A Representation Scheme for Finite Length Strings

Gaurav Tandon **Debasis Mitra**

Department of Computer Sciences Technical Report CS-2003-07
Florida Institute of Technology
Melbourne, FL 32901
{gtandon, dmitra}@cs.fit.edu

ABSTRACT

This study is an attempt to create a canonical representation scheme for finite length strings to simplify the study of the theory behind different classes of patterns and to ease the understanding of the underlying separability issues. This could then be used to determine what kinds of techniques are suitable for what class of separability (linear, multi-linear or non-linear). This representation can then be used in intrusion detection, biological sequences, pattern recognition/classification and numerous other applications.

INTRODUCTION

Various approaches have been proposed to detect intrusions based upon detecting deviations from normal host based traffic. Host based systems attempt to detect anomalies based upon system call information, an idea first suggested by Forrest et al [8]. Sequences of system calls are used to determine if an intrusion has taken place or not. This sequence of system calls is nothing but a sequence of finite length strings. This study is an attempt to define an abstract representation for the strings representing the various system calls of a host based anomaly detection system. Such a representation can also be used in other areas where such sequences of strings come into the picture, examples being the study of DNA sequences and pattern recognition.

The motivation for such a representation is to use it to study and understand the theory behind different classes of patterns and the underlying separability issues easily. With such a canonical representation, we would be able to determine what kinds of techniques are suitable for what class of separability - linear, multi-linear or non-linear. The simplest case of a linearly separable decision problem is one consisting of two sets of points (patterns) in a 2-D vector space that belong to different classes, where the two classes can be separated by a straight line. In 3 dimensions the equivalent problem is one where the points are separable by a plane. We can extend this idea to any number of dimensions, though 4 or 5 dimensions are harder to visualize. Planes and straight lines are just hyperplanes of dimension 1 and 2. There are hyperplanes of higher dimension. In higher dimensional cases two classes in an N dimensional space are termed as linearly separable if they are separable by a hyperplane of dimension N-1.

The strings in our representation scheme could represent the various system calls of a host based anomaly (intrusion) detection system. Such a representation can also be used in other areas where such sequences of strings come into the picture, examples being the study of DNA sequences and pattern recognition.

RELATED WORK

There is no work in this particular area available to our knowledge that we can cite here. But there are some works that we studied which could assist us in our representation scheme. These works are described next.

Linear Separability of Sets

Two sets X_1 and X_2 in Euclidean space E^n, which means each element x in those sets is represented as an n-tuple: $x = (E_1, E_2, ..., E_n)$. The hyperplane which seperates the two sets is represented as a nonzero vector in E^{n+1}. This is the normal to the separating plane: $a = (p_1, p_2, ..., p_n, p)$.

The following algorithm [2] is a recursive solution to find the vector a, the normal to the hyperplane separating the two sets X_1 and X_2. It is defined as the function $\boldsymbol{a = A(s, P)}$ taking two parameters:

1. s, the number of elements in the subset Y_s of Y:
   $$Y_s = \{ y_1, y_2, ..., y_s \}$$
2. P, a subspace of Euclidean space E^{n+1} of dimension $k+1$.

The algorithm also assumes that the two sets X_1 and X_2 are not existing in the same hyperplane; if this is the case, then $Y = X_1 \cup X_2$ contains a basis of E^{n+1}. The algorithm (from [6]) is as follows:

1. Find a subset B of Y which, when unioned with $P^{\perp}$, demonstrates linear closure under E^{n+1}:

   $$B = \{ y_{r0}, y_{r1}, ..., y_{rk} \}$$
   and
   $$B_1 = \{ y_{r1}, y_{r2}, ..., y_{rk} \}$$

2. Find a vector x such that:

   $$x \text{ exists in } P \cap B_1^{\perp}$$
   and
   $$x \cdot y_{r0} > 0$$

3. If $k = 0$, then x defined by step 2 is returned as the solution a Otherwise, we define the sequence $a_0 = x, a_1, a_2, ..., a_s = a$
4. Set $a_0 := x$
5. Iterate from $i = 1...s$ the recursive step of the algorithm:

   if $a_{i-1} \cdot Y_i \geq 0$, then
   $a_i := a_{i-1}$
   else
   $a_i := \boldsymbol{A(}\, i\text{-}1, P \cap \{ y_i \}^{\perp} \boldsymbol{)}$

The algorithm will terminate in step 3, returning the vector a as the normal of the hyperplane separating X_1 and X_2.

Threshold Logic Unit

In a Threshold Logic Unit (TLU) [9], the output of the unit in response to a particular input pattern is calculated in two stages. First the activation is determined. The activation is passed through a threshold function to obtain the output. Learning in a TLU is concerned with using an automatic procedure for adjusting the weights and threshold so that the decision boundary minimizes the error function.

A simple learning procedure for a TLU is called the perceptron learning rule. The error function used for the perceptron training rule is based on the number of incorrectly classified points in the training set. The perceptron learning procedure minimizes this error function. The Least Mean Squares rule minimizes an error measure called the mean squared error. In the TLU implementation of this rule the error for each pattern is calculated by finding the difference between the desired and actual activation, i.e. the error is calculated using the activation of the TLU *before* it is passed through the threshold function and weights are updated. If the weights are updated after every input pattern then the decision boundary will continue to move around the point with the lowest error.

Support Vector Machines

Support vector machine [4] finds a nonlinear decision function in the input space by mapping the data into a higher dimensional feature space and separating it there by means of a maximum margin hyperplane. The computational complexity of the classification operation does not depend on the dimensionality of the feature space, which can even be infinite. Overfitting is avoided by controlling the margin. The separating hyperplane is represented sparsely as a linear combination of points. The system automatically identifies a subset of informative points and uses them to represent the solution. Finally, the training algorithm solves a

simple convex optimization problem. All these features make Support Vector Machines an attractive classification system.

Convex Hull Methods

Another strategy used to separate a set of points from another set of points on a plane is to compute the convex hull of the two sets of points [5]. If the convex hulls overlap, then the given sets are not separable. Quite a few strategies are known for convex hull. These include Graham's scan [7], Jarvis' March [1] and Quick Hull [3]. Convex hulls in two and even three dimensions are fairly easy to work with. However, as the dimension of a space increases, certain assumptions that were valid in lower dimensions break down. For example, any *n*-vertex polygon in two dimensions has exactly *n* edges. However, the relationship between the numbers of faces and vertices is more complicated even in three dimensions.

OUR EXERCISE

The idea is to create a lattice based structure which can be used to create an abstract representation so that we can differentiate between "normal" and "abnormal" sequences of system calls, that is, two different sets of strings. The system calls are finite in number. Let the various system calls be represented by a unique natural number, which is 1, 2, 3, and so on. Our goal is to define an abstract representation of these system calls (which are inherently strings). We would like to start with a simple model consisting of 3 system calls and then build upon the same, adding more complexity and venturing into higher dimensions, as we will explain later.

Let system call s_1 be represented by *1*,
system call s_2 be represented by *2*, and
system call s_3 be represented by *3*.

There are a total of 3! = 6 combinations corresponding to these 3 numbers (representing strings/system calls), namely 1 2 3, 1 3 2, 2 1 3, 3 1 2, 3 2 1 and 2 3 1.

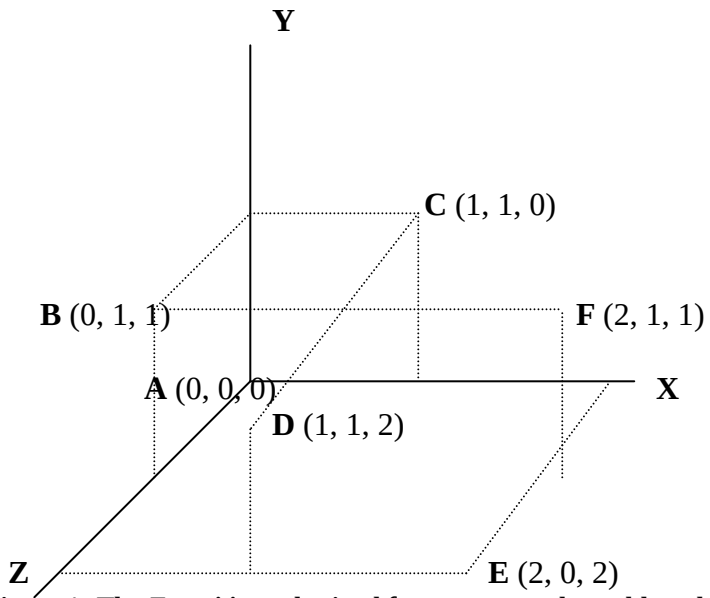

Figure 1: The *Transitions* obtained from our sample problem domain can be represented as points in 3-dimensional space.

We define the term *Transition* for each digit from one combination to the other as the number of places the digit moves, with a positive sign being assigned if the transition is from left to right and a negative sign being assigned if the transition is from the right to the left. For example, consider the transition from 123 to 132. The digit '1' remains at the same position, so *Transition* for '1' is zero. The digits '2' and '3' are interchanged. '2' has a *Transition* of +1 as it moves to the right and '3' has a *Transition* of -1 since it

moves one position to the left. Thus, if the *Transition* for the initial combination, that is 1 2 3, is depicted by (0 0 0), then the *Transition* to 1 3 2 is represented by (0 1 -1). The *Transition* for the entire combination set is represented as follows:

(1) 1 2 3 (0 0 0)
(2) 1 3 2 (0 1 -1)
(3) 2 1 3 (1 -1 0)
(4) 3 1 2 (1 1 -2)
(5) 3 2 1 (2 0 -2)
(6) 2 3 1 (2 -1 -1)

The *Transition* for the entire set can be represented in the 3-D plane, where *Transition* for each of the 3 digits corresponds to an axis(x, y or z). It can be shown that we can start from any particular combination and represent it as (0 0 0) and arrive at the other combinations. In order to simplify the representation and understand it in a better way by limiting ourselves to the positive axis only, we take the absolute values of *Transition*. We emphasize that our 3-D model, depicted in *Figure 1*, is discussed to simplify our understanding of the underlying theory. It can be extended to higher dimensions and adding more complexity.

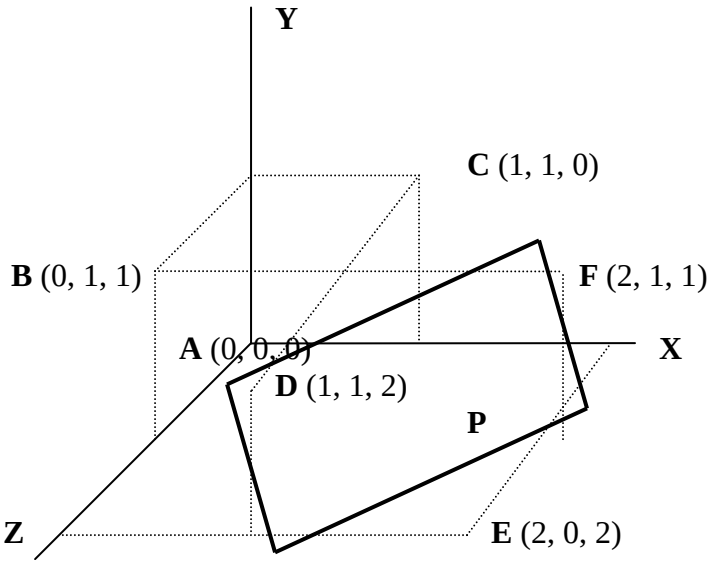

Figure 2: The plane *P*, represented by the bold lines, separates the two sets {A, B, C} and {D, E, F}. But no single plane can separate the sets {A, D} and {B, C, E, F} in this 3-dimensional space.

We reiterate that the idea is to form a concept lattice from this information and see if we can if we can use this abstraction to study different classes of patterns and understand the separability issues. In *Figure 1*, if {A, B, C} form one set of points (corresponding to one set of sequences of system calls) and {D, E, F} form another set of points (corresponding to another set of sequences of system calls), then they can be separated by a plane *P* in our model as shown in *Figure 2*. But a single plane will not suffice to separate the sets {A, D} and {B, C, E, F}. More than one plane would be required to demark boundaries between these two sets.

Relating our representation for the purpose of intrusion detection, the different strings correspond to the finite number of system calls. Intrusive sequences of system calls show a deviation from sequences during normal system behavior. These sequences are represented as points in our explanation above. We can then compute the complex hulls of the two sets of points. Thus, our task narrows down to map those sequences

as points using our representation and try to find if there exists a hyperplane which can demarcate the two complex hulls corresponding to the two different sets of points.

CONCLUSIONS AND FUTURE WORK

Some advances were made in the direction of obtaining an abstract representation for finite length strings, but we still haven't been successful in finding a definite one. Many questions remain unanswered and new questions arise – is the adjacency that we tried to create beneficial in any way? The adjacency representation in our model seemed trivial. What other attributes should be included in the model to successfully build the representation? Is the space we are trying to study really Euclidean or is it Non-Euclidean? Also, using our model we can try to find the intersection of the 2 sets of points using the concept of overlap of their respective convex hulls. The convex hull overlap condition is sufficient, but is it necessary? Studying our representation for application in intrusion detection systems, a point worth noting is that we observe sequences using a fixed sized window (i.e. short term correlations between strings). So should the number of axes be equal to the size of the window we use or should it be equal to the cardinality of the finite set of system calls? The answer to this question is the latter option, but we could map it to a lower dimension by drawing its projection. Again, the issue that comes up is that the convex hull overlap might not occur in higher dimensions but they may overlap in lower dimensions. Here again we face the aforementioned issue of necessary and sufficient conditions for convex hull overlap. A possible solution might be to map the convex hulls to a higher dimension. A detailed study needs to be pursued in this direction.

REFERENCES

[1] A. Jarvis. *On the identification of the convex hull of a finite set of points in the plane*. Info. Proc. Letters, 2:18-21, 1973.

[2] A. Jozwik, *A Recursive Method for the Investigation of the Linear Separability of Two Sets*, Pattern Recognition, Vol. 16, No. 4, pp. 429-431 (1983).

[3] C. Barber, D. Dobkin, and H. Huhdanpaa. *The Quickhull algorithm for convex hull, ACM Trans. on Mathematical Software*, 22:469-483, 1997.

[4] Christopher J. C. Burges. *A Tutorial on Support Vector Machines*, Data Mining and Knowledge Discovery, 1998.

[5] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, 1994.

[6] *Linear Separability of Sets* – http://www.sable.mcgill.ca/~flynn/644.index.html

[7] R. Graham. *An efficient algorithm for determining the convex hull of a finite planar point set*. Info. Proc. Letters, 1:132-133, 1972.

[8] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. *A Sense of Self for Unix Processes*. In Proceedings of 1996 IEEE Symposium on Computer Security and Privacy (1996).

[9] *Threshold Logic Units* - http://www.cs.bham.ac.uk/~jlw/sem2a2/Web/