Classification of Malicious Network Activity

W. Nicholas Greene
Nathan Newsom

Due 12/16/2011

1 Introduction

As more and more vital services today (e.g. email, Facebook, quantitative trading) depend on machine learning algorithms, there is a greater incentive than ever for adversaries to manipulate these algorithms for malicious ends (e.g. spam, identity theft, cyberattacks). The field of adversarial learning has arisen out of a need to design learning algorithms that are robust to these sort of disruptive manipulations.

Spam filtering has often been cited as the classic adversarial machine learning problem [5]. As filtering methods grow more accurate, spammers employ more and more sophisticated methods to work around them (e.g. deliberate misspellings of keywords, etc.), resulting in an ongoing "arms race". A more grave example, however, that illustrates the need for robust learning methods is the network intrusion problem, where potentially sensitive, confidential, or damaging information is compromised through sophisticated cyberattacks on computer networks. Cybercrime and cyberwarfare are real problems [2, 6] that advances in adversarial machine learning may mitigate.

2 Project Outline

Our project made use of an online-to-batch (O2B) method based on the perceptron algorithm from [3] to train a linear classifier to identify malicious network activity under adversarial conditions, including feature deletion and feature corruption of the data. We tested the algorithm against standard, non-adversarial classifiers using a subset of the KDD Cup 1999 data set, available through the UCI Machine Learning Repository [4]. The data set was generated by MIT Lincoln Laboratory for the 1998 DARPA Intrusion Detection Evaluation Program and subsequently used for the 1999 KDD Cup competition. The full dataset consists of nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN subject to several types of attacks including buffer overflows and IP sweeps. A total of 41 features (connection duration, protocol, number of transmitted bytes, etc.) describe the roughly 5 million TCP connections.

3 Adversarial Models

Following [3], we must impose some limits on our adversary to make the classification problem tractable. While it is assumed that our adversary has full knowledge of our classification scheme and learned classifier, we limit her influence to the testing phase, since it will be assumed that a relatively clean training set can be obtained (although we may model the adversary's actions in the training phase). We allow the adversary to perform two kinds of manipulations: (1) the adversary can delete features entirely (i.e. replace them with 0) or (2) the adversary can replace feature data with white-Gaussian noise of the same mean and variance of the original feature.

In addition we choose a noise tolerance parameter N and assign each feature j a cost v_j (we will refer to v as the cost vector). We then allow our adversary to corrupt any subset of the features such that $\sum_{j \notin J} v_j \leq N$ where J is the set of features that are uncorrupted. Our adversary will choose to manipulate a subset of features that causes the most damage to our prediction, subject to this noise constraint.

Concretely given a test example $(\mathbf{x}, y) \in \mathbb{R}^n \times \{-1, +1\}$ and the parameters to the learned, linear decision boundary $(\mathbf{w}, b)$, our adversary will first rank each feature according to the following metric: yw_jx_j/v_j. It will then greedily manipulate features (deletion or noise replacement) as long as $yw_jx_j > 0$ (i.e. the feature is informative) and $\sum_{j \notin J} v_j \leq N$. The ranking metric balances the damage the adversary can inflict (the yw_jx_j term) with how costly that feature is to manipulate (the v_j term).

We choose two such cost vectors for our tests, normalizing both so that $\sum_{j=1}^{n} v_j = n$. First we weight the features uniformly such that $v_i = v_j = 1$. In this case the adversary simply chooses to corrupt the N features that will cause the most damage. In the other case we choose v_j to correspond to the mutual information of the optimal one-feature linear classifier using the following formula:

$$v_j = \frac{1}{Z} \max_{c \in \mathbb{R}} I(\mathbf{1}\{X_j > c\}; Y)$$

where (X_j, Y) are random variables jointly distributed according to the uniform distribution over the training data and Z is our normalizing constant.

4 O2B Perceptron Algorithm

As its name implies, the O2B perceptron algorithm from [3] converts an online perceptron - modified for adversarial robustness - to a batch learning algorithm using a simple averaging procedure. In the online training phase, corrupted training examples are iteratively passed to the algorithm to be classified using the current parameter settings. If the algorithm classifies poorly, the parameters are updated. In addition, the parameter settings at each iteration are saved. After the classifier has seen all the training data, these saved parameter settings from each iteration are averaged to give the final parameter settings.

More formally, given a training dataset $\mathcal{S} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$, we initialize our parameters $\mathbf{w}$ and b to zeros. At the i-th iteration, we pass example $(\mathbf{x}^{(i)}, y^{(i)})$ to the algorithm to classify using the current parameters $\mathbf{w}^{(i)}$ and $b^{(i)}$. Our adversary allows a set J_i of features (via one of our models) to pass through to the classifier intact, which results in the following prediction of $y^{(i)}$:

$$h(\mathbf{x}^{(i)}) = \text{sign}\left(b^{(i)} + \sum_{j \in J_i} w_j^{(i)} x_j^{(i)}\right).$$

This hypothesis is compared to the true $y^{(i)}$, producing an empirical hinge-loss:

$$\hat{\xi}(\mathbf{w}^{(i)}, b^{(i)}; \mathbf{x}^{(i)}, y^{(i)}) = \left[\frac{V(J)}{P} - y^{(i)} h(\mathbf{x}^{(i)})\right]_+,$$

where $[\alpha]_+ = \max\{\alpha, 0\}$.

If $\hat{\xi}(\mathbf{w}^{(i)}, b^{(i)}; \mathbf{x}^{(i)}, y^{(i)}) = 0$, then the algorithm has classified $(\mathbf{x}^{(i)}, y^{(i)})$ correctly and we set

$$\begin{aligned} \mathbf{w}^{(i+1)} &\leftarrow \mathbf{w}^{(i)} \\ b^{(i+1)} &\leftarrow b^{(i)}. \end{aligned}$$

Otherwise, we perform the following update:

$$\begin{aligned} w_j^{(i+1)} &\leftarrow \begin{cases} \left[w_j^{(i)} + y^{(i)} \tau x_j^{(i)}\right]_{\pm C} & j \in J_i \\ w_j^{(i)} & \text{o.w.} \end{cases} \\ b^{(i+1)} &\leftarrow \left[b^{(i)} + y^{(i)} \tau\right]_{\pm C}, \end{aligned}$$

where $\tau = C\sqrt{n + 1/2m}$ and $[\alpha]_{\pm C} = \max\{\min\{\alpha, C\}, -C\}$. This update rule is a modified version of the standard perceptron update rule, where we factor in knowledge of the adverary's manipulations, learn at a rate τ, and constrain our parameters to the hypercube of radius C.

After the m training examples have been passed to the algorithm we compute our final parameters $\hat{\mathbf{w}}$ and $\hat{b}$ by averaging the previous m parameter settings:

$$\hat{\mathbf{w}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{w}^{(i)}$$
$$\hat{b} = \frac{1}{m}\sum_{i=1}^{m}b^{(i)}.$$

Note that aforementioned C is an l_∞ regularization parameter and effectively keeps $\mathbf{w}$ and b "small" and "dense" in order to build robustness into the classifier. Other regularization schemes typically use l_1 or l_2 norms, which can promote sparse solutions - the opposite of what is needed for this problem. Sparse parameters suggest that the algorithm has singled out a few powerful features that dominate the classification decision, which makes them susceptible to corruption (i.e. the adversary can cause more damage by manipulating fewer features). With dense parameters, the algorithm has essentially "hedged its bets" on which features to rely on since it knows that some features cannot be trusted.

5 Experiments

We tested the O2B perceptron on a subset of the KDD Cup 1999 Dataset with 250 "normal" examples and 250 "attack" samples that represent IP sweeps. While the entire KDD Cup 1999 dataset contains nearly 5 million examples, we were limited by our available resources to processing this 500 example subset. Each example containes 41 features, including connection duration (continuous), protocol type (symbolic), number of transmitted bytes (continuous), etc. The symbolic features were discretized using a one-to-one integer mapping.

We compared the performance of the O2B perceptron against two conventional binary, linear classification algorithms: logistic regression (LR) and a support vector machine (SVM) with a linear kernel. For our LR implementation, we used a learning rate $\alpha = 0.1$ and we declared the algorithm had converged when the absolute difference between iterations for each parameter was less than $tol = 0.001$. Other than specifying the linear kernel, we used the default parameters from the libSVM package [1]. The regularization parameter C for the O2B set to 1 after cross-validation with a 70-30 split of the data.

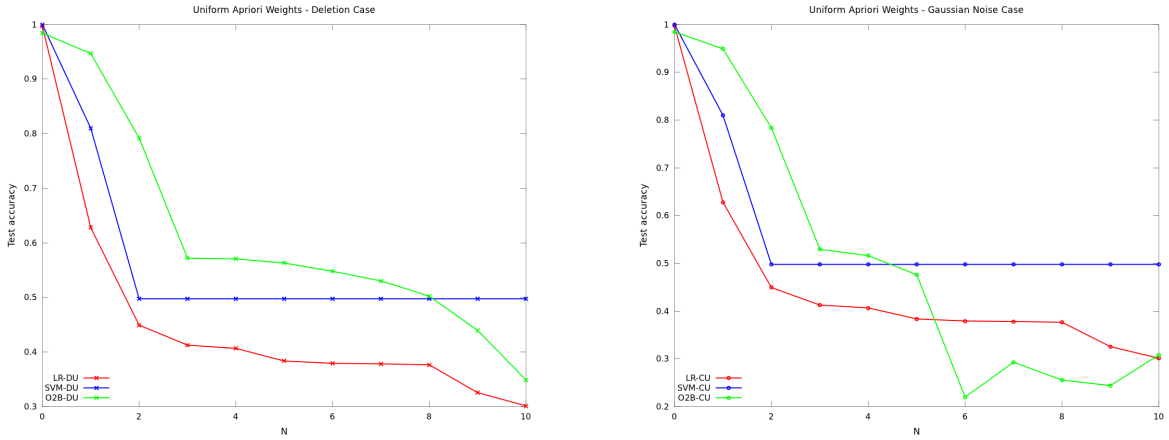
The classifiers were trained on a random 70 percent of the 500 examples and were tested on the remaining 30 percent. This process was repeated 20 times and the accuracy results averaged. For each classifier, we simulated our adversary using one of the aforementioned models. For the LR and SVM only the testing data was corrupted by the adversary, while both the training and testing data was corrupted for the O2B percetron (this follows the approach of [3]). The results are presented in Figure 1.

6 Discussion

For the deletion case it is clear that the O2B perceptron algorithm performs significantly better than the other classifiers, for both choices of cost vectors. For example, with the deletion/uniform cost pairing, when $N = 2$ the O2B algorithm outperforms the SVM by nearly 30 percent and the LR by 35 percent. With the deletion/MI pairing, when $N = 8$ the O2B algorithm again outperforms the SVM by about 30 percent and outperforms the LR by about 40 percent. In addition, the SVM algorithm performs better than Linear Regression; this is to be expected since SVM seeks to maximize the geometrical margin. Intuitively since the prediction will generally be farther from the decision boundary, our adversary will have to cause more damage to affect the prediction, as measured by $\sum_{j\in J} w_j x_j$.

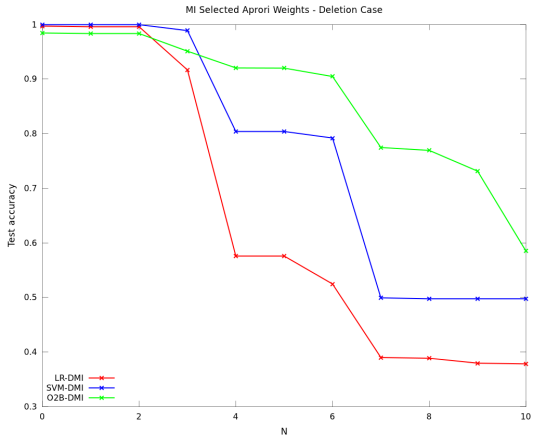
Similarly, the Gaussian noise/uniform cost pairing shows that - up to about $N = 5$ - the O2B perceptron algorithm outperforms the other classifiers. When $N = 2$, the O2B algorithm outperforms the other classifiers by roughly 30 percent.

However, in the Gaussian noise/MI case the adversary is much less effective at influencing the classifiers, demonstrated by the fact that all three perform roughly at the same level - from 90-100 percent. This should be the case in general, since the expected damage an adversary can inflict is $\sum_{j\in J} w_j(x_j - \mu_j)$ where μ_j is

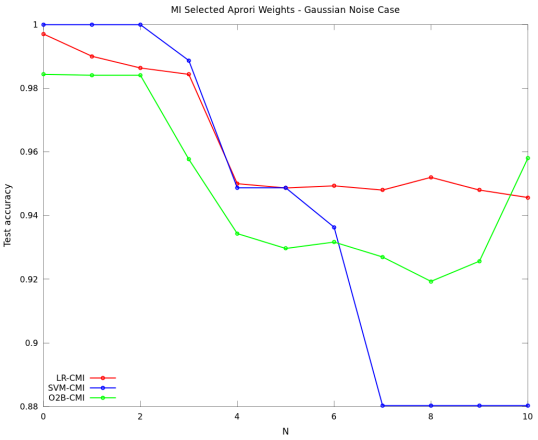


(a) DU

(b) CU



(c) DMI



(d) CMI

Figure 1: Classification accuracy for the three linear classifiers under different adversarial models. (a) and (b) show the results using uniform feature weights and the feature deletion model, respectively. (c) and (d), in turn, show results using MI selected feature weights and the Gaussian noise feature corruption model. The progression from (a) to (d) represent a decrease in classification difficulty. Note the scales on the y-axes.

the mean of the feature j as opposed to $\sum_{j \in J} w_j x_j$. In addition the data for our problem is very sparse (calculated to be $\sim$ 65 percent sparse) with each feature having a small variance, so for most features x_j will be close to μ_j. Thus it is very difficult for our adversary to affect the data, especially when we limit her ability to change those features which do not differ greatly from their mean.

The results also highlight the importance of the cost vector. Allowing the adversary to select the N most important features is much worse than when we weight the feature cost according to their mutual information, even though the adversary can choose to corrupt more features overall. The uniform cost tests also show that each classification is dependent on only a handful of features since deleting even one causes a significant drop in accuracy. This, again, is expected due to the sparsity of the data.

Overall we have demonstrated the sensitivity of these algorithms to such feature manipulation. We have also shown that significant improvements can be made by choosing an algorithm tailored to the problem and by selecting an appropriate cost vector for the training data.

7 Future Work

To apply the results in [3] as we did here there were a number of problems we faced. First, as was demonstrated, the performance gain of the O2B model is not as drastic for the case of feature corruption and does not address uncertainty in the classifications. Secondly the adversarial model used has certain problems when ranking features with uncertainty. Finally it is difficult to extend the O2B algorithm to problems using kernelization and to respect the condition that the adversary can only modify the original input features. We hope however to address these difficulties adequately and to develop useful diagnostic tools from such experiments.

In particular, we are both interested in applying these results to the diagnosis of certain psychological disorders. This problem is particularly difficult due to uncertainty in both the input features and classifications. As opposed to other diagnostic problems, there are rarely tests which can provide objective data and the underlying mechanics are often poorly understood. Many of the tests that attempt to provide such objective data are often prohibitively expensive (such as MRIs) and are often not covered under health insurance plans.

References

[1] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[2] M. Clayton. Cyberattack on Illinois water utility may confirm stuxnet warnings. *The Christian Science Monitor*, 11 2011.

[3] O. Dekel and L. Xiao. Learning to classify with missing and corrupted features. *Machine Learning*, 81(2):149–178, 2009.

[4] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

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

[6] T. Shanker. U.S. weighs its strategy on warfare in cyberspace. *The New York Times*, 10 2011.