A Genetic Algorithm for Solving the Binning Problem in Networked Applications Detection

Maxim Shevertalov, Edward Stehle, Spiros Mancoridis
Department of Computer Science
College of Engineering
Drexel University
3141 Chestnut Street, Philadelphia, PA 19104, USA
{ms333, evs23, spiros}@drexel.edu

Abstract

Network administrators need a tool that detects the kind of applications running on their networks, in order to allocate resources and enforce security policies. Previous work shows that applications can be detected by analyzing packet size distributions. Detection by packet size distribution is more efficient and accurate if the distribution is binned. An unbinned packet size distribution considers the occurrences of each packet size individually. In contrast, a binned packet size distribution considers the occurrences of packets within packet size ranges. This paper reviews some of the common methods for binning distributions and presents an improved approach to binning using a Genetic Algorithms to assist the detection of network applications.

1 Introduction

This paper describes a genetic algorithm approach to solving a binning problem in the context of detecting network applications using packet size distributions. We begin by collecting samples of training data. This data is stored as histograms where the x-axis represents packet sizes and the y-axis represents the number of packets observed at those sizes. Even though the majority of collected packets only fall into a few bins, there are few collected packets at each size ranging between 0 and 1500 bytes.

The large range presents a problem for classification algorithms. First, performing classification over such large histograms without doing some sort of optimization is resource intensive. Second, in certain cases, the bins with the smallest count are most representative of an application and, thus, need to be weighted higher. However these bins are overshadowed by the larger spikes. However, the smaller bins may just be noise and, thus, only confuse the classification algorithm.

A solution to these problems can be found by binning the histograms. A binned histogram contains fewer bins than the original one. Binned histograms can also be used to aggregate smaller bins together to give them more weight and smooth out the noise found in the distribution. In their paper, Trivedi *et.al.* [15] present a binning they found through analysis. We want to confirm that their binning was effective for the applications we want to detect, and if an improved binning can be determined automatically.

Trivedi *et.al.*[15] and Parish *et.al.*[12] demonstrated that TCP-based network traffic posed the most significant problems when using the network application detection method described by Trivedi *et.al.* These problems are due to the way TCP manages traffic. While UDP does not impose limits and allows applications to send as much data as they wish, at the rate they wish, TCP attempts to maximize throughput while adhering to fairness constraints. In addition, TCP has built-in features to guarantee delivery. Due to these complexities, TCP streams look more similar to each other than UDP streams. These complexities of TCP present us with a good test case to see if binning can improve classification.

The rest of this paper concentrates on stating the binning problem (Section 2), describing the Genetic Algorithm used to find a good solution to this problem (Section 3), discussing a case study which uses web applications such as gMail and YouTube (Section 4), stating conclusions drawn from this work and identifying opportunities for future work (Section 5).

2 The Binning Problem

The Binning problem involves converting one histogram into another such that the new histogram is created by summing up the bins found in the original histogram (Figure 1). The new histogram allows various classification algorithms to compute more accurate results at a performance

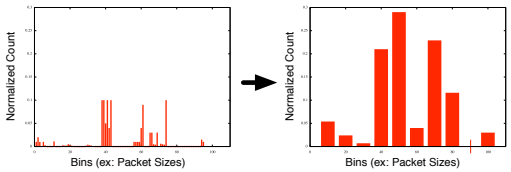

Figure 1: Illustrating the binning problem. In essence we are trying to summarize a histogram with a large amount of bins into one with less bins. Using this new histogram classification algorithms will be able to improve in performance as well as accuracy.

boost [2]. This is because the original, unbinned, histogram contains a lot of noise and is much larger than the newly created one.

We define a histogram O as having N ordered bins, contained in set I, such that each bin I_i contains a count, in our case the number of packets of a specific size. The new binned histogram P can be defined by summing the values of a range of bins from the original histogram O. We need to define a mapping such that each bin in P maps to a range of bins in O. Further, P includes all the bins in O and, thus, if a bin of P contains the range $[a, b)$, the next bin in P contains the range $[b, c)$. We define the range for bin 0 as $[0, J_0)$ and the range for each subsequent bin i as $[J_{i-1}, J_i)$. Thus the optimization problem becomes finding the values of J_0, J_1, J_2,...,J_K where $K \leq N$ and the classification is improved. We need to determine both the range of each bin in the new histogram, and the number of bins.

There are various supervised and unsupervised methods for determining a binning [3]. Two of the unsupervised methods that compared to our work to are *range* and *frequency* based binning. These methods only require statistical information about the distribution, and user input for the optimal number of bins. Both of these methods are computationally inexpensive.

Range based binning creates N buckets of identical size, where N is specified by the user. We first determine the domain size of the original data either through user input or by looking at the learning data. We then divide the domain by N to determine the size of each bin. For example, assume that we were looking at a size distribution with the smallest observable size being 60, the largest being 1000, and we choose $N = 10$. Then, the new histogram will have 10 bins each of size $(1000 - 60)/N = 94$. Thus bin 0 in the new histogram would contain the sum of bins 0 to 93 from the original histogram.

Frequency based binning works in a manner similar to range based binning by also working on a set of learning samples. Frequency based binning also requires a user specified parameter N in order to separate the domain into N bins. However, unlike range based binning, frequency

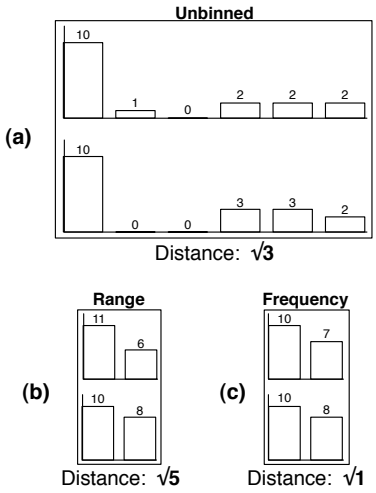

Figure 2: This example demonstrates a case where using range based binning can improve the result. Diagram **(a)** shows two histograms belonging to different classes of networked applications. We wish to separate them from one another, meaning we wish to apply a binning on them such that the distance between the two histograms increases. In this case range binning, presented in part **(b)**, achieves the goal and frequency binning, presented in part **(c)**, makes the problem worse.

based binning attempts to ensure that if the learning sample is combined into a single histogram, each bin will contain roughly the same quantity. To put it another way, we can say that frequency based binning attempts to split the domain so that, when the new histograms are combined, the variance between each bin is minimized.

When evaluating a binning we can observe how it affects classifications. A better binning will bring histograms representing the same class closer together, while separating those of different classes. A simple classification method is nearest neighbor classification. Using this method we calculate the euclidean distance between two histograms to decide how similar they are. Thus, if we have two histograms A and B with N bins each, we can define the distance between them as:

$$D(A, B) = \sqrt{\sum_{i=0}^{i<N} (A_i - B_i)^2}$$

Both range and frequency binning algorithms work well in certain cases. For example, range-based binning produces better results when the data is mostly uniform with a few spikes. By enabling the user to combine a number of small bins into a single large bin we can accommodate for small differences in the distribution and give them more

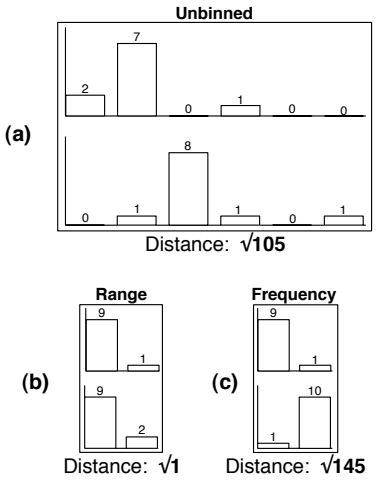

Figure 3: This example demonstrates a case where using frequency based binning can improve the result. Diagram **(a)** shows two histograms belonging to different classes of networked applications. We wish to separate them from one another. In this case frequency binning, presented in part **(c)**, achieves the goal and range binning, presented in part **(b)**, makes the problem worse.

weight. Figure 2 demonstrates a case where the range binning approach is more effective. Because we are comparing the histograms using euclidean distance, the figures present the distance without evaluating the square root. Looking at Figure 2 we can see that the original distance between the unbinned histograms is $\sqrt{3}$.

Frequency-based binning is useful when there are a few spikes, signifying unique classes, found in the data such that the bins around those spikes are there because of noise or error in the sensors collecting the data. Figure 3 demonstrates the case where the frequency-based binning approach is more effective. Similar to Figure 2, we used the euclidean distances without evaluating the square root in order to compute distance between two histograms.

This paper describes three algorithms that can be used for binning, Range, Frequency, and our GA. However, there are other algorithms that take advantage of information entropy [13, 8, 9], and iterative algorithms that start with unbinned data and progress upward by merging bins together [7, 14].

3 Genetic Algorithm

Previous work by Kyoung-jae Kim and Ingoo Han used a GA to perform discretization as a part of an artificial neural network (ANN) system to predict the stock price index [6]. They determined that an ANN had problems with

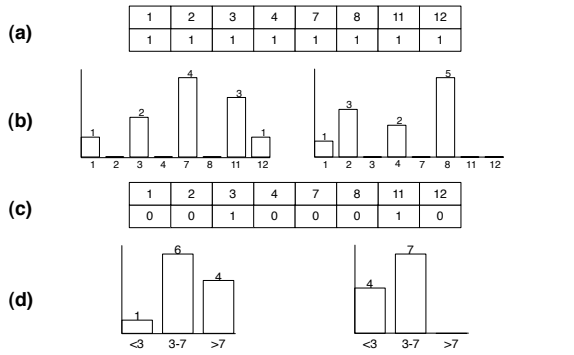

Figure 4: demonstrates how the encoding can be used to combine multiple histograms. Part **(a)** presents the encoding where it is composed of all 1's and thus every bucket is its own bin. Part **(b)** demonstrates the results of the string encoding described in **(a)**. Part **(c)** presents another sample encoding such that the result is three buckets: 0-2, 3-7, and 8-12. Part **(d)** demonstrates the results of the binning described in **(c)**.

large, noisy, and continuous data sets. Therefore, they implemented a GA to mitigate those limitations. They used the ANN as a part of the evaluation function to evolve good discretization policies.

In a similar approach, we coupled our GA with our initial problem of classification. We wish to find a binning that would provide us with good classification results. For this purpose we split the learning set into two separate sets, l_1 and l_2. We used l_1 as the learning set during the evaluation function and l_2 as the corresponding test set.

This section describes, in detail, the elements of our genetic algorithm. Section 3.1 describes how the data set is encoded, Section 3.2 presents the evaluation function used in the GA process, and Section 3.3 discusses the GA's breeding function.

3.1 Encoding

We chose a binary encoding scheme because it is the most generic and most commonly used [16]. This encoding enabled us to use work developed by others in the GA arena. It also allowed us to extend our GA to become a generic solution for a greater variety of related problems.

To encode a binning we need to identify break points, namely those values that begin and end each new bin. In our scheme, we represent the locations of break points by using a 1 to represent a break (Figure 4). For example, if the original histogram has the following buckets 2,4,5,7, and 8, an encoding of `01001` corresponds to three buckets: $x < 4$, $4 \leq x < 8$, and $8 \leq x$.

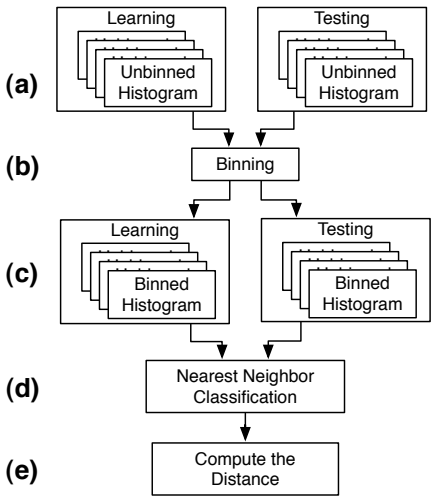

Figure 5: demonstrates the process used to compute the parameters of the evaluation function. We first take two sets of histograms, learning and testing (Part **(a)**). We then perform the binning we are trying to evaluate (Part **(b)**). That produces a binned version of the learning and testing sets (Part **(c)**). We then perform a nearest neighbor classification on the sets to get evaluation metrics (Part **(d)**). Finally, we compute the fitness using equation 1 **(e)**.

3.2 Evaluation Function

In order to compute the evaluation we must first compute four parameters:

- The number of misclassifications;
- the number of bins;
- the distance from each classified entity to the correct class;
- the distance from each classified entity to the incorrect class.

We accept a chromosome corresponding to a binning as input and convert the unbinned histograms into binned histograms. We then have binned learning and testing sets. The learning set is used as expert knowledge by the classifier. The evaluation function then performs a nearest neighbor classification of the data. In a nearest neighbor classifier, samples from the testing set are matched to the ones in the learning set based on the euclidean distance of the histograms. When creating a nearest neighbor classifier, we need to find a set of representative histograms for each class of network applications. For example, we can average all of the histograms in the learning set belonging to the same class together into a single representative histogram. We chose to include every learning histogram belonging to a particular class from the learning set in the representative set. We can then use the test set of histograms to evaluate a binning. When we attempt to determine the class of a histogram from the testing set it belongs to, we need to compare it to every histogram in the learning set using the euclidean distance. Once we determine which learning histogram it is closest to, we can label it as that class.

The performance of this nearest neighbor classifier is improved by reducing the number of histograms in the learning set or by reducing the number of bins in each histogram. Once classification is complete, we have the metrics to compute the evaluation value of a given chromosome (Equation 1).

$$E(x) = \frac{1}{100a + b + c - d} \tag{1}$$

In equation 1, a is the number of misclassifications, b is the number of bins in the binned histograms, c is the average euclidean distance between histograms in the testing set and the closest histogram of the same class in the learning set, and d is the average euclidean distance between histograms in the testing set and the closest histogram in the learning set of a different class. Because we use classification accuracy as the most important criterion, the number of misclassifications is multiplied by an arbitrarily large number, 100 in this case.

The next criterion we wish to minimize is the number of bins, the b variable. As we discussed previously, the number of bins has a direct impact on how fast the nearest neighbor classifier executes. Finally, the c and d variables allow us make small improvements. As the classification improves, the histograms in the testing set get closer to the members in their correct class and further from the histograms in other classes. We found that when we ignored the c and d parameters, the algorithm would converge quickly to a local optimum.

3.3 Breeding Function

The breeding function is responsible for generating new populations from a selection of pairs of parents. We employed a roulette wheel selection procedure where the probability that a chromosome is chosen to be a parent is weighted by its relative fitness. Thus, more fit individuals are more likely to pass on their genes to the next generation.

Once a number of parents is chosen into the breeding pool, two of them at a time are selected at random with replacement. The selected parents produce two children with a probability that is a configurable parameter in our implementation. We used 0.9 in our case study. If they do not mate, the parents are added to the next generation. In addition, we ensure that the next generation does not contain

identical chromosomes. We believe that by using this restriction we prevent the GA from converging prematurely.

Once a new set of children is generated, we apply a mutation function. The mutation is applied with a probability p, which is different based on the current performance of the algorithm. When a mutation is applied, it changes a bit from 1 to 0 with a probability j, and from 0 to 1 with a probability $1 - j$. We chose to set $j > 0.5$ because we want to spend more time exploring spaces with fewer bins. The strings we are dealing with could be as large as 1500 bits, most of which are 0. If the mutation function simply flips a random bit, it is more likely to change a 0 to a 1 and thus increase the number of buckets. We found that by controlling which was modified, a 0 or a 1, we are able to emulate two different binning operations, splitting a bucket, in the case of a 0 changing to a 1, and merging two buckets, in the case of changing a 1 to a 0. This abstraction allows the algorithm to converge faster.

When choosing the probability of mutation we decided to take advantage of a technique used in simulated annealing [1]. We specified that a generation improves, if the average fitness of the population improves. We increase the probability of mutation by a small amount, in our case 0.001, if a generation does not improve, thus introducing more randomness to the search. We decrease the probability of mutation by a large amount, in our case 0.01, when there is improvement. We found that this helps the algorithm escape from local optima in the search space.

4 Case Study

We want to find a binning that reduces the error in the classification of unknown packet streams. Previous work in the area of application detection using packet size distributions demonstrates that classifying TCP traffic is problematic [4, 15]. Therefore, we turned to this class of application to see if we could do better. We decided to restrict ourselves further by concentrating only on HTTP traffic.

In recent years the nature of web traffic has evolved from simple static pages to web applications such as email, video browsing, and image viewing. These applications now dominate web traffic and a number of popular web sites are converting to what has been dubbed Web 2.0. While previously we could have thought of web traffic as a simple transfer of data, today's web traffic is far more complicated.

We chose 14 different web applications:

- Image Browsing
  - Flickr (www.flickr.com)
  - Google Image Search (images.google.com)
- News
  - BBC News (news.bbc.co.uk)
  - CNN News (www.cnn.com)
- Video
  - YouTube (www.youtube.com)
  - Google Video (video.google.com)
  - MSN Video (video.msn.com)
- Commerce
  - eBay (www.ebay.com)
  - Amazon (www.amazon.com)
- Email
  - Yahoo Send Mail (mail.yahoo.com)
  - Yahoo Read Mail (mail.yahoo.com)
  - gMail Send Mail (mail.google.com)
- Other
  - Google Maps (local.google.com)
  - mySpace (www.myspace.com)

In our tests, we generated roughly 200 different samples of each application. Each sample contained 1000 packets. We automated the process using Firefox [11] and the Chickenfoot [10] plug-in. During each test, Chickenfoot instructed Firefox to browse a website by emulating a real user. We would begin by choosing a random page on the site and then following links found on the page for each test. For example Chickenfoot was made to watch a video on YouTube and then follow links to other, suggested, videos just as a real user would likely do. Chickenfoot would also reinitialize the entire process by searching for new videos on YouTube. In this way we were able to collect realistic data describing user behavior.

We split the 200 samples per application into training and testing data with 100 samples in each. No samples belonging to the testing set were ever considered when choosing a binning, using any of the algorithms. We used IB1, a nearest neighbor classifier, as the final step to test the various binning strategies. IB1 was given the first 100 binned samples per application class as the learning set to train on and the rest as testing data to evaluate the classification. When we describe the results of our experiments we are only counting the classification of the testing set.

Not all collection went smoothly. For instance in the test case where we wished to send email messages using gMail we had considerable trouble automating the collection process. We were able to get 153 different samples. In this case, and others like it, we used the first 100 as the learning

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<-- classified as
0	0	10	62	0	0	0	0	0	0	0	2	25	0	a = flickr
0	**14**	40	6	0	0	0	0	1	0	0	2	37	0	b = bbcNews
0	0	**81**	1	0	0	0	0	0	0	0	2	15	0	c = youtube
0	0	1	**90**	0	0	0	0	0	0	1	2	5	0	d = yahooSendText
0	0	12	53	**5**	0	0	0	1	0	0	7	21	1	e = cnn
0	0	16	34	1	**3**	0	0	2	0	1	9	30	4	f = ebay
0	1	11	35	0	0	**2**	0	0	0	2	3	44	2	g = amazon
0	0	0	0	0	0	0	**11**	89	0	0	0	0	0	h = googleMaps
0	0	0	0	0	0	0	0	**100**	0	0	0	0	0	i = googleVid
0	0	0	0	0	0	0	0	31	**69**	0	0	0	0	j = googleImage
0	0	20	22	0	0	0	0	1	0	**1**	2	44	7	k = yahooReadText
0	0	39	36	0	0	0	0	0	0	0	**8**	17	0	l = myspace
0	0	7	0	0	0	0	0	0	0	0	3	**90**	0	m = msnVideo
0	0	2	28	0	0	0	0	1	0	0	0	13	**8**	n = gmailSendText

Figure 6: Confusion matrix generated by classifying unbinned histograms.

set and the rest as the testing set. In the end we had a total of 1400 learning samples and 1346 testing samples.

When using the GA we developed, we further separated the learning data into two pieces, l_1 and l_2, with 50 samples each. The first set, l_1, became the learning set used internally in the GA. We then used l_2 in the evaluation function by classifying it against the histograms gathered from l_1. Then, once a binning was settled on, just as with all other binning strategies, we used the entire learning set to compute the histograms to classify against.

In our experiments we used five different binning strategies:

- No Discretization
- Frequency Based Discretization
- Range Based Discretization
- Discretization Described by Trivedi[15]
- GA Based Discretization

We then used Weka [5], a classification framework written in Java, to perform classifications. We used the IB1 algorithm because it was based on the nearest neighbor classification, which is what the GA optimized the binning for.

For our first test we wanted to base line the IB1 algorithm by using it on unbinned data. The unbinned histograms contained 1500 bins. Most of the bins contained relatively small values in each of the samples. For example only 9 samples contained a value greater than 0 in bin 54. There were, however, a few peak bins. Meaning bins that contained most of the data. For example only 289 training samples, out of 1400 had a count of zero in bin 40 and more than half of the samples contained more than 30% of their packets in that bin.

Due to the large number of bins, the IB1 classification algorithm took on the order of 10 minutes to execute before providing a result which had 35.8% of its networked applications classified correctly. Figure 6 presents the confusion matrix for the unbinned base case. A row in the table represents results for a particular class of networked applications.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<-- classified as
54	0	0	4	8	16	11	0	0	0	0	2	0	4	a = flickr
1	**96**	0	0	0	2	0	0	0	0	0	0	0	1	b = bbcNews
0	0	**51**	4	1	3	0	0	0	0	1	8	31	0	c = youtube
0	14	0	**61**	0	5	0	0	0	0	10	8	0	1	d = yahooSendText
14	0	0	7	**59**	4	8	0	0	0	0	2	0	6	e = cnn
10	2	1	6	12	**33**	9	1	0	2	0	17	0	7	f = ebay
2	0	0	1	10	13	**55**	0	0	0	3	0	0	16	g = amazon
0	0	0	0	0	0	0	**99**	0	1	0	0	0	0	h = googleMaps
0	0	0	0	0	0	0	3	**97**	0	0	0	0	0	i = googleVid
0	0	0	0	0	0	0	1	0	**99**	0	0	0	0	j = googleImage
2	25	0	14	11	13	3	0	0	4	**9**	2	0	14	k = yahooReadText
0	7	8	11	1	20	1	0	0	0	0	**51**	1	0	l = myspace
0	0	31	0	0	0	0	0	0	0	1	1	**67**	0	m = msnVideo
0	0	0	0	1	0	3	1	0	0	0	0	0	**47**	n = gmailSendText

Figure 7: Confusion matrix generated by classifying histograms binned using the frequency based binning algorithm.

The column specifies to which class the number of samples were classified. For instance, from Figure 6, we can observe that 62 samples from the `Flickr` class were classified as the `yahooSendText` class.

The confusion matrix paints an immediate picture of the results. We want the highest values to appear on the diagonal. Each value on the diagonal signifies the number of samples that were matched correctly. All other values in the matrix are misclassifications. Thus by glancing at the matrix we get a rough idea of the success of the classification.

The next binning algorithm we tried was the one based on frequency. We were able to classify 65.2% of the testing samples correctly using this algorithm. Figure 7 presents the confusion matrix generated by using frequency based binning. While there is a significant improvement over the unbinned case, we believe that the reason frequency based binning does not achieve an optimal result is because there are a number of dominating bins. The frequency algorithm generated only 5 bins: 0-39, 40-40, 41-63, 64-1469, and 1470-1500. As expected, most of the packets in a TCP stream are at the lower end for acknowledgments and at the highest end for MTU sized packets. When using frequency based binning the entire middle range is subsumed by a single bin, removing significant detail. However, using frequency based binning produced the results faster than any other approach we tried, only a few seconds, primarily because the number of bins was small.

The next binning technique we attempted was range based binning. In this case we set the bucket size to be 20 and were able to achieve 79.4% correct classification. The confusion matrix is provided in Figure 8. This confusion matrix illustrates that some of the problems in classification arise because a number of the applications should belong to the same class. For example 12 of the samples belonging to class `YouTube` were categorized as class `msnVideo`. Both of these classes are video browsing applications and it may be acceptable to group them together.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<-- classified as
94	0	0	1	0	1	3	0	0	0	0	0	0	0	a = flickr
0	**99**	0	0	0	0	0	0	0	0	0	0	0	1	b = bbcNews
0	0	**77**	3	0	1	0	0	0	0	0	6	12	0	c = youtube
1	0	29	**51**	1	0	0	0	0	0	7	2	6	2	d = yahooSendText
0	1	0	1	**91**	0	1	0	0	0	2	0	1	3	e = cnn
0	2	1	2	6	**59**	6	0	1	0	2	7	0	14	f = ebay
3	2	0	4	5	9	**71**	0	0	0	0	0	0	6	g = amazon
0	0	0	0	0	0	0	**100**	0	0	0	0	0	0	h = googleMaps
0	0	0	0	0	0	0	1	**98**	1	0	0	0	0	i = googleVid
0	0	0	0	0	0	0	1	0	**99**	0	0	0	0	j = googleImage
4	5	0	16	6	7	2	1	0	0	**21**	2	0	33	k = yahooReadText
0	0	5	7	0	3	0	0	0	0	1	**83**	1	0	l = myspace
0	0	7	0	0	0	0	0	0	0	0	1	**92**	0	m = msnVideo
1	0	0	0	1	0	14	0	0	1	0	1	0	**34**	n = gmailSendText

Figure 8: Confusion matrix generated by classifying histograms binned using the range based binning algorithm.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<-- classified as
94	0	0	3	0	2	0	0	0	0	0	0	0	0	a = flickr
0	**99**	0	0	0	0	0	0	0	1	0	0	0	0	b = bbcNews
0	0	**78**	5	0	0	0	0	0	0	6	10	0	1	c = youtube
0	1	27	**42**	1	2	0	0	0	0	12	6	7	1	d = yahooSendText
0	1	0	3	**93**	0	0	0	0	0	2	0	0	1	e = cnn
0	4	2	4	3	**62**	7	1	0	0	2	2	1	12	f = ebay
0	3	0	6	1	6	**81**	0	0	0	0	0	0	3	g = amazon
0	0	0	0	0	0	0	**100**	0	0	0	0	0	0	h = googleMaps
0	0	0	0	0	0	0	2	**98**	0	0	0	0	0	i = googleVid
0	0	0	0	0	0	0	0	0	**100**	0	0	0	0	j = googleImage
0	18	0	12	6	9	3	13	0	1	**15**	2	0	18	k = yahooReadText
0	0	6	0	0	1	0	0	0	0	1	**90**	2	0	l = myspace
0	0	7	0	0	0	0	0	0	0	0	0	**93**	0	m = msnVideo
0	0	0	16	0	1	1	1	0	0	0	0	0	**33**	n = gmailSendText

Figure 9: Confusion matrix generated by classifying histograms binned using the binning provided by Trivedi *et. al.*.

Next we tried the binning provided by Trivedi *et. al.* [15]. They binned the entire space into 50 buckets. This binning was created manually after examining distributions of a number of histograms. Their binning produced a result of 80.1% correct classification. While this is only a slight improvement over range binning, there are subtle differences that may suggest a larger improvement. These differences can be found by looking at the confusion matrix presented in Figure 9. For example, unlike range binning, the error in classifying the `gMailSend` class came by misclassifying it to the `yahooSendClass`. Both of these classes represent sending email. There are a number of other examples in the confusion matrix that demonstrate similar misclassifications.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	<-- classified as
90	1	0	4	0	1	1	0	0	0	0	1	0	1	a = flickr
0	**99**	0	0	0	0	0	0	0	0	0	0	0	1	b = bbcNews
0	0	**74**	3	0	0	1	0	0	0	3	7	11	0	c = youtube
2	4	9	**66**	1	0	0	0	0	0	4	8	4	1	d = yahooSendText
0	3	0	1	**92**	0	0	0	0	0	2	0	1	1	e = cnn
1	5	0	7	1	**60**	2	0	1	0	1	10	0	12	f = ebay
5	1	0	2	5	8	**71**	0	0	0	1	0	0	7	g = amazon
0	0	0	0	0	0	0	**100**	0	0	0	0	0	0	h = googleMaps
0	0	0	0	0	0	0	0	**99**	0	1	0	0	0	i = googleVid
0	0	0	0	0	0	0	1	0	**99**	0	0	0	0	j = googleImage
1	17	0	6	5	9	0	0	0	1	**28**	2	1	27	k = yahooReadText
0	2	0	4	0	6	0	0	0	0	1	**86**	1	0	l = myspace
0	0	9	0	0	1	0	0	0	0	0	0	**90**	0	m = msnVideo
1	1	0	0	1	0	0	0	0	0	0	0	0	**49**	n = gmailSendText

Figure 10: Confusion matrix generated by classifying histograms binned using the binning computed by our GA.

After trying all of these alternative methods, we used the implemented GA described in Section 3 to compute what we thought would be an optimal binning. The algorithm ran for about 36 hours and presented us with 40 bins. The final result using this binning was 81.9%. While this is only a slight improvement over the binning provided to us by Trivedi *et. al*, the confusion matrix (Figure 10) points out subtle differences. For example the mail applications now classify to the correct mail application. Most of the samples of class `gmailSendText` now classify to `gmailSendText`, as apposed to the similar `yahooSendText` class with range binning.

This demonstrates that a GA can be used in this area as a replacement to human expertise. It appears to perform at least as well as the human experts, while producing fewer bins and thus increasing performance of the classification algorithm.

5 Conclusions and Future Work

This paper presents a GA as a good solution to the binning problem. By binning histograms we are able to improve the performance, as well as the accuracy, of classification algorithms for classes of networked applications. We tailored our GA to the nearest neighbor classification algorithm, but there is no reason one could not replace it with another classification algorithm. The results achieved using the GA were close to ones achieved by other algorithms and even to the results produced by the hand-crafted binning of Trivedi *et. al*. This demonstrates that our GA helped us create detection algorithms that are is human competitive.

Range binning provided us with fairly accurate results for little cost. It was able to get an accuracy of 79.4% with only 20 bins and took less than a minute to compute. However, as we have shown, range binning depends on the distribution of data for its success. We believe that makes it unreliable. On the other hand, we believe that the GA technique is able to optimize the binning for a larger variety of situations. It is able to generate a better binning because it relies on the data for learning.

We believe that we can use binning to solve the more general discretization problem of converting a continuous variable into a set of discrete intervals. For example, if we use height as a way to characterize data, we can categorize the continuous variable of height into discrete categories such as `short`, `medium`, and `tall`. However, this discretization presents us with a new problem of how to choose the exact break points.

There are a number of techniques to determine these kind of splits. We can use expert knowledge to define break points globally, or we can use the samples and perform a clustering mechanism to choose breaks in an automated fashion. The second approach has the added advantage that

it is dependent on the domain of the sample data. If our samples only contained people between 4 and 5 feet, we would want a different discretization than if the people in the sample were between 3 and 7 feet.

This discretization problem can be converted into the binning problem if we have access to learning data. Because we have a finite set of learning samples, we can arrange each continuous variable into a histogram built from that data. For example, if our data set contained 10 elements with 7 distinct height measurements, the resulting histogram would contain seven bins. We can then run any number of algorithms to compute a binning that would improve classification. Once a good binning has been computed, it can be converted back into the original data set by using each bin as the discrete interval. In the future we would like to try this approach on various repositories of classification data for a wide array of problems in the domain of software forensics.

References

[1] E. K. Burke and G. Kendall, editors. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005.

[2] J. Catlett. On changing continuous attributes into ordered discrete attributes. In *Proceedings of the European working session on learning on Machine learning*, pages 164–178, 1991.

[3] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, 1995.

[4] A. Feldman and S. Muthukrishnan. Tradeoffs for packet classification. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1193–1202, March 2000.

[5] G. Holmes, A. Donkin, and I. Witten. Weka: a machine learning workbench. In *Intelligent Information Systems,1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, December 1994.

[6] K. jae Kim and I. Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, 19(2):125–132, August 2000.

[7] R. Kerber. Chimerge: Discretization of numeric attributes. In *Proc. Ninth Int'l Conf. Aritificial Intelligence*, pages 123–128, 1992.

[8] L. Kurgan and K. Cios. Caim discretization algorithm. *IEEEE Transactions on Knowledge and Data Engineering*, 16(2), 2004.

[9] R.-P. Li and Z.-O. Wang. An entropy-based discretization method for classification rules with inconsistency checking. In *Machine Learning and Cybernetics*, volume 1, pages 243–246, 2002.

[10] MIT. Chickenfoot. `http://groups.csail.mit.edu/uid/chickenfoot/`.

[11] Mozilla. Firefox - rediscover the web. `http://www.mozilla.com/en-US/firefox/`.

[12] D. Parish, K. Bharadia, A. Larkum, I. Phillips, and M. Oliver. Using packet size distributions to identify real-time networked applications. In *Communications, IEE Proceedings*, volume 150, pages 221–227, August 2003.

[13] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 22:379–423, July, October 1948.

[14] F. Tay and S. Lixiang. A modified chi2 algorithm for discretization. *Knowledge and Data Engineering, IEEE Transactions on*, 14(3):666–670, 2002.

[15] C. Trivedi, H. Trussell, A. Nilsson, and M. Chow. Implicit traffic classification for service differentiation. *ITC Specialist seminar*, 2002.

[16] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, June 1994.