Scientific Research Publishing

Journal of Information Security, 2018, 9, 70-84
http://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

# On the Use of *k*-NN in Anomaly Detection

**Theocharis Tsigkritis, George Groumas, Moti Schneider**

PCCW Global, Athens, Greece
Email: ttsigkritis@crypteianetworks.com, ggroumas@crypteianetworks.com, mschneider@crypteianetworks.com

**How to cite this paper:** Tsigkritis, T., Groumas, G. and Schneider, M. (2018) On the Use of *k*-NN in Anomaly Detection. *Journal of Information Security*, **9**, 70-84.
https://doi.org/10.4236/jis.2018.91006

**Received:** December 11, 2017
**Accepted:** January 13, 2018
**Published:** January 16, 2018



## Abstract

In this paper, we describe an algorithm that uses the *k*-NN technology to help detect threatening behavior in a computer network or a cloud. The *k*-NN technology is very simple and yet very powerful. It has several disadvantages and if they are removed the *k*-NN can be an asset to detect malicious behavior.

## Keywords

*k*-NN, Fuzzy Logic, Matching Process, Network Security

## 1. Introduction

### 1.1. General

Machine learning is one way to handle threatening behavior on the cloud or a network. Given a database containing the activities on the network, we can utilize some classifying algorithms in order to classify suspicious behavior. The database contains **attributes** (columns) and **records** (rows). Each attribute can be associated with a weight that describes the attribute's importance. Given a new record, R, we need to find K records from the database $\{r_1, r_2, \cdots, r_k\}$ that are R's nearest neighbors. Finding the neighbors is easy, but how do we determine the value of K? If we can determine that value, the *k*-NN algorithm becomes very simple, useful and autonomous algorithm. In the section below, we describe other popular classifiers, and then justify our decision to choose *k*-NN as the algorithm that will solve the anomaly problem in a network or a cloud.

### 1.2. Literature Survey

Classification, which is the task of assigning objects to one of several predefined categories, is a problem that encompasses many diverse applications [1]. A classifier is a supervised function (machine learning tool) where the learned (target) attribute is categorical ("nominal"). This can be viewed as a simple function in the form

$$Y = f(X)$$

where $X$ is a set of attributes and $Y$ is the set of classes. The classification model is used after the learning process classifies new records (data) by giving them the best target attributes (prediction). The target attribute can be one of $k$ class membership. This can be shown in Figure 1.

In machine learning , classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known [2].

So, the input data for the classification task is a collection of records. Each record is characterized by a set of descriptive attributes and class attributes (output class). Also each attribute is associated with a weight [2].

There are two major classification models:

- Descriptive models, in which we explain the difference between several classes, and
- Predictive model in which we are able to predict to which class a new incoming data will belong to.

There are many tools that classify data. One of the most popular tools used in classification is the decision tree [3]. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node. There are many specific decision-tree algorithms. C4.5 is probably the most popular of them [4].

Other important classifiers are neural networks [5], and Support Vector Machines (SVM) [6]. Artificial Neural networks (Ann) are system that emulates the way the human brain learns. SVMs are supervised learning models that analyze data used for classification. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Probably the simplest classifying algorithm is the "$k$-NN" algorithm. $k$-NN—The $k$-Nearest Neighbors algorithm (or $k$-NN for short) is a non-parametric method used for classification [7]. The input consists of the $k$ closest training examples in the feature space. The output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its $k$ nearest neighbors. In the classical

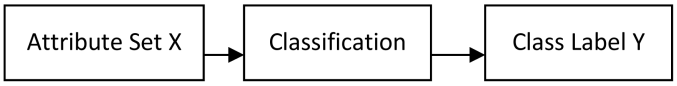


**Figure 1.** Classifying X into Y.

approach, one needs to specify the value of k [8]. One interesting application using *k*-NN is in the area of intrusion detection. This approach, based on the *k*-Nearest Neighbor (*k*-NN) classifier, is used to classify program behavior as normal or intrusive. This method shows that the *k*-NN classifier can effectively detect intrusive attacks and achieve a low false positive rate [9].

In our approach we use a dynamic assignment of ***k*** together with the fuzzification process defined by the fuzzy set theory [10]. This will be explained later.

## 2. Fuzzy Set Theory

The most fundamental concept of the fuzzy set theory is the membership function. This function computes the distance of an element to the set. If we assume that the "center" of the fuzzy set is a point in the membership function where the membership grade is 1, then we can rephrase the approach to the fuzzy set concept as described in Definition 1 below:

Definition 1: "Let C be the center of some attribute of some cluster, then C' becomes the same center after being converted to a fuzzy term".

This means that C is transformed from being a point to being a fuzzy term. The membership grade of 1 represents the original center point of the cluster, and the two end points represent the borders of the membership function (**Figure 2**).

Now, we need to match between a new data D and C'. To do so, we only need to perform the Fuzzification process. The Fuzzification process filters the domain data D thru the membership function to get the membership grade as shown in **Figure 3**.

The Fuzzification process has 2 major advantages here:

1) It avoid the need to normalize the data base

2) The Fuzzification process is the matching process itself

Next, we explain the process of cluster creation.

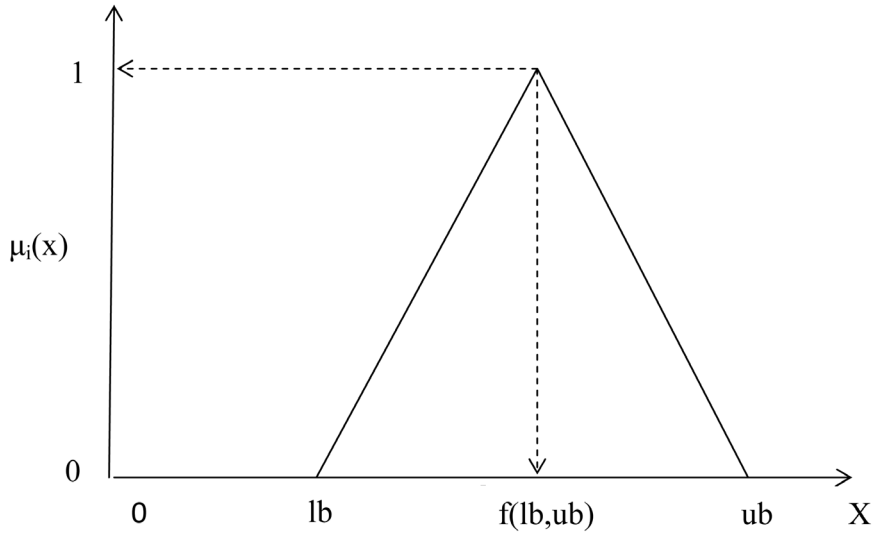


**Figure 2.** Logical structure of the fuzzy term.

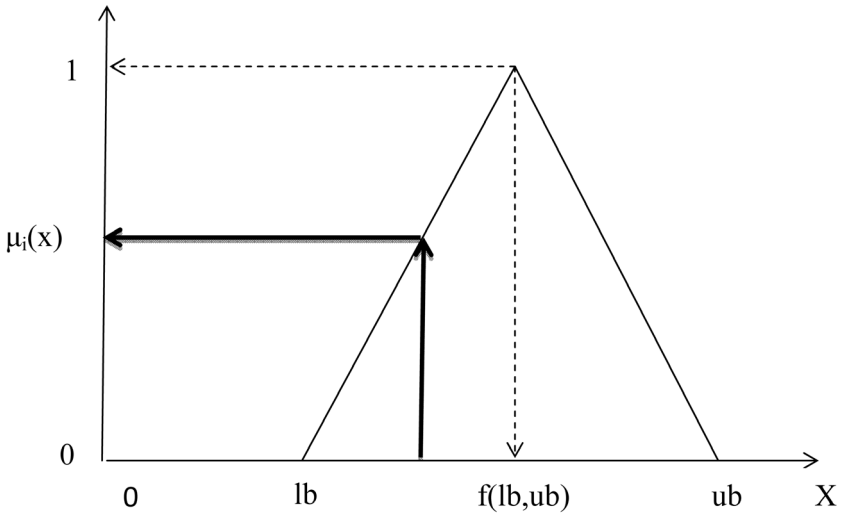


**Figure 3.** Example of a fuzzy variable.

## 3. *k*-NN

### 3.1. General

The ***k*-Nearest Neighbor's algorithm (*k*-NN)** is a non-parametric method used for classification [7]. The input consists of the $k$ closest training examples in the feature space. The output in $k$-NN *classification* is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its $k$ nearest neighbors ($k$ is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor [11]. The test sample (the circle) should be classified either to the first class of blue squares or to the second class of red triangles. If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

The general idea of the $k$-NN technology is described in **Figure 4** [11].

The $k$-NN algorithm is among the simplest of all machine learning algorithms. It can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of $1/d$, where $d$ is the distance to the neighbor [12]. In this work we will use a user defined weights on the attributes. It is important to note that this weight is different from the weight assigned to each attribute. This weight is really the distance of $r_i$ from the record $R$ and is computed via a fuzzification process as described above.

As was described above, the $k$-NN is a non-parametric algorithm where $k$ was defined in advanced. In our system we link this $k$ to some $T$, which is the threshold of similarity. As will be described later, $T$ is the same threshold used in the matching process (will also be described below). This will make k a very dynamic number that depends on the values of the neighboring records.

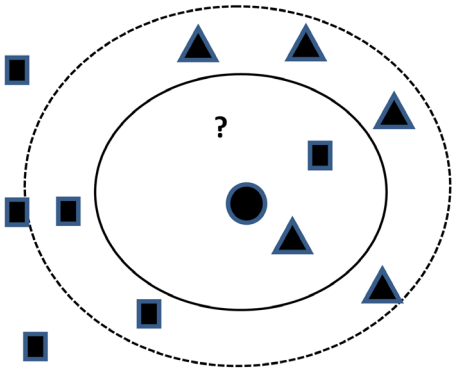


**Figure 4.** Example of *k*-NN classification.

The process of using *k*-NN is divided into several parts:

1) Set the matching threshold. This is done to determine if two records (the new one and the one from the training set) are similar.

2) Match the new record with all training records. This will be described below.

3) Determine the final result. Which means to decide if the new record is a treat or not.

Based on the above steps we can depict the logical structure as described in **Figure 5**.

### 3.2. Creating the Training Data Set

As mentioned above, the most important concern we have is to make sure that the learning data set and the testing data set will be different. To avoid bias, we require that the learning data set will be created randomly. In the following, we present our learning data set creation process.

1) Let $n$ be the size of the complete data set. Choose a random number r such that

$$0.4 * n < r < 0.6 * n$$

In other words, we want the size of the learning data set be about half of the size of the entire data set.

2) From the entire data set choose r different records:

For $i = 1$ to $r$ do

- Generate a random number between 1 and $n$ ($m$)
- If $m$ is not in the learning data set, add it to the learning data set
- If $m$ is in the data set do nothing

End.

At the end of this process we have created a data set containing $r$ different and randomly chosen data points from the initial data set.

### 3.3. The Matching Process

The matching process compares two records and computes its similarity. A record is defined as a collection of attributes, each one with a different type and

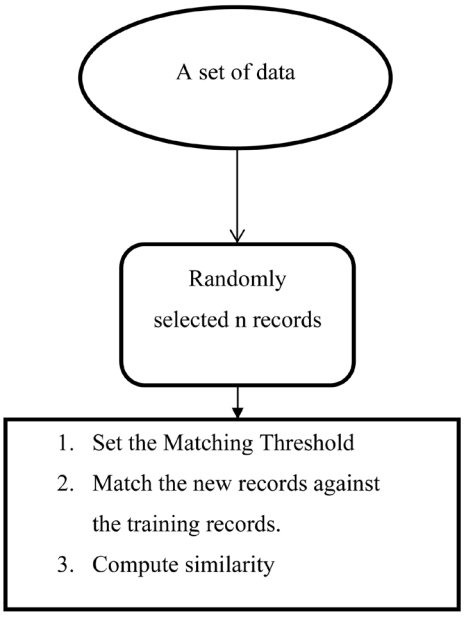


**Figure 5.** Logical structure of the *k*-NN system.

each has a different weight associate with it. In other words, the data that is collected and analyzed contain many attributes. Some of these attribute may be more important (or influential) than others. This importance can be expressed as the "weight" of the attribute (or variable). In our final decision making we will take the weight into an account. If the variables are numeric then the result is numeric (in the interval [0 ··· 1]). Otherwise it is Boolean (1 if the two attributes identical and 0 otherwise). Also note that not all variables have to participate in the matching process. We will touch on that in the analysis phase discussion. If the attribute is a non-numeric attributes then we match the attributes in a Boolean fashion. If the attributes match, assign the value of 1, otherwise assign the value of 0.

If the attributes are numeric then we perform the following steps:

1) The number representing the attribute within the training set we expend to a fuzzy term (**Figure 2**).

2) Match the value of the testing record by performing a Fuzzification process (**Figure 3**).

3) The result of the Fuzzification process is the matching result since it represents the degree to which one number belongs to the fuzzy set created by the other.

4) Add the results from all the matches performed on the attributes of the two records (one represents a record from the testing set, and the other represents a record from the learning set), such that the final result $R$ is defined as:

$$R = \frac{\sum r_i w_i}{\sum w_i} \tag{1}$$

where $r_i$ is the matching result and $w_i$ is the importance (weight) of the attribute.

Let $a_c$ be the center of the fuzzy term generated for some attribute value from the testing data and $a_d$ be the value of the attribute from the historical data then:

1) If $a_c$ and $a_d$ are Boolean or linguistic data then:

$$r_i = \begin{cases} 1 & \text{if } a_c = a_d \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

2) Else, If $a_c$ and $a_d$ are numeric then the matching $r_i$ is the Fuzzification process of $a_d$ in $a_c$:

$$r_i = \mu_{a_c}(a_d) \quad (3)$$

Equation (1) adds the weight factor into the matching process. The matching process can be summarized in the following flowchart (Figure 6).

## 4. The Testing Procedure

In this stage we match a new incoming record with the training set defined above. We used 7 variables (attributes) in the matching process: ([*firewall_block_count, failed_login_count, email_count, spam_count, msg_bytesize, orgid, value*])

Basically, we take a record from the test file and compare it with the entire training set. If the similarity is above a certain threshold, then we have a match. It is important to note that contrary to the "classic approach" in which we define a priory the number of similar records ($k$ in the $k$-NN). These $k$ records will be stored on a bin (or a list L) for further evaluation. In our system we select all

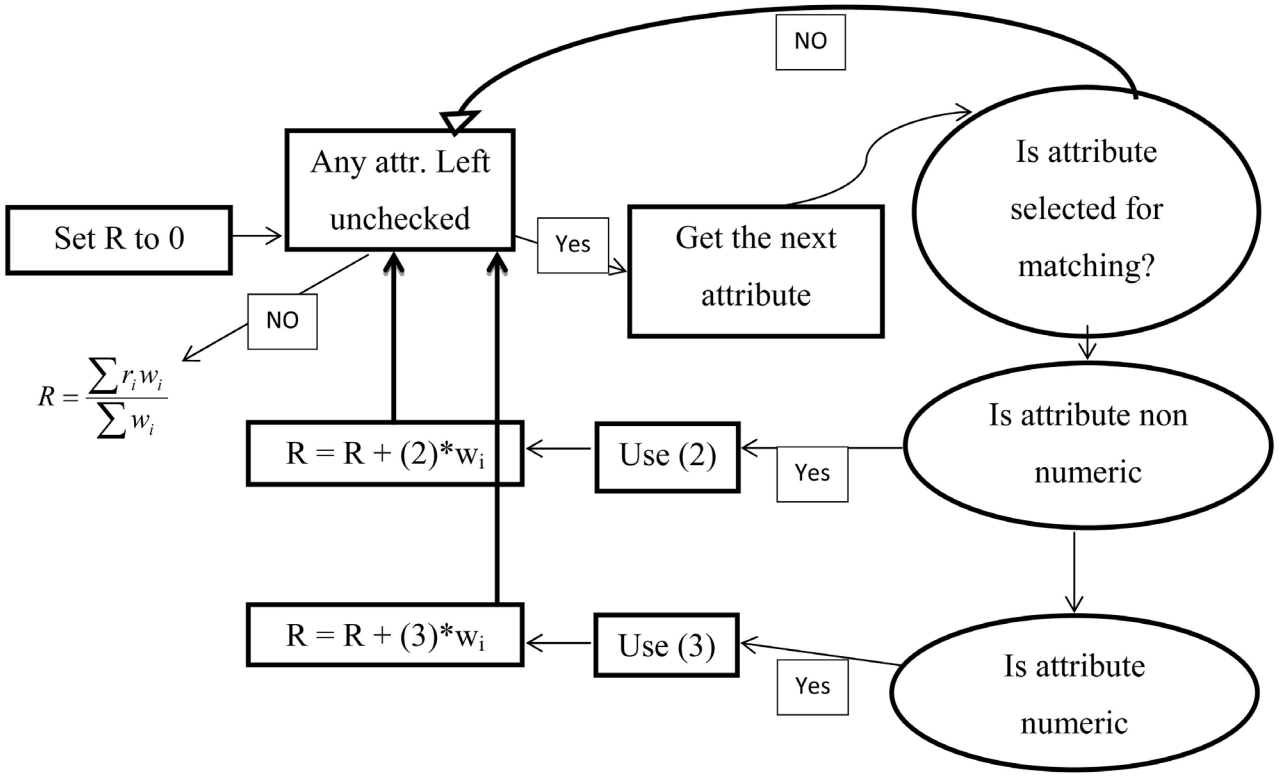


Figure 6. The matching process.

records that their similarity to the record in question is above a pre-defined threshold. So $k$ can be any positive value and therefore the bin can be of variable size. Formally, let $x$ be a record in the training set, $r$ be the record in question (the new record), and $T$ be some threshold, then

$$L = \forall x\left(sim\left(x,r\right) > T\right) \tag{4}$$

Figure 7 depicts the idea.

The process is as follows:

**For each record in the testing set do:**

**1) Match the new record against the entire training set.**

**2) If the matching result is above a given threshold, then go to step 5, else go to 3.**

**3) If the matching result is less than the threshold, try the next option in the list described below.**

**4) Go to 1.**

**5) Add the record to a given list ($L$) of matched records.**

Algorithm 1: Creating the list $L$

Algorithm 1 is summarized in Figure 8.

At the end of this process, the list $L$ contains $n$ records.

Now, each record contains many attributes, some are used for the matching process (as was described above) and 3 other attributes for postariori evaluation. These 3 important attributes are:

- IP addr (IP address)
- IP_Group addr (IP Group address). The Ip_Group is a variable containing information regarding sets of IP addresses, such as network, organization, cloud, etc.

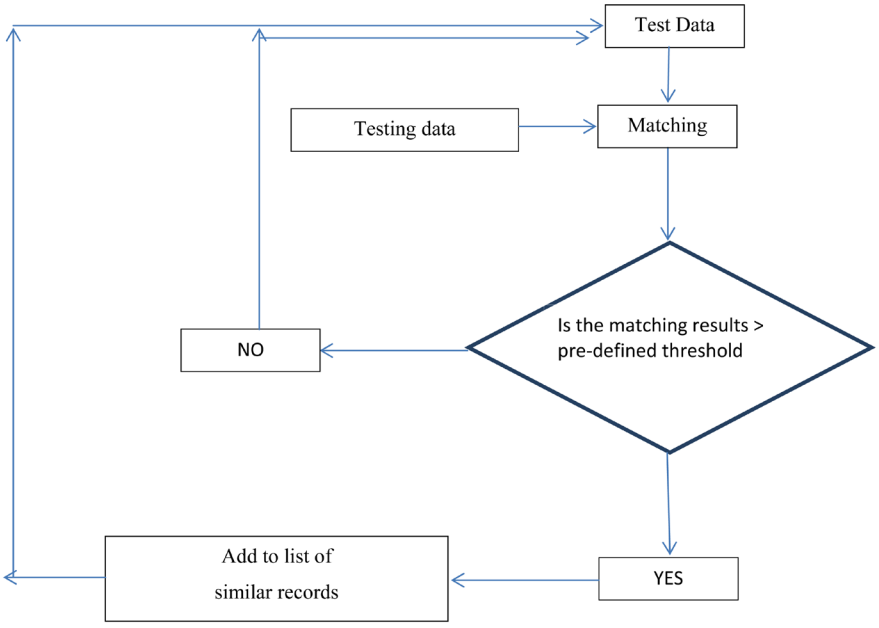


Figure 7. The testing procedure.

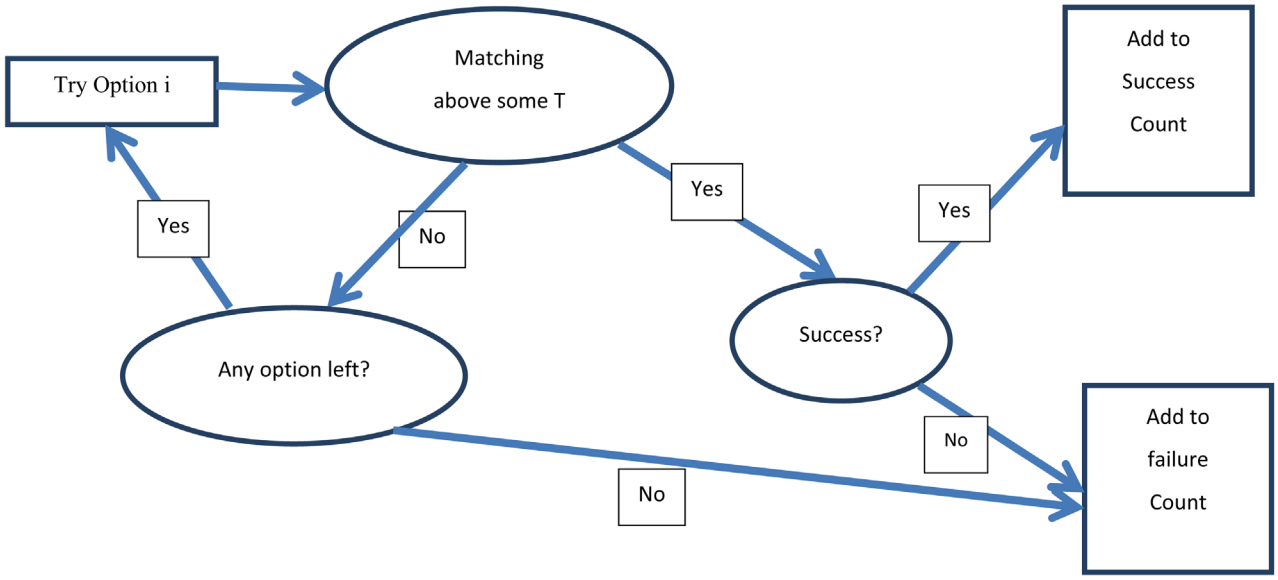


**Figure 8.** Testing the system using *k*-NN.

- Treat/normal Boolean flag

Also let NR abbreviate the new record and L[] be the list of similar records, then we can perform the following algorithm:

**Loop: I = 1 to n**
**If (NR.IP_addr=L[i].IP_addr) and (L[i].Treat=True) and (NR.Treat=True) then**
**Increment IP_Treat_positive_count (False Positive)**
**Else If (NR_IP_addr=L[i].IP_addr) and (L[i].Treat=False) and (NR.Treat=True) then**
**Increment IP_Treat_negative_count (False Negative)**
**Else If (NR.IP_addr=L[i].IP_addr) and (L[i].Treat=True) and (NR.Treat=False) then**
**Increment IP_Normal_negative_count (True Negative)**
**Else If (NR_IP_addr=L[i].IP_addr) and (L[i].Treat=False) and (NR.Treat=False) then**
**Increment IP_Normal_positive_count (True Positive)**
**Else If (NR.IP_Group_addr=L[i].IP_Group _addr) and (L[i].Treat=True) and (NR.Treat=True) then**
**Increment IP_Group _Treat_positive_count (False Positive)**
**Else If (NR_IP_Group_addr=L[i].IP_Group _addr) and (L[i].Treat=False) and (NR.Treat=True) then**
**Increment IP_Group_Treat_negative_count (False Negative)**
**Else If (NR.IP_Group _addr=L[i].IP_Group _addr) and (L[i].Treat=True) and (NR.Treat=False) then**
**Increment IP_Group _Normal_negative_count (True Negative)**
**Else If (NR_IP_Group_addr=L[i].IP_Group _addr) and (L[i].Treat=False) and (NR.Treat=False) then**

**Increment IP_Group _Normal_positive_count (True Positive)**

Algorithm 2: Computing different counts

Based on Algorithm 2 we define:

$$\text{True Positive} = \text{IP\_Normal\_positive\_count} + \text{IP\_Group\_Normal\_positive\_count} \tag{5}$$

$$\text{False Positive} = \text{IP\_Treat\_positive\_count} + \text{IP\_Group\_Treat\_positive\_count} \tag{6}$$

$$\text{True Negative} = \text{IP\_Normal\_negative\_count} + \text{IP\_Group\_Normal\_negative\_count} \tag{7}$$

$$\text{False Negative} = \text{IP\_Treat\_negative\_count} + \text{IP\_Group\_Treat\_negative\_count} \tag{8}$$

So, in the above procedure we check the new record against the list L to look for similarity with respect to threat. If we fail we want to repeat the process with respect to normal behavior.

The idea is to check one option at the time until we find a similarity. If nothing found we place the result in the outlier bin. This may indicate a new type of record or a corrupted one.

Using (5), (6), (7), and (8) we can:

1) Use the *k*-NN algorithm, to check if the incoming record IP is a threat (False Positive).

2) Use the *k*-NN algorithm, to check if the incoming record IP is not a threat.

3) Use the *k*-NN algorithm, to check if the incoming record IP group is a threat.

4) Use the *k*-NN algorithm, to check if the incoming record IP group is not a threat.

We define **Precision** and **Recall** as:

$$\text{precision} = \frac{\sum \text{True Positive}}{\sum \text{True Positive} + \sum \text{False Positive}} \tag{9}$$

$$\text{recall} = \frac{\sum \text{True Positive}}{\sum \text{True Positive} + \sum \text{False Negative}} \tag{10}$$

At the end of the process we compute **Recall** and **Precision** for final evaluation.

## 5. The Simulation

In this section we describe how to choose the best attributes from the data base. The learning system determines, which variables will be used in identifying a *threat*, and the best values for the chosen variables.

### 5.1. Choosing the Variables and Possible Values

In our system we chose all the variables that can contribute to the identification of a threat. Different variables have different values. Due to the fact that some of

the variables are Boolean, we can assign them the values 0 or 1. Some variables are linguistic (*i.e.* have a finite non numeric values), so we need to check all possible values. Other variables are numeric. In this case we have to choose some representative values. For example, the variable "threshold" is numeric. It represents how well a certain record matches the set of clusters. It is logical to assume that the matching should be between 0.7 and 1. In other words, if the matching between the record and the cluster centre is below 0.7 then it should not be accepted as a member of that cluster. We divided the range of accepted matches to low (above 0.7), medium (above 0.85) and high (above or equal to 0.99). The more options we provide, the more complex will be the computation (this will be described below).

After examining all the variables, we create a multivariable loop. Let $x_i$ be a variable of type integer. Also, $x_i$ corresponds to a real variable (such as "threshold", etc.), $y_i$. Also let $a_i$ be the lowest value in the domain of $x_i$ and $b_i$ be the highest value in this domain. So a simple loop will be:

$$\text{For} \quad x_i = a_i \quad \text{to} \quad b_i \quad \text{do } S,$$

where $S$is an executable statement.

In the example above the loop will execute from 0 to 2, such that the value 0 in $x_i$ corresponds to 0.7 in the real variable "threshold". It should be noted that the number of permutations can be large. In particular, for a set of $n$ variables, let $P$ be the number of permutations, so, we have:

$$P = x_1 \times x_2 \times \cdots \times x_n - 1$$

Generally, the algorithm, without optimization, for variable selection is $O\left(t * 2^n\right)$, where $t$ is the number of threshold's levels (e.g. 3, levels: [0.7, 0.85, 0.99]), and $n$ is the number of features (e.g. 7, as described above)

Each Boolean variable shows if some numerical variable participates in the computation. For example, If the 6th Boolean is set to false, this means that the numerical variable "orgid" does not participate in the simulation. So, in this case $P = 2^7 * 3 * 3 - 1 = 1151$. So, in the first stage we simulate the system with all possible values. This means that we run the simulation 1151 times (we omit the case where all Boolean variables are false) and place the simulation results on some file. The result file contains all the results computed above. It also contains the information regarding the variables participated in the simulation.

The problem is that if the variable set is large, then $t * 2^n$ becomes unmanageable as the problem is NP-hard. To solve the problem, we divide the process into 2 parts. In the first part we run each attribute independently. This will result with $O\left(t * n\right)$. Next, we divide the results into 3 groups: The first group contains attributes that consistently led to unsatisfiable results (their matching results were below 0.7). The next group is the group that produced satisfiable results (matching results above 0.9). The third group contains attributes that their matching results are inconsistent. So, the first group and the third group does not require any permutation, and the second group contains a small number of attributes (say $m$). So the complexity of the computation is reduced from $O\left(t * 2^n\right)$ to $O\left(t * 2^m\right)$, where $m \ll n$.

### 5.2. Tuning Up

The tuning up process deals with finding the best values for the variables chosen to be part of the system. The learning system determines which variables will be used in identifying a threat, and the best values for the chosen variables.

As was stated, first we choose all the variables that can contribute to the identification of a threat. Basically, these are all the variables in the database containing numeric or Boolean information. Different variables have different values. After running the system once, we get $P$ different results (depends on the size of $m$). From this set we choose the cases that their Success rate is the highest. In other words, we choose the permutations that generated the best results in the matching of the new data point with the clustering system to find out if the new data point is a *threat* or not.

If we select more than one case, we observe the number of outliers (# of cases the system rejects). In our system we allow less than 10% of outliers (from the given test files). The reason for having a threshold for checking the number of outliers in the system is only for reducing the amount of valid results. If the number of the outliers is more than 10%, then the possibility of rejecting non threat data increases. Note that the idea here is to reduce the number of possible permutations. The goal is to generate one set of values that will stand the tests we will describe later.

If we still have more than one option we will select the case that the integer values are the highest. We have 2 numbers in the variable pool, each having the values 0.7, 0.85, 0.99. One of the two variables is the threshold. It determines if the new record belongs to a cluster or not. We test the data with low threshold (0.7), medium threshold (0.85) and a high threshold (0.99). We obviously look for the high threshold. The problem is that high threshold will not guarantee stability. Therefore, we need to test several thresholds. The second numeric variable is responsible for the shape of the centre of each attribute in the centre vector of the cluster (we denote it as the "shape" variable). Let $x$ be the value representing the centre of an attribute of the cluster and S be the "shape" value. Then we want to expend $x$ to a fuzzy term with a trapezoid shape as described in Figure 9.

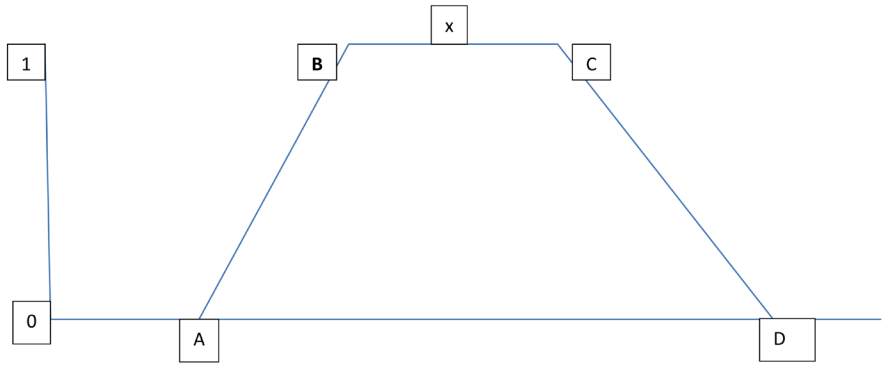


Figure 9. Fuzzy term described as a trapezoid.

The values of the points $A$, $B$, $C$, and $D$ in Figure 9 are computed as:

$$\begin{aligned} B &= xS \\ A &= BS \\ C &= x+(1-S)x = x(2-S) \\ D &= C+(1-S)C = C(2-S) \end{aligned} \tag{11}$$

For example, if $x = 100$, then $B = 70$, $A = 49$, $C = 130$, $D = 169$. The trapezoid created is not symmetric. If we want to make the trapezoid symmetric, we define $y$ such that

$$y = xS \tag{12}$$

And

$$\begin{aligned} A &= x-2y \\ B &= x-y \\ C &= x+y \\ D &= x+2y \end{aligned} \tag{13}$$

If we still have more than one option we will select the case that has the most Boolean variables having the value "true". This means that we chose the case where as many as possible variables from the variable list described above are participating in the simulation.

This guarantee that the system will choose only one case and this case has the best characteristics.

After selecting the best variables and the best values for those variables we move to the final stage, or the final test.

After choosing the best set of values for the variables to determine if some IP is a threat or not, we use these values to simulate the system 200 times to ensure consistency. The reason for choosing this large number is because of the rule of large numbers. If we show consistency in this simulation we can ensure that statistically the consistency will hold. Consistency is defined as having the same results (or very close to it). The results are stored on a file for further analysis. The analysis showed that in almost all cases we simulated (98%), we got very high success. Success is defined as a case where the prediction (threat/normal) is the same as the actual values.

We have repeated the entire process described above 500 times and got consistent and very good results.

### 5.3. Example

As was stated above, in the first run we simulate results with all permutated values (P). The system sorts the results and selects only the results that their success rate is above 98% (Figure 10).

After cutting the results with less than 99% success we are left with 9 cases. From the 9 cases remained, the system selected 1 cases in which the number of outliers is less than 10%, $T$ was 0.70 and $S$ was 0.85. This permutation was chosen.

Then we run the system 200 times to check consistency. The success rate was 99%. This is shown in Figure 11.

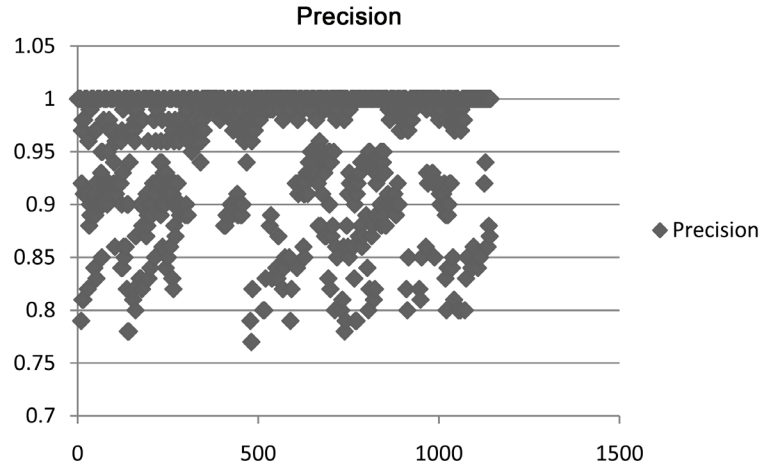


**Figure 10.** Initial results.

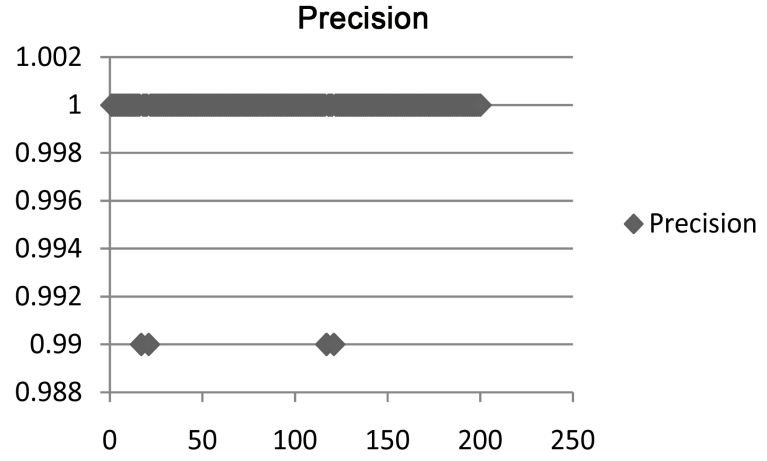


**Figure 11.** Simulation results.

Success is measured by counting the number of cases we predicted correctly divided by the number of cases tested. The outlier rate was less than 8%, that is, the number of records that were rejected due to the fact that their matching result was below the given threshold, was less than 8%. The matching threshold T was set to 0.99 (almost a binary case) and the shape value S was set to 0.84. That concluded that the system was very consistent.

We repeated the process 500 times and the results were consistent with the selected permutations.

## 6. Conclusions

The algorithm above was implemented using log database from a communication system developed in PCCW Global. We used random numbers to generate the learning and testing data. We run the simulations 500 times and got recall and precision close to 1.

We implemented and tested using actual network traffic from PCCW Global backbone network. It is used to generate early notifications regarding suspicious

IPs that although no security information was available, they were observed with similar traffic behavior with IPs that have been involved in network security incidents in given time context. We used random numbers to generate the learning and testing data, by running the simulations 200 times to optimize parameters. The presented approach is related to intellectual property protected by the US patent (provisional) with US Application No.: 62/439332.

## References

[1] Cherkassky, V. and Mulier, F. (1998) Learning from Data: Concepts, Theory, and Methods. Wiley Interscience, Hoboken.

[2] Duda, R.O., Hart, P.E. and Stork, D.G. (2001) Pattern Classification. 2nd Edition, John Wiley & Sons, Inc., New York.

[3] Rokach, L. and Maimon, O. (2008) Data Mining with Decision Trees: Theory and Applications. World Scientific Pub Co Inc., Singapore.

[4] Quinlan, J.R. (1993) C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, Burlington, Massachusetts.

[5] LeCun, Y., Bengio, Y. and Hinton, G. (2015) Deep Learning. *Nature*, **521**, 436-444. https://doi.org/10.1038/nature14539

[6] Figueiredo, M.A.T. and Jain, A.K. (2002) Unsupervised Learning of Finite Mixture Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**, 381-396. https://doi.org/10.1109/34.990138

[7] Altman, N.S. (1992) An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, **46**, 175-185.

[8] Zadeh, L.A. (1965) Fuzzy Sets. *Information and Control*, **8**, 338-353. https://doi.org/10.1016/S0019-9958(65)90241-X

[9] Hall, P., Park, B.U. and Samworth, R.J. (2008) Choice of Neighbor Order in Nearest-Neighbor Classification. *Annals of Statistics*, **36**, 2135-2152. https://doi.org/10.1214/07-AOS537

[10] Samworth, R.J. (2012) Optimal Weighted Nearest Neighbour Classifiers. *Annals of Statistics*, **40**, 2733-2763. https://doi.org/10.1214/12-AOS1049

[11] *k*-Nearest Neighbors Algorithm (2017). https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[12] Yihua, L. and Vemuri, V.R. (2002) Use of K-NN for Intrusion Detection. *Computer & Society*, **21**, 439-448.