Active Learning in Adversarial Distributed Domains Involving Networks (ALADDIN)

Chris Magnano[1], **Ronak Mehta**[1]

[1]Department of Computer Sciences - University of Wisconsin-Madison

cmagnano@cs.wisc.edu, ronakrm@cs.wisc.edu

Abstract. *Active learning allows models to choose training instances in an online setting. One unexplored domain where this comes into practice is in adversarial learning. As well as a number of new theoretical considerations, using active learning in adversarial domains may give valuable insight into how to design security systems to deter malicious users. Applying active learning in an adversarial setting results in a number of new considerations when designing a query method. We evaluate a number of query methods that may have value in the adversarial problem domain of network intrusion detection, and present a new method, k-Safe Axes. K-Safe Axes simulates a malicious user by attempting to find network attacks which could bypass a network intrusion detector. Our results are inconclusive, but show the need for further exploration of k-Safe Axes and other active learning query methods in this domain.*

1. Introduction

1.1. Active Learning

Active learning is an online learning method in which the learner can choose which instances should be labeled [Settles 2009]. There are a variety of active learning settings, but here we will focus on pool-based active learning. In the pool-based active learning scenario, the learner has a large pool of unlabeled instances. Each iteration, the learner chooses a instance from the pool to query to an oracle. The oracle, often a human expert in the problem domain, labels the instance and returns it to the learner. The learner can then add this instance to a training set. Often, the learner begins with a small set of labeled instances before active learning begins, as in many domains a small number of baseline instances are known before the data labeling process begins [Settles 2009].

Active learning is advantageous in domains which involve a large amount of unlabeled data and domains where the cost of labeling data is very high. It has been used in a range of learning applications, from handwriting recognition to passage classification [Settles 2011]. In these domains, it is infeasible to create a strong classifier with a training set large enough to reasonably represent the underlying distribution. Active learning is used to select key instances to add to the training set; those instances which might most accurately define the classification boundaries.

1.2. Adversarial Learning

Adversarial learning is the application of machine learning methods by a learner and an adversary. Here, we define an adversary as a secondary learner attempting to predict the model being used by their opponent.

One of the most common adversarial domains is in network intrusion detection. A network intrusion detector (NID) is overseeing some network traffic (generally in the form of packets) and attempts to reject traffic that it believes is malicious. An attacker may attempt to learn what the NID can accurately predict, and attempt to get malicious packets through the NID's 'blind spot.' Previous work has shown the effectiveness of rejecting attacks through standard learners, including Support Vector Machines [Eid et al. 2010] and Naive Bayes networks [Panda et al. 2010].

1.3. Adversarial Learning as Active Learning

The network intrusion detection problem can be formulated as an active learning problem from the perspective of the attacker. In this scenario, the oracle can be viewed as the NID system. The attacker sends exploratory and attack network traffic to a target server. The NID 'labels' the network traffic through its acceptance or rejection of these packets. The attacker can then use this information to update its attack pattern.

We believe that modeling an attacker as an active learner will better simulate malicious user behavior than giving an adversarial classifier a fixed training set, as modeling the pattern of malicious behavior and how the attacker responds NID actions can allow for more dynamic defensive action. If an attacker can be accurately modeled in this setting, it may be possible for stronger defensive intrusion detection systems to be built in the future which specifically focus on deterring malicious users instead of only detecting their presence.

The adversarial setting is a challenging problem for active learning techniques. As the query method is attempting to learn which instances another classifier was incorrect on, it is trying to classify which instances are difficult for the NID to learn correctly. This may mean that noisy or outlier instances, types of instances which often present problems to learning methods, become important positive instances to the adversarial active learner. The problem also becomes more interesting if the oracle learns from the queries of the active learner. In this case an 'arms race' can develop between learners [Laskov and Lippmann 2010]. We do not examine this possibility here, but it may be an interesting place for future work.

In this paper, we examine current query methods, and their advantages and disadvantages in the adversarial setting. We then propose a new method, k-Safe Axes, which attempts to improve upon shortcomings of active learning methods when applied to this problem domain.

The rest of the paper is laid out as follows: In section 2 we introduce our dataset, methods, and experimental setup. In section 3 we present the results of our experiment. We analyze our results and propose future work in section 4, and conclude in section 5.

2. Experimental Methodology

2.1. Dataset

We used the NSL-KDD dataset in our analysis [Tavallaee et al. 2009]. Each instance in the NSL-KDD dataset is a packet coming into a server on a network. Included are 41 features which a server would be able to immediately determine from a data packet, such as the packet size, service type, and protocol. There are $158,518$ instances total in the dataset. Each packet is labeled as either normal (non-attack) or as an anomaly (attack) packet.

The NSL-KDD dataset was created in 2009 as an improved version of the KDD 1999 Cup dataset. The KDD cup dataset was designed for use in the KDD Cup competition in 1999.

However, the KDD Cup dataset includes a large number of redundant packets and inconsistent attack type distributions [Tavallaee et al. 2009].

2.2. Query Methods

In recent years, a wide range of query methods have been developed in the field of active learning. Here we outline four methods we chose to apply, and propose a novel method designed for the adversarial setting.

The Simple algorithm greedily chooses the instance closest to the decision boundary of its currently-learned SVM. The instance chosen is the one the algorithm is least-certain about its corresponding class. The algorithm chooses the instance most likely to bisect the version space. [Baram et al. 2004]

The Self-Conf algorithm was designed specifically to minimize error on future test instances [Roy and McCallum 2001] . Using Monte Carlo methods, the algorithm estimates a 'self-confidence' for each instance, and attempts to choose instances which maximize the probability that the current model is correct [Baram et al. 2004].

The KFF (Kernel-Farthest First) algorithm chooses unlabeled instances that are the farthest (in kernel space) from the labeled ones currently being used to build the model. In this way, the model explores areas of the kernel space that are furthest from its current known space. Because the algorithm chooses queries only based on known instances, it can be applied to any type of classifier or learning algorithm. In this case, we apply it to the base SVM used across all query methods [Baram et al. 2004].

The last algorithm examined was Balancing Exploration and Exploitation, or Balanced_EE [Osugi et al. 2005]. This algorithm attempts to solve the issue of focusing on refining the decision boundary while ignoring potential areas of the hypothesis space that are unexplored. The balancing of these two methods is done by updating the probability of exploration on each step of the algorithm.

While these methods are designed to work well in a normal learning setting, their applicability in the adversarial domain has a number of possible setbacks. In particular, the cost of being wrong on a certain instance may have a variable cost depending on the NID's response, and thus it may not be valuable to refine the decision boundary greedily. An attacker may have a limited number of queries it can apply before its subject stops all traffic from the attacker, and so it might not be worthwhile for the attacker to continually choose instances it knows nothing about. For example, in the methods above we might expect Self-Conf to do better in an adversarial context than KFF, because Self-Conf will attempt to validate its current model, choosing safer instances than the distant ones chosen by KFF. The following section outlines our proposed method, which attempts to solve some of these issues.

2.2.1. K-Safe Axes

The proposed method was designed specifically for the adversarial setting. It combines exploration and exploitation while taking into account the high cost of a false positive error. The method first attempts to determine features which would likely not be used by a network intrusion detector (NID). It does this through choosing the features among attack instances which have the highest variance. This choice is made under the assumption that features with a high

variance within the same class are not useful for classification, and thus are likely given less importance by learning systems. More formally, we choose the rth 'safe' feature S_r by calculating:

$$S_r = \arg \max_{i \in N, i \notin S} \frac{\sum_{j=1}^{m} (x_{ij} - \bar{x_i})^2}{m} \tag{1}$$

Where N is the number of features, m is the number of instances, x_{ij} is the value of feature i in instance j, and $\bar{x_i}$ is the mean value of feature x_i over all instances. This is done k times, resulting in the set S of r safe features.

After finding safe features, a query is selected by choosing an instance which minimizes the distance to known positive instances in 'unsafe' features and maximizes the distance to known positive instances in safe features. If no positive instances are known, a random instance is chosen as a query. The query instance Q_j on iteration j is found by the following minimization:

$$Q_j = \arg \min_{j \in m} [k \sum_{i \notin S} (x_{ij} - \bar{x}^i_{pos}) - \sum_{i \in S} (x_{ij} - \bar{x}^i_{pos})] \tag{2}$$

Where m is the number of instances, S is the set of safe features, and $\bar{x}^i_{pos}$ is the mean of feature i over all known positive instances. The parameter k determines a trade off between the relative importance of staying close to known positive instances and finding instances which are distant on safe axes. It can be thought of as an exploration-exploitation trade off, with higher values of k favoring exploitation.

From here, we refer to our proposed method as k-Safe Axes, where k refers to the exploration exploitation trade-off parameter k.

2.3. Experiments

We performed two experiments in order to analyze the effectiveness of the chosen query methods in the adversarial setting and to explore k-Safe Axes under a variety of parameter settings. Every run consisted of an adversarial learner performing 100 queries on the NID learner.

In our first experiment, all five query methods (Simple, Self-Conf, Balanced_EE, KFF, and the k-Safe Axes) were evaluated over 10 trials, each trial using a random 80% stratified subset of the data for a total of 50 runs.

In our second experiment, we evaluated the k-Safe Axes over 50 different values of the parameter k, for 0 to 100 by increments of 2. Again, we used 10 trials for each value of k on 10 random 80% stratified subsets of the data, for a total of 500 runs. The parameter r was set to 5 for all experiments.

2.3.1. Overall Adversarial Design

All of our experiments involved two learners, the network intrusion detector (NID) learner and the adversarial-active learner. Each run, we load one of 10 previously generated stratified 80% subsets of the total dataset. We then split this subset into an 80% training subset and a 20% testing subset. The training subset is given to the NID learner. After the NID learner is trained, the NID learner's classification of each instance in the testing subset is obtained.

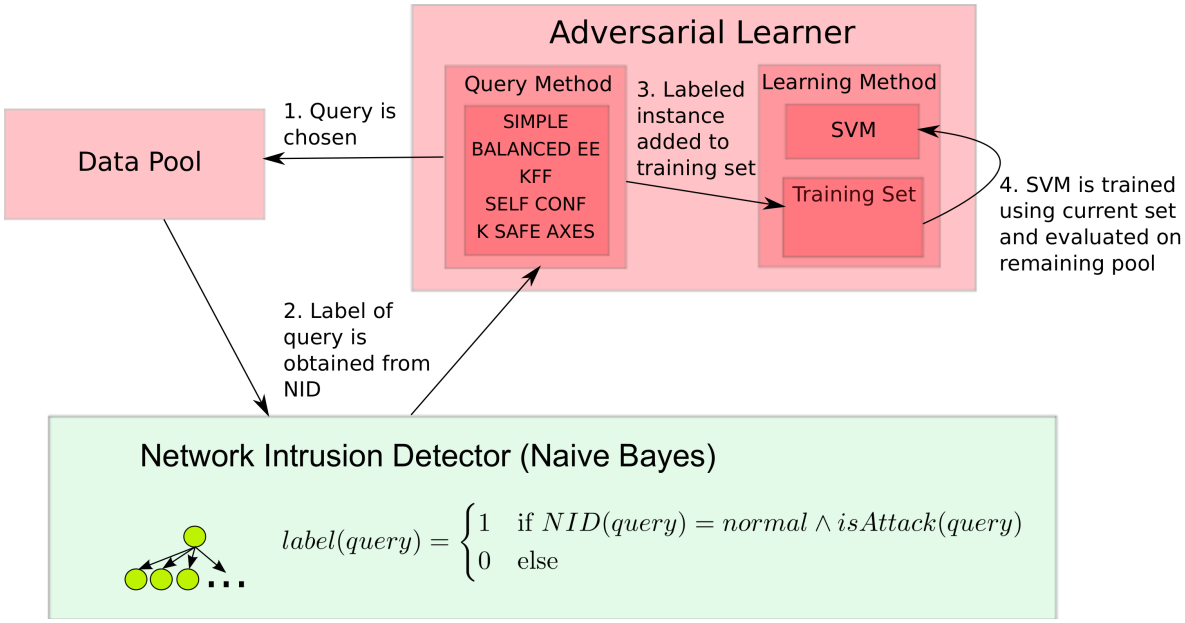

Figure 1: The main active learning loop of all experiments. This figure shows one iteration of active learning, where the adversarial learner chooses a query instance, obtains the label from the NID, learns from the new training set, and is then evaluated on the remaining pool instances. It is important to note that the pool only consists of attack instances. In order to offset this, a small number of normal instances are initially added to the training set of the SVM.

These classifications become the labels on the testing subset for the adversarial learner, where incorrectly labeled attack instances are given a label of 1 and all other instances are given a label of 0 (Figure 1). The adversarial learner is only interested in finding attacks it could send to a server without that server classifying them as attacks.

The first learner, the NID, remained constant for all experiments. The NID learned from a training set of both normal and attack network packets. We chose to use Naïve Bayes as the NID in our experiments, as it has shown moderate success on the NSL-KDD dataset previously [Tavallaee et al. 2009], but still has a high enough misclassification rate to create a meaningful amount of training data for the adversary.

The second learner, the adversarial and active learner, is comprised of 2 components. The first component is the query method, which chooses instances from the attack pool and queries them to the NID. This can be thought of as an exploratory attack on the server. The second component is a base learner, in this case a support vector machine (SVM). The SVM component in all experiments used the radial-basis function (RBF) or Gaussian kernel.

Each experimental run, the adversarial learner is given a small (approximately 50) labeled set of normal packets from the testing subset as an initial training set and all (approximately 11000) of the attack instances in the testing subset. These unlabeled attack instances become the attack pool for querying. The adversarial learner is not given the labels of these attack instances. Each iteration, the adversarial learner uses its query method to choose an instance. In our adversarial setting, this is equivalent to a malicious user choosing an attack it believes it can get through a server's intrusion detection system. The adversarial learner is then given the label of the queried instance. The second component, the SVM, is then re-trained on it's current training set and the new queried instance. Thus each iteration the adversarial

training set grows by one instance. A visual overview of a single iteration is shown in Figure 1.

3. Results

The Naïve Bayes network intrusion detector was able to classify between 80% and 85% of testing attack instances correctly. This means that the pool for the adversarial learner had approximately 4 times as many negative instances as positive instances. Thus all accuracies should be viewed against a baseline accuracy of 80%.

In experiment 1, we found that only Self-Conf was able to consistently choose queries which improved its accuracy (Figure 2). Balanced_EE improved at approximately the same pace as Self-Conf but after about 60 queries began to behave erratically. KFF and K-Safe axes both showed little improvement over the 100 queries.

K-safe axes outperformed all other methods in the percentage of safe queries chosen (Figure 3). A safe query is an attack instance which the NID missclassifies as normal. KFF had by far the lowest rate of safe queries, with Simple, Self-Conf, and Balanced_EE falling somewhere in the middle.

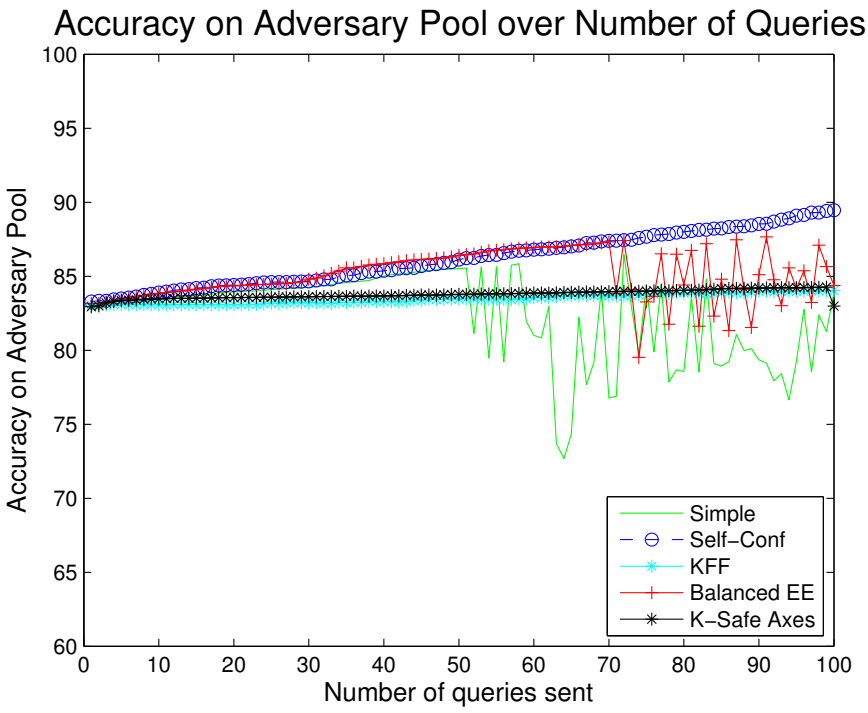

Figure 2: Accuracy of the 5 examined query methods on current pool instances over the course of 100 active learning iterations. Each point represents the mean accuracy of the base learner SVM over 10 trials on a stratified 80% random sampling of the total data set. The 10 random subsets were the same for all experiments. Only Self-Conf was able to consistently improve beyond the baseline 80% accuracy.

In experiment 2 we found no significant correlation between accuracy and the percent of safe queries or the parameter k (Figures 4 and 5). The percent of safe queries quickly improves for values of k less than 10, but quickly converges to around 90%. Accuracy remained constant at around 80% with noise between different values of k.

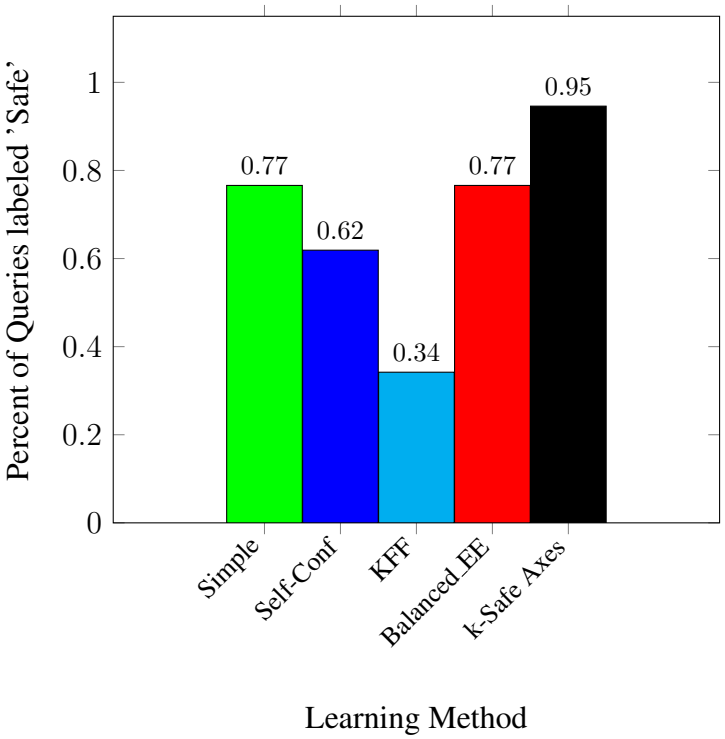

Figure 3: The mean proportion of safe queries of the 5 examined query methods. Each bar represents the average proportion of 100 queries over 10 trials using a stratified randomly selected 80% subset of the total dataset each trial. A safe query is an attack instance which the NID classifies as normal. K-Safe Axes outperformed all other methods by a margin of 18%. KFF performed the worst by a margin of 28%.

4. Discussion and Future Work

Most methods did not improve much beyond the initial baseline accuracy, which is likely a product of the adversarial problem. As the learning methods are attempting to classify instances which are difficult for the NID learner to classify, the adversarial learner looks for difficult or inconsistent instances. The NSL-KDD dataset contains a small number of instances which were unable to be correctly classified by 21 different common learning methods [Tavallaee et al. 2009]. These instances would all be included in the pool of the adversarial learner, and in general the more difficult an instance is to classify, the more likely it is to be in the active learning pool as a positive instance. This may have caused the noise observed in experiment 2 over our k parameter.

There was a drop-off in accuracy in some experiments, particularly with the Simple and Balanced_EE algorithms (Figure 2). For Simple, this may be because instances chosen near the decision boundary are more likely to become new support vectors, changing the separating hyperplane. For Balanced_EE, once safe axes have been exploited to the extent of the trade-off, exploration into unknown areas of the feature-space would cause a drop in accuracy. One interesting future experiment may be to see what percentage of queries are subsequently used as support vectors as a measure of how useful those queries were.

Both K-Safe Axes and KFF were unable to improve in accuracy over the 100 queries. It is logical that these two methods performed in a similar way, as both methods query instances which are maximally dissimilar, in either some feature dimensions or kernel space, from known

instances in an attempt to improve accuracy. The Self-Conf method was able to consistently improve accuracy as more instances were added to its pool. This may be attributed to its 'reassurance' algorithm, which queries instances it is most confident about.

K-Safe Axes was able to outperform all other methods in the percentage of safe queries chosen (Figure 3). However, the other examined methods were not designed to choose safe queries. This result thus shows that while we do perform better than a baseline in terms of the percentage of safe queries chosen, more work is needed to determine how high that percentage can go. A possible direction to explore are active learning methods with variable cost functions [Greiner et al. 2002].

The results of experiment 2 were inconclusive. While there is a clear initial gain in the percent of safe queries for values of k, the benefit quickly plateaus for values greater than 20. This initial gain indicates that potential future analysis may include refining the range over which to test.

Another possibility was that the pool of attack instances was not sufficiently preprocessed to allow variance or distance to be accurate measures between different features. It is possible that some features are on average much more noisy and inconsistent in attack packets than other features, but fall at fewer extremes. This would mean that there may be bias toward choosing features with extreme values as one of the k safe features. A possible solution for this issue may be to use mutual information instead of variance.

One likely method of improving stability may be to change the number of safe features chosen. Additionally, finding optimal values for the k parameter may be improved using a hill-climbing search to find a better range of values. Another possibility may be to perform strong smoothing methods on features so that a few extreme values have less of an effect on safe feature choices.

5. Conclusion

Existing query methods, while relatively accurate, are unsuited to the adversarial setting due to their inability to adjust to variable query costs and search specifically for 'difficult' instances. While the proposed method was able to consistently outperform previous methods at choosing safe instances in an adversarial setting, it was unable to outperform current methods in overall classification accuracy and was sensitive to small changes in parameter values.

Due to this sensitivity to small changes in data and parameter changes, k-Safe Axes is currently unsuited for being able to make a conscious choice of trade off between exploration and exploitation. Further work is needed to increase the stability of this method and to further adapt it to the adversarial setting. Once query methods that more realistically mimic a malicious user are found, these methods can be used to design intrusion detection systems which will be able to better focus on misleading or deterring malicious users.

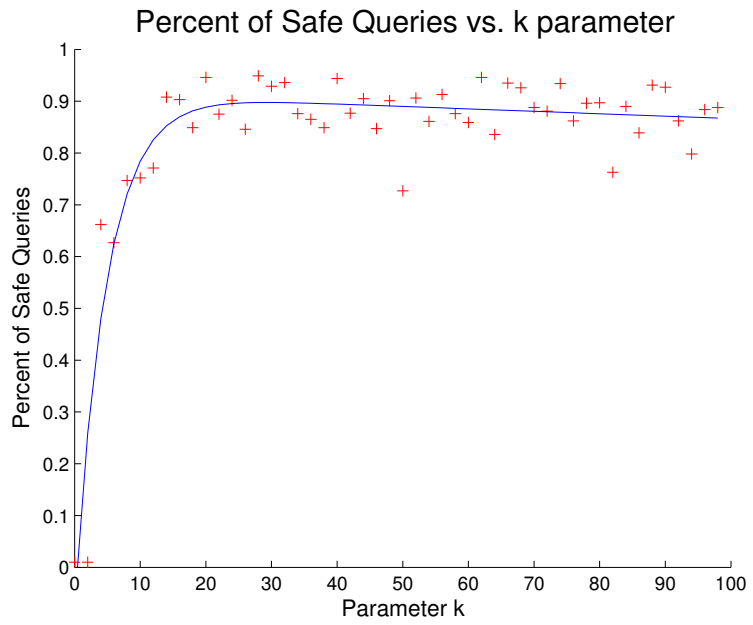

Figure 4: The percentage of queries that were labeled 'normal' by the NID plotted against our parameter k. There is a clear gain for k increasing from 0, but the benefit plateaus for k greater than 20.

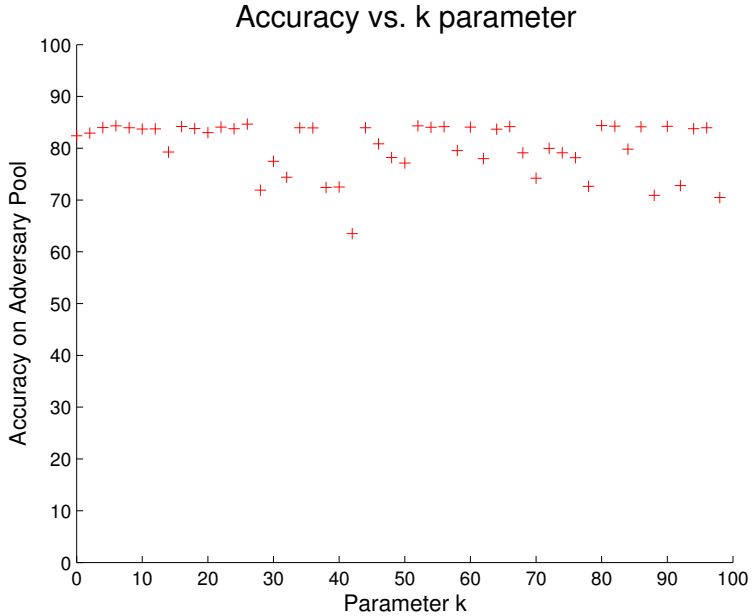

Figure 5: Accuracy of our adversarial learner on its own training pool after 100 queries against our parameter k. There does not appear to be any correlation between accuracy and our parameter.

References

Baram, Y., El-Yaniv, R., and Luz, K. (2004). Online choice of active learning algorithms. *J. Mach. Learn. Res.*, 5:255–291.

Eid, H., Darwish, A., ella Hassanien, A., and Abraham, A. (2010). Principle components analysis and support vector machine based intrusion detection system. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 363–367.

Greiner, R., Grove, A. J., and Roth, D. (2002). Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2):137–174.

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

Osugi, T., Kim, D., and Scott, S. (2005). Balancing exploration and exploitation: a new algorithm for active machine learning. In *Data Mining, Fifth IEEE International Conference on*, pages 8 pp.–.

Panda, M., Abraham, A., and Patra, M. (2010). Discriminative multinomial naïve bayes for network intrusion detection. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 5–10.

Roy, N. and McCallum, A. (2001). Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*.

Settles, B. (2009). Active learning literature survey.

Settles, B. (2011). From theories to queries: Active learning in practice. *Active Learning and Experimental Design W., pp. 1-18*.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. (2009). A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6.