Attacking an AI classifier in a realistic context

Bachelor Thesis Artificial Intelligence

Radboud University Nijmegen

Bart van Delft[1]

Supervised by dr. I.G. Sprinkhuizen-Kuyper and prof. dr. T.M. Heskes.

December 15, 2008

[1]s0513377, b.vandelft@student.ru.nl

Abstract

In this thesis a theoretical attack on an AI classifier as introduced by Barreno et al. (2006) is described and lifted to a more realistic setting. The dataset, classifier and attack algorithm were updated in order to increase reality. The KDD '99 cup-data on network intrusion was used as a data set and a combination of kMeans and Learning Vector Quantization was used for classification.

The hypothesis was confirmed that an increase in realism would result in a significant increase in the number of iterations needed for a successful attack. However in some attampts the attack still succeeded.

Subsequently the randomization defense as suggested by Barreno et al. was implemented and tested in both abstract and realistic contexts. The defense was effective in all tested contexts, however seemed to be less effective in the most realistic one.

Since both the realistic context and the randomization defense increase the number of iterations needed for the attack, further research can be performed on how an external network notifying suspicious changes in the primary classifier may benefit from this.

Contents

1 Introduction

Applications of AI-algorithms in the field of computer security often deal with classification. For example, classifying e-mail traffic into normal and spam mail. If an adversary wants to work his[1] way around such a classifier he often tries to make his spam e-mail look like a normal e-mail (in this example). If the features added to the e-mail are sufficient the classifier will label it as normal and the message of the spammer appears unfiltered in someones inbox.

The creators of the classifier will sooner or later discover this new type of spam and update their classifier. The adversary will try (and probably succeed) finding new features that distract the classifier. This may rapidly result in an arms race and causes the maintenance of the classifier to become expensive in both time and money.

To prevent this continuous need of updating, online semi-supervised algorithms can be used. Instead of generating new data by hand to retrain the classifier it is assumed that the label given by the classifier to new incoming data is correct. This labeled data is then used to retrain the classifier and keep it up to date, which makes the classifier more resistant to novelties in the incoming data (Powers & He, 2008; Sung & Mukkamala, 2003; Lazarevic et al., 2003).

In the article "Can Machine Learning Be Secure?" from Barreno et al. (2006) an example of such an online semi-supervised classifier is given. This is continued with an attack on this classifier based on the characteristics of the semi-supervised classifier, and not, as described earlier, by disguising the adversarial data. This makes it a rather unique method of attack.

However, the algorithm and thus the attack are set in an abstract, unrealistic micro-world. In this thesis I will investigate what the effects will be of lifting the attack to a more realistic setting. What I expect is that the attack becomes tougher or even impossible to be performed when more realistic restrictions are added. Next to that I'll implement and test the effects of one of the defenses suggested in the article by Barreno et al.. Summarized, the questions researched in this thesis are:

- What is the effect of lifting the attack as described by Barreno et al. (2006) to a more realistic context?
- What effect does the suggested randomization defense have on this attack in both the abstract and the realistic context?

The remainder of this thesis can be outlined as follows. In Section 2 the microworld and its classifier as used by Barreno et al. are described. This is followed by the theoretical attack on this classifier as well as some shortcomings of this attack related to the real world. In Section 3 this abstract world is lifted to a realistic context, and the effects of this are tested in Section 4 and 5. The randomization defense and its effects are discussed in Section 6.

The thesis finishes with conclusions and discussion on the results of the earlier sections in Section 7.

[1]In this thesis I'll refer to potential hackers / adversaries with 'he'. History teaches us there certainly have been female hackers, but their number is small (Adam, 2003) and it facilitates both the writing and the reading of this thesis to only make reference to adversaries with 'he'.

2 A theoretical framework

2.1 Microworld

In the paper of Barreno et al. (2006) a supportive hypersphere is used as an example of an online semi-supervised classifier for outlier detection. This hypersphere is an n-dimensional sphere in an n-dimensional space. In every dimension the sphere is fixed at the mean of the training-samples that were labeled as normal and it has a radius such that all data-entries that fall within the radius from the mean (or: center) of the hypersphere are labeled as normal (supported) and all the others abnormal (unsupported / outliers).

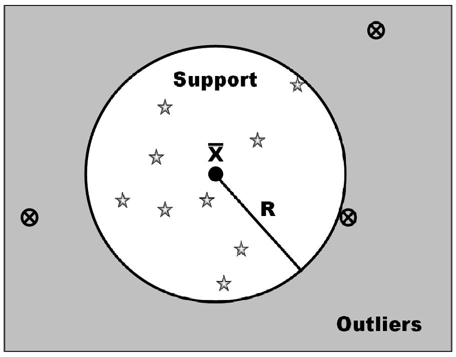

Figure 1: The mean of the hypersphere is denoted by $\overline{X}$ and its radius by R. Objects that fall within the sphere are labeled as normal, ones that fall outside the sphere are outliers and labeled as abnormal. *Image obtained from the article by Barreno et al. (2006).*

When a new data point is given to the algorithm it is classified as either normal or abnormal. The normal ones are used to recalculate the center of the sphere. This can be characterized in the following formula, where $\overline{X}$ is the center of the sphere, n the number of data points already in the hypersphere, i the iterator over all dimensions and v the added data point:

$$\overline{X_i} = \frac{n\overline{X_i} + v_i}{n+1}$$

This denotes that the whole sphere will move in the direction of the newly learned data-point, making the classifier adaptive to new types of normal activities and able to recognize them as such.

2.2 A theoretical attack

The description of the microworld is followed by the definition of a theoretical attack on this on-line unsupervised learning algorithm, abusing the adaptivity of the hypersphere. The attack is only possible if the adversary knows how the learning algorithm is implemented, what the current state of the classifier is and if he has, to some extent, control over the data set. This almost directly implies an attack from an *inside-man*, although it might be possible for an adversary to get a good impression on these things with *explanatory attacks*, for instance by sending data points to the classifier and mapping them with the returned labels (Barreno et al., 2006).

Assuming the attacker has sufficient knowledge about the algorithm, the current state of the classifier and that he has (to some extent) control over the data set he could start the following attack.

He chooses a data point (*target point*) that is correctly labeled as *abnormal* by the classifier. He wants to get this target point labeled as *normal.* He determines the point that is located exactly where the line between the mean of the hypersphere and the target point intersects with the border of the supportive hypersphere. This point we will call the *attack point.* The algorithm labels the attack point as normal and recalculates the mean of the hypersphere with this newly learned information, hereby moving the sphere in the direction of the target point. After enough iterations of this process the center of the sphere has moved so close to the target point that the sphere now embodies it and the target point is successfully labeled as normal.

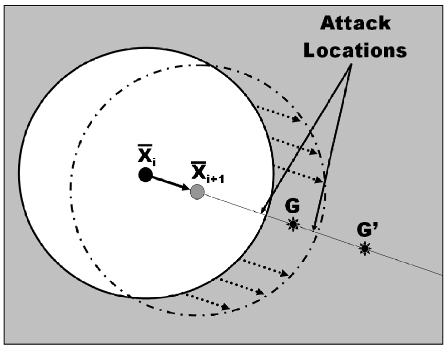

Figure 2: Presenting data points at the shown attack locations will cause the sphere to move to the target (or goal) point denoted by G or a possible secondary goal denoted by G'. *Image obtained from the article by Barreno et al. (2006).*

2.3 Some remarks on this attack

The theoretical attack described in the previous section takes place in an *abstract* world where the number of dimensions is n and the values in every dimension can vary from minus infinity to plus infinity. The classifier used for the classification is a hypersphere. All these properties make the model a welcome tool for mathematical calculations but it is far from a *realistic* environment.

A hypersphere is not a very effective classifier in an environment where normal data points are not clustered in one group (this depends on the feature set you are using, but in most realistic cases they will not). Even more, it is assumed that the attacker can choose from an infinite number of attack points to reach his goal, but we have to bear in mind that these attack points will have to be generated in a real world. The values in one dimension (remember that every dimension corresponds to a feature of the data point to be classified) could indeed have values between minus and plus infinity, but there are also features that have restrictions. For instance, the feature 'word-count' for e-mails cannot have the value of 144.23, nor can the feature 'launch-time' for a virus be minus 3.2 seconds in a realistic world.

There may also be interactions between the features (or dimensions, in the view of a hypersphere) that restrict certain values. For instance, you have a classifier for detecting malicious connections to a computer network. Once the feature *protocol-type* is set to *http*, knowing that *http* and *ftp* are both protocol types and you cannot use both in the same connection, the feature *ftp_transmitted_bytes* cannot be different from zero.

3 Lifting the attack to a realistic setting

To lift the abstract example of the hypersphere by Barreno et al. to a realistic context, changes were made in the environment (i.e. the dataset), the semi-supervised classification algorithm and in the attack algorithm. These changes are described in the following subsections.

3.1 A realistic environment

In order to have a dataset with realistic restrictions a copy of the DARPA-dataset from the KDD '99 cup was obtained (SIGKDD, 1999). This dataset was gathered from a nine weeks simulation of a LAN network of the US Military Air force. One that attracted many hackers, because over 80% of the 5 million connections available in the training set were labeled as an attack. Every *connection* is a set of 41 raw features, all either a binary, natural, positive real or symbolic value. For a full overview of these features see Appendix A.

Since this dataset is very realistic and often used in research (Sung & Mukkamala, 2003; Yu & Tsai, 2004; Lazarevic et al., 2003) I decided to use this dataset to lift the environment of the attack to a more realistic setting.

Connections were either being labeled as *normal* or received one of the 24 possible attack labels. Since this thesis is not on creating an outstanding classifier, all the labels of the malicious connections were changed to *abnormal*, no matter what type of attack it was.

Some of the features were symbolic, but since the algorithms that will be described in Section 3.2 cannot handle this type of data these features were quantified. For instance, the values of the symbolic feature *protocol* would be transformed to the quantitative features f_0, f_1 and f_2 as can be seen in Table 1.

Protocol	Quantified		
	f_0	f_1	f_2
tcp	1	0	0
icmp	0	1	0
udp	0	0	1

Table 1: Changing symbolic values to quantitative features

In this case the symbolic feature is represented with 3 values in the data point. Since there were two more symbolic features that were converted this way, this extended the feature-vector defining a connection from 41 to 119 features.

To prevent time-consuming training and testing a random set of 10.000 connections was selected from KDD's training and test set respectively, each containing 5.000 normal and 5.000 abnormal connections.

3.2 A realistic classifier

The hypersphere used by Barreno et al. was mainly used because it simplified the mathematical calculations they wanted to perform. In a real dataset such as the KDD Cup '99 however, the set of normal data points is not that easily covered with one hypersphere; it is far more scattered through the multi-dimensional feature-space.

In order to make a classifier that has sufficient performance, is semi-supervised and has the properties that still give way to the attack as described in Section 2.2, a combination of LVQ and kMeans was implemented to classify the dataset.

These algorithms and their combination are described in the following subsections.

3.2.1 LVQ

LVQ, or Learning Vector Quantization, is a particular type of artificial neural network (Kohonen, 2003). Every class is represented as an output node in the output layer. When a data point needs to be classified its feature-values are set on the corresponding input node. So in the case of classifying quantified connections from the KDD-cup we have an input layer consisting of 119 nodes. This networked is fully connected, i.e. every input node has a weighted connection to every output node, see Figure 3.

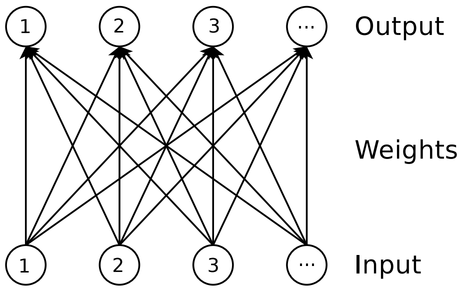

Figure 3: An LVQ-network consists of two layers. The input layer represents the data point to be classified, the output layer has a node for every existing class.

To select a winning output node that defines the class of the data presented on the input nodes, the following calculation is computed for the activation of every output node j:

$$output_j = \sum_{i=0}^{n} (input_i - weight_{i \to j})^2$$

The output node with the lowest activation is selected as the winner and matching class for the input vector. Looking closely at the algorithm one notices this will be the output node that has the weight vector which resembles the input vector most. The weight vector from the input layer to an output node could therefore be seen as the prototype of the class this output node belongs to.

During training the resulted outcome is compared with the correct label given and the weights from the input layer to the winning output node are updated. When the given label was correct, the weights are updated in the direction of the input values, when it was incorrect they are updated in the opposite direction. The algorithm also ensures values will remain positive, since values below zero do not exist in the data set, as can be seen in Figure 4.

Data points are presented in a random order. As for initializing the weights from the input to the output level, the first occurrences of every output class are set as initial values of the weights corresponding to this class.

The training continues until all data points from the training set have been seen by the algorithm or until some minimum recognition percentage has been reached. This last method may lead to other problems such as overtraining and since the focus of this thesis is not on the performance of the classifier, training was halted after all data points in the training set were seen once.

```
for (int i = 0; i < N_FEATURES; i++) {
  double update_value = eta * (input[i] - weight[i][o]);
  if (given_label == correct_label)
    weight[i][o] = max(weight[i][o] + update_value, 0.0);
  else
    weight[i][o] = max(weight[i][o] - update_value, 0.0);
}
```

Figure 4: The updating of winning output node `o` with learning rate `eta` (η) in LVQ.

3.2.2 kMeans

A semi-supervised classifier consists of a phase in which the classifier is trained using labeled data, followed by a phase in which it is updated using unlabeled data. LVQ is now used for the first phase, but since it requires the correct labels to update it cannot be used in the second, unsupervised phase. Because kMeans is able to update without having the correct labels and because it resembles the classifier as used by Barreno et al., it is used for this unsupervised stage.

kMeans starts with a set of initial centroids, each of them representing a class. A *centroid* is a vector of which each element is the average value of elements belonging to its class. See for instance Table 2.

Members of class c			**Centroid for class c**
m_1	m_2	m_3	
13	14	15	14
-2	0	-1	-1
4	6	-2	2.666667

Table 2: Example of a centroid for class c to which three data points belong.

When a new data point is presented, kMeans calculates which centroid is the closest using the Euclidean distance algorithm, and gives the corresponding label to the new entry. The center of this winning centroid is now recalculated to keep its position in the center of the data points of his class, which can be seen in Figure 5. Because of limits on computational power, previous assignments to a centroid are not recalculated to check whether they are still closest to this centroid. An example of this algorithm in a 2-dimensional world can be seen in Figure 6.

```
for (int i = 0; i < N_FEATURES; i++)
  centroid[i][o] = (n_points[o] * centroid[i][o] + input[i]) /
                   (n_points[o] + 1);
n_points[o] += 1;
```

Figure 5: The updating of winning centroid `o` in kMeans.

3.2.3 Combining LVQ and kMeans

Both the vector of weights in LVQ that point to the same output node and the centroids in kMeans can be seen as prototypical vectors of a certain class. The only difference is that in kMeans the number of data points already belonging to that centroid is taken into account. As mentioned earlier, LVQ can be used for supervised and kMeans for unsupervised learning. The algorithms are combined to create a semi-supervised classifier as follows:

- Prototypical weight vectors are constructed with a labeled training set using LVQ.
- For every output node of LVQ, the number of entries in the training set is counted that are classified to that node (using the weights resulted from the training phase).
- Every weight vector serves as an initial centroid for the kMeans algorithm. The initial number of points assigned to a centroid is set equal to the number of entries counted in the previous step.

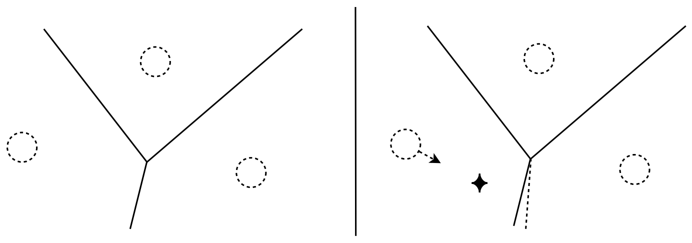

Figure 6: Example of kMeans in a 2-dimensional world. Shown are three centroids and the borders where the distances to two centroids are equal. A new data point is classified with the closest centroid, and this centroid is then moved in the direction of the new data point.

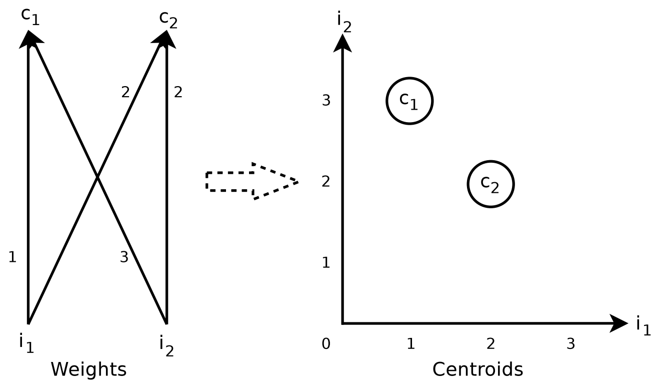

Figure 7: An example of converting weights in an LVQ network to centroids for the kMeans algorithm in two dimensions.

An example of this in two dimensions can be seen in Figure 7.

To ensure that the classifier is able to separate *normal* and *abnormal* connections even when both classes are scattered through the feature-space, several (8) centroids (or, in LVQ: output nodes) were used per class.

3.2.4 Data standardization

Because values for features varied within different domains (for example, *logged_in* varied between 0 and 1 and *src_bytes* varied between 0 and values as high as 6000) these were standarized by taking the logarithmic value and dividing the result by the highest logarithmic value found for that feature in the training set. In this way all features gained a value in $[0, 1]$ (except for possible values of a feature in the test set that are higher than the highest value to the same feature in the training set).

3.3 A realistic attack algorithm

3.3.1 The algorithm

With the adaptation of the classifier, the algorithm determining the attack locations (*attack points*) to move the normal centroids was required to change as well. Two features of the new classifier forced the attack algorithm to change.

The first feature is that in the new classifier, as opposed to the hypersphere model, not only normal but also abnormal data points are represented. To give a clear understanding why the old algorithm will not suffice, consider the example of a 2-dimensional feature space in Figure 8 of a normal centroid N, an abnormal centroid A and a target point t.

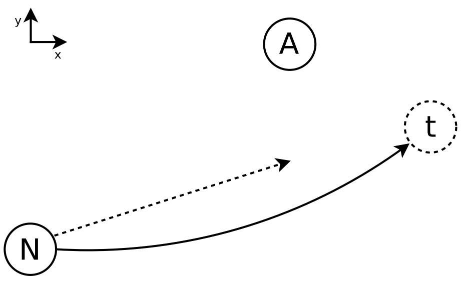

Figure 8: Figure of a normal centroid (labeled N), an abnormal centroid (A) and a target point (t) in a 2-dimensional world.

Consider the x-values of every element in Figure 8: N_x, A_x and t_x. In the same way a_x will be the x-value of the attack point returned by the algorithm. The old algorithm would place the attack point's value at the border of the hypersphere, which translates in this context to a value closer to N_x than to A_x and as close as possible to t_x. Doing this for both the x- and y-coordinate of the attack point however, N would move along the dotted arrow but stop at its head, since a_x will never be able to grow larger than A_x. This occurs because the attack does not profit from the fact that there are more dimensions. When the algorithm includes the other dimensions in determining the value of each feature in the attack point, the attack could follow a path like the solid arrow, and will be able to reach the target point.

The second feature of the used classifier hindering the attack is the usage of multiple representations for each class. Consider the situation in Figure 9, where a secondary abnormal centroid is present, closer to the target point than the normal centroid, but on a greater distance from the target point than the primary abnormal centroid.

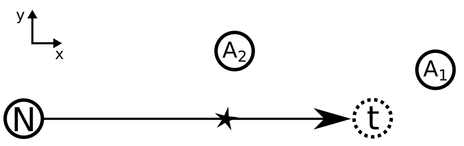

Figure 9: Figure of a normal centroid (labeled N), two abnormal centroids (A_1 and A_2) and a target point(t) in a 2-dimensional world. The star denotes the planned attack point.

A_1 is selected by the algorithm as the abnormal centroid closest to the target point. Since it is not in between the normal centroid and the target point, the algorithm will decide to place its attack at the location denoted with the star. However, there is another abnormal centroid A_2 that is not closest to the target point but still is closer to the attack point returned by the algorithm than N is. Thus the attack point will receive the label of this centroid and the normal centroid is not moved at all. This results in the fact that the attack point should not only be closer to N_1 than to A_1, but closer to N_1 than to any other abnormal centroid.

The new attack algorithm is described in pseudo code in Figure 10. The algorithm will return a data point called the *attack point*, which will cause the normal centroid closest to the target point to move in the direction of this attack point and thus in the direction of the target point. First the attack point is set equal to the target point. As long as it is closer to an abnormal centroid than to a normal centroid, the features of this attack point are shifted one by one to the value of the closest normal centroid[2]. When the attack point comes to lay closer to the normal centroid, the last changed feature is recalculated for optimal displacement of the normal centroid, for which the algorithm can be found in Appendix B. An example of the attack algorithm can be found in Appendix C.

```
function AttackAlgorithm(Targetpoint t) returns AttackPoint {
        AttackPoint a = copy of t;
        Centroid    N = closest normal centroid to a;
        Centroid    A = closest abnormal centroid to a;
        List        l = list describing processing order of features;
        while (distance(a,A) < distance(a,N)) {
                int i = next index from l;
                a[i]  = N[i];
                A     = closest abnormal centroid to a;
        }
        update the last changed feature in a;
        return a;
}
```

Figure 10: Pseudocode of the attack algorithm

3.3.2 Increasing realism

At this moment the attack algorithm is updated to function in the new context, however its results remain unrealistic since the resulting attack point's features can still have impossible values.

There are two points in the algorithm where these unrealistic values arise. The first one is when a value from the normal centroid is copied, since this is an average value and could be for instance 0.236 for a binary feature. The second point is when the value of the last changed feature is recalculated, as described in Appendix B.

These parts can be restricted and in that way help to test the hypothesis that indeed more iterations are needed when the attack is lifted to a realistic setting. In order to do so, three different types of the algorithm were created:

No restrictions This algorithm type has no restrictions whatsoever on the values of the attack point and is thus most comparable with the attack algorithm used by Barreno et al. No adaptations were made to the algorithm as described in Section 3.3.1.

Good types This algorithm required that the attack points returned by the algorithm had correct values regarding the type of the feature associated with it. These types were binary, natural and real values. *Real* values were not altered, *natural* values were either floored or 'ceiled' depending on which operation moved it closer to the normal centroid and *binary* values were set to the target point's value if this kept it close to the normal centroid and set to the floored or 'ceiled' normal centroid's value otherwise.

Symbolic This algorithm had the same restrictions as *good types*, but had the extra restriction that in a sub-vector of the attack point representing a *symbolic* feature only one element had a value of 1 and all the others were set to 0. The element set to 1 was set equal to that element in the target point if possible and equal to the element with the highest value in the normal centroid otherwise.

[2]Several policies were tried in which order to process these features. The one used was to start with the feature that had the biggest Euclidean distance between the normal centroid an the target point, followed by the second most distant value et cetera.

4 Tests

Training was done using the LVQ algorithm and the training set described in Section 3 with an η of 0.01. The percentage correctly classified connections on the test set was 93.6% with 99.5% on the normal and 87.6% on the abnormal class. Since recognition on the training set was 99.3% for normal and 96.5% for abnormal this suggests possible overtraining on the training set or it may be due to the 14 types of attack that were only available in the test set.

All abnormal connections in the test set were used as target points for the attack algorithms. The kMeans algorithm as described in section 3.2.2 was used to update the centroids obtained from the LVQ algorithm with the attack points from the three different attack algorithms described in Section 3.3.2.

For every target point the number of iterations required for each attack algorithm to get it labeled as being normal was logged, as well as the Euclidean distances between the centroids (of both the normal and abnormal class) closest to the target point, the target point and the attack point after every 50 iterations and at the end of the attack.

As a computational bound the attack was stopped when after 20.000 iterations the target point was still labeled abnormal. Target points that were already closer to a normal centroid at the start of the attack were skipped and not taken into account in the statistical tests, as well as target points with an Euclidean distance smaller than 1 to an abnormal centroid, since these could certainly not be successfully attacked within the limit of 20.000 iterations. This limit of 1 was chosen because this can be caused by a single feature-change, e.g. a binary feature being on instead of off, and can thus be seen as such small a difference that the target point might as well have been on top of the abnormal centroid.

When the attack is halted because of the computational bound, the distance between the attack point and the target point can be seen as an indicator. If the attack point is closer to the target point than the closest abnormal centroid is, it is sure that the attack would have succeeded had it not been stopped.

5 Results

The test results are summarized in Tables[3] 3 and 4.

	Attack algorithm		
Measure:	**No restrictions**	**Good types**	**Symbolic**
n_total	5000	5000	5000
n_skipped	3591	3591	3591
n_unneeded	620	620	620
n_success	789	789	758
n_failed	0	0	31
n_no_indication			29
mean iterations	92.90	193.92	836.28
mean ||a-t||	0.04	0.09	2.03

Table 3: Results of the 5000 target points processed by the different attack algorithms. Shown are how many of these points were and were not successfully reached within 20.000 iterations, as well as the average value for certain measures, explained below.

In these tables, the variables stand for the following measures:

n_total refers to the total number of processed target points. This is equal to the number of abnormal data points in the test set and therefore has the same value for every algorithm.

n_skipped is the number of target points skipped because their Euclidean distance to the closest abnormal centroid was smaller than 1.

n_unneeded is the number of target points skipped because they were already labeled as normal instead of abnormal. This is 12.4% of all data, which could already be inferred from the performance of LVQ mentioned in Section 4. As the previous two items, this has the same value for every attack algorithm.

n_success is the number of target points that remained and were successfully labeled as normal within the upper bound of 20.000 iterations.

n_failed is the number of target points that remained and were not successfully labeled as normal within the first 20.000 iterations. Only the algorithm with the most restrictions had target points that fell into this category.

n_no_indication refers to the number of failed target points that showed no clear indication of being successfully reached had the attack not been stopped (as defined in Section 4).

mean iterations is the average number of iterations needed for a successful attack. Only the target points that were not in the 'skipped' or 'unneeded' category were taken into account. That this increase in iterations is significant is shown in Table 4.

mean ||a-t|| is the average distance between the attack point and the target point at the end of the attack. The same target points as in the category 'mean iterations' were taken into account. The lower this value, the closer the attack point had come to lay with the target point, and thus the easier the attack was. That this increase is significant is also shown in Table 4.

[3]For attacks that were stopped because of the computational bound, 20.000 was taken as value for the number of iterations

Comparison	**On measure**	**p**
μ(No restrictions) = μ(Good types)	Iterations	< 0.0005
	$\|\|$a-t$\|\|$	< 0.0005
μ(Good types) = μ(Symbolic)	Iterations	< 0.0005
	$\|\|$a-t$\|\|$	< 0.0005

Table 4: Statistical results on the number of iterations needed to get the target point labeled as normal and on the distance between the attack and target point after the attack was stopped. Tests were done with a student's t-test (paired).

Between the levels of restrictions there is significant increase in iterations, as well in distance between the final attack point and target point, as can be seen in Table 4. These results prove that an increased realistic context causes a significant grow in the number of iterations needed to successfully get an abnormal data point labeled as normal. The results on distance between the attack point and the target points suggest that this is indeed due to the limited number of attack points available.

6 Defense

6.1 Definition

Although the number of iterations increases with the realism of the context in which the attack takes place, even in the most realistic context tested in Section 5 some target points were still successfully classified as normal at the end of an attack. This implies that the attack remains a possible threat to the classifier. Barreno et al. (2006) suggest several defense-mechanisms for this attack, of which one will be implemented and tested in this section.

The implemented defense is the one that adds randomization. In the original attack this would mean placing randomization around the border of the hypersphere, such that the attack points become labeled normal one half of the time and abnormal the other half of the time. In the realistic setting described in Section 3 this can be translated to randomization placed on the areas where two centroids are on about the same shortest distance.

Consider centroid A being the closest centroid to some data point d and centroid B being the second-closest centroid. When they are at about the same distance to d, d should have about the same chance of getting labeled as A or as B. If d becomes closer to A however, it should have a higher chance to get labeled as A than as B. The distribution of these chances is symbolized in Figure 11.

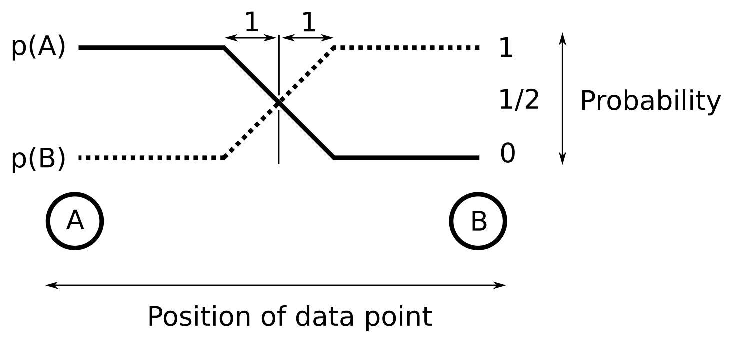

Figure 11: Probabilities for a data point's closest centroids A and B to be selected as a winner depending on the position of this data point.

It was chosen to let the randomization 'area' have size 1 because of the same reasons described in Section 4 to omit target points with a distance smaller than 1 to an abnormal centroid from the tests.

6.2 Results

For the attack algorithm with no restrictions on the attack points, the following sequence often occurred, as visualized in Figure 12.

The attack point is placed almost exactly in between the closest normal and the closest abnormal centroid, therefore both are about equally often selected as the winning centroid for the attack points. Therefore they both move towards the attack locations, meeting somewhere in the middle. But as the abnormal centroid moved towards the attack points, it moved away from the target point and another abnormal centroid came closer. The normal and the first abnormal centroid continue to move to the attack points, but the normal centroid will never be moved closer to the target point than this second abnormal centroid.

The same situation arises in the second attack algorithm that has the 'good types' for each feature. However in the third algorithm, with the extra restriction that also 'symbolic' aspects need to be taken into account, this situation does not arise that often as can be seen in the results in Table 5, where the number of iterations and failed attacks does not increase that much as compared to the other two algorithms.

This can be explained by the limitations in the number of possible attack points for this last type of algorithm. Because of all the restrictions in realism it is almost never possible for the algorithm to place

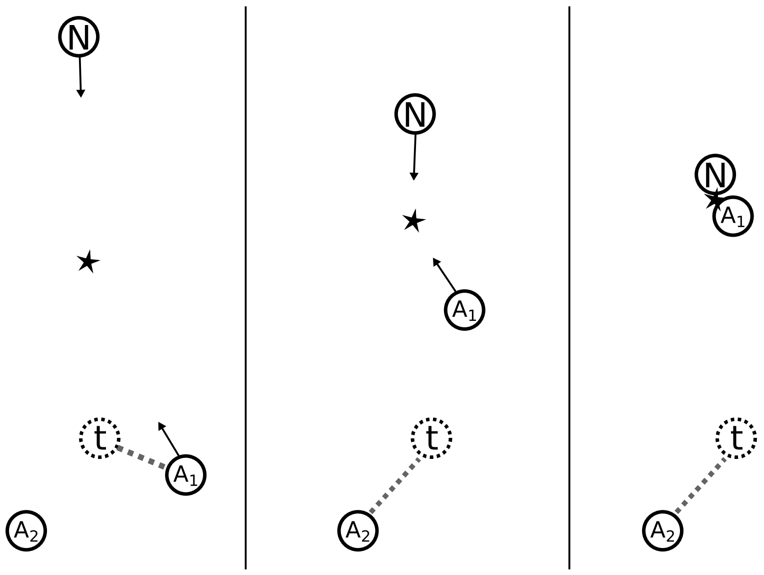

Figure 12: Often occurring sequence in the restriction-free attack algorithm when randomization defense is applied. The dotted gray line indicates the centroid closest to the target point. The star denotes the attack point.

	Attack algorithm		
	No restrictions	**Good types**	**Qualitative**
Mean iterations without randomization	92.9	139.9	836.3
Mean iterations with randomization	1131.5	1133.6	849.4
p (μ(With rand.) = μ(Without rand.))	< 0.0005	< 0.0005	< 0.02
n_failed without randomization	0	0	31
n_failed with randomization	44	36	32

Table 5: Results of the different attack algorithms when randomization defense is present. The defense had only a significant effect on the first two algorithms. Tests were done with a student's t-test (paired).

its attack points exactly in between the normal and abnormal centroid, but they will always be closer to the normal centroid. So much closer in fact that the difference in distance from the 'border' as denoted in Figure 11 was often bigger than 1 and thus the normal centroid almost always was selected as the winning centroid.

6.3 Conclusions

The randomization defense works in all three attack algorithms, however its influence seems to be smaller in the more realistic context. This rises the question whether the defense will have any success at all when even more restrictions are added to the algorithm. Another interesting question is what will happen if the attacker chooses his attack points just outside the randomization area.

Because the randomization defense increases the number of iterations needed for a successful attack, attacks may become more easily detectable. For instance an external, secondary network detecting suspicious adaptations in the centroids, another suggested defense (Barreno et al., 2006), might profit from this.

7 Conclusions and future research

In this thesis the theoretical attack on an AI-classifier as described by Barreno et al. (2006) was lifted to a realistic environment. The context was made more realistic in terms of the dataset, classifier and attack algorithm used. The results show that this led to a significant increase in the effort an attacker has to make to perform his attack successfully. However, even in the environment with the highest number of realistic restrictions there remained successful attacks, although with increased effort.

One of the defenses suggested by Barreno et al., adding randomization to borderline cases, was implemented and tested as well. In all levels of realism in the attack algorithms this lead again to a significant increase in the number of iterations needed for a successful attack. However this increase lessened with the increase in restrictions on the attack algorithm. This rises the question whether the defense will still have an effect in an even more realistic context.

Since both the increase in realism and the defense mechanism resulted in an increase in the number of iterations required for a successful attack, this could make the attack more detectable for a secondary classifier sensitive to abnormal changes in the primary classifier. Further research is required to investigate whether the creation of such a classifier is possible and whether the raised effort for successful attacks will make them more detectable.

Acknowledgments

I would like to thank dr. I.G. Sprinkhuizen-Kuyper and prof. dr. T.M. Heskes, my supervisors, for their support, suggestions and remarks.

Appendix A List of features in the KDD dataset

Here follows an overview of the features in the KDD dataset (SIGKDD, 1999):

Feature name	**Description**	**Type**
duration	length (number of seconds) of the connection	natural
protocol_type	type of the protocol, e.g. tcp, udp, etc.	symbolic (3)
service	network service on the destination, e.g., http, telnet, etc.	symbolic (66)
src_bytes	number of data bytes from source to destination	natural
dst_bytes	number of data bytes from destination to source	natural
flag	normal or error status of the connection	symbolic (12)
land	1 if connection is from/to the same host/port; 0 otherwise	binary
wrong_fragment	number of wrong fragments	natural
urgent	number of urgent packets	natural
hot	number of hot indicators	natural
num_failed_logins	number of failed login attempts	natural
logged_in	1 if successfully logged in; 0 otherwise	binary
num_compromised	number of compromised conditions	natural
root_shell	1 if root shell is obtained; 0 otherwise	binary
su_attempted	1 if su root command attempted; 0 otherwise	binary
num_root	number of root accesses	natural
num_file_creations	number of file creation operations	natural
num_shells	number of shell prompts	natural
num_access_files	number of operations on access control files	natural
num_outbound_cmds	number of outbound commands in an ftp session	natural
is_hot_login	1 if the login belongs to the hot list; 0 otherwise	binary
is_guest_login	1 if the login is a guest-login; 0 otherwise	binary
count	number of connections to the same host as the current connection in the past two seconds	natural
serror_rate	% of connections that have SYN errors	real
rerror_rate	% of connections that have REJ errors	real
same_srv_rate	% of connections to the same service	real
diff_srv_rate	% of connections to different services	real
srv_count	number of connections to the same service as the current connection in the past two seconds (from server)	natural
srv_serror_rate	% of connections that have SYN errors (from server)	real
srv_rerror_rate	% of connections that have REJ errors (from server)	real
srv_diff_host_rate	% of connections to different hosts (from server)	real

Table 6: List of the features in the KDD-cup '99 database.

Appendix B Recalculation of the last feature moved

In order to make the movement of the normal centroid as optimal as possible we need to recalculate the value of the last feature shifted, since it has most likely moved more than needed to make the attack point come closer to the normal centroid. This can be done with the following calculation.

We want to have the vector distance between the normal centroid and the attack point equal to the distance between the abnormal centroid and the attack point. Say that:

i	=	index of last changed feature
a	=	abnormal centroid's value of feature i
n	=	normal centroid's value of feature i
d_A	=	vector distance between the abnormal centroid and the attack point (without feature i)
d_N	=	vector distance between the normal centroid and the attack point (without feature i)
x	=	attack point's value of feature i

We want that:

$$(a-x)^2 + d_A = (n-x)^2 + d_N$$

resulting in:

$$x = \frac{a^2 - n^2 + d_A - d_N}{2a - 2n}$$

This calculation was used to determine x, the optimal value of feature i of the attack point. However, this sets d_A equal to d_N, but we want the attack point to be slightly closer to the normal centroid to ensure that it will move the normal centroid and not the abnormal one. Therefore we subtract 0.0001 from the value of d_A before calculating x.

Appendix C Example of the attack algorithm

In this section an example of how the attack algorithm in the realistic context is working is given in a 2-dimensional world.

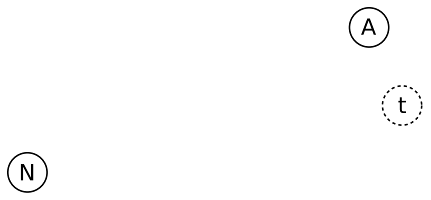

Figure 13: There is a target point t, with as closest normal centroid N and closest abnormal centroid A.

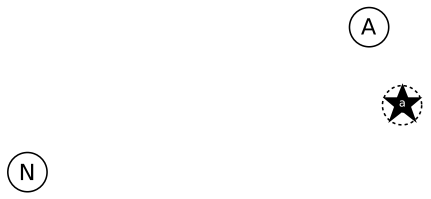

Figure 14: The attack point a is first set equal to the target point t.

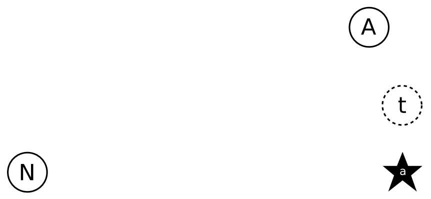

Figure 15: Since it is closer to A than to N, one feature is changed to the corresponding value in N.

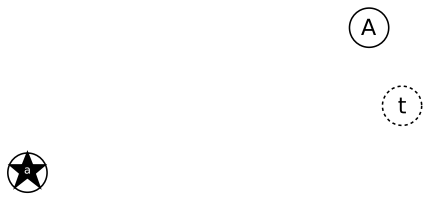

Figure 16: Since it is still closer to A than to N, again a feature is changed to the value of N.

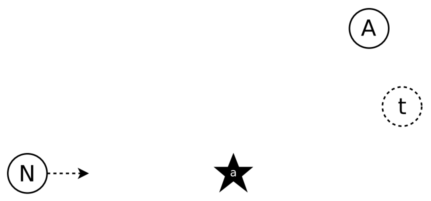

Figure 17: This last update caused a to be closer to N, and after updating the last adaptation using the algorithm described in Appendix B, the resulting attack point is given to the classifier causing the normal centroid to move in the shown direction.

References

Adam, A. E. (2003). Hacking into Hacking: Gender and the Hacker Phenomenon. *SIGCAS Comput. Soc.*, *33*(4), 3.

Barreno, M., Nelson, B., Sears, R., Joseph, A. D., & Tygar, J. D. (2006). Can Machine Learning Be Secure? In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security* (pp. 16–25). New York, NY, USA: ACM.

Kohonen, T. (2003). *Self-Organizing Maps* (3rd ed.). Springer.

Lazarevic, A., Ertöz, L., Kumar, V., Ozgur, A., & Srivastava, J. (2003). A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In *Proceedings of the Third SIAM International Conference on Data Mining.*

Powers, S. T., & He, J. (2008). A hybrid artificial immune system and Self Organising Map for network intrusion detection. *Inf. Sci.*, *178*(15), 3024–3042.

SIGKDD. (1999). Available from `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

Sung, A. H., & Mukkamala, S. (2003). Identifying Important Features for Intrusion Detection Using Support Vector Machines and Neural Networks. In *SAINT '03: Proceedings of the 2003 Symposium on Applications and the Internet* (p. 209). Washington, DC, USA: IEEE Computer Society.

Yu, Z., & Tsai, J. J. P. (2004). A Multi-Class SLIPPER System for Intrusion Detection. In *COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)* (pp. 212–217). Washington, DC, USA: IEEE Computer Society.