Security Project Report:

Sequential Decision Making in Intrusion Detection Systems

Dave Ackley & Allen Williams

December 2021

1 Introduction

In recent years there has been an explosion in papers utilizing reinforcement learning to accomplish impressive tasks including superhuman performance on a large number of Atari games [1], beating the world champion humans in games like Chess, Go and Shogi [2], and learning immensely complex actions via competition or self-play [3]. There have also been several papers outlining systems using these agents in intrusion detection settings [4, 5]. These papers cite several distinct advantages to using reinforcement learning in this setting including

- Reinforcement learning agents can be easily applied in an online setting
- It is easy to incorporate new parameters without having to retrain from scratch
- The networks used are often very fast to train and predict with

To this list we add another, which is that the *adaptation of reinforcement learning agents to settings which are not obviously sequential is an important proof of concept, which future innovations can be built on top of.*

The reinforcement learning in IDS papers [4, 5] use custom built environments with the agents built for their task, we demonstrate that powerful and modular reinforcement learning libraries can be adapted to this problem, which allows the agent to be easily chosen from a wide variety of prebuilt and highly optimized algorithms. For this project we use the Rllib library, from the Ray project, because it supports multiagent environments and distributed algorithms [6].

2 Methods

We demonstrate our model on the NSL-KDD dataset and evaluate the model using two metrics: *accuracy*, and *F1 Score*. We choose these metrics because they can be computed with classifiers that do not output probabilities, and

because accuracy gives an indicator of real-world performance, while F1 score gives an indicator of performance while taking into account the unbalanced nature of the dataset.

2.1 Dataset

The NSL-KDD consists of 43 columns, 29 numeric columns, 8 boolean valued columns, and 6 categorical columns. Of note, two columns (*attack* and *level*) are used only for the target. Then ultimately there are 41 features in the dataset. Attacks are categorized into 5 broad types, see figure 1 for counts of attacks by category in the training set.

1. Normal
2. DoS
3. Probe
4. R2L
5. U2R

Where *normal* means no attack. Several of the features are categorical in nature, specifically *protocol type, service*, and *flag*. There are 3 values of protocol type (TCP, UDP, ICMP), 11 values for flag, and 66 values for service. After one-hot encoding the categorical features, there are 127 columns in the dataset, of which 125 are potential features, one is our label, and one describes the difficulty of a particular sample. For our evaluation we use binary labels, that is samples which are *normal* are labeled 0, while samples which feature an attack are labeled 1.

2.2 Baseline

We use out of the box XGBoost as a baseline for comparison. We chose XGBoost because it often works well without significant hyperparameter tuning, so we could quickly get a baseline to compare against and focus on building our environment and agent. XGBoost is a popular library implementing a gradient boosted trees algorithm, which can learn highly nonlinear decision boundaries.

2.3 Agent & Environment

We create two environments in this project. Initially we setup the environment as follows:

1. The State Space consists of samples from the dataset. These are provided as observations to the DQN Agent.
2. The Action Space consists of the binary labels 1 or 0

3. The episodic environment has a horizon of just 1 timestep. The reward provided is 1 if the action is the same as the sample's label, and 0 otherwise.

We use a *DQN* agent from the *Stable Baselines 3* repository [7]. *DQN* is a value-based reinforcement learning agent, which was one the first highly successful deep reinforcement learning agents [1], and it featured two important innovations over previous attempts at deep reinforcement learning. The first innovation is that *DQN* uses an *experience replay buffer* which holds *experiences* of the form (s, a, r, s'), meaning the agent was in state s and received reward r after taking action a and transitioning to state s'. Learning is then done by randomly sampling the replay buffer rather than learning from experiences as they are encountered in real time. The second innovation made by the *DQN* network is the use of a *target network* to estimate the values in value networks update rule. The target network is initialized with the same weights as the value network, but they are not updated in the learning step, instead they are copied from the value network occasionally (for example every 10,000 episodes). The effect of these innovations was to stabilize the learning process and make it more like supervised learning. The *DQN* agent then learns a *value function* $Q(s, a)$ which estimates the reward from being in state s and taking action a and following the agent's policy for the rest of the episode. The policy of the *DQN* agent is implicitly derived from the value function as follows. Let $\varepsilon \in [0, 1]$ then if the agent observes the state s, with probability $1 - \varepsilon$ the agent chooses $a = argmax_a Q(s, a)$, and with probability ε the agent chooses uniformly at random from the allowed actions. Generally in practice ε follows an annealing schedule during training, so initially ε will be very high and the agent will mostly explore the state space, while in later episodes, after the agent has done significant learning, ε will be very low, allowing the agent to explore regions of the state space which would be unlikely to be explored under a uniform random policy.

2.4 Adversarial Training

In the second paper using reinforcement learning in IDS applications, the environment is designed to prevent the agent from making accurate classifications. This forces the agent to learn from samples which it classifies poorly. In this paper the environment was designed as an agent who chooses the attack type that the classifier agent observes. We use RLLib to design a multiagent environment to accomplish a similar task. In our task there are 2 agents, the classifier agent, as in the first setting, and the environment agent, which is newly introduced for this task. In this environment the classifier agent has the same state and action space as above, but the environment has an action space consisting of the different attack types. The reward is -1 if the classifier incorrectly classifies the sample and $+1$ if the classifier gets it wrong. Thus both agents observe the same state, and inverted rewards. In this setting the environment can be seen as an adversary, deliberately giving the classifier attack types that it misclassifies

most often.

3 Results

In the first experiment we report a *baseline accuracy* of **0.78** and a *baseline F1* of **0.77**. We report an accuracy of **0.76** for our agent and and F1 of **0.74**. We also see that the two models were very similar in the attacks they misclassified most frequently. Both struggled with the 'guess passwd' and 'warezmaster' attacks the most.

In the second experiment we report an accuracy of **0.79** and and F1 of **0.81**. We see that the model misclassified very different attacks than the first two, and has a much more flat histogram of correct classifications when normalized the size of the class (the number of samples with that particular type of attack). The results are shown in figure 4

4 Discussion

The results show that it is possible to improve the performance of a classifier via an adversarial environment, and also that the misclassification of minority samples could be greatly reduced by adversarial reinforcement learning environments. Future work could include a more exhaustive search of reinforcement learning agents to use, and their hyperparameters. It would also be interesting to examine the speed of both training prediction when compared with other models in an online environment.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[3] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

[4] Manuel Lopez-Martin, Belen Carro, and Antonio Sanchez-Esguevillas. Application of deep reinforcement learning to intrusion detection for supervised problems. *Expert Systems with Applications*, 141:112963, 2020.

[5] Guillermo Caminero, Manuel Lopez-Martin, and Belen Carro. Adversarial environment reinforcement learning algorithm for intrusion detection. *Computer Networks*, 159:96–109, 2019.

[6] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

[7] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

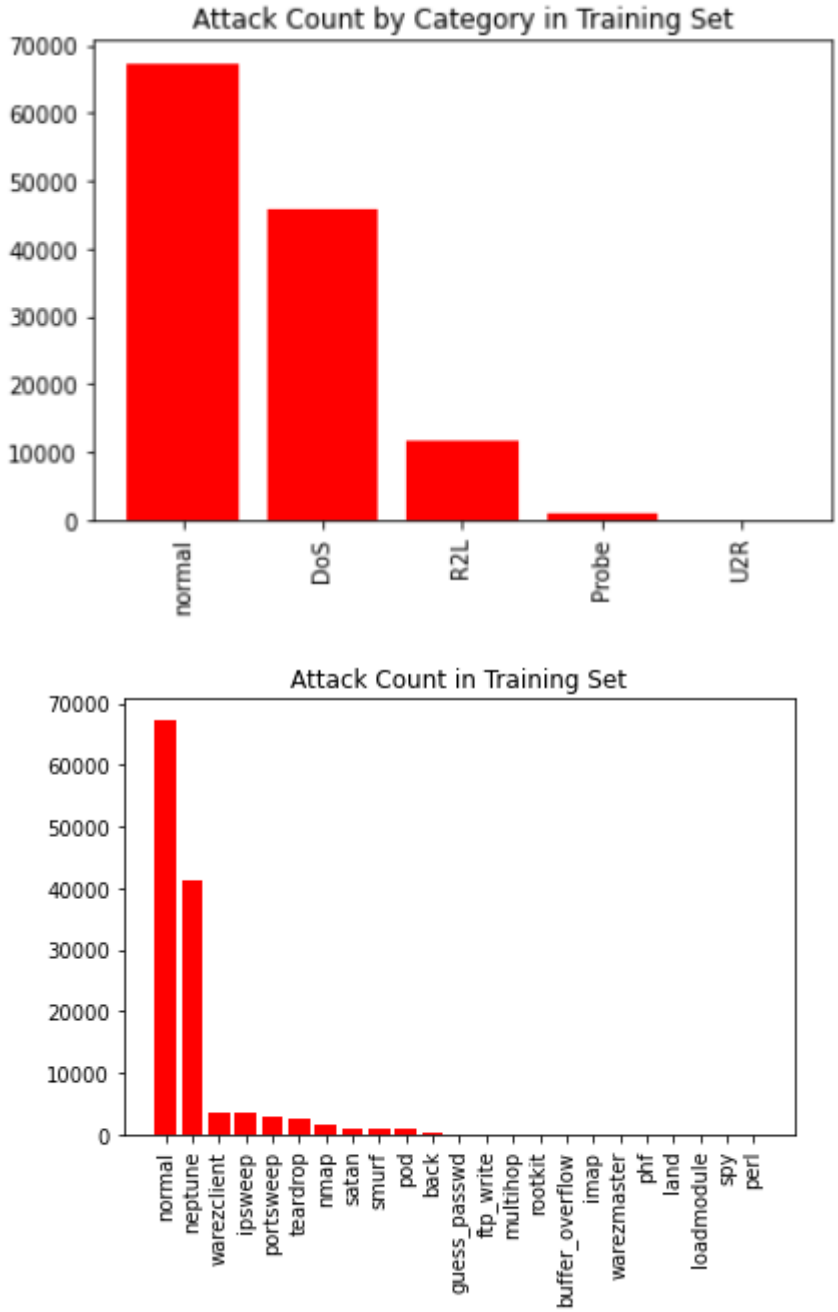

Figure 1: A Count of the attacks, and a count of attacks by category in the NSL-KDD Training set

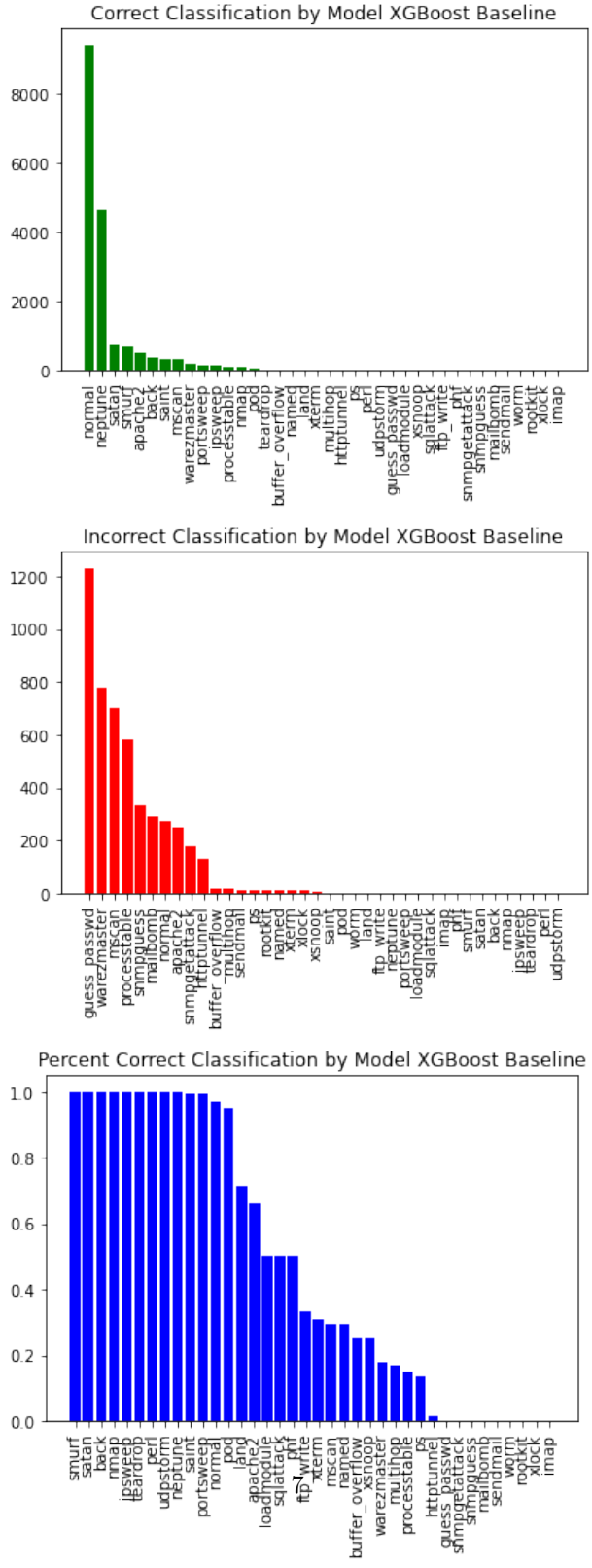

Figure 2: Histograms of Correct and Incorrect Classification by the baseline model, and the percent correct for each attack

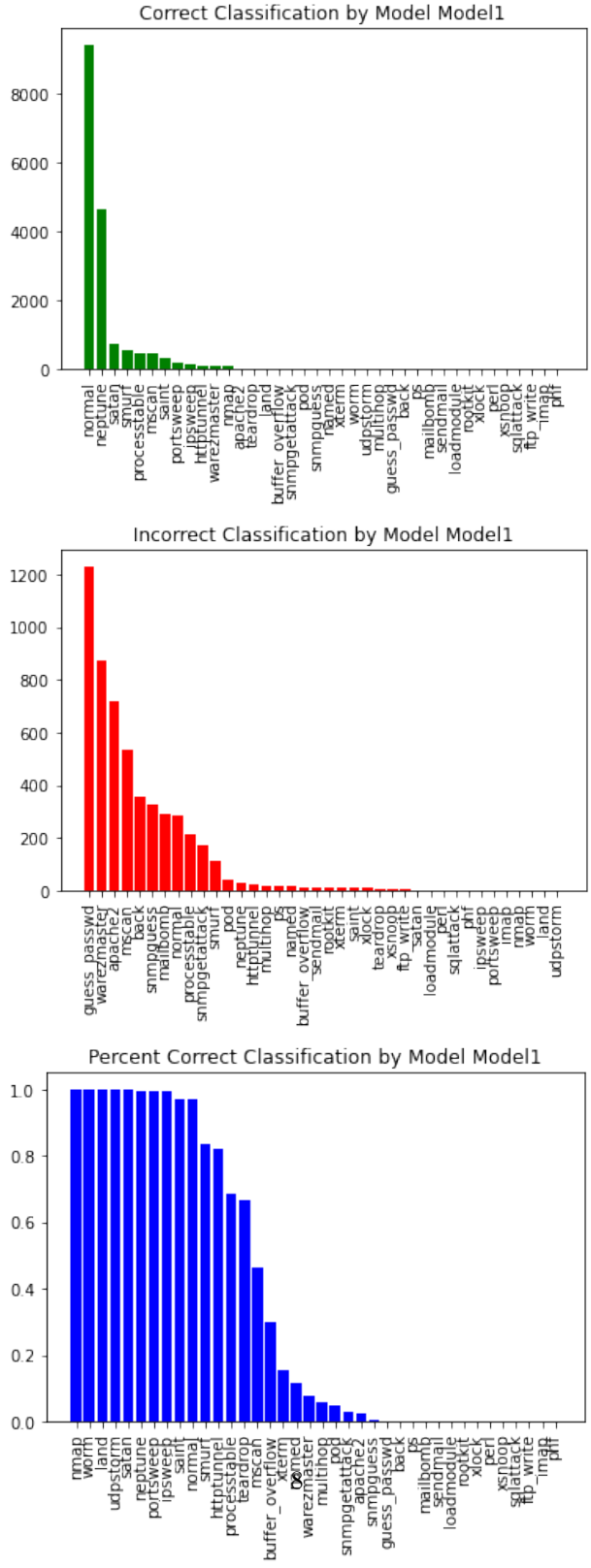

Figure 3: Histograms of Correct and Incorrect Classification first RL Model, and the percent correct for each attack

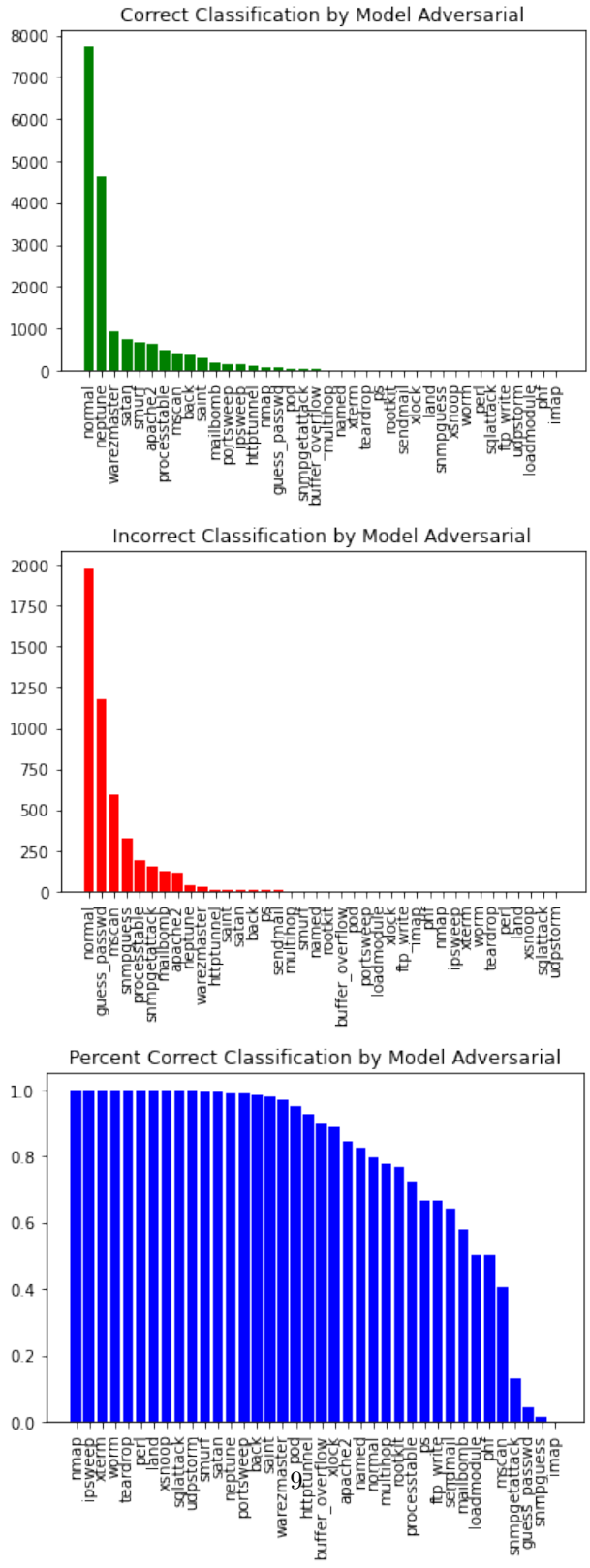

Figure 4: Histograms of Correct and Incorrect Classification Adversarial RL Model, and the percent correct for each attack