Benny Tsai
600.425
Declarative Methods
Final Project Report

Problem

In network security, intrusion detection is considered a highly important task, as one cannot defend attacks one does not know about. The task is significantly easier if we can somehow infer what application generated a particular sequence of packets. Given this ability, it would be a simple matter to filter packets based on whether the application that generated the packets is allowed or not. An intuitive solution to the problem is simply to analyze the content of individual packets in the sequence. However, this particular approach is easily defeated by the use of encryption. A method that works despite the presence of encryption would be highly valuable. Charles Wright of the JHU Information Security Institute, working with Dr. Fabian Monrose and Dr. Gerald Masson, has come up with a possible solution to the problem.

Wright's method infers the application layer protocol that generated a TCP packet sequence using only two features: size and inter-arrival times. These features were chosen because they remain intact after encryption. Wright builds two HMM models for each protocol to be detected, one based on packet sizes and one based on packet inter-arrival times. To classify a new packet sequence, one just needs to see which protocol's HMM models reports the highest likelihood of having generated the new sequence.

Wright was able to achieve a high degree of accuracy using this approach (see Results section for details). Having learned about other classifiers (K-nearest neighbor classifiers, neural networks, support vector machines) in this course, I was curious to see how each of them would compare to Wright's HMM-based approach. Hence this project.

Solution

HMM models are a natural choice for modeling sequential data. In order to apply other classifiers to the problem, I needed to come up with a way to encode the problem properly. I eventually decided to model packet sequences as a collection of bigrams.

Specifically, the range of possible packet sizes and inter-arrival times are split into discrete bins. Here I ran into the first problem: while there is a definite maximum packet size, as defined by the network link-layer technology and protocol, there is no such maximum for packet inter-arrival times! I discussed this with Wright over lunch, and he shared with me that a standard solution to this problem is to take the logarithm of inter-arrival times. This cuts down the possible range of packet inter-arrival times greatly, making binning practical.

After analyzing the data, I determined that 24 64-byte bins allows me to cover all possible packet sizes (60 to 1514), and 24 size-1 bins allows me to cover all packet inter-arrival times encountered in practice (logarithms range from –11.51 to 10.89). An additional 24 bins for each category are used to represent packets flowing in the opposite direction. Packets from X to Y are represented using the first 24 bins, while packets from Y to X are represented using the second 24 bins.

A toy example will serve to illustrate how packet sequences are encoded. Suppose we have a simple 3-packet TELNET sequence between hosts A and B. The first packet, sent from A to B, is 60 bytes long, and its arrival times is designated time 0. The second packet, 1500 bytes from A to B, arrives 1 second after the first. The third packet, 60 bytes from B to A, arrives 2 seconds after the second packet. The first packet size goes into bin 0, the second packet size goes into bin 23, and the third packet size goes into bin 24 (because it is flowing in the opposite of the original direction). The packet times in logarithm are 0 (bin 12), 0 (bin 12), and 0.6931 (bin 24, as it's flowing in the opposite of the original direction). The size bigrams, then, are 0 -> 23 and 23 -> 24, while the time bigrams are 0 -> 0 and 0 -> 24, all with count 1. These counts are normalized by dividing them by the total number of packets in the sequence. This packet sequence gets encoded as:

3 … 0.5000 … 0.5000 … 0.5000 … 0.5000 … 2

The first number represents the number of packets in the sequence, while the last number, a 2 in this case, encodes the protocol that generated the packet (0 = FTP, 1 = SMTP, 2 = TELNET, 3 = HTTP, 4 = Other).

I chose to use the 3 classifier packages we played with in homework 3, because I was already familiar with their operation and data encoding format. In order to make the comparison between these classifiers and Wright's HMM's meaningful, I made use of the same data he used. The data sets were released by MIT's Lincoln Laboratory, and can be obtained at http://www.ll.mit.edu/IST/ideval/data/data_index.html. Following Wright's footsteps, I used the first week of the 1999 data set as training data, and the third week of the 1999 data set as test data. These were chosen because these were the only weeks that were attack-free.

Results

Wright's HMM's results on the MIT Lincoln Laboratory data:

Model	ftp	smtp	telnet	http
Size Model	99.4	97.9	21.7	96.9
Timing Model	82.2	96.6	14.2	95.3

Note that the poor classification performance on telnet; the interactive nature of the protocol makes it difficult to classify correctly.

My results:

Classifier	ftp	smtp	telnet	http
1-Nearest Neighbor	88	94	92	95
3-Nearest Neighbors	87	99	99	80
25-Nearest Neighbors	76	86	99	71
Neural Network	93	92	95	93
Support Vector Machine	95	99	94	99

These results were obtained by training each of the models on 500 training samples (100 samples each from ftp, smtp, telnet, and http, and 100 samples generated by other protocols), all randomly extracted from the first week of the 1999 data set. The test data consisted of 500 samples, with the same distribution amongst protocols, randomly extracted from the third week of the 1999 data set.

Both the neural network and the support vector machine were built using the default options. The neural network had a single layer consisting of 64 nodes. The support vector machine used a Gaussian kernel.

What caught my eye immediately was how well all 3 classifiers performed when attempting to identify telnet traffic. My guess is that the being able to combine both timing and size features helped the models distinguish between the various protocols, something the separate timing and size HMM models were not able to take advantage of. That, or perhaps the bigram encoding created some features that did a good job of distinguishing telnet from other protocols.

It seems that on this task, one does not benefit from increasing the number of neighbors used in the K-nearest neighbor algorithm.

Overall, support vector machines seemed to perform the best on this task. It compares very favorably with both the timing and the size HMM models.

Wright told me that he trained his models on 300 samples from each protocol, which took roughly an hour or so. I expected these packages to take at least that much time to train, if not more, since each individual sample was represented by 4607 features (48^2 for the size bigrams, 48^2 for the timing bigrams, and 1 for the packet length), which is a great deal more than the hundreds of features we used in homework 3 However, I was pleasantly surprised to find that each classifier only took seconds to train, and only seconds more to classify the test data!

Conclusion

I believe that using these classifiers in lieu of or in conjunction with HMM's to perform application inference from TCP traffic holds great promise. These classifiers, especially support vector machines, seem to compare favorably with HMM's in performance using just a simple encoding scheme, and train orders of magnitudes faster. When I spoke with Dr. Eisner, we discussed many other features that can be encoded by building on this simple scheme, such as modeling burstiness of traffic by calculating the standard deviation of packet sizes and/or inter-arrival times. In general, there are many aspects of traffic behavior that can be encoded by applying statistical techniques on packet sizes and inter-arrival times.

One general problem with this technique, whether done by HMM's or some other classifier, is that while timing and size information remains intact after encryption, they are easily spoofed features. Padding packets with extra bytes of information distorts packet sizes, and coding random delays into an application distorts timing information. I feel that this issue must be addressed if the technique is to remain useful against determined adversaries.