as a maximum matching in this bipartite graph. This already gives a polynomial time algorithm, which runs in time polynomial in the input domain size. Our goal is to have algorithms that run in time polynomial in the logarithm of the input domain size, i.e., polynomial in the dimension. For this we develop an efficient local algorithm for approximating the minimum vertex cover in bipartite graphs. The end result is that for the inference setting we have an algorithm that predicts, given corrupted inputs, and its error rate is near optimal.

1. Other motivating examples from spam detection and hardware failures are given in Mansour et al. (2015).

An interesting aspect of this work is the connection between classical machine learning notions, such as classification and inference, and the design of sub-linear algorithms, more specifically local algorithms. Such a connection already appeared in Mansour et al. (2015), where a local algorithm was designed for solving a linear program that came up in Mansour et al. (2015). The extra interest in the current work is that our learning algorithm give rise to classical graph theoretical notions such as matching and vertex cover, and the connections that we establish with learning can serve as additional motivation to the existing work on local algorithms for classical graph theoretical notions.

Related work

The most related work is that of Mansour et al. (2015). They model robustness as a zero-sum game between a predictor and an adversary. The adversary selects one of m modification rules to corrupt the input. Their main result is an efficient algorithm to derive a near optimal policy for the predictor. The main difference is that in their model, the modification rule is selected *before* the uncorrupted input is realized, while in our model the adversary selects the corruption (modification rule) *after* the input is realized. This implies that if we like to translate our model to the modification rules model, we would have $m' = m^{|\mathcal{X}|}$ different modification rules, since each uncorrupted input in $\mathcal{X}$ can be mapped arbitrarily to one of m corrupted inputs. Allowing a stronger adversary makes it possible to handle a more adversarial learning setting, but may on the other hand increase the resulting error rate.

There is a vast literature in statistics and machine learning regarding various noise models. At large, most noise models assume a random process that generates the noise. In computational learning theory the popular noise models include: random classification noise (Angluin and Laird, 1988), and malicious noise (Valiant, 1985; Kearns and Li, 1993). In the malicious noise model, the adversary gets to arbitrarily corrupt some small fraction of the examples; in contrast, in our model the adversary can always corrupt every example, but only in a limited way.

Another vast literature which is remotely related is that of robust optimization and stability, where the main task is to understand how minor changes in the input can affect the outcome of a (specific) algorithm (see, Ben-Tal et al. (2009)). We differ in two important ways from that literature. First, we do not assume any metric for our corruptions, while in large part the robustness there is measured as a function of the norm of the modification. Second, that literature, to a large extent, discusses the stability and robustness of specific algorithms, while our starting point is trying to derive algorithms immune to such corruption.

Our work establishes a solid connection between local computation algorithms (LCAs) and efficient learning. Local Computation Algorithms (LCA) where introduced in Rubinfeld et al. (2011), and LCAs for combinatorial problems such as maximal independent set and hypergraph 2-coloring were derived in (Rubinfeld et al., 2011; Alon et al., 2012; Mansour et al., 2012). We build on the work of Mansour and Vardi (2013) that gave an LCA which finds a $(1-\epsilon)$-approximation to the maximum matching to derive our near optimal inference. (An alternative deterministic algorithm is given by Even et al. (2014).)

2. Model

There is a finite domain $\mathcal{X}$ from which uncorrupted inputs are generated, and a finite domain $\mathcal{Z}$ (possibly the same as $\mathcal{X}$) which is the domain of corrupted inputs. An arbitrary unknown target

function $f : \mathcal{X} \to \{0,1\}$ maps each uncorrupted input $x \in \mathcal{X}$ to its classification $f(x) = y \in \{0,1\}$.

The adversary maps an uncorrupted input $x \in \mathcal{X}$ to a corrupted input $z \in \mathcal{Z}$ (thus implicitly corrupting the input). There is a mapping ρ which for every $x \in \mathcal{X}$ defines a set $\rho(x) \subseteq \mathcal{Z}$, such that $|\rho(x)| \leq m$, and $|\rho^{-1}(z)| \leq m$. The adversary can map an uncorrupted input x to any corrupted input $z \in \rho(x)$. We assume that $\rho(\cdot)$ and $\rho^{-1}(\cdot)$ are known and the learner has access to both.[2]

There is a distribution D over the uncorrupted inputs $\mathcal{X}$. (We will discuss both the case where D is known and unknown to the learner.) The basic scenario is the following. First, an uncorrupted input x is selected using D. Then, the adversary, given x, selects some $z \in \rho(x)$. The learner observes a corrupted input z and predicts $h(z) \in [0,1]$ for the value of $f(x)$. Finally, the learner incurs a loss $\ell(h(z), f(x))$. We use the absolute loss to measure the quality of the prediction, i.e., $\ell(h(z), y) = |h(z) - y|$. For $y \in \{0,1\}$ this is equivalent to the prediction error.

The goal of the learner is to minimize the expected loss, while the adversary would like to maximize it. This defines a zero-sum game which has a value $error^*$. We say that the learner's hypothesis is ϵ-optimal if it guarantees a loss which is at most $error^* + \epsilon$, and the adversary policy is ϵ-optimal if it guarantees a loss which is at least $error^* - \epsilon$. We refer to a 0-*optimal* policy as an *optimal* policy.

We study two settings. The first setting is the *learning* setting, which is very similar to standard supervised learning. There is a fixed hypothesis class over corrupted inputs. Given an uncorrupted sample which is sampled from an unknown distribution over uncorrupted inputs, the learner needs to select a hypothesis (or a mixture of hypotheses) from the hypothesis class. The main difference from classical learning models is that the learner would be tested on adversarially corrupted inputs, and needs to select the hypothesis with this in mind. The second setting is the *inference* setting, in which the learner has no restriction on the hypothesis it uses for prediction. Here, there is no explicit uncorrupted sample, but the learner can query the target function on uncorrupted inputs. In addition the learner has access to the distribution over the inputs.

In the learning setting, there is a hypothesis class $\mathcal{H}$ of hypotheses $h : \mathcal{Z} \to \{0,1\}$, and the distribution D over the uncorrupted inputs is unknown. The learner observes a sample $S = \{(x_i, y_i)\}$, where x_i is drawn i.i.d. from D and $y_i = f(x_i)$. Note that the sample S is uncorrupted, and the challenge is to select a classifier that can handle future corrupted inputs. The learner selects a mixture of hypotheses $h \in \mathcal{H}$ so as to minimize its loss, assuming that the adversary corrupts the inputs. Formally, the loss of the learner when selecting such a mixture $h \in \Delta(\mathcal{H})$ is

$$\mathcal{L}(h) = E_{x \sim D}[\max_{z \in \rho(x)} \ell(h(z), f(x))]. \tag{1}$$

The $\max$ operator is the adversary selection of a corrupted input z based on x. The goal of the learner is to match the best $h \in \Delta(\mathcal{H})$, i.e., guarantee an error close to $error^* = \min_{h \in \Delta(\mathcal{H})} \mathcal{L}(h)$ which is the best error rate it can guarantee on future corrupted input. (Note that $error^*$ is the value of the zero-sum game in this case.)

For the *inference* setting we assume a known distribution D over the uncorrupted inputs $\mathcal{X}$ (namely, for any x we can compute $D(x)$). This model is an inference model, where we can model the distribution over uncorrupted inputs and the label of the target function using some generative model, say, with a Bayesian network. In this setting there is no hypothesis class, and any hypothesis $h : \mathcal{Z} \to [0,1]$ is legitimate. The learner can access the target function f through a query oracle

2. This also implicitly implies that $|\mathcal{X}|/m \leq |\mathcal{Z}| \leq m|\mathcal{X}|$.

(this can be thought of as solving the inference problem for the uncorrupted inputs, i.e., given an uncorrupted input deduce its label). In the inference setting there is no explicit sample (note that the leaner can generate uncorrupted inputs using the distribution D).

The learner, given a corrupted input $z \in \mathcal{Z}$, outputs $h(z) \in [0,1]$. In order to define $h(z)$ the learner can query the target function f on various uncorrupted inputs x. Recall that the learner has access to both $\rho(\cdot)$ and $\rho^{-1}(\cdot)$. The optimal loss of the learner is,

$$\mathcal{L}^* = \min_{h:\mathcal{Z}\to[0,1]} E_{x\sim D}[\max_{z\in\rho(x)} \ell(h(z), f(x))].$$

Intuitively, the $\min$ operator is the learner selection of a hypothesis h and the $\max$ operator is the adversary selection of a corrupted input z based on the realized uncorrupted input x. The goal of the learner is to (implicitly) define a hypothesis $h : \mathcal{Z} \to [0,1]$ whose expected loss is close to $\mathcal{L}^*$.

3. Learning from corrupted inputs

In this section we discuss the learning setting. The main idea is to show that for a bounded hypothesis class, we can also learn with corrupted inputs using an ERM oracle for the hypothesis class. Formally, let $\mathcal{H}$ be a hypothesis class. An ERM oracle for $\mathcal{H}$ receives a sample $S = \{(z_i, y_i)\}$, where $z_i \in \mathcal{Z}$ and $y_i \in \mathcal{Y}$, and outputs $h \in \mathcal{H}$ which minimizes the classification loss on S. Our goal is to use an ERM oracle for $\mathcal{H}$ to find a near optimal hypothesis $h \in \Delta(\mathcal{H})$ for the case of adversarially corrupted inputs.

We first give a characterization of the solution for the case of bounded hypothesis class (section 3.1). We use a suitably modified regret minimization algorithm to derive near optimal policies for the given sample (section 3.2). Finally we give a generalization bound that guarantees that the output hypothesis will also work well on new unseen instances drawn from the unknown distribution D (section 3.3). The end result is an efficient algorithm that uses an ERM oracle to learn a near optimal mixture hypotheses from $\mathcal{H}$ for corrupted inputs.

3.1. Characterization

There is a fixed hypotheses class $\mathcal{H}$ which the learner uses. Given an uncorrupted input $x \in \mathcal{X}$, we define a $|\rho(x)| \times |\mathcal{H}|$ matrix M_x with entries $M_x(z,h) = I(h(z) \neq y)$, where $y = f(x)$, $z \in \rho(x)$, $h \in \mathcal{H}$, and I denotes an indicator function.

The learner strategy is a distribution Q over the hypotheses in $\mathcal{H}$ (the columns of M_x). (Note that the learner will not observe the uncorrupted input x; rather the learner is selecting a hypothesis that will map a corrupted input $z \in \mathcal{Z}$, where $z \in \rho(x)$, to a prediction for the value of $y = f(x)$.) For each $x \in \mathcal{X}$, the adversary's strategy $P_x \in \Delta(\rho(x))$ is a mixed strategy (probability distribution) over corrupted inputs $z \in \rho(x)$. The goal of the learner is to minimize $\mathcal{L}(h)$ as in Eq. (1), which can be re-stated as that of selecting Q^* such that,

$$Q^* = \arg\min_Q E_x[\max_{P_x}(P_x^T M_x Q)].$$

We can also treat P as a giant vector over all $x \in \mathcal{X}$, allowing us to take its maximization out of the expectation, yielding

$$error^* = \min_Q \max_P E_x[P_x^T M_x Q].$$

Using the minimax principle, the optimal value is preserved when interchanging the $\max$ and $\min$:

$$error^* = \max_P \min_Q E_x[P_x^T M_x Q].$$

For any given P, a minimizing learner strategy Q is simply a hypothesis that minimizes the error when the distribution over pairs $(z, y) \in \mathcal{Z} \times \mathcal{Y}$ is D^P, where

$$D^P(z, y) = \sum_{x: f(x)=y \wedge z \in \rho(x)} P_x(z)\ D(x).$$

Here, z is a corrupted input and D is the unknown distribution over $\mathcal{X}$. Given P, the learner strategy Q then simply selects the hypothesis h^P such that,

$$h^P = \arg\min_h E_{(z,y)\sim D^P}[\ell(h(z), y)]. \tag{2}$$

Note that when D is instead the empirical distribution over a training sample, the distribution D^P has small support (at most a factor m larger than the training set), and the hypotheses h^P can be found using our ERM oracle. We will make use of this shortly.

3.2. Learning over an uncorrupted sample using regret minimization

In this section we show how for a given sample S one can compute an optimal strategy for both the learner and the adversary. The main tool that we use is an oracle for solving the ERM for the hypothesis class $\mathcal{H}$. Essentially we are showing that given an oracle to learn uncorrupted inputs, we can use it to learn near optimally corrupted inputs.

Our methodology will be similar to that of (Freund and Schapire, 1999). We will do an explicit analysis of a specific regret minimization algorithm for our case (since the vector P has a unique structure, and not simply a distribution over its entries). The regret minimization algorithm we use is based on the regret minimization algorithm of (Cesa-Bianchi et al., 2007).

We maintain weights $w_t(z, (x, y))$ for each $(x, y) \in S$ and $z \in \rho(x)$, which are set initially to 1. At time t, given a hypothesis h_t, we update the weights $w_t(z, (x, y))$ as follows: if $h_t(z) = y$ then the weight does not change, i.e., $w_{t+1}(z, (x, y)) = w_t(z, (x, y))$. Otherwise, we set $w_{t+1}(z, (x, y)) = (1 + \eta) w_t(z, (x, y))$. This defines a vector P^t where,

$$P^t(z, (x, y)) = \frac{w_t(z, (x, y))}{\sum_{z' \in \rho(x)} w_t(z', (x, y))}.$$

Note that we are normalizing for each (x, y) separately and thus implicitly have a distribution P_x^t for each $(x, y) \in S$.

Given P^t we select the hypothesis $h_t = h^{P^t}$ using Eq. (2), which minimizes the error with respect to P^t. As noted above, this can be done using the ERM oracle for $\mathcal{H}$. Given h_t we define the losses for each $(z, (x, y))$ as $\ell_t(z, (x, y)) = I(h_t(z) \neq y) = M_x(z, h)$. We use the losses ℓ_t to update the weights as defined above.

After T steps, we have $L_T(z, (x, y)) = \sum_{t=1}^T \ell_t(z, (x, y))$. The loss of our algorithm is

$$L_T^{ON} = \sum_{t=1}^T P^t \cdot \ell_t = \sum_{t=1}^T \sum_{(x,y)\in S} \sum_{z \in \rho(x)} P^t(z, (x, y)) \ell_t(z, (x, y)).$$

Our benchmark is

$$\mathcal{L}^* = \sum_{(x,y)\in S} \max_{z\in\rho(x)} L_T(z,(x,y)).$$

Note that the benchmark is equivalent to fixing the sequence of selected hypotheses h_t and allowing the adversary to select its optimal policy to maximize the classification error. The following theorem (whose proof is in Appendix A) states that for sufficiently large T we have near optimal policies for the learner and the adversary for domain S, i.e., assuming that the uncorrupted inputs are $\{x : (x,y) \in S\}$.

Theorem 1 *Fix a sample S, and let $T \geq (4|S|\ln m)/\epsilon^2$. For an uncorrupted input domain S we have that the strategy $P = (1/T)\sum_{t=1}^{T} P^t$ for the adversary is an ϵ-optimal strategy and the strategy $Q = (1/T)\sum_{t=1}^{T} h^t$ for the learner is an ϵ-optimal strategy.*

Note that the above theorem is mainly about the resulting strategies of the learner and the adversary. It shows that if we run the regret minimization algorithm long enough we will have near optimal strategies, if we limit out attention to inputs from the given sample S.

3.3. Generalization Bounds

In the previous section we presented a near optimal learner policy for a given sample S. We would like to show that if the sample S is large enough, then the same policy will generalize well. This will give us an efficient algorithm to learn a near optimal learner policy using an ERM oracle.

Let $\mathcal{H}^N$ be the set of hypotheses which are a simple average of at most N hypotheses from $\mathcal{H}$. That is, there exists a multiset $T \subseteq \mathcal{H}$ with $|T| \leq N$ such that $h(x) = \sum_{h_i\in T} h_i(x)/|T|$. The first step is to show that we can limit N and get the same approximation.

Lemma 2 *Fix a distribution D. For any hypothesis $h(x) = \sum_{i=1}^{N} q_i h_i(x)$, where $q \in \Delta([1,N])$, there exists a hypothesis $h'(x) = \sum_{j=1}^{N_0} h'_j(x)/N_0$, where $N_0 = O(\epsilon^{-2}\log\delta^{-1})$ such that $\Pr_{x\sim D}[|h(x) - h'(x)| \geq \epsilon] \leq \delta$*

Proof Fix $x \in \mathcal{X}$. Sample h'_j from the distribution q i.i.d. N_0 times. Note that $E_{h'}[h'_j(x)] = h(x)$, where $E_{h'}$ denotes expectation with respect to the hypotheses comprising h. Using a Chernoff bound, for $N_0 \geq (\log 1/\delta)/\epsilon^2$, with probability at most δ (over the choice of h') we have $|h(x) - h'(x)| \geq \epsilon$. Hence,

$$E_{h'}[\Pr_{x\sim D}[|h(x) - h'(x)| \geq \epsilon]] = E_{x\sim D}[\Pr_{h'}[|h(x) - h'(x)| \geq \epsilon]] \leq \delta.$$

This implies that there exists an h' such that $\Pr_{x\sim D}[|h(x) - h'(x)| \geq \epsilon] \leq \delta$. ■

The following theorem (whose proof is in Appendix B) gives the generalization bound.

Theorem 3 *For a sample size of at least $|S| \geq (1/\epsilon^4)\log(|\mathcal{H}|/\delta)$ we have that for every $h \in \mathcal{H}^N$*

$$\left| \left(E_{(x,y)\sim D}[\max_{z\in\rho(x)} \frac{1}{N}\sum_{i=1}^{N} I[h_i(z) \neq y]] \right) - \frac{1}{|S|}\sum_{(x,y)\in S} \left(\max_{z\in\rho(x)} \frac{1}{N}\sum_{i=1}^{N} I[h_i(z) \neq y] \right) \right| \leq \epsilon$$

with probability at least $1-\delta$.

Combining Theorem 1 and Theorem 3 we derive the main result for the bounded hypothesis class, showing that with an ERM oracle we can learn a near optimal learner hypothesis.

Theorem 4 *Given a hypothesis class $\mathcal{H}$ and an ERM oracle for $\mathcal{H}$, there is an algorithm that, with probability $1-\delta$, computes an ϵ-optimal learner hypothesis. The running time of the algorithm is polynomial in $1/\epsilon$, $\log 1/\delta$, m and $\log|\mathcal{H}|$.*

4. Characterizing the optimal policies in the inference setting

In this section we study the optimal robust classification in the inference setting where the learner can use an arbitrary hypothesis mapping observables to predictions, characterizing the optimal policies of the learner and the adversary. We start with preliminaries that define an abstraction of the problem to graphs (section 4.1), continue by defining the optimal policy of the adversary (section 4.2), then end with the related optimal policy for the learner (section 4.3).

4.1. Preliminaries

Given a Boolean function $f : \mathcal{X} \to \{0,1\}$, we define the *corruption graph*, which reflects the adversary's ability to corrupt inputs. The corruption graph is a directed bipartite graph where the left nodes are the uncorrupted inputs $\mathcal{X}$, the right nodes are the corrupted inputs $\mathcal{Z}$, and an edge from $x \in \mathcal{X}$ to $z \in \mathcal{Z}$ reflects the adversary's ability to transform x to z. Formally, the corruption graph is a directed bipartite graph $G(\mathcal{X}, \mathcal{Z}, E)$ where $(x,z) \in E$ iff $z \in \rho(x)$. In addition, we have weights on the nodes of $\mathcal{X}$ which reflect the distribution D, namely the weight of $x \in \mathcal{X}$ is $D(x)$. (Sometime, for intuition, we consider an unweighted graph, which is equivalent to assuming a uniform distribution D over $\mathcal{X}$.)

We also define an *interference graph* which will be helpful in characterizing the optimal policies. The interference graph is an undirected bipartite graph $G(F_0, F_1, E)$, where the left side is $F_0 = \{x : f(x) = 0\}$ and the right side is $F_1 = \{x : f(x) = 1\}$. There is an edge $(x_0, x_1) \in E$ between $x_0 \in F_0$ and $x_1 \in F_1$ if $\rho(x_0) \cap \rho(x_1) \neq \emptyset$, i.e., there is a $z \in \mathcal{Z}$ such that both x_0 and x_1 can be mapped to z. We denote by $z(x_0, x_1)$ an arbitrary $z \in \rho(x_0) \cap \rho(x_1)$, if one exists. Again, we have weights that reflect the distribution D where the weight of $x \in \mathcal{X}$ is $D(x)$. (Note that the weights are both on the left and right side unlike the corruption graph where the weights are only on the left side.)

4.2. Optimal policy for the adversary

To derive the adversary policy we can consider the interference graph. For simplicity assume the unweighted case (uniform distribution). A possible adversary policy is to select a matching between F_0 and F_1. Given a matching of size k the adversary can force k mistakes using the following policy π. For each edge (x_0, x_1) in the matching, the adversary maps both x_0 and x_1 to $z(x_0, x_1)$. Namely, $\pi(z(x_0, x_1)|x_0) = \pi(z(x_0, x_1)|x_1) = 1$. (Such a $z(x_0, x_1)$ exists by the construction of the edges of the interference graph.) Clearly for any such pair (x_0, x_1) the adversary will force any learner to make exactly one error, regardless of the learner hypothesis. Since we have a matching, the pairs are disjoint and this implies that the adversary can force that the number of errors will be at least the size of the maximum matching in the interference graph.

For a weighted graph, the equivalent quantity will be the size of the maximum fractional matching (where there are weights are on the vertices). Each vertex x will have weight $D(x)$. A

fractional matching assigns non-negative weights $y(\cdot)$ to edges such that for each vertex x, the sum of its incident edges is at most the vertex weight, i.e., $D(x) \geq \sum_{x':(x,x')\in E} y(x,x')$. The weight of the fractional matching is the sum of the weights of the edges, i.e., $\sum_{(x_0,x_1)\in E} y(x_0,x_1)$. Given the fractional matching, the adversary can define the following policy $\pi(z|x)$: given an uncorrupted input x, it selects an incident edge (x,x') with probability $y(x,x')/D(x)$, and corrupts x to $z(x,x')$, and with the remaining probability it maps x to an arbitrary z'. Namely, $\pi(z|x) = \sum_{x':z=z(x,x')} y(x,x')/D(x)$, and with probability $1 - (\sum_{x':(x,x')\in E} y(x,x'))/D(x)$ it sets x to some arbitrary z'. Using the same argument as for the unweighted graph, we show the following guarantee for the adversary.

Theorem 5 *The adversary can guarantee that the expected fraction of errors (relative to the distribution D) is at least the weight of the maximum fractional matching in the weighted interference graph.*

4.3. Optimal policy for learner

For the learner policy we consider the interference graph. For simplicity assume the unweighted case (uniform distribution). A possible learner policy is to select a vertex cover $S \subseteq F_0 \cup F_1$ in the interference graph. (Recall that S is a vertex cover if for any edge (x_1,x_0) either $x_1 \in S$ or $x_0 \in S$.) Given a vertex cover S we show that the learner can guarantee that the number of mistakes is at most $|S|$. First let us define the prediction policy h given the vertex cover S and its complement $\bar{S} = (F_0 \cup F_1) \setminus S$, which is an independent set. Given a corrupted input z, we consider $\rho^{-1}(z) \cap \bar{S}$, the set of possible uncorrupted inputs that can generate z and are not in the vertex cover. We have the following cases:

1. If $\rho^{-1}(z) \cap \bar{S} \subseteq F_1$ then predict $h(z) = 1$,
2. If $\rho^{-1}(z) \cap \bar{S} \subseteq F_0$ then predict $h(z) = 0$.
3. If both $\rho^{-1}(z) \cap \bar{S} \subseteq F_1$ and $\rho^{-1}(z) \cap \bar{S} \subseteq F_0$, since $F_1 \cap F_0 = \emptyset$, it must be that $\rho^{-1}(z) \subseteq S$. In this case we make an arbitrary prediction $h(z)$.
4. For $\rho^{-1}(z) \cap \bar{S} \not\subseteq F_1$ and $\rho^{-1}(z) \cap \bar{S} \not\subseteq F_0$, we claim that this is impossible, i.e., we can not have $x_0 \in F_0$ such that $x_0 \in \rho^{-1}(z) \cap \bar{S}$ and $x_1 \in F_1$ such that $x_1 \in \rho^{-1}(z) \cap \bar{S}$. If there is such a pair x_0 and x_1, then there is an edge $(x_0,x_1) \in E$ in the interference graph, since $z \in \rho(x_0) \cap \rho(x_1)$. Since both $x_0, x_1 \in \bar{S}$ it implies that $x_0 \notin S$ and $x_1 \notin S$, and hence S does not cover the edge (x_0,x_1) and therefore it is not a vertex cover. This contradicts the assumption that S is a vertex cover.

The hypothesis h guarantees that for any uncorrupted input $x \notin S$ the learner does not make an error, and hence the number of errors is at most $|S|$. Also note that h is deterministic.

For a weighted graph, we have a non-negative weight $D(x)$ on vertex x. The size of a weighted vertex cover is the sum of the weights of its vertices, i.e., $D(S) = \sum_{x\in S} D(x)$. We use the same prediction policy as for an unweighted graph (since S is a vertex cover). Since we make errors only on inputs in S, the expected error rate is at most the minimal weight of a vertex cover in the interference graph.

Theorem 6 *The learner can guarantee that the expected fraction of errors (relative to the distribution D) is at most the weight of the minimum weighted vertex cover in the weighted interference graph.*

Since in vertex-weighted bipartite graphs the weight of the minimum weight vertex cover and the weight of the maximum fractional matching are identical, it implies that those are the optimal policies for the learner and the adversary. Since our policy for the learner based on a vertex cover is deterministic, we also show that there is a deterministic learner policy which is optimal.

Corollary 7 *The value of the prediction game is the minimal weight vertex cover, which equals the maximum fractional matching. Moreover, there is an optimal learner policy which is deterministic.*

5. Computing efficiently a near-optimal learner policy - inference setting

Since the learner optimal policy involves computing a vertex cover in a bipartite graph, we can clearly compute it in polynomial time in the graph representation. This implies that the running time would be polynomial in $|\mathcal{X}|$. However, in most applications this would be prohibitively inefficient, and efficient algorithms are running in time polynomial in $\log |\mathcal{X}|$, i.e., the dimension. For this reason we would like to derive near-optimal policies that can be computed in time poly-logarithmic in $|\mathcal{X}|$, i.e., $O(\log^c |\mathcal{X}|)$. Clearly we cannot output the complete policy h in such a time, since there are $|\mathcal{Z}|$ different outputs. The idea is that there is no real need to output a complete policy h; rather we would like to compute it only on the given observation z, which is the corrupted input we need to predict for.

This is exactly the idea behind *local algorithms* (Rubinfeld et al., 2011), where one needs to reply to various queries in polylogarithmic time, while utilizing only polylogarithmic space. The main requirement is that the various outputs of the queries will be consistent with some global solution.

In our case, the local queries are simply given an observable $z \in \mathcal{Z}$ output a prediction $h(z) \in [0, 1]$. In this respect, we have a simple setting since we do not have any global constraints. However, since we reduce the optimal learner policy to computing a vertex cover and the optimal adversary policy to computing a matching, we will need to make sure that the computed quantity (vertex cover and matching) are indeed a feasible solution (in addition to being near optimal).

An important building block in our algorithms will be an algorithm to compute a near optimal unweighted matching in bipartite graphs. We will utilize the following theorem from Mansour and Vardi (2013). (A similar result appears in Even et al. (2014).)

Theorem 8 *(Mansour and Vardi, 2013, Theorem 3) There exists a local computation algorithm with the following properties. Given $\epsilon > 0$, and a graph $G = (V, E)$ of bounded degree d, the algorithm replies whether any given edge of G is in some matching in time $O(\log^4 |V|)$, using memory $O(\log^3 |V|)$, and with failure probability at most $1/|V|$. Furthermore, its responses to such edge queries are all consistent with some maximal matching of G which is a $(1-\epsilon)$-approximation to the maximum matching. (The big-Oh notation used here and latter hides constants which may depend on d and $1/\epsilon$.)*

The above theorem immediately gives a near optimal adversary policy for the unweighted case.

Corollary 9 *Let D be the uniform distribution. There is an algorithm for that computes an adversary policy π in time $O(\log^4 |\mathcal{X}|)$, memory size $O(\log^3 |\mathcal{X}|)$ and the failure probability (some query returning a incorrect output) is $1/|\mathcal{X}|$ and guarantees an error rate of at least $(1-\epsilon)OPT - 1/|\mathcal{X}|$, where OPT is the optimal error rate an adversary can guarantee.*

While computing the adversary policy efficiently is intellectually interesting, being the good guys, our main goal is to compute a near optimal learner policy. For computing a learner policy we need a local algorithm for vertex cover. A simple corollary from what we have is that if we get a vertex cover S of weight $w(S)$ then we can define a learner's prediction policy that has error probability at most $w(S)$, and more importantly, we can compute the the prediction policy from S locally. (The proof is in Appendix C.)

Theorem 10 *Given an oracle for membership in a weighed vertex cover S of weight $w(S)$, which runs in time T and failure probability δ, we can locally compute a prediction in time $O(mT) = O(T)$. The expected error of the resulting prediction policy is at most $w(S) + \delta$.*

In the next section we give local algorithms for computing near minimum vertex cover in a bipartite graph. This will imply an efficient near optimal prediction algorithm.

6. Local algorithm for vertex cover in bipartite graphs

In this section we derive a local algorithm for weighted vertex cover. We first derive the algorithm for the unweighted case, which includes most of the ingredients we later need for the weighted case. The weighted vertex cover local algorithm appears in Appendix D.

Recall that given a matching M in a graph, an *exposed* vertex is an unmatched vertex, an *alternating path* is a simple path in which edges alternate as being out of or in M, and an *augmenting path* is an alternating path connecting two exposed vertices (hence its length is odd).

The matching produced by (Mansour and Vardi, 2013; Even et al., 2014) has the property that it does not have any augmenting path of length $2\ell + 1$ or shorter, for $\ell \leq 1/\epsilon$. By the classical result of (Hopcroft and Karp, 1973) such a matching is guaranteed to have a size which is at least within $1 - 1/\ell$ of the maximum matching. Since we use the algorithm of Mansour and Vardi (2013) as a subroutine for computing an approximate maximum matching, we can also assume that the resulting output does not have an augmenting path of length $2\ell + 1$ for any $\ell \leq 1/\epsilon$.

Let $G(F_0, F_1; E)$ be a bipartite graph. Let $M \subset E$ be a matching for which there are no augmenting paths of length $2\ell + 1$ or less. We wish to compute a vertex cover S using a local algorithm. That is, for every vertex v the algorithm decides whether $v \in S$. We propose a randomized algorithm, where the only randomization is in two aspects:

1. Choose an odd integer parameter r uniformly at random in the range $[1, \ell]$.
2. Choose at random one side of G (either F_0 or F_1) to be the distinguished side W.

Given a vertex v, the algorithm checks only $N^r(v)$, the r-local neighborhood of v (namely, the subgraph induced on those vertices of distance at most r from v), and is told for each edge in this neighborhood whether it belongs to M. It places v in one of three sets, S, T_1 or T_2. The intended vertex cover is S, and the distinction between T_1 and T_2 is only for sake of the analysis.

Algorithm LVC (local vertex cover):

1. If v is exposed, then $v \in T_2$.
2. If there are no exposed vertices in $N^r(v)$, then $v \in S$ if $v \in W$, and $v \in T_1$ if $v \notin W$.
3. For every exposed vertex $u \in N^r(v)$, let $d_{alt}(u, v)$ denote the length of the shortest alternating path in $G(N^r(v))$ connecting u to v, and ∞ if there is no such alternating path. Here an alternating path starts at an exposed vertex u and ends at a non-exposed vertex v, and hence its length might be either even or odd. Let $d = \min\{d_{alt}(u, v)\}$, where the minimum is taken over all exposed $u \in N^r(v)$.

(a) If $d > r$ then $v \in S$ if $v \in W$, and $v \in T_1$ if $v \notin W$.
(b) If $d \leq r$ is odd then place v in S.
(c) If $d < r$ is even then place v in T_2. (Step 1 of the algorithm is a special case of this.)

Theorem 11 *For every choice of odd $r \in [1, \ell]$ and $W \in \{F_0, F_1\}$, the corresponding set S output by algorithm LVC is a vertex cover.*

Proof To prove that S is a vertex cover, we need to show that there are no edges in the subgraph induced on $T_1 \cup T_2$. Within T_1 there are no edges because T_1 is disjoint from W, and W is either F_0 or F_1, and hence its complement is an independent set.

Within T_2 there are no edges. Suppose for the sake of contradiction that there is an edge between $u, v \in T_2$. Then by being in T_2, there are exposed vertices u' and v' such that there is an alternating path starting at u', taking an even number of steps (at most r) to reach u, then takes the edge (u, v) (which is not in M because u was matched to the vertex leading to it in the alternating path from u'), and then takes an even number of steps (at most r) to reach v'. This is an augmenting path of length at most $2r + 1 \leq 2\ell + 1$. If the path is simple it is a contradiction to our assumption about the matching M. If the path is not simple, let w be the first node in the alternating path from u' to u that appears also in the alternating path from v' to v. If the path from v' to w and then to u' is an odd length path, then it is an augmenting path of length at most $2r + 1 \leq 2\ell + 1$, contradicting the nonexistence of such paths. Otherwise, w to v to u and back to w is an odd length cycle, contradicting the fact that the graph is bipartite.

There are no edges between T_1 and T_2 because as we shall show, for every vertex $v \in T_2$, all its neighbors are in S. In other words, we need to show that v cannot have a neighbor in $T_1 \cup T_2$. We have already shown that T_2 is an independent set, so it remains to show that v does not have a neighbor in T_1. Let d denote the length of the shortest alternating path from v to an exposed vertex v', and recall that $v \in T_2$ implies that $d < r$ and d is even. One neighbor u of v precedes v on this alternating path (via an edge in M), and hence $u \in S$. All other neighbors of v have an alternating path of length $d+1 \leq r$ through v to v'. The neighbors of v have shortest alternating path of length either $d - 1$, d or $d + 1$ (otherwise v will have an alternating path shorter than d). If their shortest alternating path is length $d-1$ or $d+1$ they are in S. Node v cannot have a neighbor w with shortest alternating path length d, since then we have a path w' to w to v to v' of odd length, and we reach a contradiction like in the proof showing there are no edges in T_2. Hence none of the neighbors of v is in T_1. ■

Theorem 12 *For a random choice of odd $r \in [1, \ell]$ and $W \in \{F_0, F_1\}$, the expected size of the output S of algorithm LVC is at most $(1 + \frac{1}{\ell})|M|$.*

The above theorem is a special case of Theorem 14 for the weighted case, which appears in Appendix D.

References

Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *SODA*, pages 1132–1139, 2012.

Dana Angluin and Philip Laird. Learning from noisy examples. *Mach. Learn.*, 2(4):343–370, April 1988.

A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

N. Cesa-Bianchi, Y. Mansour, and G. Stoltz. Improved second-order bounds for prediction with expert advice. *Machine Learning*, 66(2/3):321–352, 2007. Preliminary version in COLT 2005.

Guy Even, Moti Medina, and Dana Ron. Best of two local models: Local centralized and local distributed algorithms. *CoRR*, abs/1402.3796, 2014. URL http://arxiv.org/abs/1402.3796.

Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999. Preliminary version in COLT 1996.

Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2): 302–343, 2002. Preliminary version in 29th STOC, 1997.

John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

Michael J. Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM J. Comput.*, 22 (4):807–837, 1993.

Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *APPROX-RANDOM*, pages 260–273, 2013.

Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *ICALP (1)*, pages 653–664, 2012.

Yishay Mansour, Aviad Rubinstein, and Moshe Tennenholtz. Robust probabilistic inference. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 449–460, 2015.

Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *ICS*, pages 223–238, 2011.

Robert E. Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. The MIT Press, 2012. ISBN 0262017180, 9780262017183.

L. G. Valiant. Learning disjunction of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'85, pages 560–566, 1985.

Acknowledgements

UF: Work partly done in Microsoft Research, Hertzelia. Work supported in part by the Israel Science Foundation (grant No. 621/12) and by the I-CORE Program of the Planning and Budgeting Committee and The Israel Science Foundation (grant No. 4/11).
YM: This research was supported in part by The Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11), by a grant from the Israel Science Foundation, by a grant from United States-Israel Binational Science Foundation (BSF).

Appendix A. Proof of Theorem 1

Proof [of Theorem 1] Define $W^t_{(x,y)} = \sum_{z\in\rho(x)} w_t(z,(x,y))$ and $W^t = \prod_{(x,y)\in S} W^t_{(x,y)}$. Let

$$F^t_{(x,y)} = \frac{\sum_{z\in\rho(x)} w_t(z,(x,y))\ell_t(z,(x,y))}{\sum_{z\in\rho(x)} w_t(z,(x,y))} = \sum_{z\in\rho(x)} P^t(z,(x,y))\ell_t(z,(x,y))$$

which is the loss of the online algorithm on (x,y) at step t, and let

$$F^t = \sum_{(x,y)\in S} F^t_{(x,y)} \ ,$$

which is the total loss of the online algorithm at time t.

Since $W^t_{(x,y)} \geq (1+\eta)^{\max_{z\in\rho(x)} L_T(z,(x,y))}$ we have that $W^T \geq (1+\eta)^{\mathcal{L}^*}$. Now we also have,

$$W^{t+1}_{(x,y)} = \sum_{z:\ell_t(z,(x,y))=0} w_t(z,(x,y)) + \sum_{z:\ell_t(z,(x,y))=1} (1+\eta)w_t(z,(x,y)) = W^t_{(x,y)}(1+\eta F^t_{(x,y)}) \ .$$

This implies that,

$$W^T_{(x,y)} = W^0_{(x,y)} \prod_{t=1}^{T}(1+\eta F^t_{(x,y)}) = m\prod_{t=1}^{T}(1+\eta F^t_{(x,y)}) \ .$$

Finally, we have

$$W^T = \prod_{(x,y)} W^T_{(x,y)} = \prod_{(x,y)} m\prod_{t=1}^{T}(1+\eta F^t_{(x,y)}) = m^{|S|}\prod_{(x,y)}\prod_{t=1}^{T}(1+\eta F^t_{(x,y)})$$

Combining the two we have,

$$(1+\eta)^{\mathcal{L}^*} \leq m^{|S|}\prod_{(x,y)}\prod_{t=1}^{T}(1+\eta F^t_{(x,y)}) \ .$$

Tacking the logarithm implies,

$$\mathcal{L}^*\ln(1+\eta) \leq |S|\ln m + \sum_{(x,y)}\sum_{t=1}^{T}\ln(1+\eta F^t_{(x,y)}) \ .$$

Since $z - z^2 \leq \ln(1+z) \leq z$ for $z \geq 0$[3] we have,

$$\mathcal{L}^*(\eta - \eta^2) \leq |S| \ln m + \sum_{(x,y)} \sum_{t=1}^{T} \eta F^t_{(x,y)} \ .$$

Since the loss of the online algorithm is $L_T^{ON} = \sum_{t=1}^{T} \sum_{(x,y)} F^t_{(x,y)}$ we have

$$\mathcal{L}^*(1-\eta) - \frac{|S| \ln m}{\eta} \leq L_T^{ON} \ .$$

Setting $\eta = \sqrt{\frac{|S| \ln m}{\mathcal{L}^*}}$ we have,

$$\mathcal{L}^* - 2\sqrt{\mathcal{L}^* |S| \ln m} \leq L_T^{ON} \ .$$

We now like to relate the learned hypothesis to the value of the zero sum game. Namely show that $P = (1/T) \sum_{t=1}^{T} P^t$ is a near optimal policy for the adversary and $Q = (1/T) \sum_{t=1}^{T} h_t$ is a near optimal policy for the learner. The analysis is similar to the one in Schapire and Freund (2012, Chapter 6).

$$\max_P \min_{Q \in \Delta(\mathcal{H})} \sum_{(x,y) \in S} \sum_{z \in \rho(x)} \sum_{h \in \mathcal{H}} P(z,(x,y)) Q(h) I(h(z) \neq y) \tag{3}$$

$$\geq \min_{Q \in \Delta(\mathcal{H})} \frac{1}{T} \sum_{t=1}^{T} \sum_{(x,y) \in S} \sum_{z \in \rho(x)} \sum_{h \in \mathcal{H}} P^t(z,(x,y)) Q(h) I(h(z) \neq y) \tag{4}$$

$$\geq \frac{1}{T} \sum_{t=1}^{T} \sum_{(x,y) \in S} \sum_{z \in \rho(x)} \min_{Q \in \Delta(\mathcal{H})} \sum_{h \in \mathcal{H}} P^t(z,(x,y)) Q(h) I(h(z) \neq y) \tag{5}$$

$$= \frac{1}{T} \sum_{t=1}^{T} \sum_{(x,y) \in S} \sum_{z \in \rho(x)} P^t(z,(x,y)) I(h_t(z) \neq y) \tag{6}$$

$$= \frac{L_T^{ON}}{T} \tag{7}$$

$$\geq \frac{\mathcal{L}^*}{T} - \frac{2\sqrt{\mathcal{L}^* |S| \ln m}}{T} \tag{8}$$

$$= \max_P \sum_{(x,y) \in S} \sum_{z \in \rho(x)} \frac{1}{T} \sum_{t=1}^{T} P(z,(x,y)) I(h_t(z) \neq y) - \frac{2\sqrt{\mathcal{L}^* |S| \ln m}}{T} \tag{9}$$

$$\geq \min_Q \max_P \sum_{(x,y) \in S} \sum_{z \in \rho(x)} \sum_{h \in \mathcal{H}} \sum_{t=1}^{T} P(z,(x,y)) Q(h) I(h(z) \neq y) - \frac{2\sqrt{\mathcal{L}^* |S| \ln m}}{T} \tag{10}$$

From line (3) to (4), we are fixing a specific P rather than the maximizing optimal P. From line (4) to (5), we are allowing the hypothesis to depend on the time step t. From line (5) to (6), we

3. At $z = 0$ the three expressions are equal and their derivatives are $1 - 2z \leq (1+z)^{-1} \leq 1$ for $z \geq 0$, and hence the inequalities hold.

are using the fact that h_t is optimal for a given P^t. From line (6) to (7), we are using the definition of L_T^{ON}. From line (7) to (8), we are using the regret bound we showed. From line (8) to (9), we are using the definition of $\mathcal{L}^*$. From line (9) to (10), we are replacing a specific Q with the optimal minimizing one.

To show that $P = (1/T)\sum_{t=1}^{T} P^t$ is a near optimal policy for the adversary, we can start at line (4). By the minimax theorem, we can rewrite line (10) and get,

$$\min_{Q\in\Delta(\mathcal{H})} \frac{1}{T}\sum_{t=1}^{T}\sum_{(x,y)\in S}\sum_{z\in\rho(x)}\sum_{h\in\mathcal{H}} P^t(z,(x,y))Q(h)I(h(z)\neq y)$$

$$\geq \max_{P}\min_{Q}\sum_{(x,y)\in S}\sum_{z\in\rho(x)}\sum_{h\in\mathcal{H}}\sum_{t=1}^{T} P(z,(x,y))Q(h)I(h(z)\neq y) - \frac{2\sqrt{\mathcal{L}^*|S|\ln m}}{T}$$

and hence P^t is a $\frac{2\sqrt{\mathcal{L}^*|S|\ln m}}{T}$-optimal policy for the adversary.

To show that $Q = (1/T)\sum_{t=1}^{T} h_t$ is a near optimal policy for the learner, we can end at line (9). By the minimax theorem, we can rewrite line (3) and get,

$$\min_{Q\in\Delta(\mathcal{H})}\max_{P}\sum_{(x,y)\in S}\sum_{z\in\rho(x)}\sum_{h} P(z,(x,y))Q(h)I(h(z)\neq y)$$

$$\geq \max_{P}\sum_{(x,y)\in S}\sum_{z\in\rho(x)}\sum_{t=1}^{T} P(z,(x,y))I(h_t(z)\neq y) - \frac{2\sqrt{\mathcal{L}^*|S|\ln m}}{T}$$

and hence h is a $\frac{2\sqrt{\mathcal{L}^*|S|\ln m}}{T}$-optimal policy for the learner.

Since $\mathcal{L}^* \leq T$, we have that for $T \geq (4|S|\ln m)/\epsilon^2$ the policies are ϵ-optimal. ■ ■

Appendix B. Generalization bound (proof of Theorem 3)

Proof [of Theorem 3] Fix $h \in \mathcal{H}^N$, where $h(x) = \sum_{i=1}^{N} h_i(x)/N$. Let h' be the hypothesis derived by Lemma 2. Let

$$Z_{x,y}(h') = \max_{z\in\rho(x)} \frac{1}{N_0}\sum_{i=1}^{N_0} I[h'_i(z)\neq y]$$

Clearly, $Z_{x,y}(h') \in [0,1]$. From the definition of h' we have that for any (x,y) with probability at least $1-\delta$ we have $|Z_{x,y}(h) - Z_{x,y}(h')| \leq 2\epsilon$. Using a Chernoff bound, with probability at most δ we have $|E_{(x,y)}[Z_{x,y}(h')] - \frac{1}{|S|}\sum_{(x,y)\in S} Z_{x,y}(h')| \geq \epsilon$, given $|S| \geq (\log 1/\delta)/\epsilon^2$. Therefore, with

probability $1-3\delta$ we have

$$\begin{aligned}\left|E_{(x,y)\sim D}[Z_{x,y}(h)] - \frac{1}{|S|}\sum_{(x,y)\in S} Z_{x,y}(h)\right| \leq & \left|E_{(x,y)\sim D}[Z_{x,y}(h)] - E_{(x,y)\sim D}[Z_{x,y}(h')]\right| \\ & + \left|E_{(x,y)\sim D}[Z_{x,y}(h')] - \frac{1}{|S|}\sum_{(x,y)\in S} Z_{x,y}(h')\right| \\ & + \left|\frac{1}{|S|}\sum_{(x,y)\in S} Z_{x,y}(h) - \frac{1}{|S|}\sum_{(x,y)\in S} Z_{x,y}(h')\right| \\ \leq & 5\epsilon\end{aligned}$$

where in the first expression we use Lemma 2 with the distribution D and in the last expression we use Lemma 2 with the empirical distribution on S. The theorem follows from a union bound over the $|\mathcal{H}|^{N_0}$ possible h', using the bound for N_0 from Lemma 2, and re-scaling 5ϵ and 3δ to ϵ and δ. ■

Appendix C. Proof from Section 5

Proof [of Corollary 9] The degree of the interference graph is at most $d = m^2 = O(1)$. This follows since an uncorrupted input x has at most m corrupted values $z \in \rho(x)$ it can be mapped to. For each $z \in \rho(x)$ we have at most m uncorrupted values in $x' \in \rho^{-1}(z)$. The edges incident to x are (x, x') such that $x' \in \rho^{-1}(\rho(x))$, and therefore the degree of x is at most m^2.

We apply the algorithm of Theorem 8 on the interference graph. Our set of nodes is $V = \mathcal{X}$, which implies a running time of $O(\log^4 |\mathcal{X}|)$, memory size $O(\log^3 |\mathcal{X}|)$ and the failure probability (some query returning a incorrect output) is $1/|\mathcal{X}|$. This implies that the effect of the failed queries on the error probability is negligible, only $1/|\mathcal{X}|$. The algorithm's output defines an implicit matching M such that $|M| \geq (1-\epsilon)OPT$.

We now need to specify the adversary policy π given an oracle for maximum matching. Given any uncorrupted input $x \in \mathcal{X}$, we compute $x' \in \rho^{-1}(\rho(x))$. For any such pair we test whether it is an edge in the matching, i.e., $(x, x') \in M$ using the algorithm of Theorem 8. If $(x, x') \in M$ we map x to $z \in \rho(x) \cap \rho(x')$, i.e., $\pi(x) = z$. Otherwise, if there is no such x', we map x arbitrarily, say $\pi(x) = x$.

The number of errors any hypothesis makes is at least $|M| \geq (1-\epsilon)OPT|\mathcal{X}|$ since for any edge $(x_0, x_1) \in M$ we are guarantee that the learner makes one error. Since the probability that the algorithm fails is at most $1/|\mathcal{X}|$ the expected probability of error is at least $|M|/|\mathcal{X}| - 1/|\mathcal{X}|$. ■

Proof [of Theorem 10] Given a corrupted input z, for each $x \in \rho^{-1}(z)$ we compute whether it is in the vertex cover S. This involves $|\rho^{-1}(z)| \leq m$ queries to the vertex cover oracle. If there is $x \in \rho^{-1}(z)$ which is not in the vertex cover we return $f(x)$ as $h(z)$, i.e., $h(z) = f(x)$. If there is no such x, i.e., $\rho^{-1}(z) \subseteq S$, then we return an arbitrary value for $h(z)$. (We already showed in the arguments leading to Theorem 6, that it is impossible to have $x_0, x_1 \in \rho^{-1}(z)$ which are both not

in the vertex cover, and $f(x_0) = 0$ and $f(x_1) = 1$.) As in the arguments leading to Theorem 6 this implies that the policy has error at most $w(S)$.

For the time complexity, we have at most m calls to the oracle each requires T time. ■

Appendix D. Weighted Vertex Cover

Let $G(U, V; E)$ be a bipartite graph, and let w be a weight function that assigns nonnegative weights to vertices. Given a vertex weighted graph, a *fractional matching* M is a function y assigning nonnegative weights to edges such that for every vertex v, the sum of weights of edges incident with v does not exceed the weight of v. Namely, $\sum_{e|v\in e} y(e) \leq w(v)$. A vertex v is *exposed* in M if $\sum_{e|v\in e} y(e) < w(v)$. An *augmenting path* in M is a path with an odd number of edges connecting two exposed vertices, in which every even edge has strictly positive weight. Given an augmenting path, a *greedy increase* changes M by adding ϵ to the weight of every odd edge in the path and subtracting ϵ from every even edge on the path, where ϵ is the maximum possible value that keeps the result a legal fractional matching.

Suppose that M is a fractional matching with no augmenting paths of length $2\ell + 1$ or less. Given the bipartite graph G and the fractional matching M, algorithm LVC extends as is to the weighted setting. The only difference is that now vertices have weights, and the given matching is a fractional matching rather than an integer one. We call the resulting algorithm WLVC (weighted local vertex cover).

The proof that algorithm WLVC produces a vertex cover is identical to the proof of Theorem 11.

To relate the weight of the vertex cover found by algorithm WLVC and the weight of the minimum vertex cover, we can use the following well known relation between fractional matchings and weighted vertex covers.

Proposition 13 *For every graph $G(V, E)$ with nonnegative vertex weights, for every fractional matching M and every vertex cover S, $y(M) \leq w(S)$.*

Proof For every edge $e \in E$, at least one endpoint is in S. Let e transfer its matching weight $y(e)$ to its endpoint that is in S (or one of the two endpoints arbitrarily if both are in S). Hence all of $y(M)$ is transferred to vertices in S, and no vertex in S receives more weight than its own weight, because M is a fractional matching. Hence $y(M) \leq w(S)$. ■

Theorem 12 is modified as follows. Let $y(M) = \sum_{e\in E} y(e)$ be the total weight of the fractional matching. Given a set S of vertices, let $w(S) = \sum_{v\in S} w(v)$ be its weight.

Theorem 14 *For a random choice of r and W, the expected weight of the output S of algorithm WLVC is at most $(1 + \frac{1}{\ell})y(M)$, assuming M is a fractional matching with no augmenting paths of length $2\ell + 1$ or less.*

Proof By construction, no exposed vertex is in S. Hence it remains to upper bound the expected weight of matched vertices that are in S. Let M' be the set of edges such that both of their vertices are in S, i.e., $M' = \{(u, v) : u, v \in S, (u, v) \in M\}$. Then, we have that $w(S) \leq y(M) +$

$\sum_{e \in M'} y(e)$. In the following we bound the probability that an edge e is in M' by $1/\ell$, which derives the desired bound.

Consider an arbitrary edge $(u, v) \in M$. Let d_u denote the length of the shortest alternating path from u to an exposed vertex (and ∞ if no such path exists), and define d_v similarly for v. Suppose without loss of generality that $d_u \leq d_v$.

Both u and v are placed in S iff $r = d_u$ (an event that happens with probability $1/\lceil \frac{\ell}{2} \rceil$ over the choice of r, and only if $d_u \leq \ell$) and $v \in W$ (an event that happens with probability $\frac{1}{2}$ over the choice of W). Hence $E[w(S)] \leq (1 + \frac{1}{\ell})y(M)$, as desired. ■

Let us now explain how the randomized local algorithm of Mansour and Vardi (Mansour and Vardi, 2013) can be modified so that it finds fractional matchings in vertex weighted graphs (rather than matchings in unweighted graphs). As in (Mansour and Vardi, 2013), the goal would be to find fractional matchings with no short augmenting paths (of length $2\ell + 1$ or less). As in (Mansour and Vardi, 2013), each vertex will inspect a neighborhood of size $O(\log n)$, where n is the size of the graph, and the O notation hides a constant the depends only on the maximum degree Δ and on ℓ. (Unfortunately, the upper bounds known on this constant might be prohibitively large.)

The main lemma that is needed in order to extend the algorithm and analysis of (Mansour and Vardi, 2013) to vertex weighted (bipartite) graphs is the following.

Lemma 15 *Let G be a vertex weighted bipartite graph and let M be a fractional matching in which the shortest augmenting path is of length $2\ell+1$. Let $\mathcal{P} = P_1, \ldots$ be a list of all augmenting paths of length $2\ell+1$ in arbitrary order. Consider a greedy algorithm that produces a new fractional matching M' by performing a sequence of greedy improvements on the given paths in the order of their appearance, where for each path the greedy improvement is with respect to the fractional matching obtained after the preceding path. (Possibly by that time the path is no longer an augmenting path, and gives no greedy improvement.) Then M' has no augmenting paths of length $2\ell + 1$ or less.*

Proof No edge e serves as an even edge on some path P_i and as an odd edge on some other path P_j. This is because the union of P_i and P_j would then necessarily include an augmenting path shorter than $2\ell + 1$. Let $e = (u, v)$, and the paths be $P_i = u_i, \ldots, u, v, \ldots v_i$ and $P_j = u_j, \ldots, u, v, \ldots v_j$. Then $u_i, \ldots, u, \ldots u_j$ is an odd length path (not necessary simple, but $u_i \neq u_j$ because G is bipartite) and so is $v_i, \ldots, v, \ldots v_j$. One of them must be shorter than $2\ell + 1$ (because the edge e was discarded).

The above implies that changes to edge weights during the greedy algorithm are monotone. Hence by the end of the greedy algorithm, no path in $\mathcal{P}$ remains an augmenting path (no greedy improvement can undo the effects of previous greedy improvements).

It remains to show that no path $P \notin \mathcal{P}$ of odd length at most $2\ell+1$ becomes an augmenting path as a consequence of the greedy algorithm. Consider for the sake of contradiction that this happens, and let $P_i \in \mathcal{P}$ be the path that was processed by the greedy algorithm at the first time that this happens. As processing P_i cannot create newly exposed vertices, the only way by which P can become an augmenting path is by sharing an edge e with P_i. Namely, e might be an odd edge in P_i and and even edge in P, and furthermore, $y(e) = 0$. In this case, processing P_i makes the weight of e positive and may by this turn P into an augmenting path. However, an argument similar to the one used for P_i and P_j shows that in this case the union of P_i and P contained an augmenting path shorter than $2\ell + 1$ already before processing P_i, which is a contradiction. ■

D.1. The need to know a bipartization

In our inference setting it is clear that one can know the bipartization of the conflict graph by simply querying the label of the target function for a given input x, i.e., $f(x)$. However, our local algorithm for vertex cover works for any bipartite graph G, and therefore it is interesting whether in general one needs to know the bipartization G in the same way that our algorithm LVC requires it.

Algorithm LVC uses its knowledge of a matching M, and of a bipartization of G (so as to determine which vertices are W). A matching M as required by the algorithm can be found locally, as shown in (Mansour and Vardi, 2013). Here we show that knowledge of the bipartization is essential for obtaining via a local algorithm a nearly minimum vertex cover in bipartite graphs.

Consider the following two distributions on 3-regular graphs with an even number n of vertices. G_1^n is the distribution over a cycle plus a random matching. G_2^n is the distribution over a cycle plus a random matching that matches the even numbered vertices to the odd numbered vertices.

Proposition 16

1. *Every graph in G_2^n has a vertex cover of size $n/2$.*
2. *For most graphs from G_1^n the minimum vertex cover has size at least $(\frac{1}{2}+c)n$, for some constant $c>0$ and all sufficiently large n.*

Proof Every graph in G_2^n is bipartite and hence has a vertex cover of size at most $n/2$. (Being regular, it also holds that graphs in G_2^n have no vertex cover smaller than $n/2$, though this fact is not needed for our future arguments.) This proves item 1.

Item 2 follows from standard probabilistic arguments (e.g., similar arguments are used in Goldreich and Ron (2002)). Details omitted. ■

Consider graph algorithms that access the input graph by queries. Namely, the vertices of the graph are named arbitrarily from 1 to n, a query specifies a vertex name, and the reply is the names of its neighbors. We use the following theorem of Goldreich and Ron (2002).

Theorem 17 *Let A be an arbitrary randomized algorithm that uses $o(\sqrt{n})$ adaptive queries to the input graph G, and outputs a single bit. Then $|Pr_{G\in G_1^n}[A(G)=1]-Pr_{G\in G_2^n}[A(G)=1]|=o(1)$.*

Corollary 18 *For some constant $\epsilon>0$ (independent of n), there is no local algorithm B that on input an n vertex bipartite graph has all of the following properties:*

1. *Given a vertex v, algorithm B makes $o(\sqrt{n})$ adaptive queries to G and outputs a single bit.*
2. *The vertices with $B(v)=1$ form a vertex cover of G.*
3. *The size of the set $|\{v|B(v)=1\}|$ approximates the size of the minimum vertex cover within a ratio of $1+\epsilon$.*

Proof Suppose for the sake of contradiction that there is an algorithm B that for bipartite graphs satisfies the three properties listed in the Corollary. (For non-bipartite graphs, we only assume that the output of the algorithm on any given vertex is either 0 or 1. Properties 2 and 3 need not hold in this case, though we remark that without loss of generality, property 1 can be assumed to hold – if the algorithm exceeds the total number of allowable queries it is terminated and the output is set to 0.) We show how B can be used in order to construct an algorithm A that contradicts Theorem 17.

Given an input graph G sampled from either G_1^n or G_2^n, algorithm A will proceed as follows.

1. Pick independently at random a set S of $\Theta\left(\frac{1}{\epsilon^2}\right)$ vertices from G. On each vertex $v \in S$ run B. If $|\{v|B(v)=1\}| \geq (\frac{1}{2}+\epsilon)|S|$ then output 1.
2. For each vertex $v \in S$ with $B(v)=0$ and each of the three neighbors u of v, compute $B(u)$. If for any such u it holds that $B(u)=0$ then output 1.
3. Output 2.

In what follows, we shall refer to a vertex v that has a neighbor u such that $B(v)=B(u)=0$ as a *witness* (for the graph coming from G_1^n).

We first show that if G is sampled from G_1^n, then algorithm A is likely to output 1. To avoid outputting 1, the fraction of vertices in S with $B(v)=0$ is at least $\frac{1}{2}-\epsilon$. Consequently, we infer that with high probability, the fraction of vertices in G with $B(v)=0$ is at least $\frac{1}{2}-2\epsilon$. Item 2 of Proposition 16 implies that there is no independent set in G larger than $(\frac{1}{2}-c)n$. If follows that G has at least $(c-2\epsilon)n$ witnesses, namely, vertices with $B(v)=0$ and a neighbor u with $B(u)=0$. (This is because after removing all witnesses, the set $\{v|B(v)=0\}$ becomes an independent set.) For $\epsilon = c/3$, the set S is likely to contain a witness (in fact, $\Omega(1/\epsilon)$ witnesses), and step 2 of algorithm A will detect this.

We now show that if G is sampled from G_2^n, then algorithm A is likely to output 2. By Proposition 16 the graph G is bipartite and has a vertex cover of size $n/2$. By Property 2 of the Corollary the vertices of G with $B(v)=1$ form a vertex cover, and by Property 3 their number is at most $\frac{1+\epsilon}{2}n$. Hence step 1 of algorithm A is likely to pass without outputting 1. Likewise, step 2 of algorithm A will pass without outputting 1, because the vertices with $B(v)=0$ form an independent set. Hence step 3 will output 2.

The number of queries made by algorithm A is $o(\frac{\sqrt{n}}{\epsilon^2})$ for some fixed constant $\epsilon > 0$, thus contradicting Theorem 17. ■