Using GA-based Feature Selection in Ensemble Classifier for Network Intrusion Detection

Vaibhav Sharma
Michigan State University
428 S. Shaw Ln., Room 3110
East Lansing, MI 48824
sharmav6@msu.edu

ABSTRACT

Classifying network connections as normal or anomalous is an important problem in the area of intrusion detection. The input to such detection systems consists of a large number of features. Reducing the number of redundant features in network intrusion detection increases accuracy and speed of detection. In this paper, we propose a genetic algorithm to select a subset of features for ensemble classifiers with a goal of improving the accuracy of the classifier. We use the NSL-KDD data set to train and test the ensemble classifier and show that the resulting feature subset can be used to improve classification accuracy.

Categories and Subject Descriptors

D.1.0 [**Software**]: Programming Techniques, General

General Terms

Algorithms, Performance, Security

Keywords

Genetic Algorithm, Ensemble Classifiers, Intrusion Detection

1. INTRODUCTION

Network Intrusion Detection is an important problem in the area of network security. Network intrusion detection systems are of 2 kinds - abnormal behavior detection in which the objective is to differentiate normal network traffic from abnormal network traffic and misuse detection in which the system already has a database of known signatures of network traffic seen during a network attack. Solutions to the former problem are good at detecting new attacks while solutions to the latter problem are good at identifying known attacks.
In this paper, we propose to identify feature subsets which improve accuracy of classifiers which address the former problem. For the purposes of training and testing, we choose to use the NSL-KDD Data Set[?]. This dataset reduces the number of redundant records in the training and test sets which makes it computationally affordable to run experiments on the entire training and test set.

2. RELATED WORK

The most similar approach to ours has been reported in Stein et al.[?] with the primary difference in our approach being that we demonstrate the effectiveness of GA-based feature selection with a ensemble classification technique. GA-based feature selection was also proposed by Kim et al.[?] which improves on kernel SVM classifier accuracy by finding the optimal parameter setting and optimal feature subset. A related but different problem was explored by Goyal and Kumar[?] which described a GA-based algorithm to classify smurf attacks with a detection rate of almost 100%. Hoque et al.[?] also used a GA-based algorithm for network intrusion detection and showed an accuracy of 99.4% when classifying DOS category of attacks in the KDD'99 dataset. Heba et al.[?] perform similar feature reduction using principal component analysis and use the feature subset with a Support Vector Machine(SVM)-based classifier on the NSL-KDD dataset.

3. GA-BASED FEATURE SELECTION FOR ENSEMBLE CLASSIFIERS

3.1 Ensemble Classifiers

We adopted an ensemble classification approach which is generally considered more robust and often more accurate than any single classifier[?]. An ensemble technique uses base learners in order to improve its accuracy and for a strong ensemble, reasonably strong base learners are a must. After trying out different base learners, we found the J48 decision tree classification algorithm had the highest kappa statistic during standalone training and testing when all the features were being used and hence decided to use the combination of the bagging ensemble classifier along with the J48 decision tree for our approach.

3.2 Using kappa as a metric

The kappa statistic[?] compares observed accuracy and expected accuracy and can be used to evaluate classifiers between themselves. It accounts for agreement with a random classifier and hence is less misleading than accuracy. The formula[?] for Kappa can be given as

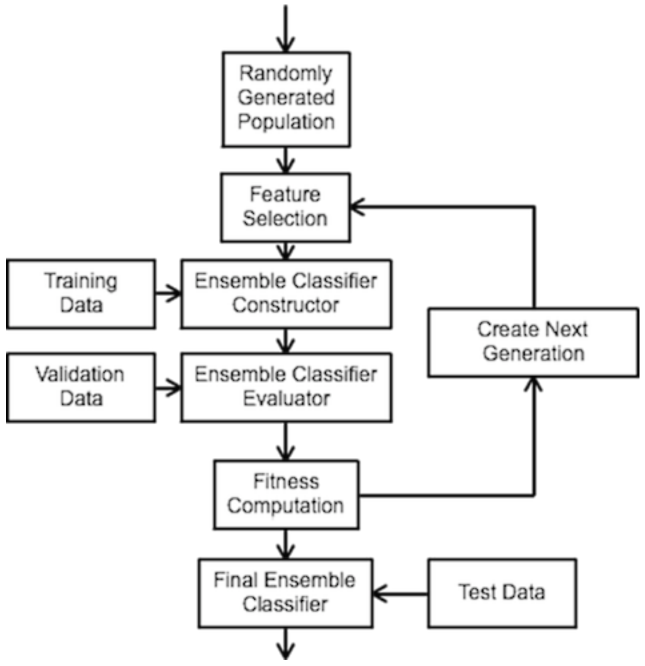
Randomly Generated Population
Feature Selection
Training Data
Ensemble Classifier Constructor
Validation Data
Ensemble Classifier Evaluator
Create Next Generation
Fitness Computation
Final Ensemble Classifier
Test Data

Figure 1: GA/Ensemble Classifier Hybrid approach

$$\text{Kappa} = \frac{ObservedAgreement - ExpectedAgreement}{1 - ExpectedAgreement}$$

3.3 Hybrid Algorithm

Parameter	Value
Population Size	20
Number of Generations	50
Crossover	2-point
Mutation if both parents different	1 per child
Mutation if both parents same	2 per child

Table 1: Parameters used in GA-Ensemble Hybrid approach

The premise we wish to test is if the training and testing data contains irrelevant or redundant features, can we use a GA to weed out such features and provide a subset of features which will be much faster and more accurate at classifying a network connection as a normal connection or as an anomaly. This approach is based on the wrapper model [**?**] in which we search for an optimal feature subset specific to the problem of network intrusion detection. Our approach is detailed in Figure **??**. We list out the parameters used for this approach in Table **??**. We create a initial population randomly where each candidate's length is equal to the number of features present in the network data. For every candidate, any feature has a gene representation of a 0 or a 1, wherein a 0 means that feature is not to be used during classification whereas a 1 means the represented feature is to be used during classification. For each candidate, a Ensemble classifier is built. Each such classifier is trained on the KDD training data and validated on the KDD test data. The fitness of this candidate is its kappa measure[**?**]. The higher the kappa measure, the better the fitness of this candidate. Once the fitness of all the individuals have been computed, there are 2 ways to move forward that we explored and found both to produce good candidates. The 2 methods have been named GA-1 and GA-2 and are explained in Sections 3.4 and 3.5 respectively.

3.4 Crossover in GA-1

The algorithm we used for this GA can be described as follows

Algorithm 1 Elitist Crossover

```
 1: procedure GA-1
 2:     $candidates \leftarrow$ reverse sorted list of $candidate$ objects
 3:     $i \leftarrow 0$
 4:     $len \leftarrow lengthof\,candidates$
 5: $top$:
 6:     $parent1, parent2 \leftarrow candidates[i], candidates[i+1]$
 7:     $child1, child2 \leftarrow crossover\text{-}2\text{-}pt(parent1, parent2)$
 8:     if $parent1 = parent2$ then
 9:         $mutated\text{-}child1 \leftarrow mutate(mutate(child1))$.
10:         $mutated\text{-}child2 \leftarrow mutate(mutate(child2))$.
11:     else
12:         $mutated\text{-}child1 \leftarrow mutate(child1)$.
13:         $mutated\text{-}child2 \leftarrow mutate(child2)$.
14:     $candidates[len-i-1] \leftarrow mutated\text{-}child1$.
15:     $candidates[len-i-2] \leftarrow mutated\text{-}child2$.
16:     $i \leftarrow i + 2$.
17:
18:     if i > len/2 then return ;
19:     goto $top$.
```

This algorithm tries to maintain elitism while performing a single bit mutation to each offspring. Due to this, after about 25 generations, the population tends to be dominated by mostly similar elite candidates which lowers the diversity of the population. In order to push the GA to explore new regions of the feature subset search space, we perform a 2-bit random mutation on each offspring if both the elite parents were the same.

3.5 Crossover in GA-2

The algorithm we used for this GA can be described as follows

Algorithm 2 Average Crossover

```
 1: procedure GA-2
 2:     $candidates \leftarrow$ reverse sorted list of $candidate$ objects
 3:     $i \leftarrow 0$
 4:     $len \leftarrow lengthof\,candidates$
 5: $top$:
 6:     $parent1 \leftarrow candidates[i]$
 7:     $parent2 \leftarrow candidates[len-i-1]$
 8:     $child \leftarrow crossover\text{-}2\text{-}pt(parent1, parent2)$
 9:     if $parent1 = parent2$ then
10:         $mutated\text{-}child \leftarrow mutate(mutate(child))$.
11:     else
12:         $mutated\text{-}child \leftarrow mutate(child)$.
13:     $candidates[len-i-1] \leftarrow mutated\text{-}child$.
14:     $i \leftarrow i + 2$.
15:
16:     if i > len/2 then return ;
17:     goto $top$.
```

This algorithm tries to maintain elitism while still trying to maintain diversity in the population. Elitism is maintained because the 1st parent is an elite candidate in the population whereas diversity is maintained because the 2nd parent is among the least fit candidates in the population. Similar to GA-1, this algorithm also tries to maintain diversity by adding an extra mutation to the child if both the parents were same.

4. EXPERIMENTS AND ANALYSIS

4.1 Fitness over time

We ran both the GAs described in Sections 3.4 and 3.5 using parameters described in Table **??**. We used the classifiers from the Weka library[**?**]. Since each classification took roughly 2 minutes to be completed, doing 20 classifications per generation for 50 generations took about 34 hours to complete for each run. During initial runs, we observed having any mutation rate of less than 1-bit mutation per offspring caused the population to converge in less than 10 generations. We also observed the rate of convergence was dependent on the base learner too. Using a simple but fast base learner like a DecisionStump[**?**] caused the population to converge too quickly(less than 10 generations). Another observation was that using GAs as described in Sections 3.4 and 3.5 above allowed us to cache the fitness calculated for the elite individuals in previous generations and thus saved some fitness computation time. Over multiple runs, the best observed kappa measure using GA-1 was 0.7723 and using GA-2 was 0.7902 both after 50 generations. As shown in Figure **??**, the fitness of the best candidate in each generation always first plateaus for a few generations and then rises abruptly. During this plateau, the average fitness of the entire population drops a bit before finally getting a push in the higher fitness direction as shown in Figure **??**.

4.2 Increase in Accuracy

We used the best candidate feature subset found after 50 runs by each GA with 3 different classifiers, Naive Bayes, J48 Decision Tree, Bagging, to check if the subsets resulted in improvement in accuracy. Each of the f-measure comparisons are shown in Figure **??**,**??**,**??**. As it can be seen, while Naive Bayes shows an improvement in accuracy, the Decision Tree and Bagging classifiers benefit the most from the feature subset selection. This can be explained by the fact that the J48 Decision Tree was used as a base learner along with the Bagging ensemble classifier in both the GAs. This shows the choice of base learner and ensemble classifier does have an impact on how much improvement a candidate feature subset causes in the accuracy of a final standalone classifier. Using the feature subset, the Naive Bayes classifier classified 3% more of the test data correctly and the J48 Decision Tree and Bagging classifiers both classified 4.8% and 5.1% more of the test data correctly. While these percentage increases may seem small, since the test data set consisted of about 22,000 instances, the actual increase in the number of correct classifications was significant.

4.3 Comparison

Finally, in Table **??**, we also show how frequent each gene was over the course of various generations for GA-1. We find that these frequencies mostly match with those calculated in Stein et al.[**?**]. Some differences such as the low frequency for a critical field like *protocol_type* can be explained by the fact that Stein et al.[**?**] show gene frequencies for only attacks belonging to the DOS category whereas our experiments classify all 4 kinds of attacks including DOS.

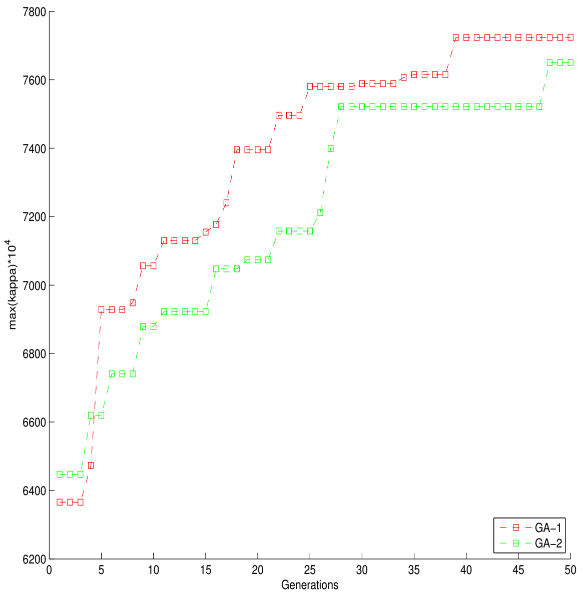

Figure 2: Fitness of best candidate over generations

5. CONCLUSION

We have shown 2 simple, effective GAs which were capable of extracting feature subsets from the NSL-KDD dataset. We showed how fitness of the candidates in a population changes over time and the improvement in accuracy when the best candidate after 50 runs of each GA was used with 3 different classifiers.

6. FUTURE WORK

While we found the kappa measure to be a good statistic for evaluating classifiers, further work needs to be done to verify its superiority or equivalence to other metrics such as f-measure. Another possible direction for further investigation would be use different combinations of base classifiers such as Naive Bayes, Bayesian Network along with other ensemble techniques such as boosting to see if it makes it any easier to maintain the right selection pressure on the population. Since each run of feature-selection GAs takes about 33 hours to complete with our parameters as described in Table **??**, if we increase the population size and number of generations, it would take even longer to get feedback from the GA . Since the most time-consuming part of the GA is the training and evaluation of each candidate, in order to reduce this overall running time, we could calculate the fitness metric of each candidate in parallel for a given generation and then merge all fitness metrics into the GA's list of candidates. Another way to speed up the candidate training and evaluation would be to rewrite the implementation

Table 2: Gene Frequency for Selected Generations for GA-1

Attribute	Gen 10	Gen 20	Gen 30	Gen 40	Gen 50
duration	0.4400	0.2525	0.4617	0.5962	0.6740
protocol_type	0.3950	0.2100	0.2200	0.3187	0.3040
service	0.1300	0.4575	0.6283	0.7175	0.7690
flag	0.3050	0.2500	0.1917	0.1825	0.2780
src_bytes	0.8350	0.9175	0.9383	0.9513	0.9610
dst_bytes	0.4700	0.3325	0.2283	0.1787	0.1440
land	0.1600	0.1150	0.1467	0.1100	0.0910
wrong_fragment	0.5250	0.6500	0.6733	0.6225	0.5060
urgent	0.5550	0.4375	0.3033	0.2750	0.3210
hot	0.4400	0.6100	0.7317	0.7975	0.8360
num_failedst_logins	0.3450	0.3050	0.4250	0.4863	0.5880
loggedst_in	0.7650	0.5525	0.3717	0.2838	0.2280
num_compromised	0.4200	0.3150	0.2167	0.1700	0.1380
root_shell	0.8750	0.9300	0.9433	0.9550	0.9620
su_attempted	0.6100	0.7725	0.8333	0.8500	0.8650
num_root	0.4400	0.6225	0.7417	0.8000	0.7870
num_file_creations	0.8800	0.9275	0.8983	0.8738	0.7830
num_shells	0.3550	0.2700	0.1800	0.1388	0.1550
num_accessrv_files	0.1700	0.4675	0.6383	0.7262	0.7720
num_ob_cmds	0.2850	0.1625	0.1183	0.1200	0.2100
isrv_host_login	0.6300	0.7275	0.7467	0.7788	0.8230
isrv_guest_login	0.3300	0.2475	0.1750	0.1475	0.1620
count	0.1600	0.0875	0.0617	0.0488	0.0410
srv_count	0.8050	0.8975	0.9283	0.9437	0.9530
serror_rate	0.6900	0.7575	0.8383	0.8762	0.9000
srv_serror_rate	0.1950	0.2175	0.1500	0.1125	0.1360
rerror_rate	0.2100	0.1075	0.1133	0.1862	0.3460
srv_rerror_rate	0.3300	0.4700	0.3983	0.3325	0.2820
same_srv_rate	0.8150	0.8375	0.8900	0.9150	0.9290
diff_srv_rate	0.6500	0.4300	0.2900	0.2213	0.1800
srv_diff_host_rate	0.1400	0.1875	0.2200	0.2587	0.3480
dst_host_count	0.4450	0.6300	0.7483	0.8025	0.8410
dst_host_srv_count	0.8300	0.9100	0.9367	0.9500	0.9580
dst_host_same_srv_rate	0.1650	0.0925	0.0650	0.0500	0.0430
dst_host_diff_srv_rate	0.1700	0.1000	0.0850	0.0663	0.0560
dst_host_same_src_prt_rate	0.3500	0.2475	0.1683	0.2050	0.3180
dst_host_srv_diff_host_rate	0.6850	0.7675	0.8400	0.8775	0.8980
dst_host_serror_rate	0.2050	0.1150	0.0867	0.0675	0.0570
dst_host_srv_serror_rate	0.7850	0.8875	0.9167	0.9375	0.9480
dst_host_rerror_rate	0.8650	0.7550	0.7617	0.7788	0.6700
dst_host_srv_rerror_rate	0.4400	0.6225	0.7217	0.7887	0.8280

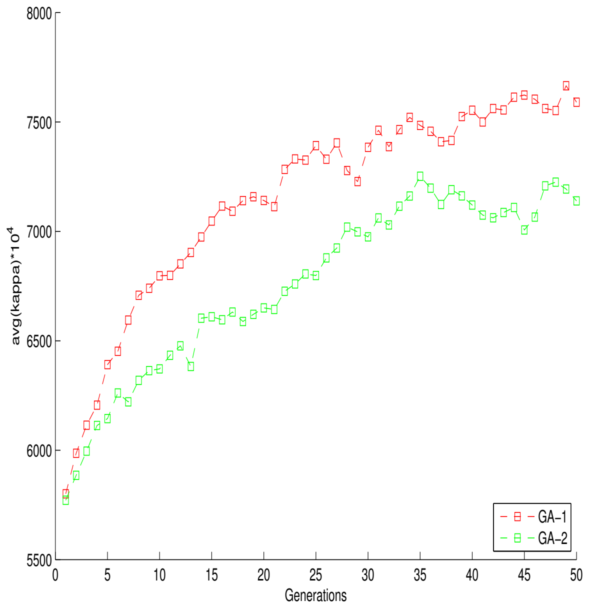

Figure 3: Avg fitness over generations

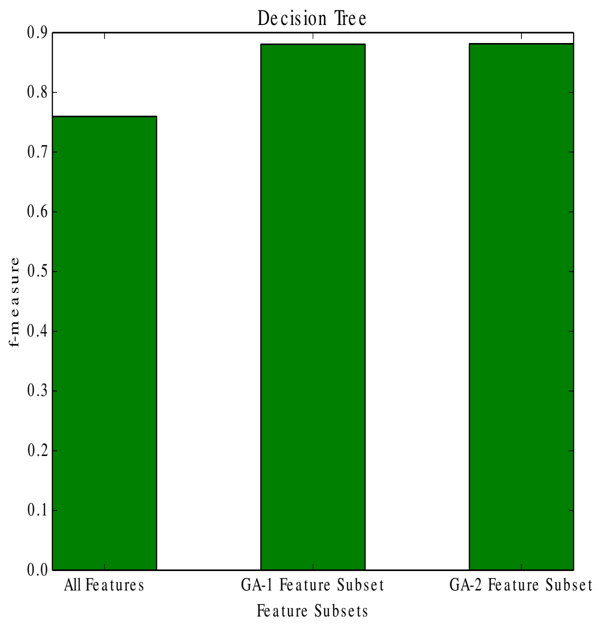

Figure 5: f-measure for Decision Tree classifier

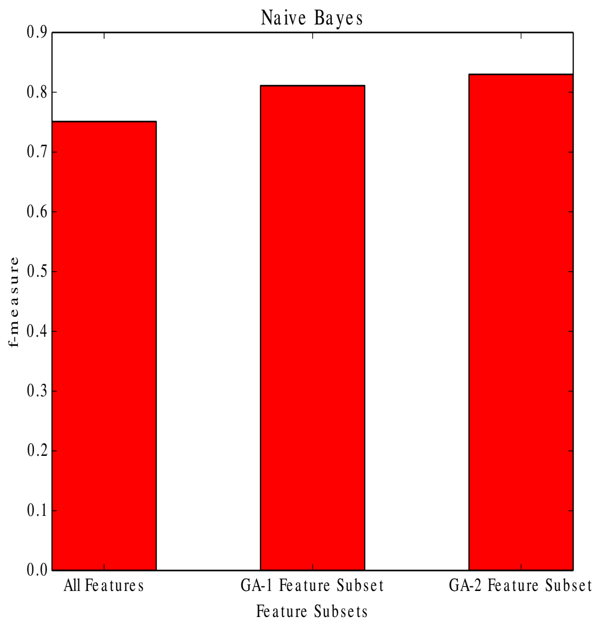

Figure 4: f-measure for Naive Bayes classifier

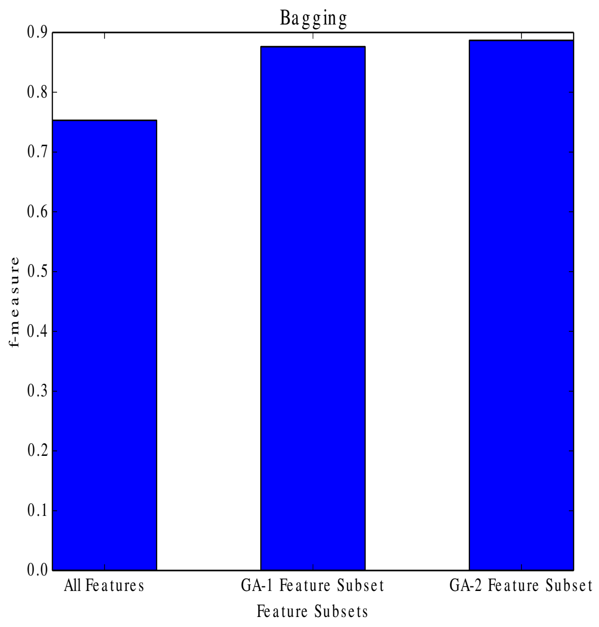

Figure 6: f-measure for Bagging classifier

of the base learners to cache from previous generations as many intermediate steps as possible.

7. ACKNOWLEDGMENTS

We would like to thank Dr. Punch(Professor, CSE, Michigan State University) for a enriching experience in the Evolutionary Computation course instructed by him.

8. REFERENCES

[1] How to Calculate Kappa. `http://epiville.ccnmtl.columbia.edu/popup/how_to_calculate_kappa.html`, 2014. [Online; accessed 09-Dec-2014].

[2] Kappa statistic in plain English. `http://stats.stackexchange.com/questions/82162/kappa-statistic-in-plain-english`, 2014. [Online; accessed 09-Dec-2014].

[3] The NSL-KDD Data Set. `http://nsl.cs.unb.ca/NSL-KDD/`, 2014. [Online; accessed 09-Dec-2014].

[4] M. L. G. at the University of Waikato. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. `http://www.cs.waikato.ac.nz/ml/weka/`, 2014. [Online; accessed 09-Dec-2014].

[5] T. G. Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

[6] A. Goyal and C. Kumar. Ga-nids: a genetic algorithm based network intrusion detection system. *Northwestern university*, 2008.

[7] F. E. Heba, A. Darwish, A. E. Hassanien, and A. Abraham. Principle components analysis and Support Vector Machine based Intrusion Detection System. In *Intelligent Systems Design and Applications*, pages 363–367, 2010.

[8] M. S. Hoque. A N I MPLEMENTATION OF I NTRUSION D ETECTION. 4(2):109–120, 2012.

[9] D. S. Kim, H.-n. Nguyen, and J. S. Park. Genetic Algorithm to Improve SVM Based Network Intrusion Detection System. *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, 2:155–158, 2005.

[10] R. Kohavi and G. H. John. The wrapper approach. In *Feature extraction, construction and selection*, pages 33–50. Springer, 1998.

[11] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[12] G. Stein, B. Chen, A. Wu, and K. Hua. Decision tree classifier for network intrusion detection with GA-based feature selection. *... of the 43rd annual Southeast regional ...*, pages 136–141, 2005.

[13] Weka. DecisionStump. `http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/DecisionStump.html`, 2014. [Online; accessed 09-Dec-2014].