Adventures in Anomaly Detection

Shahzad Ali
cheeko@cs.cmu.edu

December 1, 1999

Abstract

Anomaly detection as a mechanism for Intrusion detection has been an active area of research. Its importance beyond the military setting is becoming more and more vital. However, there is a distinct lack of evaluation schemes for various anomaly detection tools. There has also been a lack of complete analysis of the techniques that can be borrowed from other fields for use in anomaly detection. In this paper, we present a simple mechanism for evaluating anomaly detectors. We design and implement three different anomaly detectors using a recently explored field of data mining and a novel approach of genetic programming. Using the simple model, we evaluate the performance of the detectors on synthetic data and report our results. It is shown the these new areas are promising, but there performance has to be taken together with the nature of data that they work with.

Contents

List of Figures

1. Introduction and Motivation

Intrusion detection has been an active area of research for more than a decade now. The importance of Intrusion Detection Systems (IDS) have grown tremendously recently because of our dependence on electronic forms of data. Sensitive information, which has to be kept secure, is kept in an electronic form on computers. Military and the Government have been the most vocal of the IDS users, but more and more private organizations are realizing the importance of such a system. There is an increase in the amount of data for which confidentiality, integrity and availability are critical characteristics. Patient records kept in hospital data-bases are a object of constant attacks. So is information stored about buying patterns of consumers, their financial status and banking habits as well as on-line browsing patterns. It has been generally known that building a secure system will definitely reduce the probability of a successful attack, but it will not completely eliminate such a incident. IDS assume that **some** of the probable attacks will get through and will have to be detected by some sophisticated techniques. Another threat that is becoming real, is insider attack. All the security built into the system will not be able to detect an attack that is initiated by someone with the allowed access. Such attacks are impossible to avoid and it is imperative that they be detected. Timely detection of an attack can lead a supervisor to either stop the attack in the middle, track down the possible intruder, or revert the system back to a consistent state after the attack.

Desgining and implementing anomaly detectors was the major goal of this project. The first phase of the project involved making a probablity based detector called a **Frequentist**. The second phase was the design and development of another anomaly detector based on some machine learning technique. A data-mining approach and a genetic programming approach were used to build two detectors during this phase. The results of all of these detectors were analyzed using standard techniques.

Despite its growing importance, very little research has been done in comparing the performance of any detector. The evaluation of a detector depends heavily on the data-set used. During the course of this project, we ran all tests on a fixed data-set that has been synthesized with great precision to depict a real life data-set. This set includes training and test sets of eleven different types and are discussed in more detail in Section 4.

The rest of the paper is divided as follows. In Section 2, we present a background of the problem and related work. To make the process of evaluation of IDSs better, we present a simple model in Section 3 that can be used to test anomaly detectors. Next we present the detectors that were built during this project in Sections 5, 6 and 7. We conclude by identifying future work opportunities and our experience in Section 8.

2. Background

Anomaly detection as an approach for IDS was first proposed by Dorothy Denning in 1987 [2] . It was suggested that by observing abnormal behavior in the data, anomolies can be detected. This is based on the concept that abnormal data is necessarily different from normal data and if some *measure* of normal/abnormal data can be deduced, further anomolies can be detected. The general architecture for using any anomaly detector is as shown in Figure 1. The detector is given some training data, which is used to get an idea of what normal/abnormal means. A test data is given to the detector after the training period and the results are compared against the actual anomolies that were present in the test data. There are many metrics that are of interest after the detector has run over the test data. Some of these that we are interested in are hit rate, miss rate and false alarm rate. They are defined as:

- **Hits** of the total number of anomalies injected in the data, how many did the detector detect as anomalous?
- **Misses** of the total number of anomolies injected in the data, how many did the detector **not** detect as anomolies?

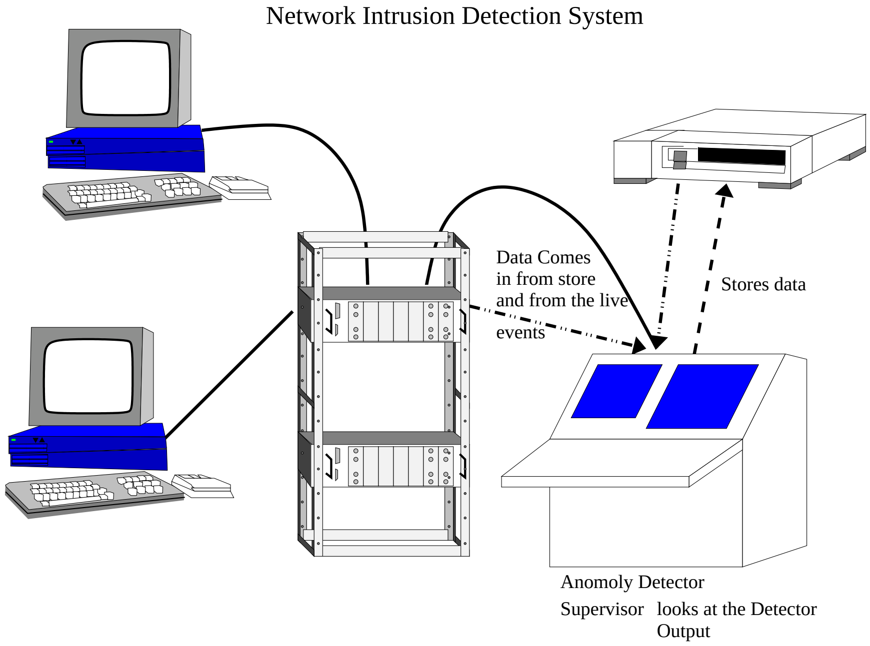

Figure 1: Generic Architecture for using an Anomaly Detector.

- **False Alarms** how many **NON**-anomolies in the data did the detector flag as anomolies?

$$Hit\_rate = \frac{hits}{total\_anomolies} \times 100$$

$$Miss\_rate = \frac{misses}{total\_anomolies} \times 100$$

$$False\_Alarm\_rate = \frac{hits}{total\_non\_anomolies} \times 100$$

IDS have been built with different techniques and their taxonomy is not readily available. Some of these include Neural Networks, Bayesian Networks, Hidden Markov Models, Genetic Algorithms etc. We present here, a taxonomy based on the learning approach that is used by the IDS. There are two major categories of IDS based on the learning algorithm that they use. These are

1. Unsupervised Learning
2. Supervised Learning

As evident from their names, an unsupervised learning anomaly detection scheme works by training the detector on normal data. No notion of an anomaly is given to the detector. This makes the detector deduce a notion of normality and any significant deviation from that behavior is marked as an anomaly. In the second category, the detector is trained on normal as well as some of the abnormal data. The detector is also given an indication of what part of the data is anomalous. This helps the detector in deducing a notion of normal as well as abnormal data.

Although anomaly detection is mostly concerned with the first category of learning, there are examples of the second category scattered all over the literature. During this project, we designed one detector from the first category and two from the second category.

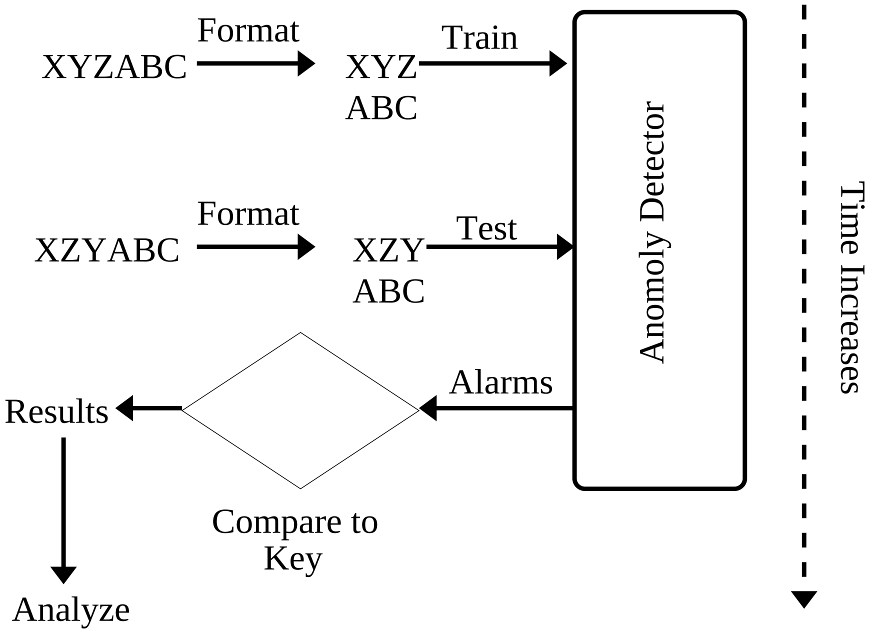

Figure 2: Simple Model for evaluating Anomaly detectors.

3. Model for an IDS using Anomaly Detection

Due to the lack of formal evaluation of ADs, we present a relatively simple scheme by which detectors can be tested. An important part of any evaluation scheme is the data used. It is important that the data be reproducible and it has some reflection of what reality is. The data has to include a training set which is the normal data-set. It also includes a test data set which has anomolies inserted artificially. Both these data sets should have some index by which one can measure their relative difficulty for any detector. Once such a data-set is available, we can just format the data-set for each type of detector. We then train the detector on the training data and then test it on the test data. Performance indices such as hit rate and false alarms as well as others should be defined and used to evaluate different detectors. This simple model is shown graphically in Figure 2.

4. Description of the data sets

As mentioned earlier, we used a data-set which has eleven different sets. The difference between the data-sets is the amount of noise that the data has normally. This is characterized by a **regularity index**. A high regularity index means that the normal data has a lot of randomness built in it. A low regularity index means that the normal data is very regular. The data-sets we used have varying regularity indices from 1 to 11 with an alphabet size of 6. Test data-sets included the training data-sets injected with some anomolies at various positions. A key is also provided with each data-set to discern where the anomaly was injected. Each of the data-sets contained about 2.5 million records with the test data containing about 160 anomolies of size 3.

5. NIDES based Frequentist Detector

NIDES is an IDS built at SRI [3]. It is built up of two components which serve as the anomaly detection back-engines for NIDES. One of these is a statistical anomaly detector often referred to as the frequentist. The concept of the frequentist is pretty straight-forward. Frequencies of various components in the training data can be thought of as a measure of the normalcy of the data. Thus if the frequency of the various components in the test data are significantly different from that of the training data, an alarm can be raised. NIDES enhances this general concept and applies it to various categories of data after the modifications. The data could be numeric, categorical or logical. NIDES has slightly modified versions of the same statistical detector for different kinds of data. As categorical data is what we will be interested in for the scope of this project, we concentrate on it mostly.

5.1. Description of the NIDES Statistical Components for Categorical Data

The component is trained on a set of categorical data. A frequency distribution of the categories is calculated. NIDES periodically updates this distribution while it is running. Any new set of data, that is to be tested for intrusions, is then passed through this detector. For n records of the test data-set, the detector calculates a value $g_{m,n}$ for each category m in the data, given by

$$g_{m,n} = 2^{-r} g_{m,n-1} + \frac{I(n,m)}{N_r}$$

where $I(j,m) = 1$ if the j^{th} record indicates category m and 0 otherwise and

$$N_r = \sum_{j=1}^{n} 2^{-r(n-j)}$$

and r =decay rate for the records, so if half-life is 100 records, then r = 0.01.

From these values of $g_{m,n}$ a running statistic Q_n is kept,

$$Q_n = \sum_{m=1}^{M} \frac{(g_{m,n} - f_m)^2}{V_m}$$

where f_m = the frequency of category m in the training data and M is all the categories and

$$V_m = f_m \frac{(1 - f_m)}{N_r}$$

This value of Q_n is compared against a frequency distribution of the Q-Statistic that is a historical distribution of values of Q_n. Various thresholds can be set so that if the current value of Q goes above certain limit, an alarm is raised.

5.2. Description of our variant of the NIDES Frequentist

One of the most obvious techniques that can be used to build a detector are based on probablity. Frequentist is such a detector where statistical measures are calculated to detect anomolies. Our frequentist detector is based on the NIDES categorical statistical detector. However there are some changes that we have made. The equation for the calculation of Q_n is changed as follows.

$$Q_n = \sum_{m=1}^{M} \frac{(g_{m,n} - f_m)^2}{f_m}$$

This allows us to use the popular chi-squared statistic instead of maintaining a distribution of the Q-Statistic. We ran our frequentist on the training data and let it collect a frequency distribution of the categories. Then we ran it on the test-data and for each additional record that the detector reads, it calculates a new value of $g_{m,n}$ and Q_n. It then runs the value of Q_n through a chi-squared detector and gets a chi-squared normalized value of Q_n. After running through all the test-data, the values of Q_n are judged against a threshold, T, and alarms are raised whenever the value of the chi-squared normalized Q_n goes above the threshold. The results are then compared against the keys to evaluate the performance of the detector on that particular data-set.

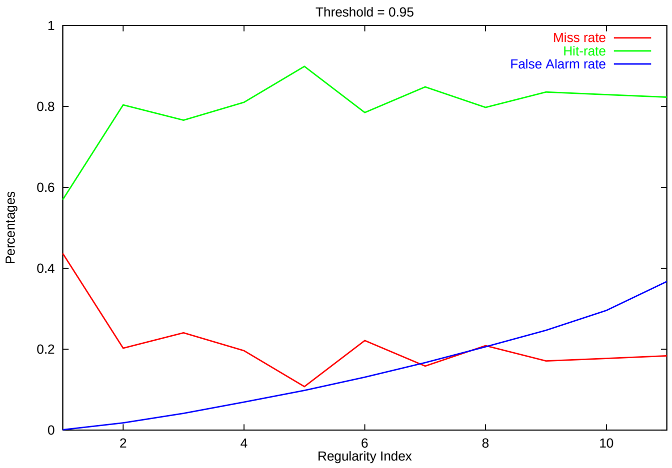

Figure 3: Performance of the NIDES based frequentist with Threshold = 0.95

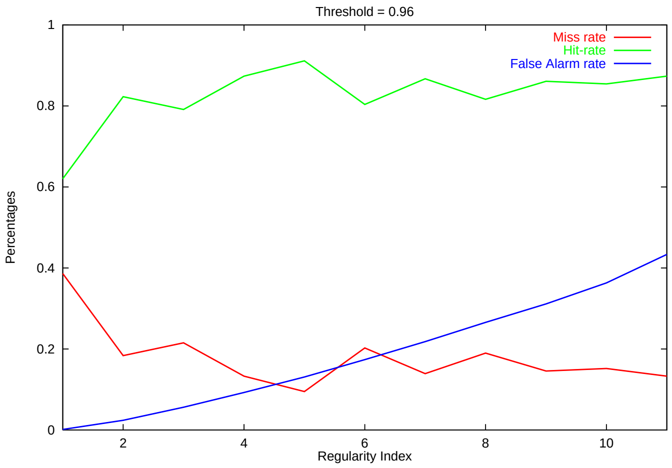

Figure 4: Performance of the NIDES based frequentist with Threshold = 0.96

5.3. Results of the Frequentist Detector

The performance of this detector was evaluated with the test data described. Simulations were run that calculated the hit, miss and false alarm rates for the different regularity indices in the data. Results are presented in Figures 3 to 6. The first thing that can be observed from these graphs is that as the regularity of data increases, the false alarm rate constantly rises. This is a reiteration of the fact that the regularity of base data effects the results of any anomaly detector significantly. Also observable from these graphs is that when the threshold is increased from 0.95 to 0.98, the results change with hit rate increasing together with the false alarm rate.

We can also plot the ratio between the hit rate and the false alarm rate as a measure of the detector accuracy. The results of these graphs are shown in Figures 7. The graph is drawn with the y-axis in log scale. This shows that the accuracy of the detector at various thresholds decreased as the regularity index of the underlying data increased. Also evident from this graph, is the decrease in the accuracy of the detector as we increase the threshold. This is basically due to the increase in the false alarm rate as the threshold is increased. This will be compared to the other detectors through the rest of the paper.

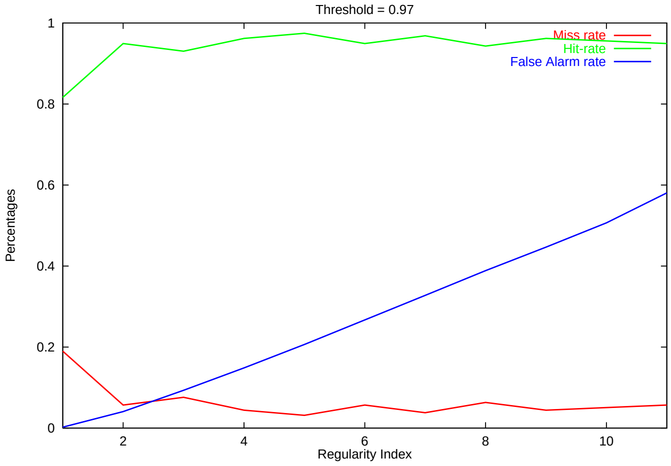

Figure 5: Performance of the NIDES based frequentist with Threshold = 0.97

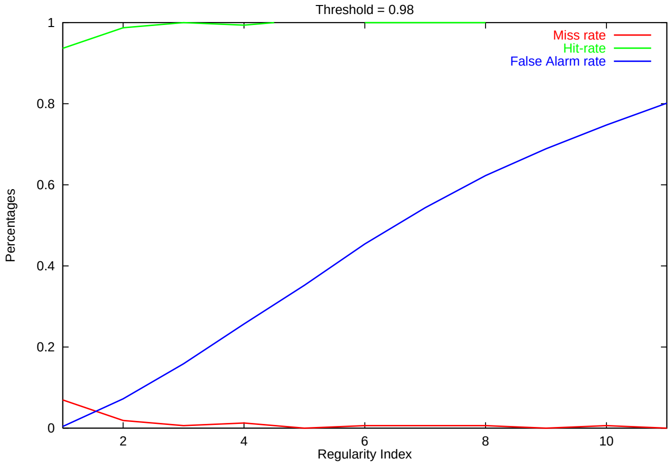

Figure 6: Performance of the NIDES based frequentist with Threshold = 0.98

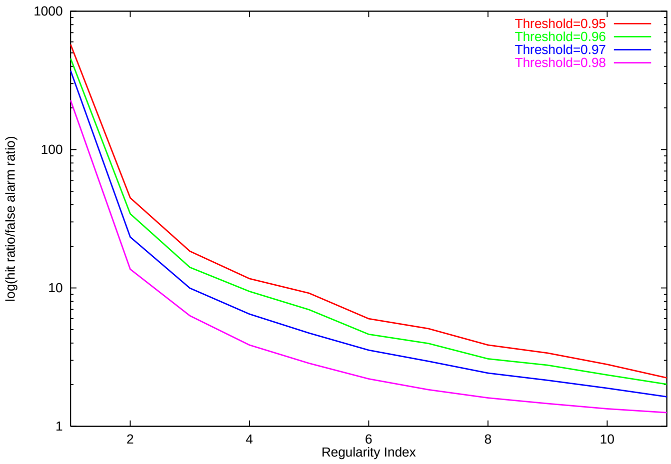

Figure 7: Accuracy of the NIDES based Frequentist Detector

5.4. Conclusions

As expected, our results show that the frequentist behaves reasonably well on data that has a low degree of randomness in it. As soon as the degree of randomness increases, the results of the detector become less and less confident. However, if real data is very regular, then it makes sense to use a frequentist because of its ease of implementation. Our detector is based on 600 lines of C-code with some perl scripts. However the simplicity is at the cost of making the detector very predictable and defeating its prediction quite simple. It can only detect the variations in the frequency of certain categories which might not be the real difference between normal and abnormal data. Also, deciding a threshold for the significance of the detectors results is a judgement call. There is no clear mechanism by which this threshold can be adjusted. As shown earlier, the performance of the detector depends heavily on the threshold chosen. The accuracy of the detector goes down as the threshold is increased because of the increase in the false alarm rate. Thus the accuracy can be controlled directly by the threshold and having no clear mechanism for doing that is a drawback.

6. A data-mining based detector

Data-mining is defined in [5] as

Using historical data to discover regularities and improve future decisions

Applications of data mining have included detection systems such as credit card fraud detection. However, the interest in Data mining tools for intrusion detection is very recent. In their work [9] [8], Lee and Stolfo try to identify a general approach for using data mining tools for intrusion detection. They also carry out some experiments using a well known data mining tool, RIPPER [1]. The results that they presented were on data collected from system calls and *tcpdump*. Their results are very interesting and they propose looking into this approach further. As mentioned earlier, a data mining based detector carries on supervised learning where it is shown some cases of anomolies and it learns from this data set. If the anomolies are very regular, the performance of the detector will be better. However, we suspect that the performance of the detector will be effected if new kinds of anomolies are injected into the data. We tried to investigate some of these questions.

6.1. Description of a data mining based detector

We present here a brief description of the detector used in the study in [8]. The detector is based on RIPPER which is a rule learning program. Using data from system call traces (normal and during an intrusion) and tcpdump data (normal and during a network intrusion), the rules were induced. For the induction process, 20 percent of the abnormal (anomalous) data was also used. The rules were then tested on the rest of the abnormal data and a threshold was set to alarm an intrusion if the rule fired with a specified significance level.

For our experiments we used another tool called CN2.

6.2. Description of our data mining tool: CN2

For our project, we used another data mining tool called CN2 [7] [6]. This tool is very similar to RIPPER as it is also rule based. A couple of different approaches have been tried out for data mining tools. Decisions trees are among the first of such techniques, which work by selecting different attributes of the data to specialize the result. Thus the tree grows from a single node and traversing down the tree with the attributes of any particular example gives the class that example belongs to. To limit the size of the tree, entropy is used to specialize a node. When the information gained from specializing a node is below a minimum, the node is not specialized at all. Eventually, for any example data set with n classes, the tool outputs a set of rules for each class.

Another common approach for data-mining tools is to use *complexes*, which can be thought of as a combinations of different paths in a decision tree. This class of algorithms works over the complete set of training data (that includes part of the abnormal data) and with each complex it covers part of the data-set. It removes the *cover* of a complex and finds other complexes on the remaining data.

CN2 combines these two approaches and searches for complexes along with specializing each node in the decision tree. Thus each node in the tree is a complex. The decision to specialize a particular node in a tree can be made based on different criteria. Entropy is the most obvious, but CN2 uses a Laplace accuracy measure to determine if a rule [1](basically a complex) is accurately describing a data set.

$$LaplaceAccuracy = \frac{(n_c + 1)}{n_{tot} + k}$$

where k is the number of classes in the domain (two in our case, normal and abnormal), n_c is the number of examples in the predicted class c by the rule and n_{tot} is the total number of examples covered by the rule. This proves to be a better measure of accuracy of a rule than entropy.

Another issue with rule-based schemes is whether the rules are ordered or unordered. In ordered rules, the first rule in the list that fires, predicts the class of the example. In an unordered rule set, all rules are tried and the one that fires with the highest confidence is the one that predicts the class of the example. CN2 uses both approaches and considers the unordered scheme better than the ordered scheme.

6.3. Results of the data-mining detector

The data had to be re-formatted to present it to CN2. There were options while reformatting data as we had to decide on a pattern length to give to CN2 as an example. We used pattern lengths of 3 to 13. This means that for example, for a pattern length of 5, the data looked like

```
#p1 p2 p3 p4 p5 class
A B C D E y;
A B C D F y;
A A D D E n;
B B C C E n;
```

where each record is a stream of data and the class determines whether this data is an anomaly or not. Thus the first record is normal data while the third is an anomaly. The decision is made based on the keys that are given with all the data sets. In all experiments, only 20 percent of the test data (data with anomolies injected) was used. The tests were then run on the remaining 80 percent of the test data and the results tabulated.

Figure 8 represents that results obtained by running the test data through the detector based on an ordered rule learning algorithm. Surprisingly, the results of the detector were very accurate. The detector has negligible percentage of false alarms (almost 0.00001 percent). However, the hit rate of the detector was low, ranging from 0 to 0.25. This suggests that the detector can probably not be used alone in an IDS. Its results have to be taken together with a different detector in such a way, that when this detector sets an alarm, it is almost 100 percent certain that there is an anomaly.

We also ran the detector with unordered rule learning algorithm but the results were even worse. Figure 9 shows that the hit rate never got above 10 percent. Interestingly, the false alarm rate was still 0 percent, signifying the accuracy of the data mining scheme in general.

Although unexpected, we still observed that the data-mining approach produces results which are very conservative. Although being accurate, it fails to detect a large percentage of the anomolies. To get a better estimate of how this will be effected by the data that the detector

[1] A rule predicts a class c based on some values of the attributes.

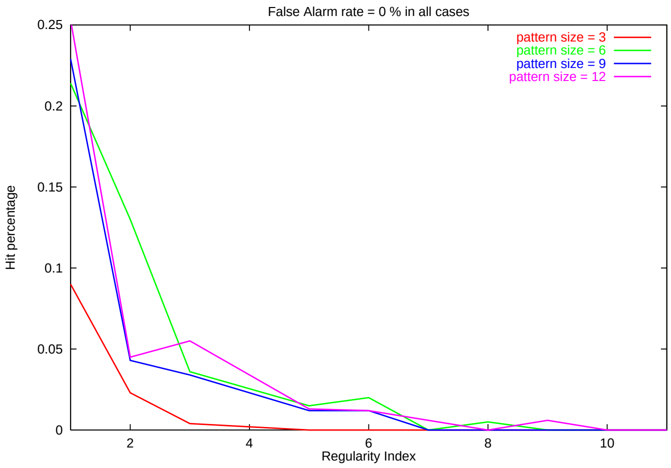

Figure 8: Performance of the ordered rule learning data mining detector

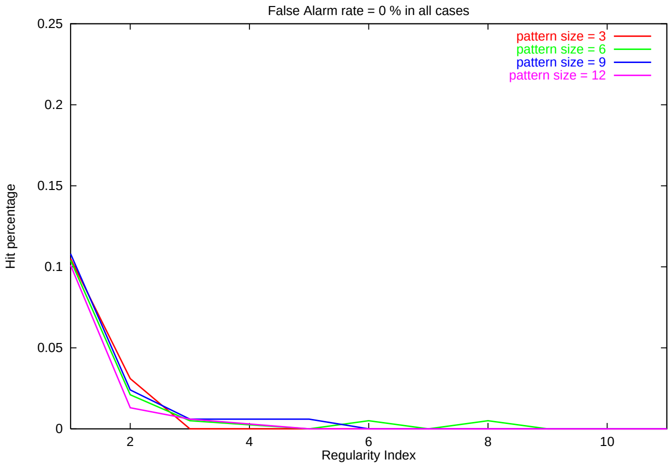

Figure 9: Performance of the UN-ordered rule learning data mining detector

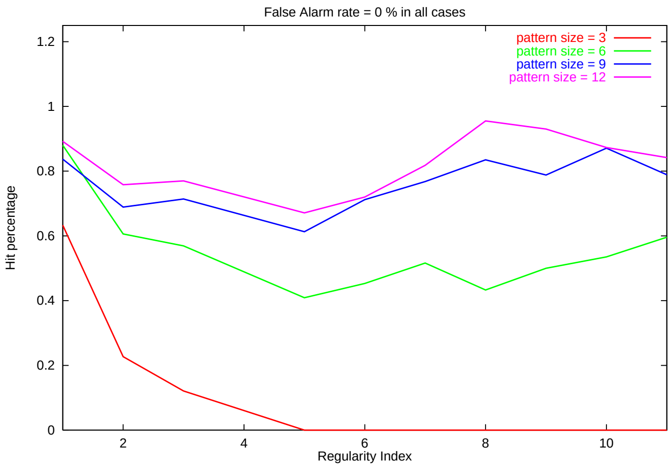

Figure 10: Reference performance of the ordered rule learning data mining detector

is trained with, we trained the detector with the test data (with the anomolies) and tested it with the same data. As expected, it performed well, detecting most of the anomolies and never raising a false alarm. The results are shown in Figure 10. With pattern sizes greater than 3, the detector performs well, even on the high end of the regularity index.

6.4. Conclusion

From the results we achieved, it can be concluded that the data mining based detector is very accurate in its prediction. Its alarm can be almost certainly assumed to be an anomaly. However, it does miss quite a few anomolies. We believe that these are the novel anomolies that the detector has never seen. Thus, if this detector is trained continuously, it can detect most commonly occurring anomolies but will not be able to detect novel scenarios.

7. A Genetic Programming based detector

Another approach of using supervised learning to build anomaly detectors is using Genetic Programming. GP is based on the more general Genetic Algorithms approach but there are significant differences between the two. There has been some work on detectors using GA [4] and a tool called **GASSATA** has been built. GASSATA uses audit trails to detect intrusions based on a genetic algorithm. The major problem in a GA approach is to come up with a binary representation of the problem and a evaluation function to judge the performance of various solutions. In his work, Ludovic uses GASSATA on a command sequence and reports interesting results.

7.1. What is Genetic Programming?

Genetic programming started in 1992 with Koza introducing the concept. The ideas in genetic programming are very similar to those in genetic algorithms but there are some major differences as well. The general approach for both GA and GP is to evolve a program through the natural process of selection. These techniques are based on the phenomenon of natural selection. A population of solutions, which have not been created by the programmer explicitly, are tested against each other using an evaluation function. The next generation of solutions is developed from the current generation using techniques such as mutation and crossover.

The differences between GA and GP is that GA chooses to represent the solution to a problem as a string of bits. GP takes this concept a step further and represents solutions to problems as

A	-3
B	-2
C	-1
D	1
E	2
F	3

Table 1: Translation of symbols to integers

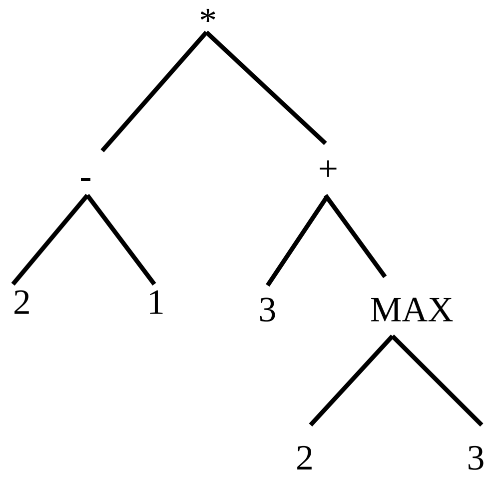

Figure 11: Sample GP tree for pattern size 3

trees. This versatility make a GP approach better solutions in general. The two main features of a GP approach are the representation of the problem and the evaluation function called the Fitness function used.

7.2. Representation of the problem: Making trees

The data that we presented to the GP is formatted in the same way as it was for the data-mining approach with varying pattern sizes. To represent the problem at hand, we use nodes, some of which are called terminals while others internal. We converted the data into numbers, taking the translation shown in Table 1. The terminals were then chosen to be integers from 1 to n-1, where n is the pattern size used. Thus if the pattern size is 3, the terminals are 1,2 and 3 and for a record A B C, terminal 1 will have a value of -3(A), 2 will have a value -2(B) and 3 will have a value -1(C).

This representation is chosen to make the GP calculation easy and to keep the resulting numbers small. Similarly, the resulting classes are given numbers: greater than 0 for an anomaly and less than 0 for normal. This means that if a GP evaluates a particular record to a value greater than 0, it considers it an anomaly.

For the internal nodes, we could have taken any operators that can work with numbers. We chose arithmetic operators $(+, -, /, *)$, relational operators$(>, <, !=, ==)$, minimum, maximum and square root. We think that this provides the GP enough flexibility to carry on its functions. Thus a potential GP for pattern size 3 could look like as shown in Figure 11. The value of this GP for a record like D A C will be $(A-D)*(C+max(A,C)) = (-3-1)*(-1+max(-3,-1)) = 8)$. As this value is greater than 0, it will be evaluated as an anomaly.

7.3. Fitness function

The next step is choosing an appropriate fitness function to evaluate the performance of a GP. The way to get a value for the performance of the GP is to run each GP on all the records in the training set. Note that the training set does include some of the test data as well, for this approach uses supervised learning. For each record we can get an estimate of how well the GP performed. We can then add up all these estimates for the entire training data to get a fitness value for the GP. We first used a basic evaluation function

$$fitness = \forall_{all\ training\ records} false\_alarms + misses$$

This measure is some what like a standard deviation of the prediction from the actual. For each wrong prediction, we added 1 to the fitness. This turned out to be a very conservative approach, especially given that there were only about 0.1 percent anomolies in the training data. The best GP in this case would be one that will always predict normal and thus incur a fitness of number of anomolies, which was really small. As the goal of the GP evolution is to reduces the value of fitness, we changed the fitness function to

$$fitness = \sqrt{false\_alarms + k \times misses}$$

where $k \approx \frac{anamolous\_records}{total\_records}$. This function represents the data better and proved to be an effective choice for the fitness function.

7.4. Evolution Schemes

As mentioned above, a GP approach works by randomly selecting a generation of solutions. Each solution is a tree, with internal nodes and terminals. An example solution for pattern size 3 is shown in Figure 11 earlier. The solutions in the generation are evaluated and the best ones are chosen. A new generation is formed from the old one with a few evolutionary techniques. In this project, we used mutations and crossovers as our evolutionary schemes. In mutation, part of the tree changes randomly. In crossovers, two trees exchange some parts to make two new tress that are a combination of the original two. These techniques have proved to be sufficient for an intelligent evolutionary algorithm. They are represented graphically in Figure 12 and 13.

7.5. Results of the GP based detector

The strategy we used for evaluation was similar to that used in the other schemes. With the eleven data-sets we had, we took the training data and combined it with about 20 percent of the test data to train the GP. The best solution of that GP was then tested with the remaining 80 percent of the test data and results were compiled. We observed the hit, miss and false alarm rates and also observed the fitness function value for a GP on the training data and the test data.

7.5.1. Inputs to a GP

GP are excellent tools if one can determine what the correct values of the various parameters in the experiments should be to produce the best results. Some of these variables which are of interest include:

1. **Population Size**
2. **Number of Generations**
3. Crossover Probablity
4. Maximum Depth of the GP

The variables that have the most influence on the results are Population Size and Number of generations. We did experiments to choose reasonable values for these variables before carrying out the experiments on all the data. We will discuss this briefly now.

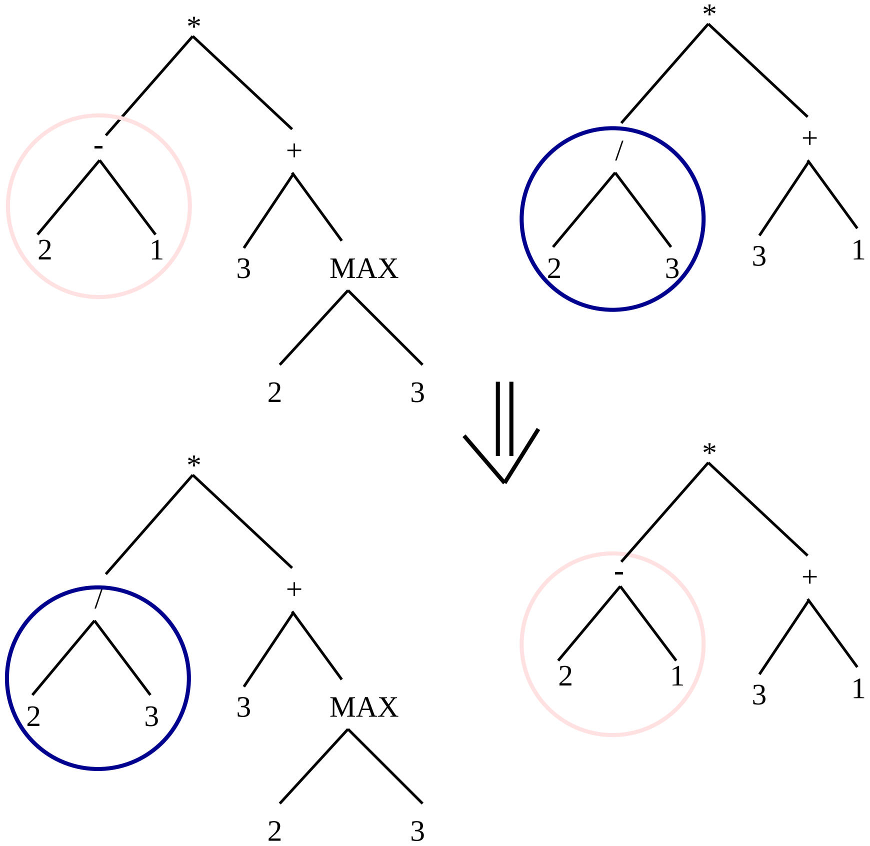

Figure 12: Crossover in a GP.

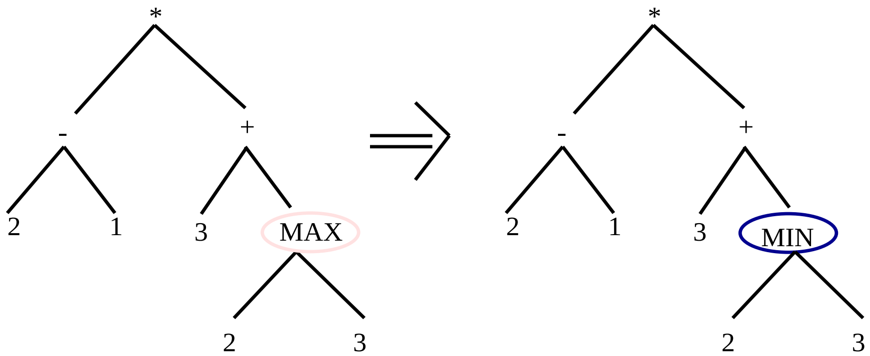

Figure 13: Mutation in a GP.

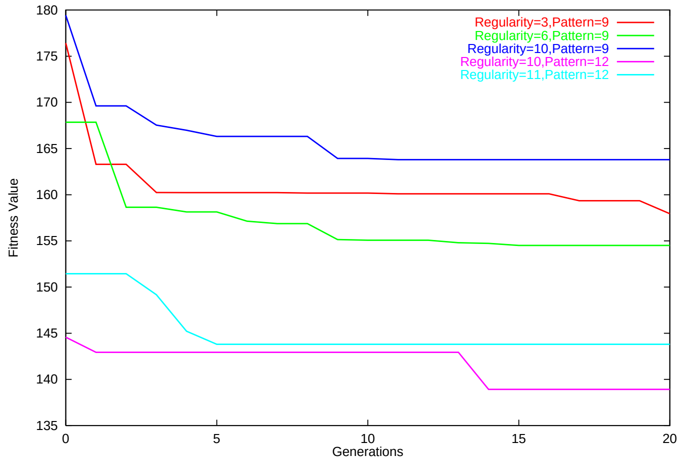

Figure 14: Fitness Value for GPs with generations for the training data: The graph shows that within 20 generations, an optimal solution is reached and there is no improvement. We could have also chosen something less than 20 without much effect, but 20 seems to be an upper bound.

7.5.2. Choosing the right values for variables

To choose the correct value for the number of generations to use, we did experiments in which we observed the fitness function value of the GP for each generation. The results are shown in Figure 14. It clearly shows that as the number of generations increase beyond 10, there is hardly any change in the fitness function value which means that the GP has converged to a solution. These results are for different regularity indices and pattern sizes and so this led us to believe that a generation size of 20 will be both sufficient and optimal for this set of experiments.

To choose the other variable, the population size for the experiments, we did experiments with population sizes of 10, 25, 50 and 100. In all these experiments the number of generation was set to 20, the regularity was 3 and the pattern size was 6. Figure 15 shows the results of these experiments. In this figure, we see that as the population size increase, the fitness value gets better. However, one thing that the results don't show is that for each data set at a regularity index, it takes as much as 1 week in some cases to get the results. At that pace, we would never have completed the entire project. So, because of these time constraints, we chose to carry out the experiments with a population size of 50. The results can be possibly improved with a bigger population size but this population size also gives promising results.

The other variables for the experiments were chosen as upper limits so as to minimize their effect on the results. We chose the crossover probablity to be 30 percent and the maximum depth to 45. We believe that these values are upper bounds and thus not restrictive. However, we haven't done any analysis to prove these claims.

7.5.3. Performance of the GP detector

We now present the actual results of the detector that was built using the values that we chose. All results in this section are with population size of 50, number of generations 20 and maximum depth of the GP tree of 45. Figure 16 shows the fitness value for various pattern sizes and regularities across generations. It is interesting to note that the corresponding figure for the training data is always a non-increasing function (Figure 14), this graph shows that the fitness of a later generation can be worse than that of its parent. However, one thing to note if the relative convergence of the fitness to one value in the 20 generations, supporting our claims earlier that 20 generations are enough.

Now we look at the fitness value of the final GP (after 20 generations) for various regularity indices. Figure 17 shows that the fitness value generally increases as the regularity in the

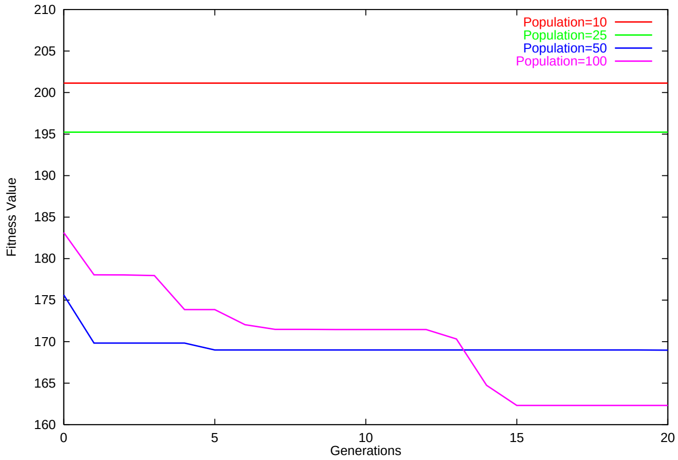

Figure 15: Fitness Value for GPs with Population Size: The graph shows that as the population size increase, the fitness of the Final GP increases. We chose a population size of 50.

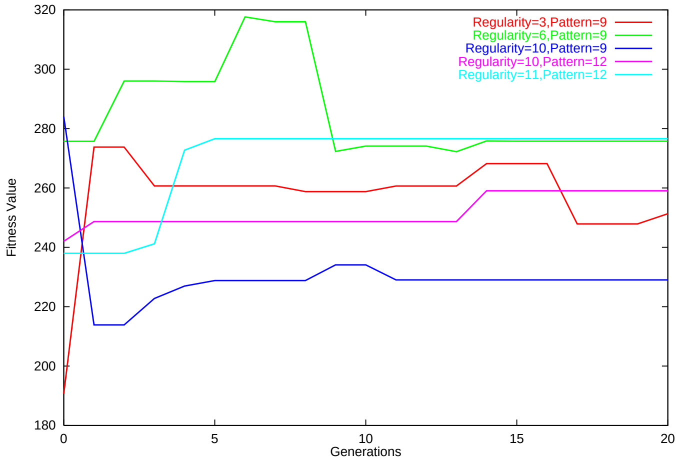

Figure 16: Fitness Value for GPs with generations for the Test data: Shows that 20 generations were enough to get convergence. Note that the fitness can get worse as generations progress.

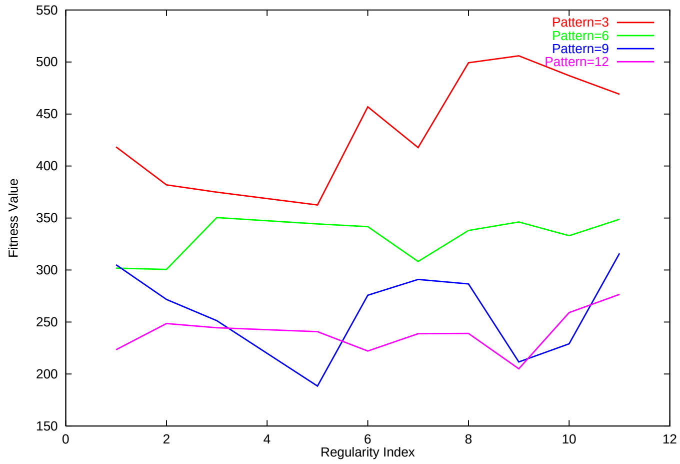

Figure 17: Fitness value of the Final GP for various regularities.

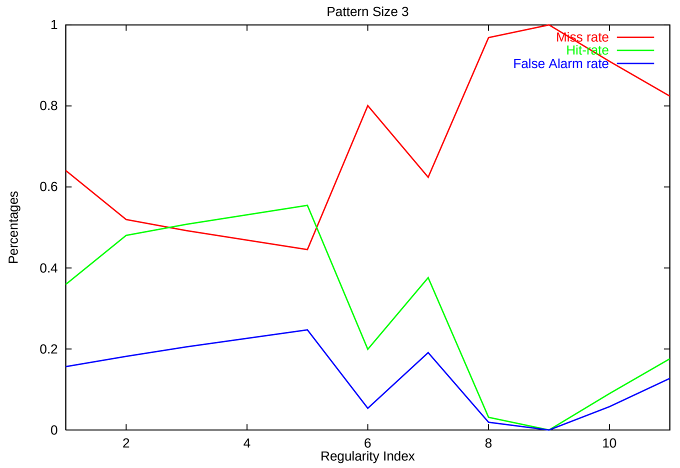

Figure 18: Performance for the GP detector with pattern size 3

base data increases. This is consistent with the performance of the other detectors getting worse as regularity increases. This figure also shows that different pattern sizes give different performances and pattern sizes 9 and 12 give better performance than pattern sizes 3 and 6. This is an interesting result as the anomaly size in the data is 3 and the alphabet size is 6. The results are also slightly skewed because of the approximation of k is the fitness calculation (Section 7.3). The difference is still more significant than the error introduced by the approximation.

Now we look at the hit, miss and false alarm rates for the GP detector. Figures 18 to 21 show the results of the detector with different pattern sizes. Clearly, the performance of the detector degrades as the regularity of the data increases. This is consistent with our previous results for other detectors. However, we see that the hit rate improves and it is best at a pattern size of 9. It is unclear from these graphs which pattern size is ideal for use.

To make the results a little more representative, we plot for each pattern size, the ratio of the hit rate to false alarms. That should give us a better idea as to the performance of the GP detector with various pattern sizes. Figure 22 shows that as the regularity increases, the ratio comes closer to 1 which means that the performance of the detector is no better than a random selection of records as anomolies. However, with regularities in the range of 3 to 5, it seems like the detector has good performance with pattern sizes of 6 and 12. It is interesting to note the

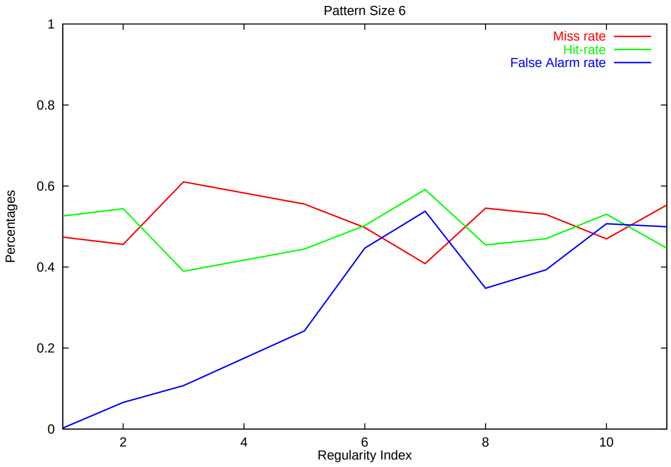

Figure 19: Performance for the GP detector with pattern size 6

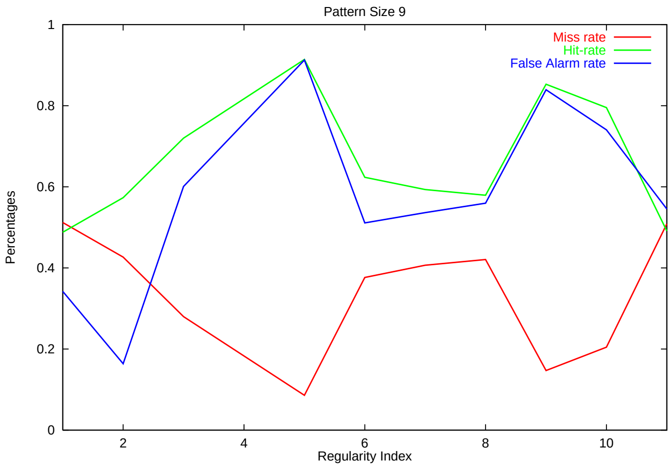

Figure 20: Performance for the GP detector with pattern size 9

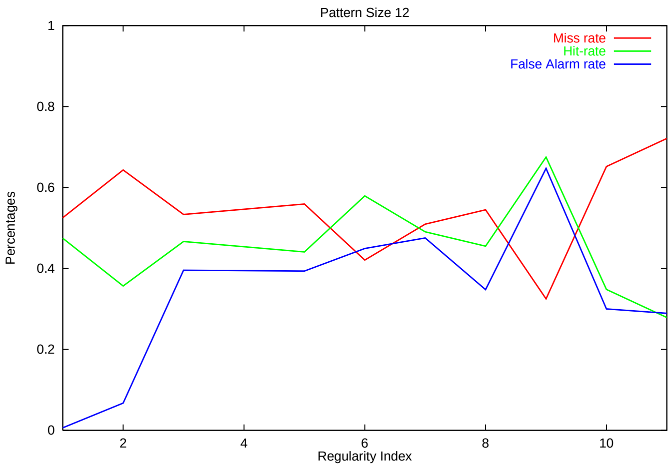

Figure 21: Performance for the GP detector with pattern size 12

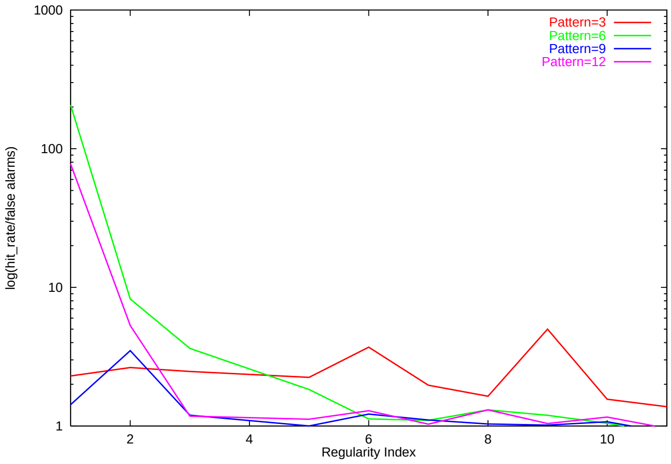

Figure 22: Performance of the detector for various pattern sizes. Note the log scale on the y-axis. Also, for pattern size = 3, the false alarm rate at regularity 9 was 0. A default value of 5 was used for the ratio.

consistency in the values for pattern sizes of 3 and 9, showing that they are probably not the best pattern sizes to use.

7.6. Conclusion

Genetic programming was tried as a possible approach for IDS. The analysis shows that like other supervised learning schemes, GP requires that the anomolies that the detector is trained on include most of the observed anomolies. It is difficult for the detector to detect a novel anomaly. Also evident is the dependence of the results on the pattern sizes used for the detection process. Different pattern sizes give different results and the decision of what pattern size to use is not automatic. Despite all these drawbacks, we think GP has promising results and its an interesting way of detecting anomolies. Further work needs to be done to explore its performance with other types of data.

8. Conclusions

In this project, we designed, implemented and evaluated three separate anomaly detectors. We used schemes with unsupervised learning techniques as well as supervised learning techniques. It was shown that the data chosen for the evaluation process is the key for a detectors performance. Also important is the effect of variations in the input parameters to the detector.

We analyzed Genetic Programming and Data mining as interesting approaches to anomaly detection and further need to investigate their appropriateness for real data sets. As future work, we also envision work on different synthetic data sets with different alphabets and anomaly sizes.

9. Acknowledgements

I would like to that Dr. Roy Maxion to assisting me with this project. Due to my limited knowledge of the field, discussions with him turned out to be very productive. I would also like to thank Kymie Tan who gave me pointers as to what detectors will be useful to implement. She also helped me with understanding the data-sets.

References

[1] W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th Internation Conference*, 1995.

[2] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.

[3] Alfonso Valdes Harold S. Javitz. The nides statistical component: Description and justification. Technical report, SRI International, March 1993.

[4] Ludovic Me. Genetic algorithms, an alternative tool for security audit trails analysis. Technical report, Supelec, France, 1992.

[5] Todd M. Mitchell. Machine learning and data mining. *Communications of the ACM*, 42(11), November 1999.

[6] Robin Boaswell Peter Clark. Rule induction with cn2: Some recent improvements. *Machine Learning - Fifth European Conference*, pages 151–163, 1991.

[7] Tim Niblett Peter Clark. The cn2 induction algorithm. *Machine Learning Journal*, 3(4):261–283, 1989.

[8] Salvatore J. Stolfo Wenke Lee. Data mining approaches for intrusion detection. Technical report, Computer Science Department, Columbia University, 1998.

[9] Salvatore J. Stolfo Wenke Lee. A data mining framework for building intrusion detection models. Technical report, Computer Science Department, Columbia University, 1998.