**University of Oregon**

# Convex Adversarial Collective Classification

by

Mohamad Ali Torkamani

The directed research project (DRP) report submitted to the CIS department of University of Oregon

Under the supervision of
Prof. Daniel Lowd
Computer and Information Science Department

March 2013

# *Abstract*

Many real-world domains, such as web spam, auction fraud, and counter-terrorism, are both relational and adversarial. Existing work on adversarial machine learning assumes that the attributes of each instance can be manipulated independently. Collective classification violates this assumption, since object labels depend on the labels of related objects as well as their own attributes. In this paper, we present a novel method for robustly performing collective classification in the presence of a malicious adversary that can modify up to a fixed number of binary-valued attributes. Our method is formulated as a convex quadratic program that guarantees optimal weights against a worst-case adversary in polynomial time. In addition to increased robustness against active adversaries, this kind of adversarial regularization can also lead to improved generalization even when no adversary is present. In experiments on real and simulated data, our method consistently outperforms both non-adversarial and non-relational baselines.

# Contents

# Chapter 1

# Introduction

In collective classification [1], we wish to jointly label a set of interconnected objects using both their attributes and their relationships. For example, linked web pages are likely to have related topics; friends in a social network are likely to have similar demographics; and proteins that interact with each other are likely to have similar locations and related functions. Probabilistic graphical models, such as Markov networks, and their relational extensions, such as Markov logic networks [2], can handle both uncertainty and complex relationships in a single model, making them well-suited to collective classification problems.

However, many collective classification models must also cope with test data that is drawn from a different distribution than the training data. In some cases, this is simply a matter of concept drift. For example, when classifying blogs, tweets, or news articles, the topics being discussed will vary over time. In other cases, the change in distribution can be attributed to one or more adversaries actively modifying their behavior in order to avoid detection. For example, when search engines began using incoming links to help rank web pages, spammers began posting comments on unrelated blogs or message boards with links back to their websites. Since incoming links are used as an indication of quality, manufacturing incoming links makes a spammy web site appear more legitimate. In addition to web spam [3, 4], other explicitly adversarial domains include counter-terrorism, online auction fraud [5], and spam in online social networks.

Rather than simply reacting to an adversary's actions, recent work in adversarial machine learning takes the proactive approach of modeling the learner and adversary as players in a game. The learner selects a function that assigns labels to instances, and the adversary selects a function that transforms malicious instances in order to avoid detection. The strategies chosen determine the outcome of the game, such as the success rate of the adversary and the error rate of the chosen classifier. By analyzing the dynamics

of this game, we can search for an effective classifier that will be robust to adversarial manipulation. Even in non-adversarial domains such as blog classification, selecting a classifier that is robust to a hypothetical adversary may lead to better generalization in the presence of concept drift or other noise.

Early work in adversarial machine learning included methods for blocking the adversary by anticipating their next move [6], reverse engineering classifiers [7, 8] (and later: [9]), and building classifiers robust to feature deletion or other invariants [10, 11]. More recently, Brückner and Scheffer showed that, under modest assumptions, Nash equilibria can be found for domains such as spam [12]. However, current adversarial methods assume that instances are independent, ignoring the relational nature of many domains.

In this project, we present Convex Adversarial Collective Classification (CACC), which combines the ideas of associative Markov networks [13] (AMNs) and convex learning with invariants [11]. Unlike previous work in learning graphical models, CACC selects the most effective weights *assuming a worst-case adversary* who can modify up to a fixed number of binary-valued attributes. Unlike previous work in adversarial machine learning, CACC allows for dependencies among the labels of different objects, as long as these dependencies are associative. Associativity means that related objects are more likely to have the same label, which is a reasonable assumption for many collective classification domains. Surprisingly, all of this can be done in polynomial time using a convex quadratic program.

In experiments on real and synthetic data, CACC finds much better strategies than both a naïve AMN that ignores the adversary and a non-relational adversarial baseline. In some cases, the adversarial regularization employed by CACC helps it generalize better than AMNs even when the test data is not modified by any adversary.

The rest of our paper is organized as follows. In chapter 2, we present a brief overview of Markov networks and associative Markov networks as applied to collective classification, and adversarial machine learning. We introduce our formulation and algorithm in chapter 3. chapter 5 contains our experiments on real and synthetic data, discussion and conclusion as well as ongoing and future work. In appendix A we have a complete proof of the theorems. Appendix B is a short tutorial on CPLEX.

# Chapter 2

# Background

## 2.1 Max-margin relational learning

*Markov networks* (MNs) represent the joint distribution over a set of random variables $\mathbf{X} = \{X_1, \ldots, X_N\}$ as a normalized product of factors:

$$P(\mathbf{X}) = \frac{1}{Z} \prod_i \phi_i(\mathbf{D}_i)$$

where $Z$ is a normalization constant so that the distribution sums to one, $\phi_i$ is the $i$th factor, and $\mathbf{D}_i \subset \mathbf{X}$ is the scope of the $i$th factor. Factors are sometimes referred to as potential functions. For positive distributions, a Markov network can also be represented as a *log-linear model*:

$$P(\mathbf{X}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\mathbf{D}_i)\right)$$

where $w_i$ is a real-valued weight and $f_i$ a real-valued feature function. For the common case of indicator features, each feature equals 1 when some logical expression over the variables is satisfied and 0 otherwise.

A factor or potential function is *associative* if its value is at least as great when the variables in its scope take on identical values as when they take on different values. For example, consider a factor $\phi$ parameterized by a set of non-negative weights $\{w^k\}$, so that $\phi(y_i, y_j) = \exp(w^k)$ when $y_i = y_j = k$ and 1 otherwise. $\phi$ is clearly associative, since its value is higher when $y_i = y_j$. An *associative Markov network* (AMN) [13] is an MN where all factors are associative. Certain learning and inference problems that are intractable in general MNs have exact polynomial-time solutions in AMNs with binary-valued variables, as will be discussed later.

TABLE 2.1: MLN formulas for a simple collective classification model.

| Weight | Formula |
|---|---|
| $w_0^k$ | $\texttt{Label}(o) = k$ |
| $w_j^k$ | $\texttt{Attribute}_j(o) \Rightarrow \texttt{Label}(o) = k$ |
| $w_e^k$ | $\texttt{Link}(o, o') \wedge (\texttt{Label}(o) = k) \Rightarrow (\texttt{Label}(o') = k)$ |

Markov logic [2] is a language for conveniently defining Markov networks over relational domains. A *Markov logic network* (MLN) is a set of weighted first-order formulas $(w_i, F_i)$, where $w_i$ is the weight of formula $F_i$; the more likely $F_i$ to "true" the larger $w_i$, i.e. $w_i = +\infty$ means $F_i$ has to be true and $w_i = -\infty$ means $F_i$ has to be false. As a corrolary if we have both $F_i$ and $\neg F_i$ we won't need negative weights. Together with a set of logical constants representing objects in the domain, an MLN induces a log-linear model where the features $n_i$ are the number of satisfied groundings of the formulas $F_i$:

$$P(\mathbf{X}) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(\mathbf{X})\right)$$

Here, the random variables in $\mathbf{X}$ are all the ground atoms that can defined with the given set of logical constants. For example, if our domain includes a binary predicate $\texttt{Friends}(x, y)$ and our set of constants is {`Anna`, `Bob`}, then the random variables are given by the ground atoms `Friends(Anna,Anna)`, `Friends(Anna,Bob)`, `Friends(Bob,Anna)`, and `Friends(Bob, Bob)`. MLNs can also be viewed as templates for creating large MNs, where each grounding of the formula $F_i$ defines a factor with value $e^{w_i}$ when that grounding formula is satisfied and 1 otherwise.

An MLN or MN can also represent a conditional distribution, $P(\mathbf{Y}|\mathbf{X})$, in which case the normalization constant becomes a function of the evidence, $Z(\mathbf{X})$.

MLNs make it easy to compactly describe very complex distributions. For example, a simple collective classification model can be defined using relatively simple formulas, as shown in Table 2.1. The subscript $j$ and superscript $k$ indicate that different formulas are defined for each attribute $j \in \{1, \ldots, M\}$ and object label $k \in \{1, \ldots, K\}$. The formula from the first line defines features for the prior distribution over labels in the absence of any attributes or links. The next line relates each object's attributes to its label. The third line relates the labels of neighboring objects. Note that the first formula may be omitted as a special case of the second if we assume that a special bias attribute $\texttt{Attribute}_0(o)$ is true for every object $o$.

The MLN from Table 2.1 can be represented as the following log-linear model:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{ijk} w_j^k x_{ij} y_i^k + \sum_{(i,j)\in E,k} w_e^k y_i^k y_j^k\right)$$

where $x_{ij}$ represents the value of the $j$th attribute of the $i$th object and $y_i^k$ is an indicator variable which equals 1 when the $i$th object is assigned the $k$th label and 0 otherwise. $E$ denotes the set of pairs $(i, j)$ such that the $i$th and $j$th object are linked.

A common inference task is to find the most probable explanation (MPE), the most likely assignment of the non-evidence variables $\mathbf{y}$ given the evidence. This can be done by maximizing the unnormalized log probability, since log is a monotonic function and the normalization factor $Z$ is constant over $\mathbf{y}$. For the simple collective classification model, the MPE task is to find the most likely labeling given the links and attributes:

$$\text{argmax}_{\mathbf{y}} \sum_{ijk} w_j^k x_{ij} y_i^k + \sum_{(i,j)\in E,k} w_e^k y_i^k y_j^k$$

In general, inference in graphical models is computationally intractable. However, for the special case of AMNs with binary-valued variables, MPE inference can be done in polynomial time by formulating it as a min-cut problem [14]. For $w_e^k \geq 0$, our working example of a collective classification model is an AMN over the labels $\mathbf{y}$ given the links $E$ and attributes $\mathbf{x}$. In general, associative interactions are very common in collective classification problems since related objects tend to have similar properties, a phenomenon known as homophily.

Markov networks and MLNs are often learned by maximizing the (conditional) log-likelihood of the training data (e.g., [15]). An alternative is to maximize the margin between the correct labeling and all alternative labelings, as done by max-margin Markov networks ($M^3$Ns) [16] and max-margin Markov logic networks ($M^3$LNs) [17]. Both approaches are intractable in the general case. For the special case of AMNs, however, max-margin weight learning can be formulated as a quadratic program which gives optimal weights in polynomial time as long as the variables are binary-valued [13]. We now briefly describe the solution of Taskar et al, which will later motivate our adversarial extension of AMNs. (We use slightly different notation from the original presentation in order to make the structure of $\mathbf{x}$ and $\mathbf{y}$ clearer.)

The goal of the AMN optimization problem is to maximize the margin between the log probability of the true labeling, $h(\mathbf{w}, \mathbf{x}, \hat{\mathbf{y}})$ (the scoring function $h(.)$ is defined in the next paragraph), and any alternative labeling, $h(\mathbf{w}, \mathbf{x}, \mathbf{y})$. Margin scaling is used to enforce a wider margin from labelings that are more different, according to some

difference function $\Delta(\mathbf{y}, \hat{\mathbf{y}})$. We thus obtain the following minimization problem with an exponential number of constraints (one for each $\mathbf{y}$):

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\xi \\ \text{s.t.} \quad & h(\mathbf{w}, \mathbf{x}, \hat{\mathbf{y}}) - h(\mathbf{w}, \mathbf{x}, \mathbf{y}) \geq \Delta(\mathbf{y}, \hat{\mathbf{y}}) - \xi \quad \forall \mathbf{y} \in \mathcal{Y} \end{aligned}$$

Minimizing the norm of the weight vector is equivalent to maximizing the margin. The slack variable $\xi$ represents the magnitude of the margin violation, which is scaled by $C$ and used to penalize the objective function. For our problem, $h$ is defined by the simple collective classification model Table 1, and $\Delta$ is Hamming distance: $h(\mathbf{w}, \mathbf{x}, \mathbf{y}) = \sum_{i,j,k} w_j^k x_{ij} y_i^k + \sum_{(i,j)\in E,k} w_e^k y_i^k y_j^k$ and $\Delta(\mathbf{y}, \hat{\mathbf{y}}) = N - \sum_{i,k} y_i^k \hat{y}_i^k$. Where N is the total number of objects. To transform this into a tractable quadratic program, Taskar et al. modify it in several ways. First, they replace each product $y_i^k y_j^k$ with a new variable $y_{ij}^k$ and add constraints $y_{ij}^k \leq y_i^k$ and $y_{ij}^k \leq y_j^k$. In other words, $y_{ij}^k \leq \min(y_i^k, y_j^k)$, which is equivalent to $y_i^k y_j^k$ for $y_i^k, y_j^k \in \{0, 1\}$. Second, they replace the exponential number of constraints with a continuum of constraints over a relaxed set of $y \in \mathcal{Y}'$, where $\mathcal{Y}' = \{\mathbf{y} : y_i^k \geq 0; \sum_k y_i^k = 1; y_{ij}^k \leq y_i^k; y_{ij}^k \leq y_j^k\}$. Since all constraints share the same slack variable, $\xi$, we can take the maximum to summarize the entire set by the most violated constraint. After applying these modifications, substituting in $h$ and $\Delta$, and simplifying, we obtain the following optimization problem for our collective classification task:

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\xi \\ \text{s.t.} \quad & \mathbf{w} \geq 0; \\ & \xi - N \geq \max_{\mathbf{y}\in\mathcal{Y}'} \sum_{i,j,k} w_j^k x_{ij} (y_i^k - \hat{y}_i^k) \\ & \qquad + \sum_{(i,j)\in E,k} w_e^k (y_{ij}^k - \hat{y}_{ij}^k) - \sum_{i,k} y_i^k \cdot \hat{y}_i^k \end{aligned} \tag{2.1}$$

Finally, since the inner maximization is itself a linear program, we can replace it with the minimization of its dual to obtain a single quadratic program (not shown). For the two-class setting, Taskar et al. prove that the inner program always has an integral solution, which guarantees that the weights found by the outer quadratic program are always optimal.

For simplicity and clarity of exposition, we have used a very simple collective classification model as our working example of an AMN. This model can easily be extended to allow multiple link types with different weights, link weights that are a function of the evidence, and higher-order links (hyper-edges), as described by Taskar et al. [13]. Our adversarial variant of AMNs, which will be described in Section 4, supports most of these extensions as well.

## 2.2 Adversarial machine learning

Most classic learning algorithms assume that training and test data are drawn from the same distributions. However, in many real world applications, an adversary will actively change its behavior to avoid detection, leading to significantly worse performance in practice. For example, spammers add and remove words from their email messages in order to bypass spam filters, and web spammers try to deceive search engines by creating "link farms" to make a web site seem more important. In computer and network security, many bots are engineered to attack network computers and change their behavior so that intrusion detection systems fail to detect them.

Designing machine learning algorithms that are robust to malicious adversaries is an area of growing interest [18]. One approach is to formulate the problem as a game between the learner and an adversary, each with its own set of strategies and rewards. Dalvi et al. [6] note that finding a Nash equilibrium is often intractable and propose a strategy to anticipate the adversary's next move instead. Brückner and Scheffer [12] present a method to find a Nash equilibrium for non-zero sum games that satisfy certain convexity conditions. In later work, they present results for finding Stackelberg equilibria as well [19]. One special case of adversarial manipulation is feature deletion, in which the adversary chooses the features to remove that would most harm the classifier's performance. This results in a zero-sum game between the learner and adversary that can be solved using robust minimax methods as in [10, 20, 21, 22]. Teo et al. [11] is more general, allowing any set of adversarial actions that afford an efficient numerical solution to be represented as an invariant while learning.

We take particular inspiration from Globerson and Roweis [10] and Teo et al. [11], which take the quadratic program of a max-margin learning problem and substitute in the adversary's worst-case modification of the evidence. By formulating the adversary's modification as a linear program and taking the dual, the learning problem remains convex.

However, none of these methods handles collective classification, in which the label of each object depends on the labels of its neighbors.

# Chapter 3

# Convex Adversarial Collective Classification

Collective classification problems are hard because the number of joint label assignments is exponential in the number of nodes. As discussed in Section 2, if neighboring nodes are more likely to have the same label, then the collective classification problem can be represented as an associative Markov network (AMN), in which max-margin learning and MPE inference are both efficient. To construct an adversarial collective classifier, we start with the AMN formulation (Eq. 2.1) and incorporate an adversarial invariant, similar to the approach of Globerson and Roweis [10]. Specifically, we assume that the adversary may change up to $D$ binary-valued features $x_{ij}$, for some positive integer $D$ that we select in advance. We use $\hat{\mathbf{x}}$ to indicate the true features and $\mathbf{x}$ to indicate the adversarially modified features. The number of changes can be written as: $\Delta(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{i,j} x_{ij} + \hat{x}_{ij} - 2x_{ij}\hat{x}_{ij}$

We define the set of valid $\mathbf{x}$ as $\mathcal{X}' = \{\mathbf{x} : 0 \leq x_{ij} \leq 1; \Delta(\mathbf{x}, \hat{\mathbf{x}}) \leq D\}$. Note that $\mathcal{X}'$ is a relaxation that allows fractional values, much like the set $\mathcal{Y}'$ defined by Taskar et al. We will later show that there is always an integral solution when both the features and labels are binary-valued.

In our adversarial formulation, we want the true labeling $\hat{\mathbf{y}}$ to be separated from any alternate labeling $\mathbf{y} \in \mathcal{Y}'$ by a margin of $\Delta(\mathbf{y}, \hat{\mathbf{y}})$ given any $\mathbf{x} \in \mathcal{X}'$. Rather than including an exponential number of constraints (one for each $\mathbf{x}$ and $\mathbf{y}$), we use a maximization

over $\mathbf{x}$ and $\mathbf{y}$ to find the most violated constraint:

$$\begin{aligned}
&\max_{\mathbf{y}\in\mathcal{Y}',\mathbf{x}\in\mathcal{X}'} h(\mathbf{w},\mathbf{x},\mathbf{y}) - h(\mathbf{w},\mathbf{x},\hat{\mathbf{y}}) + \Delta(\mathbf{y},\hat{\mathbf{y}}) \\
&= \max_{\mathbf{y}\in\mathcal{Y}',\mathbf{x}\in\mathcal{X}'} \sum_{i,j,k} w_j^k x_{ij} y_i^k + \sum_{(i,j)\in E,k} w_e^k y_{ij}^k \\
&\qquad - \sum_{i,j,k} w_j^k x_{ij} \hat{y}_i^k - \sum_{(i,j)\in E,k} w_e^k \hat{y}_{ij}^k \\
&\qquad + N - \sum_{i,k} y_i^k \cdot \hat{y}_i^k
\end{aligned} \tag{3.1}$$

Next, we convert this to a linear program. Since $x_{ij}y_i^k$ is bilinear in $\mathbf{x}$ and $\mathbf{y}$, we replace it with the auxiliary variable $z_{ij}^k$, satisfying the constraints: $z_{ij}^k \geq 0$; $z_{ij}^k \leq x_{ij}$; and $z_{ij}^k \leq y_i^k$. The removes the bilinearity and is exactly equivalent as long as $x_{ij}$ or $y_i^k$ is integral.

Putting it all together and removing terms that are constant with respect to $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, we obtain the following linear program:

$$\begin{aligned}
\max_{\mathbf{x},\mathbf{y},\mathbf{z}} \quad & \sum_{i,j,k} w_j^k (z_{ij}^k - \hat{y}_i^k x_{ij}) + \sum_{(i,j)\in E,k} w_e^k y_{ij}^k - \sum_{i,k} y_i^k \cdot \hat{y}_i^k \\
\text{s.t.} \quad & 0 \leq x_{ij} \leq 1; \quad \sum_{i,j} x_{ij} + \hat{x}_{ij} - 2x_{ij}\hat{x}_{ij} \leq D \\
& 0 \leq y_i^k; \quad \sum_k y_i^k = 1; \quad y_{ij}^k \leq y_i^k; \quad y_{ij}^k \leq y_j^k \\
& z_{ij}^k \leq x_{ij}; \quad z_{ij}^k \leq y_i^k \quad \forall i,j,k
\end{aligned} \tag{3.2}$$

Given the model's weights, this linear program allows the adversary to change up to $D$ binary features. Recall that, in the AMN formulation, the exponential number of constraints separating the true labeling from all alternate labelings are replaced with a single non-linear constraint that separates the true labeling from the best alternate labeling (Eq. 2.1). This non-linear constraint contains a nested maximization. We have a similar scenario, but here the margin can also be altered by changing the binary features, affecting the probabilities of both the true and alternate labelings. By substituting this new MPE inference task (Eq. 3.2) into the original AMN's formulation, the resulting program's optimal solution will be robust to the worst manipulation of the input feature vector:

$$\begin{aligned}
\min_{\mathbf{w},\xi} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\xi \quad \text{s.t.} \quad \mathbf{w} \geq 0; \\
\xi - N \geq \max_{\mathbf{x},\mathbf{y},\mathbf{z}} \quad & \sum_{i,j,k} w_j^k (z_{ij}^k - \hat{y}_i^k x_{ij}) + \sum_{(i,j)\in E,k} w_e^k y_{ij}^k \\
& - \sum_{i,k} y_i^k \cdot \hat{y}_i^k \qquad \text{s.t.} \\
& 0 \leq y_i^k; \;\; \sum_k y_i^k = 1; \;\; y_{ij}^k \leq y_i^k; \;\; y_{ij}^k \leq y_j^k \\
& 0 \leq x_{ij} \leq 1; \;\; \sum_{i,j} x_{ij} + \hat{x}_{ij} - 2x_{ij}\hat{x}_{ij} \leq D \\
& z_{ij}^k \leq x_{ij}; \quad z_{ij}^k \leq y_i^k
\end{aligned} \tag{3.3}$$

The mathematical program in Eq. (3.3) is not convex because of the bilinear terms and the nested maximization (similar to solving a bilevel Stackelberg game). Fortunately, we can use the strong duality property of linear programs to resolve both of these difficulties. The dual of the maximization linear program is a minimization linear program with the same optimal value as the primal problem. Therefore, we can replace the inner maximization with its dual minimization problem to obtain a single convex quadratic program that minimizes over $\mathbf{w}$, $\xi$, and the dual variables (not shown). A similar approach is used by Globerson and Roweis [10]. As long as this relaxed program has an integral optimum, it is equivalent to maximizing only over integral $\mathbf{x}$ and $\mathbf{y}$. Thus, the overall program will find optimal weights. Taskar et al. [13] prove that the inner maximization in a 2-class AMN always has an integral solution. We can prove a similar result for the adversarial AMN:

*Theorem* 1. Eq. 3.2 has an integral optimum when $\mathbf{w} \geq 0$ and the number of classes is 2.

*Proof Sketch.* The structure of our argument is to show that an integral optimum exists by taking an arbitrary adversarial AMN problem and constructing an equivalent AMN problem that has an integral solution. Since the two problems are equivalent, the original adversarial AMN must also have an integral solution. First, we use a Lagrange multiplier to incorporate the constraint $\Delta(\mathbf{x}, \hat{\mathbf{x}}) \leq D$ directly into the maximization. The extra term acts as a "per-change" penalty, which remains linear in $\mathbf{x}$. Minimizing over the Lagrange multiplier effectively adjusts this per-change penalty until there are at most $D$ changes between $\mathbf{x}$ and $\hat{\mathbf{x}}$, but does not affect the integrality of the inner maximization. Next, we replace all $\mathbf{x}$ variables with equivalent variables $\mathbf{v}$. Assume that either $w_j^1 = 0$ or $w_j^2 = 0$, for all $j$. (If both are positive, then we can subtract the smaller value from both to obtain a new set of weights with the same optimum as before.) We define $\mathbf{v}$ as

follows:

$$v_{ij}^1 = \begin{cases} x_{ij} & \text{if } w_j^1 > 0, \\ 1 - x_{ij} & \text{if } w_j^1 = 0. \end{cases}$$
$$v_{ij}^2 = 1 - v_{ij}^1$$

By construction:

$$\sum_{i,j,k} w_j^k x_{ij}(y_i^k - \hat{y}_i^k) = \sum_{i,j,k} w_j^k v_{ij}^k (y_i^k - \hat{y}_i^k)$$

Thus, we can replace the $\mathbf{x}$ variables with $\mathbf{v}$. Since the connections between the $v_{ij}^k$ and corresponding $y_i^k$ variables are all associative, this defines an AMN over variables $\{\mathbf{y}, \mathbf{v}\}$, which is guaranteed to have an integral solution when there are only two classes.

By translating $\mathbf{v}$ back into $\mathbf{x}$, we obtain a solution that is integral in both $\mathbf{x}$ and $\mathbf{y}$. □

Many extensions of our model are possible. One extension is to restrict the adversary to only changing certain features of certain objects. For example, in a web spam domain, we might assume that the adversary will only modify spam pages. We could also have different budgets for different types of changes, such as a separate budget for each web page, or even separate budgets for changing the title of a web page and changing its body. These are easily expressed by changing the definition of $\mathcal{X}'$ and adding the appropriate constraints to the quadratic program. Our model can also support higher-order cliques, as described by Taskar et al. [13], as long as they are associative. For simplicity, our exposition and experiments focus on the simpler case described above.

One important limitation of our model is that we do not allow edges to be added or removed by the adversary. While edges can be encoded as variables in the model, they result in non-associative potentials, since the presence of an edge is not associated with either class label. Instead, the presence of an edge increases the probability that the two linked nodes will have the same label. Handling the adversarial addition and removal of edges is an important area for future work, but will almost certainly be a non-convex problem.

# Chapter 4

# Results and Conclusion

## 4.1 Experiments

In this section, we describe our experimental evaluation of CACC. Since CACC is both adversarial and relational, we compared it to four baselines: AMNs, which are relational but not adversarial; SVMInvar [11], which is adversarial but not relational; and SVMs with a linear kernel, which are neither. AMNs, SVMInvar, and SVMs can be seen as special cases of CACC: fixing the adversary's budget $D$ to zero results in an AMN, fixing the edge weights $w_e^k$ to zero results in SVMInvar, and doing both results in an SVM.

### 4.1.1 Datasets

We evaluated our method on three collective classification problems.

**Synthetic.** To evaluate the effectiveness of our method in a controlled setting where the distribution is known, we constructed a set of 10 random graphs, each with 100 nodes and 30 Boolean features. Of the 100 nodes, half had a positive label ('+') and half had a negative label ('−'). Nodes of the same class were more likely to be linked by an edge than nodes with different classes. The features were divided evenly into three types: positive, negative, and neutral. Half of the positive and negative nodes had different feature distributions based on their class; that is, the positive nodes had more positive attributes and the negative nodes had more negative attributes, on average. In such nodes, on average there are 6 words, one of which is of the opposite class's words, two words are consistent with the class label and three words are neutral. The other half of the nodes had an ambiguous distribution consisting mainly of the neutral words (on average one word is consistent with class label, one word is not consistent and 3 words are neutral). Therefore, an effective classifier for these graphs must rely on both the

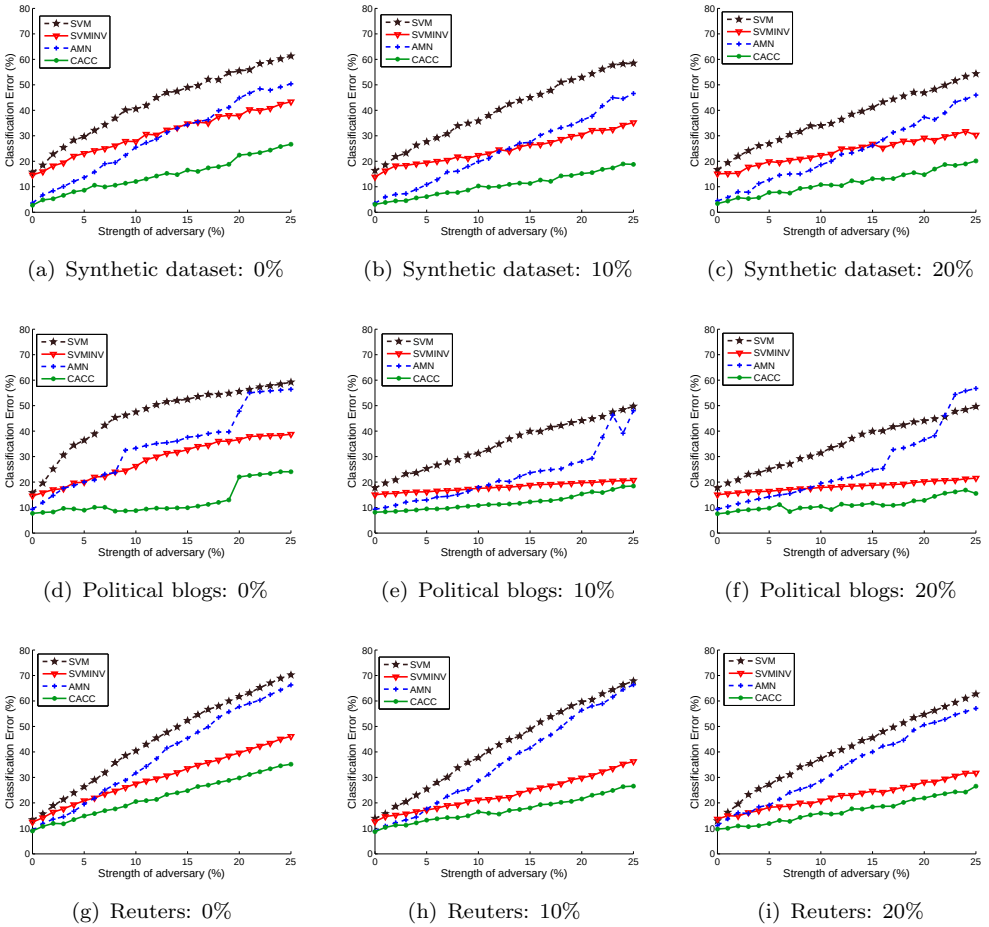
attributes and relations. On average, each node had 8 neighbors, 7 of which had the same class and 1 of which had a different class.

**Political Blogs.** Our second domain is based on the Political blogs dataset collected by Adamic and Glance [23]. The original dataset contains 1490 online blogs captured during the 2004 election cycle, their political affiliation (liberal or conservative), and their linking relationships to other blogs. We extended this dataset with word information from four different crawls at different dates in 2012: early February, late February, early May and late May. We used mutual information to select the 100 words that best predict the class label [24], only using blogs from February and half of the blogs in early May, in order to limit the influence of test labels on our training procedure. We found that some of the blogs in the original dataset were no longer active, and had been replaced by empty or spam web pages. We manually removed these from consideration. Finally, we partitioned the blogs into two disjoint subsets and removed all edges between nodes in the different subsets.

**Reuters.** As our third dataset, we prepared a Reuters dataset similar to the one used by Taskar et al. [13]. We took the ModApte split of the Reuters-21578 corpus and selected articles from four classes: crude, grain, trade, and money-fx. We used the 200 words with highest mutual information as features. We linked each document to the two most similar documents based on TF-IDF weighted cosine distance. We split the data into 7 sets based on time, and performed the tuning and then the training phases based on this temporal order (as explained in 4.1.3).

### 4.1.2 Simulating an Adversary

In real world adversarial problems, the adversary does not usually have perfect access to the model parameters. Researchers have widely studied the different ways that an adversary can acquire access to the model parameters actively or passively [7, 8]. In this paper we have examined two extreme cases. In the first, the adversary has perfect access to the model parameters and manipulates the features in order to maximize the misclassification rate. Since exactly maximizing the error rate is typically NP-hard, our intelligent adversary instead maximizes the margin loss by solving the linear program in Eq. (3.2). The simulation results of such adversarial manipulation is reported in Fig. 4.1 In the second scenario, the random adversary randomly toggles $D$ binary features, representing random noise or perhaps a very naive adversary (Fig. 4.2).




(a) Synthetic dataset: 0% (b) Synthetic dataset: 10% (c) Synthetic dataset: 20%

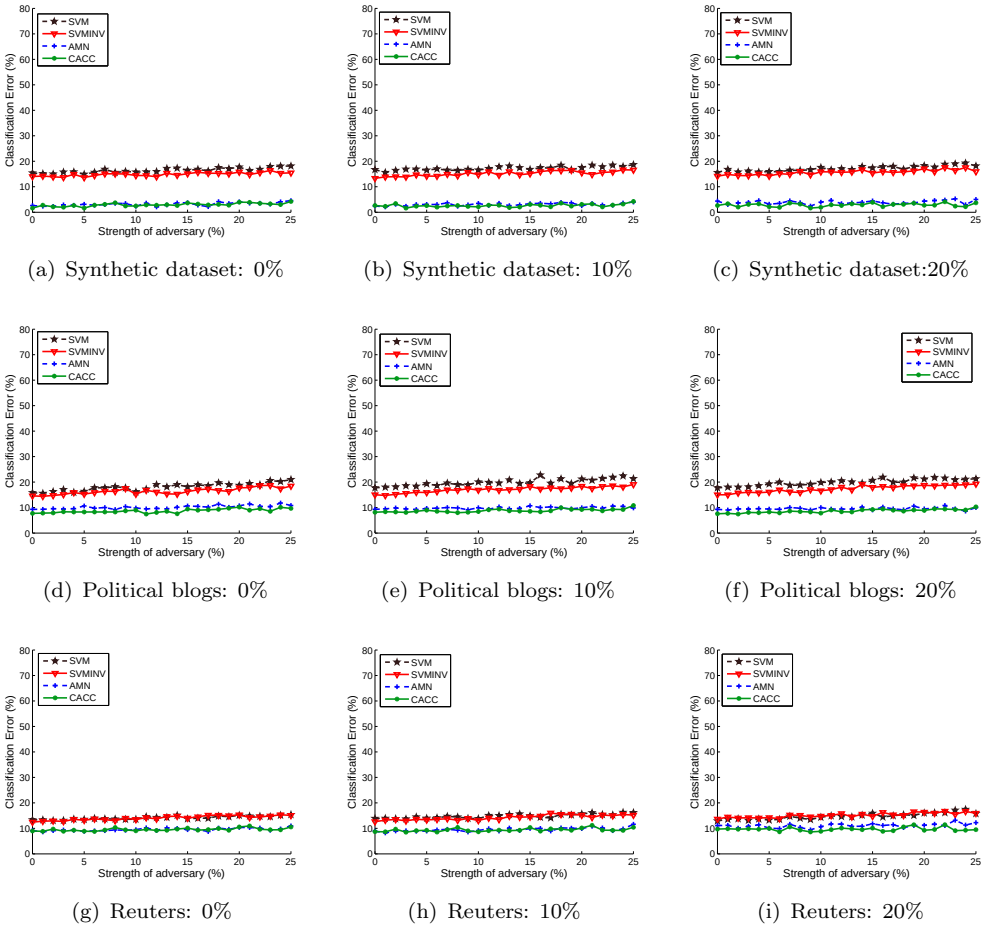
(d) Political blogs: 0% (e) Political blogs: 10% (f) Political blogs: 20%

(g) Reuters: 0% (h) Reuters: 10% (i) Reuters: 20%

FIGURE 4.1: Accuracy of different classifiers in presence of worst-case adversary. The number following the dataset name indicates the adversary's strength at the time of parameter tuning. The x-axis indicates the adversary's strength at test time. Smaller is better.

### 4.1.3 Methodology and Metrics

In order to evaluate the robustness of these methods to malicious adversaries, we applied a simulated adversary to both the tuning data and the test data. We assumed the worst-case scenario, in which the adversary has perfect knowledge of the model parameters and only wants to maximize the error rate of the classifier. Since exactly maximizing the error rate is typically NP-hard, our intelligent adversary instead maximizes the margin loss by solving the linear program in Eq. (3.2) for a fixed budget. Each model was attacked separately. On the validation data, we used adversarial budgets of 0% (no adversarial manipulation), 10%, and 20% of the total number of features present in the data. This allowed us to tune our models to "expect" adversaries of different strengths. Of course, we rarely know the exact strength of the adversary in advance. Thus, on the test data, we used budgets that ranged from 0% to 25%, in order to see how well different models did against adversaries that were weaker and stronger than expected.




(a) Synthetic dataset: 0% (b) Synthetic dataset: 10% (c) Synthetic dataset:20%

(d) Political blogs: 0% (e) Political blogs: 10% (f) Political blogs: 20%

(g) Reuters: 0% (h) Reuters: 10% (i) Reuters: 20%

FIGURE 4.2: Similar experiments as in Fig.4.1, but this time some naive adversary has randomly changed the features. All other settings are as in the paper.

We used the fraction of misclassified nodes as our primary evaluation criterion. For all methods, we tuned the regularization parameter $C$ using held-out validation data. For the adversarial methods (CACC and SVMInvar), we tuned the adversarial training budget $D$ as well. All parameters were selected to maximize performance on the tuning set with the given level of adversarial manipulation.

For political blogs, we tuned our parameters using the words from the February crawls, and then learned models on early May data and evaluated them on late May data. In this way, our tuning procedure could observe the concept drift within February and select parameters that would handle the concept drift during May well. For Synthetic data, we ran 10-fold cross validation. For Reuters, we split the data into 7 sets based on time. We tuned parameters using articles from time $t$ and $t+1$ and then learned on articles at time $t+1$ and evaluated on articles from time $t+2$.

During the tuning procedure, we performed a coarse to fine tuning; For tunung $C$, we start with the log space of $C \in \{C_1 = 1e-4, C_2 = 1e-3, ..., C_9 = 1e4\}$, then for the

best $C_i$, we expand the logspace of $e^{((logC_{i-1}+logC_i)/2)}$ to $e^{((logC_i+logC_{i+1})/2)}$; for the best $C_i$, if $i = 1$ or $i = 9$, we correspondingly set $C_i = C_i/2$ or $C_i = C_i * 2$ and continue the expansion. The tuning will be continued until $C_9 - C_1 < \epsilon$. For the robust methods, we perform a grid search on the joint space of adversarial budget at train time and $C$, where for each budget value we perform the same coarse to fine tuning.

In addition to evaluating on the unmodified test data, we also modified the test data by applying simulated adversaries of different strengths, in order to see how each model degraded under direct attack. We ran all experiments in three settings, where in each setting we used a different adversarial worst-case manipulation on the tuning data; we used 0%, 10%, and 20% powerful adversaries. These different settings show how different algorithms behave when being tuned on some data that is not original and some adversary has already manipulated them.

We used CPLEX to solve all quadratic and linear programming problems (see appendix B for a short tutorial on CPLEX). Most problems were solved in less than 1 minute on a single core.

All of our code and datasets are available upon request.

### 4.1.4 Results and Discussion

Figure 4.1 shows the performance of all four methods on test data manipulated by rational adversaries of varying strength (0%-25%), after being tuned against adversaries of different strengths (0%, 10%, and 20%). Lower is better. On the far left of each graph is performance without an adversary. To the right of each graph, the strength of the adversary increases.

When a rational adversary is present, CACC clearly and consistently outperforms all other methods. When there is no adversary, its performance is similar to a regular AMN. On political blogs, it appears to be slightly better, which may be the result of the large amount of concept drift in that dataset.

As expected, tuning against stronger adversaries (10% and 20%) makes CACC more effective against stronger adversaries at test time. Surprisingly, tuning against a stronger adversary does not significantly reduce performance against weaker adversaries: CACC remains nearly as effective against no adversary when tuned for a 20% adversary as when tuned for no adversary. Specifically, when there is no adversary at test time, the increase in error rate from training against a 20% adversary is less than 1% on Synthetic and Reuters, and on Political the error rate actually decreases slightly. Thus, this additional robustness comes at a very small cost.

In each of the experimental settings with 0%, 10%, and 20% powerful adversaries, we observe that methods mostly perform slightly better in presence of adversary at test time if they are tuned with some adversarially manipulated tuning data. In particular, CACC outperforms other methods at almost every point.

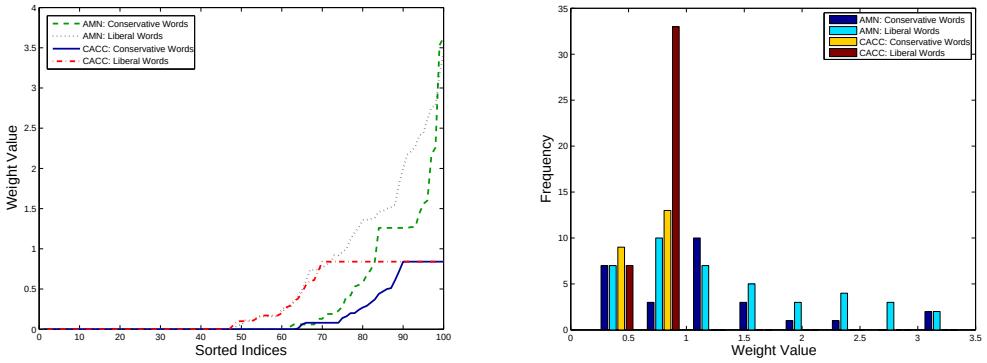
We observe that up to some certain point AMN outperforms SVMInvar, but as the adversary is allowed to change more features, it will exploit the structure of the graph to mislead the classifier even more. Therefore, the AMN performs very poorly in the presence of an active adversary, while CACC is still robust against adversaries of any strength.

In Figures 4.1(d), 4.1(e), and 4.1(f), the AMN classification error jumps sharply as the adversary budget increases. This is the point when enough nodes are mis-classified that links are actively misleading in one or two of the eight cross-validation folds, leading to worse performance than the SVM for those folds. This demonstrates that relational classifiers are potentially more vulnerable to adversarial attacks than non-relational classifiers. A smoother version of this effect can also be observed on both the synthetic dataset and Reuters.

Another interesting result was that our solutions on Reuters were always integral, even though the number of classes is 4 and integrality is not guaranteed.

We also performed additional experiments against irrational adversaries that modify attributes uniformly at random. These random attacks had little effect on the accuracy of any of the methods; all remained nearly as effective as against no adversary.

We also looked at the distribution of learned weights by naive and adversarial methods. We observe that adversarial methods distribute the weights over a small range of values and results in a smooth distribution of model weights, while the naive methods assign very high weights to some certain features. This is one of the main reasons that naive methods are vulnerable to feature addition/deletion; adding a good feature that has a high weight in a naive model, to a bad sample will increase its score, and may lead to a misclassification. In Fig. 4.3(a) , we are showing the sorted values of 100 of the weights learned for the Political blogs dataset; one can see that in a naive method like AMN, the weight values have a high variance, but in an adveresarial method like CACC, the variance of the weights is much lower. Fig. 4.3(b) show the histogram of the same weights, where one can see how the adversarial method has changed the distribution of learned weights.




(a) Sorted weights

(b) Histogram of weights

FIGURE 4.3: Distribution and variance of 100 of the weights learned for political blogs dataset (by AMN and CACC); Note that we have removed weights with zero value for better readability.

## 4.2 Conclusion

In this paper we provide a generalization of SVMInvar [11] and AMN [13] which combines the robustness of SVMInvar with the ability to reason about interrelated objects. Our approach represents the adversarial learning task as a bilevel quadratic Stackleberg optimization problem and uses the linear programming relaxation of MPE inference to convert it into a convex quadratic program with a tractable number of constraints. The number of constraints is linear in the number of nodes and edges in the graph. This resulting program can be solved in polynomial time using existing solvers. When there are only two class labels, the MPE subproblem is guaranteed to have an integral solution, and the overall learning problem is guaranteed to find optimal weights. In experiments on real and synthetic data, we find that CACC finds consistently effective and robust models, even when there are more than two labels.

In future work, we intend to extend our methods to learn adversarially regularized variants of non-associative relational models and general Markov logic networks, using approximate inference and constraint generation methods as necessary to cope with the intractability of inference. We would also like to apply our methods to larger, more realistic adversarial problems, such as web-spam. In addition to larger size, many of these problems are semi-supervised and include numeric attributes, which would require some modifications to CACC.

# Appendix A

# Proofs of theorems

*Lemma* 1. For K=2, any fixed $j$ and $0 \leq x_{ij}, y_i^k \leq 1$, $\hat{y}_i^k \in \{0,1\}$, if $A_j^k = \sum_{i=1}^N \min(x_{ij}, y_i^k) - x_{ij}\hat{y}_i^k$, then $\sum_{k=1}^K A_j^K \geq 0$.

*Proof.* $A_j^1 + A_j^2 = \sum_{i=1}^N \min(x_{ij}, y_i^1) - x_{ij}\hat{y}_i^1 + \min(x_{ij}, y_i^2) - x_{ij}\hat{y}_i^2$. Since $y_i^1 + y_i^2 = 1$ and $\hat{y}_i^1 + \hat{y}_i^2 = 1$, we can rewrite it as $\sum_{i=1}^N \min(x_{ij}, y_i^1) - x_{ij}(\hat{y}_i^1 + \hat{y}_i^2) + \min(x_{ij}, 1 - y_i^1)$ $= \sum_{i=1}^N \min(x_{ij}, y_i^1) + \min(x_{ij}, 1 - y_i^1) - x_{ij}$. Now three cases can happen:

(a) If $x_{ij} \geq \max(y_i^1, 1-y_i^1)$, then $\min(x_{ij}, y_i^1) + \min(x_{ij}, 1-y_i^1) - x_{ij} = y_i^1 + 1 - y_i^1 - x_{ij}$ $= 1 - x_{ij} \geq 0$.

(b) If $\min(y_i^1, 1 - y_i^1) \leq x_{ij} \leq \max(y_i^1, 1 - y_i^1)$, then $\min(x_{ij}, \min(y_i^1, 1 - y_i^1)) + \min(x_{ij}, \max(y_i^1, 1-y_i^1)) - x_{ij} = \min(x_{ij}, \min(y_i^1, 1-y_i^1)) + x_{ij} - x_{ij} = \min(x_{ij}, y_i^1, 1 - y_i^1) \geq 0$.

(c) If $x_{ij} \leq \min(y_i^1, 1 - y_i^1)$, then $\min(x_{ij}, y_i^1) + \min(x_{ij}, 1 - y_i^1) - x_{ij} = x_{ij} + x_{ij} - x_{ij}$ $= x_{ij} \geq 0$.

Therefore $\min(x_{ij}, y_i^1) + \min(x_{ij}, y_i^2) - x_{ij}$ is always nonnegative and consequently $A_j^1 + A_j^2 = \sum_{i=1}^N \min(x_{ij}, y_i^1) - x_{ij}\hat{y}_i^1 + \min(x_{ij}, y_i^2) - x_{ij}\hat{y}_i^2$ is always nonnegative. □

*Lemma* 2. For $K = 2$, in the optimal solution of the final quadratic program , $W^*$ satisfies the following property: $\min(w_j^1, w_j^2) = 0 \ \forall j = 1 \ldots m$.

*Proof.* Let $\theta_j = \min(w_j^1, w_j^2)$, we define $u_j^1 = w_j^1 - \theta_j$ and $u_j^2 = w_j^2 - \theta_j$, by substitution the objective of the constraint's linear program will be:

$$\sum_{i,j,k}(u_j^k+\theta_j)z_{ij}^k-(u_j^k+\theta_j)x_{ij}\hat{y}_i^k+\underbrace{\sum_{(i,j)\in E,k}w_e^k y_{ij}^k-\sum_{i,k}y_i^k\cdot\hat{y}_i^k+\sum_{i,j}\delta_{ij}(1-2\hat{x}_{ij})x_{ij}}_{B}$$

$$=\sum_j\sum_i u_j^1 z_{ij}^1-u_j^1x_{ij}\hat{y}_i^1+u_j^2z_{ij}^2-u_j^2x_{ij}\hat{y}_i^2+\theta_j(z_{ij}^1-x_{ij}\hat{y}_i^1+z_{ij}^2-x_{ij}\hat{y}_i^2)+B$$

$$=\sum_j\left[\sum_i F_{ij}+\theta_j\underbrace{\sum_i H_ij}_{\geq 0}\right]+B$$

In which $F_{ij}$ and $H_{ij}$ are:

$$\begin{aligned}F_{ij} &= u_j^1z_{ij}^1-u_j^1x_{ij}\hat{y}_i^1+u_j^2z_{ij}^2-u_j^2x_{ij}\hat{y}_i^2\\ H_{ij} &= z_{ij}^1-x_{ij}\hat{y}_i^1+z_{ij}^2-x_{ij}\hat{y}_i^2\end{aligned}$$

According to Lemma 1, $\sum_i(z_{ij}^1-x_{ij}\hat{y}_i^1+z_{ij}^2-x_{ij}\hat{y}_i^2)\geq 0$, therefore the coefficient of each $\theta_j$ is non-negative. Since $\theta_j=\min(w_j^1,w_j^1)\geq 0$, thus:

i. If optimization algorithm chooses smaller value for $\theta_j$, the relaxed inequality constraint will not be violated, and also smaller $\theta_j$ will not imply larger $\xi$.

ii. Smaller $\theta_j$ will directly reduce the objective value.

Therefore, the optimization algorithm chooses the smallest possible $\theta_j$, which is $\theta_j=0\ \forall j$. So $\min(w_j^1,w_j^2)=0$ or equivalently $w_j^1w_j^2=0\ \forall j=1\dots m$. □

*Theorem* 2. Adversary's problem in Eq. (3), has integral solution for both $\mathbf{X}$ and $Y$.

*Proof.* According to Lemma 2, we know that $\min(w_j^1,w_j^2)=0$ for all $j$. So we can rewrite Eq. (3) as:

$$\max_{\mathbf{y}\in\mathcal{Y}',0\leq\mathbf{x}\leq 1}\sum_{i,j}D_{ij}+\sum_{(i,j)\in E,k}w_e^k y_{ij}^k-\sum_{i,k}y_i^k\cdot\hat{y}_i^k+\sum_{i,j}\delta_{ij}(1-2\hat{x}_{ij})x_{ij}\qquad(A.1)$$

Where $D_{ij}=w_j^1z_{ij}^1-w_j^1x_{ij}\hat{y}_i^1+w_j^2z_{ij}^2-w_j^2x_{ij}\hat{y}_i^2$. Here we assume that one the $w_j^1$ or $w_j^2$ is not zero because this the interesting case otherwise the proof is trivial, therefore since either $w_j^1$ or $w_j^2$ is zero, we have:

$$\begin{aligned} D_{ij} &= w_j^1 \min(x_{ij}, y_i^1) - w_j^1 x_{ij} \hat{y}_i^1 + w_j^2 \min(x_{ij}, y_i^2) - w_j^2 x_{ij} \hat{y}_i^2 \\ &= I(w_j^1 = 0) \left[ w_j^1 \min(1 - x_{ij}, y_i^1) - w_j^1 (1 - x_{ij}) \hat{y}_i^1 + w_j^2 \min(x_{ij}, y_i^2) - w_j^2 x_{ij} \hat{y}_i^2 \right] + \\ & \quad I(w_j^2 = 0) \left[ w_j^1 \min(x_{ij}, y_i^1) - w_j^1 x_{ij} \hat{y}_i^1 + w_j^2 \min(1 - x_{ij}, y_i^2) - w_j^2 (1 - x_{ij}) \hat{y}_i^2 \right] \end{aligned}$$

Let $v_{ij}^k = x_{ij} I(w_j^k > 0) + (1 - x_{ij}) I(w_j^k = 0)$, where $I(.)$ is the indicator function, then:

$$\begin{aligned} D_{ij} &= I(w_j^1 = 0) \left[ w_j^1 \min(v_{ij}^1, y_i^1) - w_j^1 v_{ij}^1 \hat{y}_i^1 + w_j^2 \min(v_{ij}^2, y_i^2) - w_j^2 v_{ij}^2 \hat{y}_i^2 \right] + \\ & \quad I(w_j^2 = 0) \left[ w_j^1 \min(v_{ij}^1, y_i^1) - w_j^1 v_{ij}^1 \hat{y}_i^1 + w_j^2 \min(v_{ij}^2, y_i^2) - w_j^2 v_{ij}^2 \hat{y}_i^2 \right] \\ &= \left( I(w_j^1 = 0) + I(w_j^2 = 0) \right) \left[ w_j^1 \min(v_{ij}^1, y_i^1) - w_j^1 v_{ij}^1 \hat{y}_i^1 + w_j^2 \min(v_{ij}^2, y_i^2) - w_j^2 v_{ij}^2 \hat{y}_i^2 \right] \\ &= w_j^1 \min(v_{ij}^1, y_i^1) - w_j^1 v_{ij}^1 \hat{y}_i^1 + w_j^2 \min(v_{ij}^2, y_i^2) - w_j^2 v_{ij}^2 \hat{y}_i^2 \end{aligned} \tag{A.2}$$

Clearly, we $v_{ij}^1 + v_{ij}^2 = 1$, because:

$$\begin{aligned} v_{ij}^1 + v_{ij}^2 &= x_{ij} I(w_j^1 > 0) + (1 - x_{ij}) I(w_j^1 = 0) + x_{ij} I(w_j^2 > 0) + (1 - x_{ij}) I(w_j^2 = 0) \\ &= x_{ij} \underbrace{\left[ I(w_j^1 > 0) + I(w_j^2 > 0) \right]}_{=1} + (1 - x_{ij}) \underbrace{\left[ I(w_j^1 = 0) + I(w_j^2 = 0) \right]}_{=1} \\ &= x_{ij} + 1 - x_{ij} = 1. \end{aligned}$$

Obviously, as a result we will have $z_{ij}^k = \min(v_{ij}^k, y_i^k)$, because otherwise increasing $z_{ij}^k$ can increase the objective, so the solver program will choose the maximum possible value for $z_{ij}^k$. By lemma 3, and reformulation of suggested $D_{ij}$ in Eq. (A.2), we conclude that Eq. (A.1) has integral solution for $y_i^k$ and $v_{ij}^k$ for all $i, j$ and $k = 1, 2$. Since inetgrality of $v_{ij}^k$ implies integrality of $x_{ij}$, proof is complete. □

*Lemma* 3. If K=2, for any $W = [W^1, W^2]$, $W^k = [w_1^k, \ldots, w_m^k]^T$, linear program in Eq. (A.1), has an integral solution.

*Proof.* Here, our argument is similar to the proof of the theorem 3.1 of [13]. We show that for any fractional solution $\mathbf{X}$ (and respectively $\mathbf{V}$) and $Y$ of Eq. (A.1), we can construct a new feasible integral assignment $\mathbf{X}'$ and $Y'$, that increases the objective or does not change it.

Since all $w_e^k$'s and $w_j^k$'s are positive, therefore, $y_{ij}^k = \min(y_i^k, y_j^k)$ and $z_{ij}^k = \min(y_i^k, x_{ij})$; this means that the slack variables corresponding to $z_{ij}^k \leq y_i^k$,$z_{ij}^k \leq x_{ij}$ and $y_{ij}^k \leq y_i^k$,$y_{ij}^k \leq y_j^k$ are zero, because otherwise by increasing $y_{ij}^k$ or $z_{ij}^k$, the objective could be increased.

Let $\lambda^k = \min(\min_{i,y_i^k>0} y_i^k, \min_{ij,v_{ij}^k>0} v_{ij}^k)$ and $\lambda = \lambda^1$ or $\lambda = -\lambda^2$. We propose a new construction of solution, that either increases the objective or does not change it, and at the same time reduces the number of fractional values in the solution.

$$\begin{aligned}
v_{ij}^{'1} &= v_{ij}^1 - \lambda I(0 < v_{ij}^1 < 1),\ v_{ij}^{'2} = v_{ij}^2 + \lambda I(0 < v_{ij}^2 < 1) \\
z_{ij}^{'1} &= z_{ij}^1 - \lambda I(0 < z_{ij}^1 < 1),\ z_{ij}^{'2} = z_{ij}^2 + \lambda I(0 < z_{ij}^2 < 1) \\
y_i^{'1} &= y_i^1 - \lambda I(0 < y_i^1 < 1),\ \ y_i^{'2} = y_i^2 + \lambda I(0 < y_i^2 < 1) \\
y_{ij}^{'1} &= y_{ij}^1 - \lambda I(0 < y_{ij}^1 < 1),\ y_{ij}^{'2} = y_{ij}^2 + \lambda I(< 0 y_{ij}^2 < 1)
\end{aligned}$$

It is obvious that by this update, at least two of the fractional values become integral. First, we show that in this new construction, values remain feasible. So we need to show that $v_{ij}^{'1} + v_{ij}^{'2} = 1$,$y_i^{'1} + y_i^{'2} = 1$, $v_{ij}^{'k} \geq 0$, $y_i^{'k} \geq 0$, $y_{ij}^{'k} = \min(y_i^{'k}, y_j^{'k})$ and $z_{ij}^{'k} = \min(v_{ij}^{'k}, y_i^{'k})$. In the following we show that all of the feasibility requirements are satisfied.

$$v_{ij}^{'1} + v_{ij}^{'2} = v_{ij}^1 - \lambda I(0 < v_{ij}^1 < 1) + v_{ij}^2 + \lambda I(0 < v_{ij}^2 < 1 = v_{ij}^1 + v_{ij}^2 = 1.$$

$$y_i^{'1} + y_i^{'2} = y_i^1 - \lambda I(0 < y_i^1 < 1) + y_i^2 + \lambda I(0 < y_i^2 < 1) = y_i^1 + y_i^2 = 1.$$

Above we used the fact that if $v_{ij}^1$ is fractional then $v_{ij}^2$ will also be fractional, and similarly if $y_i^1$ is fractional then $y_i^2$ will also be fractional, since $v_{ij}^1 + v_{ij}^2 = 1$ and $y_i^1 + y_i^2 = 1$. To show $v_{ij}^{'k} \geq 0$ and $y_i^{'k} \geq 0$, we prove that $\min_{ij}\ v_{ij}^{'k} \geq 0$ and $\min_i\ y_i^{'k} \geq 0$.

$$\begin{aligned}
\min_{ij}\ v_{ij}^{'k} &= \min_{ij}(v_{ij}^k - (\min(\min_{i,y_i^k>0} y_i^k, \min_{ij,v_{ij}^k>0} v_{ij}^k))I(0 < v_{ij}^k < 1)) \\
&= \min\left(\min_{ij}\ v_{ij}^k, \min_{ij}\left[v_{ij}^k - (\min(\min_{i,y_i^k>0} y_i^k, \min_{ij,v_{ij}^k>0} v_{ij}^k))\right]\right) \\
&\geq \min\left(\min_{ij}\ v_{ij}^k, \min_{ij}\left[v_{ij}^k - (\min_{ij,v_{ij}^k>0} v_{ij}^k)\right]\right) \\
&\geq \min_{ij}\left[v_{ij}^k - (\min_{ij,v_{ij}^k>0} v_{ij}^k)\right] = 0.
\end{aligned}$$

$$\begin{aligned}
\min_i \ y_i^{'k} &= \min_i(y_i^k - (\min(\min_{i,y_i^k>0} y_i^k, \min_{ij,v_{ij}^k>0} v_{ij}^k))I(0<y_i^k<1)) \\
&= \min\left(\min_i \ y_i^k, \min_i\left[y_i^k - (\min(\min_{i,y_i^k>0} y_i^k, \min_{ij,v_{ij}^k>0} v_{ij}^k))\right]\right) \\
&\geq \min\left(\min_i \ y_i^k, \min_i\left[y_i^k - (\min_{i,y_i^k>0} y_i^k)\right]\right) \\
&\geq \min_i\left[y_i^k - (\min_{i,y_i^k>0} y_i^k)\right] = 0.
\end{aligned}$$

The last step in showing that the proposed construction is feasible is showing that $y_{ij}^{'k} = \min(y_i^{'k}, y_j^{'k})$ and $z_{ij}^{'k} = \min(v_{ij}^{'k}, y_i^{'k})$.

$$\begin{aligned}
y_{ij}^{'1} &= y_{ij}^1 - \lambda I(0<y_{ij}^1<1) \\
&= \min(y_i^1, y_j^1) - \lambda I(0<\min(y_i^1, y_j^1)<1) \\
&= \min(y_i^1 - \lambda I(0<y_i^1<1), y_j^1 - \lambda I(0<y_j^1<1)) \\
&= \min(y_i^{'1}, y_j^{'1}).
\end{aligned}$$

$$\begin{aligned}
y_{ij}^{'2} &= y_{ij}^2 + \lambda I(0<y_{ij}^2<1) \\
&= \min(y_i^2, y_j^2) + \lambda I(0<\min(y_i^2, y_j^2)<1) \\
&= \min(y_i^2 + \lambda I(0<y_i^2<1), y_j^2 + \lambda I(0<y_j^2<1)) \\
&= \min(y_i^{'2}, y_j^{'2}).
\end{aligned}$$

$$\begin{aligned}
z_{ij}^{'1} &= z_{ij}^1 - \lambda I(0<z_{ij}^1<1) \\
&= \min(v_{ij}^1, y_i^1) - \lambda I(0<\min(v_{ij}^1, y_i^1)<1) \\
&= \min(v_{ij}^1 - \lambda I(0<v_{ij}^1<1), y_i^1 - \lambda I(0<y_i^1<1)) \\
&= \min(v_{ij}^{'1}, y_i^{'1}).
\end{aligned}$$

$$\begin{aligned} z_{ij}^{'2} &= z_{ij}^2 + \lambda I(0 < z_{ij}^2 < 1) \\ &= \min(v_{ij}^2, y_i^2) + \lambda I(0 < \min(v_{ij}^2, y_i^2) < 1) \\ &= \min(v_{ij}^2 + \lambda I(0 < v_{ij}^2 < 1), y_i^2 + \lambda I(0 < y_i^2 < 1)) \\ &= \min(v_{ij}^{'2}, y_i^{'2}). \end{aligned}$$

So far we have shown that the new variable construction is feasible, and it remains to show that we can increase the objective. We substitute the newly constructed feasible values in Eq. (A.1) and subtract the objective with unchanged values from it. Then we show that with proper choice of $\lambda = \lambda^1$ or of $\lambda = -\lambda^2$, we can improve the objective.

$$\begin{aligned} V_{old} &= \sum_{i,j} D_{ij} + \sum_{(i,j)\in E,k} w_e^k y_{ij}^k - \sum_{i,k} y_i^k \cdot \hat{y}_i^k + \sum_{i,j} \delta_{ij}(1 - 2\hat{x}_{ij}) x_{ij} \\ &= \sum_{i,j} w_j^1 z_{ij}^1 - w_j^1 v_{ij}^1 \hat{y}_i^1 + w_j^2 z_{ij}^2 - w_j^2 v_{ij}^2 \hat{y}_i^2 \\ &\quad + \sum_{(i,j)\in E,k} w_e^k y_{ij}^k - \sum_{i,k} y_i^k \cdot \hat{y}_i^k + \sum_{i,j} \delta_{ij}(1 - 2\hat{x}_{ij}) x_{ij} \\ &= \sum_{i,j} w_j^1 z_{ij}^1 - w_j^1 v_{ij}^1 \hat{y}_i^1 + w_j^2 z_{ij}^2 - w_j^2 v_{ij}^2 \hat{y}_i^2 \\ &\quad + \sum_{(i,j)\in E,k} w_e^k y_{ij}^k - \sum_{i,k} y_i^k \cdot \hat{y}_i^k \\ &\quad + \sum_{i,j} \delta_{ij}(1 - 2\hat{x}_{ij}) \left[ \left( I(w_j^1 > 0) - I(w_j^1 = 0) \right) v_{ij}^1 + I(w_j^1 = 0) \right] \\ &= \sum_{i,j} w_j^1 z_{ij}^1 - w_j^1 v_{ij}^1 \hat{y}_i^1 + w_j^2 z_{ij}^2 - w_j^2 v_{ij}^2 \hat{y}_i^2 \\ &\quad + \sum_{(i,j)\in E,k} w_e^k y_{ij}^k - \sum_{i,k} y_i^k \cdot \hat{y}_i^k \\ &\quad + \sum_{i,j} \left[ \delta_{ij}(1 - 2\hat{x}_{ij}) \left( I(w_j^1 > 0) - I(w_j^1 = 0) \right) \right] v_{ij}^1 + C. \end{aligned}$$

Above we have used the fact that $x_{ij} = I(w_j^{'k} > 0) v_{ij}^{'k} + I(w_j^{'k} = 0)(1 - v_{ij}^{'k}) = I(w_j^{'1} > 0) v_{ij}^{'1} + I(w_j^{'1} = 0)(1 - v_{ij}^{'1}) = \left( I(w_j^{'1} > 0) - I(w_j^{'1} = 0) \right) v_{ij}^{'1} + I(w_j^{'1} = 0)$.

$$
\begin{aligned}
V_{new} &= \sum_{i,j} w_j^1 z_{ij}^{'1} - w_j^1 v_{ij}^{'1} \hat{y}_i^1 + w_j^2 z_{ij}^{'2} - w_j^2 v_{ij}^{'2} \hat{y}_i^2 \\
&\quad + \sum_{(i,j)\in E,k} w_e^k y_{ij}^{'k} - \sum_{i,k} y_i^{'k} \cdot \hat{y}_i^k \\
&\quad + \sum_{i,j} \left[\delta_{ij}(1-2\hat{x}_{ij})\left(I(w_j^1>0) - I(w_j^1=0)\right)\right] v_{ij}^{'1} + C \\
&= \sum_{i,j} [w_j^1(z_{ij}^1 - \lambda I(0<z_{ij}^1<1)) - w_j^1 \hat{y}_i^1 (v_{ij}^1 - \lambda I(0<v_{ij}^1<1)) \\
&\quad + w_j^2(z_{ij}^2 + \lambda I(0<z_{ij}^2<1)) - w_j^2 \hat{y}_i^2 (v_{ij}^2 + \lambda I(0<v_{ij}^2<1))] \\
&\quad + \sum_{(i,j)\in E} \left[w_e^1(y_{ij}^1 - \lambda I(0<y_{ij}^1<1)) + w_e^2(y_{ij}^2 + \lambda I(0<y_{ij}^2<1))\right] \\
&\quad - \sum_i \hat{y}_i^1 \cdot (y_i^1 - \lambda I(0<y_i^1<1)) + \hat{y}_i^2 \cdot (y_i^2 + \lambda I(0<y_i^2<1)) \\
&\quad + \sum_{i,j} \left[\delta_{ij}(1-2\hat{x}_{ij})\left(I(w_j^1>0) - I(w_j^1=0)\right)\right](v_{ij}^1 - \lambda I(0<v_{ij}^1<1)) + C \\
&= V_{old} + \sum_{i,j} [w_j^1(-\lambda I(0<z_{ij}^1<1)) - w_j^1 \hat{y}_i^1 (-\lambda I(0<v_{ij}^1<1)) \\
&\quad + w_j^2(\lambda I(0<z_{ij}^2<1)) - w_j^2 \hat{y}_i^2 (\lambda I(0<v_{ij}^2<1))] \\
&\quad + \sum_{(i,j)\in E} \left[w_e^1(-\lambda I(0<y_{ij}^1<1)) + w_e^2(\lambda I(0<y_{ij}^2<1))\right] \\
&\quad - \sum_i \hat{y}_i^1 \cdot (-\lambda I(0<y_i^1<1)) + \hat{y}_i^2 \cdot (+\lambda I(0<y_i^2<1)) \\
&\quad + \sum_{i,j} \left[\delta_{ij}(1-2\hat{x}_{ij})\left(I(w_j^1>0) - I(w_j^1=0)\right)\right](-\lambda I(0<v_{ij}^1<1)).
\end{aligned}
$$

Therefore, we can write $V_{new} - V_{old}$ as:

$$
\begin{aligned}
V_{new} - V_{old} &= \lambda[\sum_{i,j} [-w_j^1 I(0<z_{ij}^1<1) + w_j^1 \hat{y}_i^1 I(0<v_{ij}^1<1) \\
&\quad + w_j^2 I(0<z_{ij}^2<1) - w_j^2 \hat{y}_i^2 I(0<v_{ij}^2<1)] \\
&\quad + \sum_{(i,j)\in E} \left[-w_e^1 I(0<y_{ij}^1<1) + w_e^2 I(0<y_{ij}^2<1)\right] \\
&\quad - \sum_i \hat{y}_i^1 \cdot (-I(0<y_i^1<1)) + \hat{y}_i^2 \cdot (+I(0<y_i^2<1)) \\
&\quad + \sum_{i,j} -\delta_{ij}(1-2\hat{x}_{ij})\left(I(w_j^1>0) - I(w_j^1=0)\right) I(0<v_{ij}^1<1)] \\
&= \lambda D.
\end{aligned}
$$

The change in objective is $\lambda D$, and since $D$ is constant with respect to $\lambda$, by choosing $\lambda = -\lambda^2$ for negative $D$, or $\lambda = \lambda^1$ for positive $D$, we can always have positive or zero

$\lambda D$.It means that the integral solution will increase the objective or will not change it, while leaving fewer fractional values.

□

# Appendix B

# Quick CPLEX Tutorial

This appendix is partially (including the example) taken from `http://pic.dhe.ibm.com/infocenter/cplexzos/v12r4/index.jsp?topic=%2Fcom.ibm.cplex.zos.help%2FGettingStar` `2Ftopics%2Ftutorials%2Fataglance%2FInteractive_Optimizer.html`.

## B.1 Linear and Quadratic programming

ILOG CPLEX is a tool for solving linear/quadratic optimization problems, commonly referred to as Linear/Quadratic Programming (LP/QP) problems, of the form:

Maximize (or Minimize) $\sum_{i,j} c_{ij} * x_i * x_j + \sum_i d_i * x_i$
subject to
$a_{11}x_1 + a_{12}12x_2 + \ldots + a_{1n}x_n \sim b_1$
$a_{21}x_1 + a_{22}12x_2 + \ldots + a_{2n}x_n \sim b_2$
$\ldots$
$a_{m1}x_1 + a_{m2}12x_2 + \ldots + a_{mn}x_n \sim b_m$

with these bounds

$l_1 \leq x_1 \leq u_1$
$\ldots$
$l_n \leq x_n \leq u_n$

where $\sim$ can be $\leq$ , $\geq$ , or $=$, and the upper bounds $u_i$ and lower bounds $l_i$ may be any real number.

Given the following coefficients: Objective function coefficients $c_{1,1}, c_{1,2}, \ldots, c_{n,n}, d_1, \ldots, d_n$ Constraint coefficients:$a_{11}, a_{12}, \ldots, a_{1n}, \ldots, a_{m1}, a_{m2}, \ldots, a_{mn}$

Right-hand sides $b_1, b_2, \ldots, b_m$ Upper and lower bounds $u_1, u_2, \ldots, u_n$ and $l_1, l_2, \ldots, l_n$ , the optimal solution that ILOG CPLEX computes and returns is: Variables $x_1, x_2, \ldots, x_n$ ILOG CPLEX also can solve several extensions to LP:

1. Network flow problem, which can be formulated as linear program; CPLEX exploits the problem structure and solves the network flow problems faster..
2. CPLEX can solve mixed Integer Programming (MIP) problems, where integrality constraint is imposed on some of the variables.

## B.2 ILOG CPLEX Components

CPLEX can be used in three ways:

1. The CPLEX Interactive Optimizer is an executable program that can read a problem interactively or from files in certain standard syntax, solve the problem, and deliver the solution interactively or into XML files. The program file is cplex.exe for Windows and cplex for UNIX platforms.
2. Concert Technology is a set of C++, Java, and .NET class libraries offering an API that includes modeling facilities to allow the programmer to embed CPLEX optimizers in C++, Java, or .NET applications.
3. The CPLEX Callable Library is a C library that allows the programmer to embed ILOG CPLEX optimizer in applications written in C, Visual Basic, FORTRAN, or any other language that can call C functions. The library is provided in files cplex.lib and cplex.dll on Windows platforms, and in libcplex.a, libcplex.so, and libcplex.sl on UNIX platforms.

## B.3 Solving an LP with ILOG CPLEX

In the following example, I explain the syntax of cplex. The problem to be solved is:

Maximize $x_1 + 2x_2 + 3x_3$
subject to
$-x_1 + x_2 + x_3 \leq 20$
$x_1 - 3x_2 + x_3 \leq 30$
with these bounds
$0 \leq x_1 \leq 40$
$0 \leq x_2$
$0 \leq x_3$

Using the Interactive Optimizer The following sample is screen output from a CPLEX Interactive Optimizer session where the model of an example is entered and solved. CPLEX> indicates the CPLEX prompt, and text following this prompt is user input.

```
Welcome to CPLEX Interactive Optimizer 9.0.0
with Simplex, Mixed Integer & Barrier Optimizers
Copyright (c) ILOG 1997-2003
CPLEX is a registered trademark of ILOG
Type 'help' for a list of available commands.
Type 'help' followed by a command name for more
information on commands.
CPLEX> enter example
Enter new problem ['end' on a separate line terminates]:
maximize  x1 + 2 x2 + 3 x3
subject to
-x1 +  x2 + x3 <= 20
x1 - 3 x2 + x3 <=30
bounds
0 <= x1 <= 40
0 <= x2
0 <= x3
end
CPLEX> optimize
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time =    0.00 sec.
Iteration log . . .
Iteration:     1   Dual infeasibility =             0.000000
Iteration:     2   Dual objective     =           202.500000
```

```
Dual simplex - Optimal:  Objective =   2.0250000000e+002
Solution time =    0.01 sec.  Iterations = 2 (1)
CPLEX> display solution variables x1-x3
Variable Name           Solution Value
x1                           40.000000
x2                           17.500000
x3                           42.500000
CPLEX> quit
```

You can also enter the program by first writing it into a text file with ".lp" extension, and then loading into the CPLEX environment by using the 'read' command:

```
CPLEX> read filename.lp
```

## B.4 Using CPLEX in batch mode

In this project, I have used the interactive optimizer in batch mode; I write the whole optimization program in the cplex syntax into a file, and generate a command file. Then use the "cat" command ("type" in windows) to pass the commands to cplex on aciss cluster machines (or cplex.exe on a windows machine) to solve the program and save the results. The command file (say "program.cmd" for solving the problem which is written in "program.lp") should have the following lines:

```
read program.lp
opt
write solution.xml sol
y
quit
```

which reads the problem ("read" command), optimizes the objective ("opt" command), writes the solution into an XML file, confirms overwriting if needed (y for yes), and quits cplex. In a unix machine the following command is executed by a system call:

```
system("cat program.cmd  |  cplex ")
```

After cplex solves the problem, it generates an XML solution file. The main program then will access the solution by parsing the XML file.

# Bibliography


[1] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008. ISSN 0738-4602.

[2] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI.* Morgan & Claypool, San Rafael, CA, 2009.

[3] J. Abernethy, O. Chapelle, and C. Castillo. Graph regularization methods for web spam detection. *Machine Learning*, 81(2):207–225, 2010.

[4] I. Drost and T. Scheffer. Thwarting the nigritude ultramarine: Learning to identify link spam. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 96–107. Springer, 2005.

[5] D. Chau, S. Pandit, and C. Faloutsos. Detecting fraudulent personalities in networks of online auctioneers. *Knowledge Discovery in Databases: PKDD 2006*, pages 103–114, 2006.

[6] N. Dalvi, P. Domingos, M., S. Sanghai, and D. Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 99–108, Seattle, WA, 2004. ACM Press.

[7] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of the Second Conference on Email and Anti-Spam (CEAS)*, pages 125–132, 2005.

[8] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 641–647. ACM, 2005. ISBN 159593135X.

[9] B. Nelson, B.I.P. Rubinstein, L. Huang, A.D. Joseph, S. Lau, S.J. Lee, S. Rao, A. Tran, and JD Tygar. Near-optimal evasion of convex-inducing classifiers. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010*, volume 9, Chia Laguna Resort, Sardinia, Italy, 2010.

[10] A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, pages 353–360, Pittsburgh, PA, 2006. ACM Press.

[11] C.H. Teo, A. Globerson, S. Roweis, and A. Smola. Convex learning with invariances. In *Advances in Neural Information Processing Systems 21*, 2008.

[12] M. Brückner and T. Scheffer. Nash equilibria of static prediction games. In *Advances in Neural Information Processing Systems 22*, 2009.

[13] B. Taskar, V. Chatalbashev, and D. Koller. Learning associative Markov networks. In *Proceedings of the twenty-first international conference on machine learning*. ACM Press, 2004.

[14] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2): 147–159, 2004.

[15] D. Lowd and P. Domingos. Efficient weight learning for Markov logic networks. In *Proceedings of the Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 200–211, Warsaw, Poland, 2007. Springer.

[16] B. Taskar, M. F. Wong, P. Abbeel, and D. Koller. Max-margin Markov networks. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[17] T.N. Huynh and R.J. Mooney. Max-margin weight learning for Markov logic networks. In *In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-09). Bled*, pages 564–579. Springer, 2009.

[18] P. Laskov and R. Lippmann. Machine learning in adversarial environments. *Machine learning*, 81(2):115–119, 2010.

[19] M. Brückner and T. Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the Seventeenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2011.

[20] G.R.G. Lanckriet, N. Cristianini, P. Bartlett, L.E. Ghaoui, and M.I. Jordan. Learning the kernel matrix with semidefinite programming. *The Journal of Machine Learning Research*, 5:27–72, 2004.

[21] L. El Ghaoui, G.R.G. Lanckriet, and G. Natsoulis. *Robust classification with interval data*. Computer Science Division, University of California, 2003.

[22] S.J. Kim, A. Magnani, and S. Boyd. Robust fisher discriminant analysis. *Advances in Neural Information Processing Systems*, 18:659, 2006.

[23] L.A. Adamic and N. Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005.

[24] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005.