The Thirty-Third International
FLAIRS Conference (FLAIRS-33)

Improving Costs and Robustness of Machine Learning Classifiers Against Adversarial Attacks via Self Play of Repeated Bayesian Games

Prithviraj Dasgupta, Joseph B. Collins, Michael McCarrick
U. S. Naval Research Laboratory, Washington D.C.
{raj.dasgupta, joseph.collins, michael.mccarrick}@nrl.navy.mil

Abstract

We consider the problem of adversarial machine learning where an adversary performs evasion attacks on a classifier-based learner by sending queries with adversarial data of different attack strengths to it. The learner is unaware whether a query sent to it is clean versus adversarial. The objective of the learner is to mitigate the adversary's attacks by reducing its classification errors of adversarial data. To address this problem, we propose a technique where the learner maintains multiple classifiers that are trained with clean as well as adversarial data of different attack strengths. We then describe a game theoretic framework based on a 2-player repeated Bayesian game called Repeated Bayesian Sequential Game with self play, that enables the learner to determine an appropriate classifier to deploy so that the likelihood of correctly classifying the query and preventing the evasion attack is not deteriorated, while reducing the costs to deploy the classifiers. Experimental results of our proposed approach with adversarial text data shows that our RBSG with self play-based technique maintains classifier accuracies comparable with that of an individual, powerful and costly classifier, while strategically using multiple, lower cost but less powerful classifiers to reduce the overall classification costs.

Introduction

Adversarial machine learning (Vorobeychik and Kantarcioglu 2018) is an important problem in machine learning based prediction systems such as email spam filters, online recommender systems, text classifier and sentiment analysis techniques used on social media, and, automatic video and image classifiers. The main problem in adversarial learning is to prevent an adversary from bypassing an ML-based predictive model such as a classifier by sending engineered, malicious data instances called adversarial examples. These attacks, called evasion attacks, could enable a malicious adversary to subvert the learner's ML model and possibly get access to critical resources being protected by the learner. Researchers have proposed techniques including adversarial training (Yuan et al. 2019) and game theory based techniques (Dasgupta and Collins 2019) to address the problem of adversarial learning. These techniques employ an approach called classifier hardening on a single classifier where the decision boundary of the classifier is refined over time via re-training with adversarial data. However, improving the robustness of single classifier hardening techniques is an open problem and these techniques are still been known to be susceptible to adversarial attacks (Madry et al. 2017). Moreover, classifier hardening techniques do not explicitly align costs to harden the classifier (e.g., costs to acquire adversarial training data, time and costs to recruit human experts for validating training and test data, and computational time and costs to retrain the classifier with adversarial data) with the data being classified. For instance, for classifying clean data, a classifier hardened over several batches of adversarial data might be excessive, as a classifier that is not hardened might achieve similar performance. In this paper, we posit that the costs of a classifier-based ML model to adversarial attacks can be improved without deteriorating classification accuracy, if instead of using a single classifier, we use multiple classifiers that are hardened separately against attacks of different strengths. Our idea is based on the well-known result of Wolpert's theorem (Wolpert 2002) that there is not a single classifier which can be optimal for all classification tasks and multiple, combined classifiers could outperform the best individual classifier. The main challenge with using multiple classifiers is to determine the appropriate pairing between a query sent to the classifier, with either clean or adversarial data of different attacks strengths, and a commensurate classifier from the collection of classifiers to handle the query most effectively, e.g., with least likelihood of classification errors and while aligning classifier hardening costs with the query's attack strength. A further wrinkle to the problem is that the classifier is not aware whether the query is with clean data from a legitimate client versus with adversarial data from an attacker. To address this problem, we propose a game theoretic framework called a Repeated Bayesian Sequential Game with self play between a learner and an adversary. The outcome of the game strategically selects an appropriate classifier for the learner. Our proposed formulation enables us to realize several practical aspects of learner-attacker interactions including uncertainty of the learner about the strengths of different attacks, costs to the learner and attacker to train the classifier and generate ad-

versarial examples respectively, rewards and penalties to attacker and learner for successes in their attacks and defenses respectively. Finally, a Bayesian game based representation enables our approach to handle asymmetric interactions between the learner and its clients for both non-competitive (legitimate clients, clean queries) and competitive (attackers, adversarial queries) settings. To the best of our knowledge, our work is one of the first attempts at using multiple classifiers deployed strategically to tackle the adversarial learning problem. We have validated our approach within a learner-adversary setting where the adversary generates queries with both clean and adversarial text data with different attack strengths while the learner's classifiers use deep network models for classification. Our results show that the learner can successfully converge to the distribution of different attacks types of the adversary and can strategically select different classifiers to reduce the overall classification cost without deteriorating the classification accuracy.

Related Work

Early work in adversarial learning modeled the interaction between the learner and adversary as a competitive, 2-player game (Dalvi et al. 2004) (Globerson and Roweis 2006). Subsequently, researchers extended the adversarial learning game using different formalisation including a sequential game (Brückner, Kanzow, and Scheffer 2012), a Bayesian game (Grosshans et al. 2013) where the learner has incomplete information about the attacker's strategy, a bi-level optimization problem (Mei and Zhu 2015) (Alfeld, Zhu, and Barford 2017), and randomization over strategies of the learner and the adversary (Bulò et al. 2017). In most of these techniques, the learner's strategy at each instance of the game is to adjust its model parameters. While that might be practical for smaller models with few parameters, as the model size increases, e.g., for a deep network with thousands of parameters, the strategy might become infeasible to realize in practice. In contrast, in our work, we use the strategy output of the game to select an appropriate classifier for the learner from an existing, pre-trained set of classifiers.

With the popularity of deep neural networks as ML models, several adversarial learning techniques based on adversarial training for deep networks have also been researched (Goodfellow, Shlens, and Szegedy 2014) (Kurakin, Goodfellow, and Bengio 2016). In adversarial training, the learner's ML model is trained with both clean and adversarial data to improve its capability to correctly classify adversarial data. Unlike game theory representations, adversarial training techniques do not explicitly model costs, penalties and rewards for learner and attacker. Most of these techniques use adversarial training to harden a single classifier. In contrast, instead of hardening one classifier, in our work, the learner maintains multiple classifiers with different degrees of hardening and strategically deploys one of them. Generative Adversarial Nets (GANs) (Goodfellow et al. 2014) also model interaction between an adversary (generator) and learner (discriminator) as a 2-player game. However, the objectives of GANs and adversarial learning are different. GANs enable an adversary to refine its data generation process, starting from a random distribution, so that the generated data is indistinguishable from legitimate data. Adversarial learning, on the other hand, aims to enable the learner to strategically defend against adversarial attacks.

Adversarial Learning as Bayesian Game

We consider a supervised learning setting for binary classification where learner, $\mathcal{L}$, receives data instances as queries from an attacker or adversary, $\mathcal{A}$. We represent this interaction between $\mathcal{L}$ and $\mathcal{A}$ as a 2-player Bayesian game for adversarial learning (Grosshans et al. 2013) while adapting Großhans' model to multiple classifiers, different attack strengths and repeated interactions between $\mathcal{L}$ and $\mathcal{A}$. We describe the different components of the game below:

Let $\mathbf{X}^{ev}$ denote a set of queries. We refer to $\mathbf{X}^{ev}$ as the clean query set. Let $X = (\mathbf{x}, y), X \in \mathbf{X}^{ev}$ denote a query data instance, where $\mathbf{x} = \{x_1, x_2, ...\}$ is its set of attributes or features and $y \in \{0, 1\}$ is its ground truth label.

Adversary. $\mathcal{A}$ sends either clean or adversarial data as queries; the latter is generated by perturbing clean data using a perturbation function $\delta : \mathbf{x} \rightarrow \mathbf{x}$. We assume that $\mathcal{A}$ uses different perturbation functions δ_i, $i = 0, 1, 2, ...$, where i denotes the strength of the perturbation. For example, perturbation strength could correspond to the number of features of $\mathbf{x}$ that are modified to convert it into an adversarial instance (Globerson and Roweis 2006). $\delta_i(\mathbf{x})$ denotes the adversarial data generated with perturbation strength i and δ_{i+1} is a stronger perturbation than δ_i. Perturbing $\mathbf{x}$ does not change its ground truth label, y. For notational convenience, we refer to clean data, $\mathbf{x} = \delta_0(\mathbf{x})$. An action for $\mathcal{A}$ is to select a δ_i, use it to convert clean instance $\mathbf{x}$ into adversarial instance $\delta_i(\mathbf{x})$, and send the adversarial instance to $\mathcal{L}$.

Learner. $\mathcal{L}$ receives a query data instance $\bar{\mathbf{x}}$ and its task is to correctly predict its category. $\mathcal{L}$ is neither aware of the perturbation strength i of $\delta_i(\mathbf{x})$ inside the data, nor is it aware of $\bar{y}$, the ground truth label of $\bar{\mathbf{x}}$. $\mathcal{L}$ uses a set of classifiers, L_j, $j = 0, 1, 2...$ for its prediction task. L_j implements a classification, $L_j : \mathbf{x} \rightarrow \{0, 1\}$, that outputs a category given the features of the query data. Classifier L_j is adversarially trained using training data $\mathbf{X}^{tr,\delta_j} \notin \mathbf{X}^{ev}$, where δ_j denotes the perturbation strength of the training data. We assume that L_{j+1} is a stronger classifier than L_j: for a query $\mathbf{x}$, L_{j+1} has a higher confidence in its output than L_j, or, mathematcally, $P(L_{j+1}(\mathbf{x}) = y) \geq P(L_j(\mathbf{x}) = y)$. An action for $\mathcal{L}$ is to select a classifier L_j and use it to classify the data instance sent by $\mathcal{A}$. We denote the action set of $\mathcal{L}$ as $Ac_{\mathcal{L}} = \{L_0, L_1, L_2...\}$. Let $\Pi(Ac_{\mathcal{L}})$ be the set of probability distributions over $Ac_{\mathcal{L}}$. $s_{\mathcal{L}} \in \Pi(Ac_{\mathcal{L}})$ denotes a strategy for $\mathcal{L}$ and $s_{\mathcal{L}}(L_j)$the probability of selecting L_j under strategy $s_{\mathcal{L}}$. Finally, recall that $\mathcal{L}$ is not aware of the perturbation δ_i that has been used by $\mathcal{A}$ on the query data instance, $\bar{X}$ that it receives. To model this uncertainty about its opponent, $\mathcal{L}$ uses epistemic types for $\mathcal{A}$ (Harsanyi 1967). $\mathcal{A}$'s type θ_i denotes that $\mathcal{A}$ uses perturbation strength i to create $\bar{\mathbf{x}}$, i.e., $\bar{\mathbf{x}} = \mathbf{x}_{\theta_i} = \delta_i(\mathbf{x})$. $\Theta_{\mathcal{A}} = \{\theta_i\}$ is $\mathcal{A}$'s set of types and $p : \Theta_{\mathcal{A}} \rightarrow [0, 1]^{|\theta_{\mathcal{A}}|}$ denotes a probability distribution over these types. $\Theta_{\mathcal{A}}$ is known to $\mathcal{L}$, $p()$ is calculated by $\mathcal{L}$. But θ_i, the exact realization of $\mathcal{A}$'s type (in other words, the perturbation strength used to create $\bar{\mathbf{x}}$) is not known to $\mathcal{L}$

when it receives $\bar{\mathbf{x}}$ from $\mathcal{A}$.

Utilities. Utilities are numeric values assigned by each player to the outcomes from the players' joint actions in a game. Each player could then preferentially rank its joint outcomes and select a suitable action such as a utility maximizing action. For our game, recall that $\mathcal{L}$ is not able to observe $\mathcal{A}$'s type θ_i (amount of perturbation in $\bar{\mathbf{x}}$). Therefore, $\mathcal{L}$ calculates an expected utility over $\mathcal{A}$'s possible types, $\Theta_{\mathcal{A}}$, using $\mathcal{A}$'s type distribution $p()$. $\mathcal{L}$'s expected utility for strategy $s_{\mathcal{L}}$ with query data $\bar{\mathbf{x}}$ and ground truth label $\bar{y}$ is given by:

$$EU_{\mathcal{L}}(s_{\mathcal{L}}, \bar{\mathbf{x}}, \theta_{\mathcal{A}}, p()) = \sum_{\theta_i \in \Theta_{\mathcal{A}}} p(\theta_i) U_{\mathcal{L}}(L_j, \bar{\mathbf{x}}, \theta_i),$$

$$U_{\mathcal{L}}(L_j, \bar{\mathbf{x}}, \theta_i) = \sum_{L_j} s_{\mathcal{L}}(L_j) \Big(P(L_j((\bar{\mathbf{x}}) = \bar{y}) | \theta_i) v_{\mathcal{L}}(L_j, \theta_i) - c_{L_j} \Big), \quad (1)$$

where $P((L_j(\bar{\mathbf{x}}) = \bar{y}) | \theta_i)$ is the probability that $\mathcal{L}$ makes a correct prediction given $\bar{\mathbf{x}}$ was generated using θ_i, $v_{\mathcal{L}}(L_j, \theta_i)$ is the value for $\mathcal{L}$ from classifying $\bar{\mathbf{x}}$ using L_j and c_{L_j} is the cost of using classifier L_j.

In adversarial settings, it is usually assumed that the adversary is aware of the learner's prediction model, e.g., model parameters of the learner's classifier (Alfeld, Zhu, and Barford 2017). In the context of our game, this can be interpreted as $\mathcal{A}$ knowing $\mathcal{L}$'s strategy, $s_{\mathcal{L}}$. $\mathcal{A}$'s utility for query data $\bar{\mathbf{x}}$ with ground truth label $\bar{y}$, for $\mathcal{L}$'s strategy $s_{\mathcal{L}}$ and its own type θ_i is given by:

$$U_{\mathcal{A}}(s_{\mathcal{L}}, \bar{\mathbf{x}}, \theta_i) = \sum_{L_j} s_{\mathcal{L}}(L_j) \Big(P(L_j(\bar{\mathbf{x}}) \neq \bar{y}) v_{\mathcal{A}}(L_j, \theta_i) - c_{\theta_i} \Big), \quad (2)$$

where $P(L_j(\bar{\mathbf{x}}) \neq \bar{y})$ represents the probability that $\mathcal{L}$ makes a mistake in prediction (in other words, $\mathcal{A}$'s adversarial perturbation of clean data was successful) and $v_{\mathcal{A}}(L_j, \theta_i)$ is the value that $\mathcal{A}$ derives from sending the query data $\mathbf{x}_{\theta_i}$ when $\mathcal{L}$'s action is L_j and c_{θ_i} is $\mathcal{A}$'s cost for generating adversarial data with type (perturbation strength) θ_i.

Bayesian Sequential Game. Using the above actions and utility functions, we can represent a Bayesian sequential game between $\mathcal{L}$ and $\mathcal{A}$ as $\Gamma = [N, Ac, U, \Theta_A, p()]$, where $N = \{\mathcal{L}, \mathcal{A}\}$ is the set of players, $Ac = Ac_{\mathcal{L}} \times \Theta_{\mathcal{A}}$ is the set of joint action-types of $\mathcal{L}$ and $\mathcal{A}$, $U = (EU_{\mathcal{L}}, U_{\mathcal{A}})$ denotes the utilities received by $\mathcal{L}$ and $\mathcal{A}$ (given in Eqns. 1 and 2), $\Theta_{\mathcal{A}}$ and $p()$ are the set of $\mathcal{A}$'s types and probability distribution over those types, as defined before.

The computational problem facing $\mathcal{L}$ and $\mathcal{A}$ is to calculate a suitable strategy $s^*_{\mathcal{L}}$ and suitable type θ^*_i respectively. To do this calculation using Eqn. 1, $\mathcal{L}$ also needs to know the value of $p()$, the probability distribution over $\mathcal{A}$'s types. To address these issues, we propose an approach using a technique called self play with repeated plays of the above Bayesian Sequential game called a Repeated Bayesian Sequential Game (RBSG), as described below.

Repeated Bayesian Sequential Game and Self-Play

The objective of $\mathcal{L}$ is to determine a suitable strategy $s^*_{\mathcal{L}}$ to play against $\mathcal{A}$ that would improve its expected utility by deploying an appropriate classifier that has been hardened commensurate to the strength of the perturbation used by $\mathcal{A}$. To achieve this, $\mathcal{L}$ uses self play, where $\mathcal{L}$ and $\mathcal{A}$ play the Bayesian Sequential game, Γ, repeatedly. The repeated interactions between $\mathcal{L}$ and $\mathcal{A}$ can be represented as a game tree with sequential moves between them. A node in the game tree denotes a player's turn to make a move. In a move, a player selects an action from its action set. $\mathcal{L}$ and $\mathcal{A}$ make alternate moves with $\mathcal{L}$ moving first. A pair of moves by $\mathcal{L}$ and $\mathcal{A}$ corresponds to an instance of the Bayesian Sequential game, Γ, realized as below.

Algorithm 1: game-play()

1 Select $s^*_{\mathcal{L}}$ using current belief of $\hat{p}$, and θ^*_i (Eqn. 3 or 4)
2 Calculate utils. recd.: $\hat{u}_{\mathcal{L}}$ and $\hat{u}_{\mathcal{A}}$ with observed values of $s^*_{\mathcal{L}}$ and θ^*_i resp. (using Eqns. 1 and 2)
3 return $(\hat{u}_{\mathcal{L}}, \hat{u}_{\mathcal{A}})$

Game Play. As shown in Algo. 1, $\mathcal{L}$'s moves by selecting a strategy $s^*_{\mathcal{L}}$. $\mathcal{A}$ then selects type (perturbation strength) $\theta^*_i \sim p()$ while observing $s^*_{\mathcal{L}}$. With the selected θ^*_i, $\mathcal{A}$ then generates q adversarial queries by perturbing q clean data instances from $\mathbf{X}^{ev}$, and sends each adversarial query, $\bar{\mathbf{x}}$, to $\mathcal{L}$. After $\mathcal{L}$ processes the queries, both $\mathcal{L}$ and $\mathcal{A}$ receive utilities given by Eqns. 1 and 2 respectively. The problem facing $\mathcal{L}$ is to calculate $s^*_{\mathcal{L}}$ without observing θ^*_i and $p()$ from $\mathcal{A}$'s moves. We solve this problem using a modified Monte Carlo Tree Search (MCTS) algorithm, as described below.

Calculating strategy $s^*_{\mathcal{L}}$. To calculate $s^*_{\mathcal{L}}$, $\mathcal{L}$ generates different paths in the game tree to discover utilities received from different sequences of moves. To systematically explore the game tree, $\mathcal{L}$ uses an MCTS-like algorithm (Browne et al. 2012), called *TreeTraverse*. shown in Algos. 2 and 3. *TreeTraverse* works by generating a sequences of moves or game plays corresponding to a path in the game tree up to a finite cutoff depth h. $\mathcal{L}$ and $\mathcal{A}$'s utilities from their moves are recorded along the path and once the bottommost level is reached, the utilities are updated along the path upwards toward the root. In this way, moves that could lead to high utility can be identified by each player.

The key aspects of MCTS are to balance exploration and exploitation while traversing the game tree by using a heuristic function called *selectBestChild* (Algo. 2, line 4), and, doing an operation called rollout to rapidly traverse unexplored parts of the game tree by selecting actions for each player up to the game tree's cutoff depth h (Algo. 3). In our *TreeTraverse* algorithm, we have used two heuristic functions for *selectBestChild*, as described below:

Bayes Nash Equilibrium (BNE). In BNE, each player selects a best reponse strategy that maximizes its utilities, given the possible strategies of its opponent (Harsanyi 1967). The strategies for $\mathcal{L}$ and $\mathcal{A}$ calculated using BNE are

given by:

$$s^*_{\mathcal{L}} = arg \max_{s_{\mathcal{L}} \in \Pi(Ac_{\mathcal{L}})} EU_{\mathcal{L}}(s_{\mathcal{L}}, \bar{\mathbf{x}}, \Theta_{\mathcal{A}}, p()),$$
$$\theta^*_i = arg \max_{\theta_i \in \Theta_{\mathcal{A}}} U_{\mathcal{A}}(s^*_{\mathcal{L}}, \bar{\mathbf{x}}, \theta_i), \quad (3)$$

where $u_{\mathcal{A}}$ is given by Eqn. 2 and $EU_{\mathcal{L}}$ is given by Eqn. 1 with $\mathcal{A}$'s actual type distribution $p(\theta_i)$ replaced by $\mathcal{L}$'s belief distribution $\hat{p}(\theta_i)$.

Upper Confidence Bound (UCB). UCB is a bandit-based technique (Browne et al. 2012) that weighs the expected utility of a move with the number of times it has been visited, so that previously unexplored or less-explored actions at a move are also tried. UCB uses the following equation to calculate $s^*_{\mathcal{L}}$ and θ^*_i:

$$s^*_{\mathcal{L}} = \arg\max_{\Pi(L_j)} \sum_{\theta_i} \left(p(\theta_i) \sum_{\bar{\mathbf{x}} \in \bar{\mathbf{X}}} U_{\mathcal{L}}(L_j, \bar{\mathbf{x}}, \theta_i) + C \sqrt{\frac{2 \ln Par_{visit}}{L_{j,visit}}} \right)$$
$$\theta^*_i = \arg\max_{\theta_i} \sum_{L_j} \left(\sum_{\bar{\mathbf{x}} \in \bar{\mathbf{X}}} s^*_{\mathcal{L}}(L_j) U_{\mathcal{A}}(L_j, \bar{\mathbf{x}}, \theta_i) + C \sqrt{\frac{2 \ln Par_{visit}}{\theta_{i,visit}}} \right) \quad (4)$$

Here, C is a constant, Par_{visit} is the number of times the parent node of the current node was visited and $L_{j,visit}$ and $\theta_{i,visit}$ are the number of times the current node has been visited for $\mathcal{L}$ and $\mathcal{A}$ respectively.

Algorithm 2: TreeTraverse(v)

Input: v: start node for traversal
Output: v_{val}: value from tree traversal (via backtracking) starting from v up to depth h

1 **if** $v_{depth} = h$ **then**
2 return
3 **else if** v *is fully expanded* **then**
4 $c_{val} \leftarrow$ TreeTraverse(selectBestChild(v)) // go down game tree along best action (Eqns. 3 or 4)
5 Update $v_{val} \leftarrow v_{val} + c_{val}$; increment v_{visit}
6 return c_{val}
7 **else if** v *is visited but not expanded* **then**
8 $\bar{c} \leftarrow$ generatedAllChildren(v) // all actions
9 $c \leftarrow$ select random child (action) from $\bar{c}$
10 $c_{val} \leftarrow$ rollout(c)
11 Update $v_{val} \leftarrow v_{val} + c_{val}$, increment v_{visit} and c_{visit}
12 return c_{val}
13 **else if** v *is not visited* **then**
14 $v_{val} \leftarrow$ rollout(v)
15 Increment v_{visit}
16 return v_{val}

Updating belief of $\mathcal{A}$'s type distribution. The *TreeTraverse* algorithm explores a sequence of moves along any single path from the root of the game tree up to the cutoff depth h. We call this a trial for the RBSG. To update its belief distribution $\hat{p}$, $\mathcal{L}$ uses multiple trials and, at the end of each trial, $\mathcal{L}$ uses an update strategy to update $\hat{p}()$. We consider two probability update strategies that can be used by $\mathcal{L}$ (Algo. 4, line 4) for updating $p_{\Theta_{\mathcal{A}}}$. **1) Fictitious Play (FP)**: In fictitious play (Shoham and Leyton-Brown 2009), the probability of type θ_i is the fraction of times it was played following

Algorithm 3: Rollout(v)

Input: v: start node for rollout
Output: v_{val}: value from rollout (via backtracking) starting from v up to depth h

1 **if** v *is terminal* **then**
2 $\hat{u}_{\mathcal{L}}, \hat{u}_{\mathcal{A}} \leftarrow$ game-play()
3 return $(\hat{u}_{\mathcal{L}}, \hat{u}_{\mathcal{A}})$
4 **else**
5 $c \leftarrow$ select child of v prop. to $u_{\mathcal{L}}$ (for $\mathcal{L}$'s move) or prop. to $p()$ (for $\mathcal{A}$'s move)
6 $c_{val} \leftarrow$ rollout(c)
7 return c_{val}

action L_j, as given by the following update rule:

$$P(\theta_i | L_j) = \frac{\text{No. of times } \theta_i \text{ selected after } L_j}{\text{Total no. of times } L_j \text{ selected}} \quad (5)$$

2) Bayesian Update (BU): Bayesian update of θ_i calculates the conditional probability of selecting θ_i when it followed L_j using Bayes rule, given by the following equation:

$$P(\theta_i | L_j) = \frac{P(L_j|\theta_i)P(\theta_i)}{P(L_j)} = \frac{P(L_j|\theta_i)P(\theta_i)}{\sum_{\theta_i} P(L_j|\theta_i)P(\theta_i)}, \quad (6)$$

where $P(L_j|\theta_i)$ is the fraction of times L_j was played following θ_i, $P(L_j)$ is known to $\mathcal{L}$ and the denominator is a normalization term. The updated probability estimate is then used by $\mathcal{L}$ to calculate the expected utilities in Eqns. 3 and 4 for its actions more accurately against $\mathcal{A}$'s in future trials.

Algorithm 4: Self-Play()

1 **for** $\tau = 1 ... n_{trials}$ **do**
2 $root \leftarrow \mathcal{L}$'s first move with randomly sel. action
3 TreeTraverse(root)
4 Update $\hat{p}$ using prob. update strategy (fic. play, Eqn. 5 or Bayes update, Eqn. 6)

Experimental Results

We have evaluated the performance our proposed RBSG with self play-based adversarial learning technique for a binary classification task with text data using the Yelp review polarity data set. (dat). Each data instance has either of two labels, 1 (negative) and 2 (positive). The clean training and test sets have $560,000$ and $38,000$ samples respectively. We used the Character Convolutional Neural Network (CharCNN) (Zhang, Zhao, and LeCun 2015) model that consists of 5 convolution layers followed by 3 fully connected layers. It uses convolution layers to identify character level features to classify text. For generating adversarial text, we used the single character gradient based replacement technique (Liang et al. 2018). Given a data instance in the form of a text character string as input to an ML model, the method works by classifying the text using the model and calculating the gradient of the loss function for each

character in the input text. It then replaces the character with the most negative gradient (most influential on the classifier output) in the text with the character that has the least positive gradient (least influential on the classifier output). The technique can be used iteratively on a data instance to replace multiple characters in the text and create adversarial text with different attack strengths, e.g., two iterations of the technique yields adversarial text with perturbation strength 2. The CharCNN was first trained with clean data, and then hardened separately with two adversarial training data sets with $200,000$ adversarial training samples of perturbation strengths 1 and 2 respectively. This gave three classifiers for $\mathcal{L}$ with increasing hardening levels, denoted by L_0, L_1 and L_2. The accuracies of these classifiers were then evaluated with $50,000$ instances of test data of perturbation strengths 1, 2 and 3 each, as reported in Table 1.

	L_0	L_1	L_2
Clean	0.9392	0.9426	0.94
Adv 1	0.8684	0.88	0.8782
Adv 2	0.7706	0.7922	0.8152
Adv 3	0.6814	0.7056	0.7502

Table 1: Testing accuracy of individual classifiers with different hardening levels (columns) on adversarial test data with different perturbation strengths (rows).

Adversary $\mathcal{A}$ generates queries with either clean data or adversarial data with perturbation strengths $1, 2$ and 3, giving $\Theta_{\mathcal{A}} = \{\theta_0, \theta_1, \theta_2, \theta_3\}$. $\mathcal{L}$ uses three classifiers, so, $Ac_{\mathcal{L}} = \{L_0, L_1, L_2\}$. The different parameters used for our experiments are: cutoff depth in self play, $h = 20$; number of trials in self play, $n_{trials} = 10$; batch size for queries sent by $\mathcal{A}$ to $\mathcal{L}$, $q = 10$; and constant in UCB calculation (Eqn. 4), $C = 2$.

UCB	L_0	L_1	L_2	Acc.
Clean	43.75%	29.46%	26.79%	0.9321
Adv 1	39.65%	24.13%	36.21	0.8716
Adv 2	24.11%	25%	50.89%	0.8062
Adv 3	39.81%	20.37%	39.81%	0.7222
BNE	L_0	L_1	L_2	Acc.
Clean	57.56%	10.37%	32.07%	0.9302
Adv 1	33.91%	46.96%	19.13	0.867
Adv 2	29.46%	27.68%	42.86%	0.808
Adv 3	31.53%	32.43%	36.04%	0.709

Table 2: Percentage of different classifiers used and accuracies (columns) obtained for clean and adversarial data of different perturbation strengths (rows). Data in the top and bottom tables are with Upper Confidence Bound (UCB) and Bayes Nash Equilibrium (BNE), respectively, for action selection during self play.

For our first set of experiments, we validated if $\mathcal{L}$, using the self play algorithm, could effectively deploy appropriate classifiers for data of different perturbation strengths. We created four different type distributions for data generated by $\mathcal{A}$, each distribution having 98% of one of the four types$\{\theta_0, \theta_1, \theta_2, \theta_3\}$. $\mathcal{L}$ used either Upper Confidence Bound (UCB) or Bayes Nash Equilibrium (Eqn. 3 or Eqn. 4) to select actions in the game tree during self play. Our results are shown in Table 2. The results show that both UCB and BNE metric for action selection perform comparably. The accuracy obtained using our RBSG-based self play technique on clean and adversarial data perturbed with different perturbation strengths (last column of Table 2 is not degraded and comparable to the best accuracies obtained with the most hardened classifier, L_2, when used individually (column 4 of Table 1). The RBSG with self play technique is also able to align adversarial data of different perturbation strengths with the commensurately hardened classifier, as shown by the maximum percentage of each row in Table 1 corresponding to the classifier hardened with adversarial data of that perturbation strength. Note that with adversarial data of perturbation strength 3, $Adv\,3$, the classifiers are selected almost uniformly. This is because none of the classifiers, L_0, L_1 or L_2 were trained with adversarial data of perturbation strength 3. L_2, which had the highest individual accuracy for $Adv\,3$ data, is used most frequently, albeit marginally, for $Adv\,3$ data in Table 2. Our Self-play technique also strategically also uses L_0 and L_1 that incur lower costs to deploy than L_2. Consequently, the utility obtained by $\mathcal{L}$ with self play is better than its utility while using individual classifier L_2 only. Fig. 1 shows the comparison of the relative utilities obtained by $\mathcal{L}$ while using the proposed RBSG with self play technique versus the utilities obtained while using the most hardened individual classifier L_2. As illustrated, the RBSG with self play technique is able to improve utilities as it deploys lower cost classifiers L_0 and L_1 along with L_2 while aligning the expected perturbation strength of the query data, estimated via $\hat{p}$, with the commensurately hardened classifier.

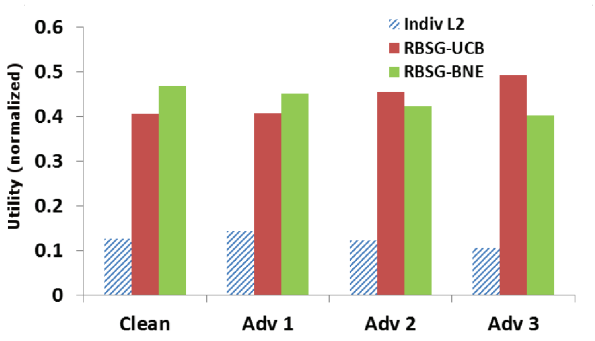

Figure 1: Relative utilities obtained by individual classifier L_2, and RBSG with self play-based techniques with UCB and BNE action selection for data reported in Table 2

.

For our next experiments, we evaluated the convergence of $\mathcal{L}$'s belief distribution $\hat{p}()$ to $\mathcal{A}$'s actual type distribution $p()$ using the fictitious play and Bayesian update probability update strategies (Eqns. 5 and 6). Results were averaged over 10 runs. For each run, $p()$ was selected as a random distribution. We report the Kullback-Liebler(KL)

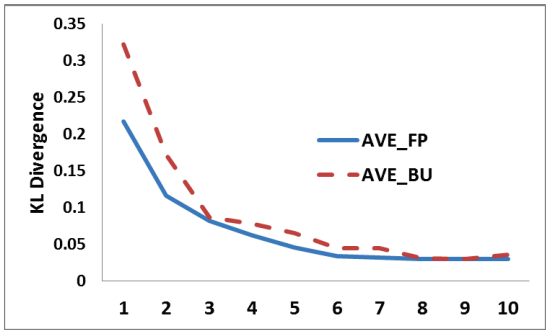

Figure 2: KL divergence between $\mathcal{A}$'s actual type distribution and $\mathcal{L}$'s belief distribution using fictitious play and Bayesian update for $n_{trials} = 10, h = 20$. Results are averaged over 10 runs.

divergence between $\hat{p}()$ and $p()$, given by $D_{KL}(\hat{p}||p) = \sum_{\theta_i \in \Theta_{\mathcal{A}}} \hat{p}(\theta_i) ln \frac{\hat{p}(\theta_i)}{p(\theta_i)}$. As shown in Fig. 2, with both strategies $\hat{p}$ is able to converge to within 5% of $p()$ within about 6 trials. Fictitious play converges faster with higher KL divergence values while Bayesian update takes a longer time to converge owing to its more complex calculations.

Conclusion

We proposed a technique for improving the costs of a classifier-based ML model against adversarial attacks of different strengths without deteriorating its performance by using repeated game-like interactions between a learner and an adversary. There are several important directions that are worthy of further investigation. First, the assumption in existing research which assumes that the learner reveals its classifier to the adversary is rather limiting. A more realistic situation would be that the adversary is able to reverse engineer the learner's classifiers, but it is not aware of the frequency with which the learner deploys them. The adversary could then also build a model of the learner via repeated interactions to determine its perturbation strength strategically. Secondly, although used as a popular solution technique in games, Nash equilibrium (NE) strategy calculation is known to have certain shortcomings such as assuming that players always behave rationally. In reality, an adversary could behave myopically, select a greedy outcome, or, adopt suboptimal, low and slow strategies to misguide the learner. To handle these situations, a direction we are interested in exploring is to use recent techniques such as regret-based techniques, safety value and exploitability of opponents, instead of Bayes Nash equilibrium-based strategy selection. Finally, integrating reinforcement learning for our adversarial learning setting promises to be another direction worthy of further investigation.

References

Alfeld, S.; Zhu, X.; and Barford, P. 2017. Explicit defense actions against test-set attacks. In *Proc. 31st AAAI Conf. Artificial Intelligence, AAAI'17*, 1274–1280.

Browne, C. B.; Powley, E.; Whitehouse, D.; and et al. 2012. A survey of monte carlo tree search methods. *IEEE Trans. on Comp. Intelligence and AI in games* 4(1):1–43.

Brückner, M.; Kanzow, C.; and Scheffer, T. 2012. Static prediction games for adversarial learning problems. *J. Mach. Learn. Res.* 13(1):2617–2654.

Bulò, S. R.; Biggio, B.; Pillai, I.; Pelillo, M.; and Roli, F. 2017. Randomized prediction games for adversarial machine learning. *IEEE Trans. Neural Netw. Learning Syst.* 28(11):2466–2478.

Dalvi, N.; Domingos, P.; Sanghai, S.; Verma, D.; et al. 2004. Adversarial classification. In *Proc. 10th ACM SIGKDD Intl. Conf. Knowledge Discovery and Data mining*, 99–108. ACM.

Dasgupta, P., and Collins, J. 2019. A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks. *AI Magazine* 40(2):31–43.

Yelp reviews polarity data set. http://goo.gl/JyCnZq. 2019-07-15.

Globerson, A., and Roweis, S. 2006. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning*, 353–360. ACM.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Grosshans, M.; Sawade, C.; Bruckner, M.; and Scheffer, T. 2013. Bayesian games for adversarial regression problems. In *Proc. 30th Intl. Conf. Machine Learning*, ICML'13, III–55–III–63.

Harsanyi, J. C. 1967. Games with incomplete information played by "bayesian" players, i–iii part i. the basic model. *Management science* 14(3):159–182.

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.

Liang, B.; Li, H.; Su, M.; Bian, P.; Li, X.; and Shi, W. 2018. Deep text classification can be fooled. In *Proc. 22nd Intl. Joint Conf on AI, IJCAI*, 4208–4215.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Mei, S., and Zhu, X. 2015. Using machine teaching to identify optimal training-set attacks on machine learners. In *Proc. 29th AAAI Conf, Artificial Intelligence*, AAAI'15, 2871–2877. AAAI Press.

Shoham, Y., and Leyton-Brown, K. 2009. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Vorobeychik, Y., and Kantarcioglu, M. 2018. Adversarial machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 12(3):1–169.

Wolpert, D. H. 2002. The supervised learning no-free-lunch theorems. In *Soft computing and industry*. Springer. 25–42.

Yuan, X.; He, P.; Zhu, Q.; and Li, X. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems* 30(9):2805–2824.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, 649–657.