Optimizing an Artificial Immune System Algorithm in Support of Flow-Based Internet Traffic Classification

Brian Schmidt, Ala Al-Fuqaha, Ajay Gupta, Dionysios Kountanis
Computer Science Department
College of Engineering and Applied Sciences
Western Michigan University
Kalamazoo, Michigan, USA
{brian.h.schmidt, dionysios.kountanis, ajay.gupta, ala.al-fuqaha}@wmich.edu

Abstract—**The problem of classifying traffic flows in networks has become more and more important in recent times, and much research has been dedicated to it. In recent years, there has been a lot of interest in classifying traffic flows by application, based on the statistical features of each flow. Information about the applications that are being used on a network is very useful in network design, accounting, management, and security. In our previous work we proposed a classification algorithm for Internet traffic flow classification based on Artificial Immune Systems (AIS). We also applied the algorithm on an available data set, and found that the algorithm performed as well as other algorithms, and was insensitive to input parameters, which makes it valuable for embedded systems. It is also very simple to implement, and generalizes well from small training data sets. In this research, we expanded on the previous research by introducing several optimizations in the training and classification phases of the algorithm. We improved the design of the original algorithm in order to make it more predictable. We also give the asymptotic complexity of the optimized algorithm as well as draw a bound on the generalization error of the algorithm. Lastly, we also experimented with several different distance formulas to improve the classification performance. In this paper we have shown how the changes and optimizations applied to the original algorithm do not functionally change the original algorithm, while making its execution 50-60% faster. We also show that the classification accuracy of the Euclidian distance is superseded by the Manhattan distance for this application, giving 1-2% higher accuracy, making the accuracy of the algorithm comparable to that of a Naïve Bayes classifier in previous research that uses the same data set.**

Keywords— artificial immune system; Internet traffic classification; multi-class classification; machine learning

1. Introduction

Because of recent changes in the regulatory climate of the Internet, as well as the business needs of Internet Service Providers, the traffic classification problem has become an important research topic. Some concerns are: the struggle between malicious users and security professionals, network neutrality, and the use of networks for sharing copyrighted material. The task of optimizing the flow of traffic across a network is also related to this problem, since some applications rely on low latency to provide Quality of Service, while others are unaffected by it. The goal of network operators has been to classify network traffic according to the application that generated it, which is highly correlated to the type of information contained in it. On the other side, application developers have sought to hide the identity of their application's packets on the network by obfuscating their signature.

In the early days of the Internet, an application could easily be identified by the port number used in the transport layer. This approach is very simple and still useful, but is not accurate enough in the modern Internet, since certain applications are able to fool this method by negotiating port numbers dynamically, making it impossible to reliably identify them.

Deep packet inspection is also useful when doing network traffic classification, and involves analyzing the contents of packets. To find the patterns used by certain applications, regular expressions are often used. There are a few shortcomings to this approach as well, since using encryption allows users to easily hide their data from this method, while also being very resource intensive, since every packet has to be examined. There are also concerns about the privacy of users [1].

A non-intrusive technique used in recent years is traffic flow classification based on features of the flow, in which the statistical properties of traffic flows are calculated and used to identify the generating application by comparing the information to previously-learned models. Some example features are: inter-packet arrival time, average packet size, and packet counts.

Another way to identify the applications that are generating network traffic is by looking at the interactions that a host engages in, and comparing them to behavior signatures that are associated with certain application servers. This approach to traffic classification depends strongly on the topological location of the monitor, and performs well when the monitor sees both directions of the flow under inspection [1-2].

The focus of this paper will be to utilize the statistical features of network flows to identify the generating application. We will accomplish this by using a multi-class Artificial Immune System inspired classification algorithm. We are encouraged to try this approach because of the use of AIS algorithms in similar network traffic classification problems.

Our proposed approach uses fewer parameters than other natural computing algorithms and does not incur the training costs associated with discovering such parameters. For example, the performance of the genetic algorithms highly depends on the mutation and cross-over operations and parameters. Similarly, the performance of artificial neural network, deep learning and extreme machine learning based approaches depends highly on the number of hidden layers, the number of neurons in each layer and the employed activation function. Also, the performance of SVM is highly dependent of the Kernel function used and its parameters. In this paper, we propose an optimized AIS algorithm that needs few parameters and produces results comparable to these produced by the optimal parameters of the aforementioned methods. Therefore, the proposed approach eliminates all the overhead and subjectivity involved in the selection of the parameters in other biologically inspired approaches.

Furthermore, Artificial Immune System algorithms are able to operate in highly distributed systems and can be easily adapted to run on networked computers. AIS algorithms are capable of learning new patterns, remember previously learned patterns, and do pattern recognition in networked systems. At the same time, their performance degrades gracefully, in the same way as Artificial Neural Networks. In past research, AIS algorithms have been used to detect malicious activity in computer networks[3]. Because of this research and the capabilities of AIS classifiers we are encouraged to explore their performance on the task of network flow classification. Research has also shown that positive selection AIS algorithms can perform very well while also being simpler to code and faster to train. For this reason, the algorithm presented in this paper is a positive selection algorithm.

The original algorithm described here is designed to be simple and fast so that it will work well in resource-constrained systems. Because of our previous findings, we have been motivated to develop optimizations for the algorithm, to make it competitive with other Machine Learning approaches while depending on lesser configurable parameters.

When testing the optimizations made to the algorithm, a speedup of about 10x-30x was achieved in the training algorithm. A speedup of around 2x was observed in the classification portion of the algorithm. No significant differences where observed in the accuracy of the optimized and unoptimized algorithms. When testing different distance functions, it was observed that Manhattan distance was 1% to 2% more accurate for the data set used.

The rest of the paper is organized as follows. Section II introduces the traffic flow classification problem along with other solutions found in the literature. Section III introduces artificial immune systems, including their biological inspiration. Section IV introduces the problem under investigation, places our own solution in context, and describes our own classifier, inspired by AIS principles. Section V describes the changes that we made to the algorithm to optimize it. Section VI deals with an analysis of the performance of the algorithm, section VII explains the tests performed on the algorithm, including the data set used. Section VII shows the results of the tests, section IX contains our conclusions and recommendations for future work.

2. Background

2.1 The Flow Classification Problem in Machine Learning

In [4], Moore and Zuev applied a Naive Bayes classifier to the traffic classification problem. A simple naive Bayes classifier did not do very well at first, with an average 65.3% classification accuracy. The accuracy rose, however, when kernel density estimation and Fast Correlation-Based Filter (FCBF) feature reduction were applied. The techniques were tested separately and jointly, with the best performance achieved when both techniques were used at the same time, achieving 96.3% classification accuracy.

In [5], a survey is carried out of the state of traffic classification, including a review of all approaches to the problem, as well as all machine learning algorithms that have been tested. Furthermore, an introduction to elephant and mice flows and early classification was provided. The authors also highlighted the processing time, memory, and directional neutrality aspects of the algorithms.

In [6], Alshammari and Zincir-Heywood, tested Support Vector Machines (SVM), Naive Bayes, RIPPER, and C4.5 algorithms using three publicly available data sets, focusing on classifying encrypted traffic. Singh and Agrawal [7] also tested several of the same ML algorithms as [6] on the task of traffic classification, the algorithms being: Bayes net, multi-layer perceptron, C4.5 trees, Naive Bayes, and Radial Basis Function Neural Networks. Both full-feature and reduced-feature data sets were tested and results compared, with the best classification accuracy achieved by C4.5 with a reduced feature data set. Lastly, [8] focused on the accurate classification of P2P traffic using decision trees, achieving between 95% and 97% accuracy.

The selection of flow features for classification has also been studied in the literature. [9] performs a survey of the reasons why some algorithms perform well on the traffic classification problem, as well as the features that are most useful. Their results show that there are three features in particular that are most useful: ports, the sizes of the first one or two packets for UDP flows, and the sizes of the first four or five packets for TCP flows. This paper also finds that Minimum Description Length discretization on ports and packet sizes improves the classification accuracy of all algorithms studied. In [10], feature selection for flow classification is tested. The best performance is achieved when using information from the first 7 packets with a one-against-all SVM classifier, confirming the findings of [9]. Specifically, [9] and [10] have shown that it is possible to classify a flow accurately with only limited information about it. Lastly, in [11], the data set is preprocessed by removing outliers and using data normalization, and performing dimensionality reduction. Decision Trees, Artificial Immune Networks, Naive Bayes, Bagging and Boosting classifiers are tested. Although Artificial Immune Networks are used in this research, they are substantially different algorithms from the one used in this research.

In [12], the authors use Extreme Learning Machines (ELM) to tackle the supervised learning network traffic classification problem. ELMs are like artificial neural networks, however, ELMs use randomized computational nodes in the hidden layer and generate their weights by solving linear equations. A similar approach is taken in [13], although the ELMs used are kernel based. In this study, over 95% accuracy was achieved using different activation functions.

In [14], the same problem is undertaken using an original approach that fuses Hidden Naïve Bayes and K* classifiers. Feature selection is done using Correlation Based feature selection and Minimum Description Length. In [15], the researchers build an anomaly detection system using machine learning techniques. The system is meant to detect anomalies within the traffic in a cellular network and it is built using Random Neural Networks. The approach is tested on synthetically generated data.

The research in [16] seeks to identify traffic flows generated by a mobile messaging app called WeChat. To achieve this, 50 features were extracted from every traffic flow within two data sets. Several different classification approaches were used, including: SVM, C4.5, Bayes Net and Naive Bayes are applied to classify the WeChat text messages traffic. Very high accuracy was achieved in both data sets. The research contained in [17] seeks to solve the traffic classification problem in the same way as other research presented in this section. They use SVM classifiers to classify flows into two categories: Video and Other. The researchers were able to achieve an accuracy of 90% and above.

Finally, some of the authors of this research have published some related work in [18] and [19]. The algorithm introduced in that research is a basic version of the one proposed in the current research. This paper extends that baseline algorithm and optimizes its performance. Furthermore, this paper presents a theoretical analysis of the proposed optimizations.

Table I contains extra information about previous research. In Table I, the references are listed along the left of the table, and different research topics are listed along the top. An "X" in the table signifies that the publication covers that topic.

2.2 Natural Immune Systems

Natural immune systems (NIS) are responsible for protecting organisms from outside threats, and are an integral part of mammals. They protect from bacteria, viruses, and parasites collectively known as pathogens. Immune systems are involved in two basic activities: recognition and removal of pathogens. The field of artificial immune systems is concerned mainly with the recognition aspects of the NIS, and how to emulate them in computers.

TABLE I. FLOW CLASSIFICATION LITERATURE

Reference	**Survey**	**Encrypted/ Obfuscated Traffic**	**Early Classification**	**Feature Selection**	**Support Vector Machines (SVM)**	**Artificial Neural Networks / Deep Learning / Extreme Learning**
[1]	X			X		
[2]	X					
[3]						
[4]	X			X		
[5]	X					
[6]	X	X		X	X	X
[7]	X			X		
[8]		X		X		
[9]	X		X	X		
[10]		X	X	X	X	
[11]	X			X		
[12]						X
[13]						X
[14]				X		
[15]						X
[16]	X				X	
[17]	X				X	

The NIS can be divided into two parts: the innate immune system, which is fixed and not adaptable, and the acquired immune system which is able to fine-tune itself to previously unseen pathogens. The qualities of the acquired immune system make it of specific interest to computer scientists, which have tried to emulate its flexibility. The space of possible pathogens and attack strategies is very large, yet the acquired immune system is capable of adapting to new pathogens and remember them for the future.

2.3 Training Methods

Two types of cells that are involved in the recognition and disposal of pathogens are T-cells and B-cells. The population of these cells present in the bloodstream is collectively responsible for the recognition and disposal of pathogens. The population acts collectively and is able to recognize new pathogens through two training methods: negative selection and clonal selection.

Through the process of negative selection, the NIS is able to protect the tissues of the host organism from being attacked by its own immune system. Certain cells generate detectors that recognize proteins, which are present on the surfaces of cells.

The detectors are called “antibodies” and are created randomly. Before the cells become fully mature, they are “tested” in the thymus, an organ located behind the sternum. The thymus is able to destroy any immature cells that recognize the tissues of the organism as “non-self.” Therefore, the process of negative selection maps the negative space of a given class, given examples of the “self” class. The negative selection algorithm first appeared in [20].

With clonal selection, the NIS is able to adjust itself to provide the most efficient response to an attack by a pathogen. Clonal selection happens when a detector cell finds a previously-seen pathogen in the organism, and clones itself to start the immune response. The cloning process, however, introduces small variations in the pattern that the detector cell recognizes. The number of clones that are created from a detector cell is proportional to the “affinity” of the cell to the new pathogen, which is a way to measure how well the cell matches the pathogen. The amount of variation allowed in the clones is negatively proportional to the affinity, as well, meaning that the cells with the most affinity are mutated less. Clonal selection is similar to natural selection, therefore the clonal selection algorithm is similar to the genetic algorithms that are based on natural selection [21]. However, clonal selection algorithms have fewer parameters than genetic algorithms, and do not require potentially complex operations such as crossover. Clonal selection algorithms are mostly used as optimization algorithms, although there have been a few used for classification. The clonal selection principle first appeared in [22].

Artificial immune systems algorithms used for classification are considered to be ensemble classifiers, since they combine the output of many simple classifiers.

2.4 Classification Methods

The classification step of the negative selection algorithm uses the whole population of antibodies. Each member of the population of classifiers, is compared against the example to be classified. If one or more antibodies recognize the example, then the example is classified as “non-self,” if no antibody recognizes the example, then it is classified as “self.” The negative selection algorithm is able to work even when only positive training examples are available. It is naturally a binary classification algorithm, however there has been research done to expand its capabilities to multi-class classification.

2.5 Multiclass Classification

The first effort to do multi-class classification with AIS was by Goodman, Boggess, and Watkins in [23]. The Artificial Immune Recognition System (AIRS) algorithm works with the principle of clonal selection and trains a population of data points called artificial recognition balls (ARBs), which are then used to perform classification using the k Nearest Neighbors strategy. In [23], the AIRS algorithm is tested against Kohonen's Learning Vector Quantization algorithm. AIRS proves to be easy to tune to problems and is not sensitive to input parameters. The work is further expanded in [24], with further tests and refinements.

White and Garrett used the clonal selection algorithm to train a multi-class classifier, calling their algorithm CLONCLAS [25]. They tested their algorithm to recognize binary character patterns, but it takes a long time to train. A similar algorithm is presented by Brownlee in [26], named CSCA.

A multi-class negative selection classification algorithm is proposed and tested by Markowska-Kaczmar and Kordas in [27] and expanded in [28]. The algorithm trains a population of antibodies by training several sub-populations of antibodies, each of which recognizes one class in the data set. Each subpopulation essentially maps the negative space of a class present in the data set. The classification is performed by comparing a test example to each antibody in the population. The class assigned to a pattern is the one whose antibodies match the testing point the least number of times.

2.6 Positive Selection

Positive selection is also present in the NIS, although it is not as widely studied in the field of AIS as negative selection. It is modeled on the major histocompatibility complex (MHC) molecule receptor filter present in immature T-cells, which allows the body to recognize (positively select) cells that have the MHC receptor. The mechanism helps the body to keep cells that have the correct MHC receptor, while also performing negative selection on cells that classify self-tissues incorrectly (negative selection). The first appearance of positive selection is in [29].

In [29], the authors build a positive selection algorithm for change detection and test it against a negative selection algorithm. The algorithm is simply the reverse of negative selection, where a detector is generated randomly and added to the population if it matches a "self" sample in the training set. The classifier shows improved change detection in some cases, and the authors suggest that a hybrid approach, combining negative and positive selection, may be useful.

In [30], the author shows a multi-layered positive selection algorithm to detect anomalous behavior. The algorithm performs both positive and negative selection in several phases to produce a population of antibodies. The framework is intended to reduce the rate of false positives in an anomaly detection algorithm, but it is untested and no experiments or results are given.

The authors of [31], have proposed a positive selection algorithm based on the work in [29] by making use of the clonal selection principle and using it to iteratively train a population of antibodies. The authors further expanded on their algorithm by creating a multi-class classifier by combining many classifiers into a one-against-all classifier. The algorithm is used to detect malware using data from API call traces and kernel mode callbacks. The algorithm outperformed all other algorithms tested against it in this task. The same authors further tested their algorithm in [32], and found that it outperformed all other classifiers on the UCI Diabetes dataset, with 79.9% accuracy, also achieving 96.7% accuracy on the UCI Iris data set.

Since the positive selection AIS classifier tested in [32] is the work most similar to our own, we will compare and contrast our classifier against it. Both classifiers can perform multi-class classification by attaching a category label to every antibody, and both classifiers use hyper sphere antibodies. Like our own classifier, their classifier uses k-NN as a fallback if the population of antibodies fails to classify a test data point. However, our classifier allows hyper spheres to overlap each other, while theirs does not. This greatly simplifies the training process and allows for faster execution. Furthermore, our classifier does not use clonal selection for training. Instead, a random selection of training data is used to train the population of antibodies, as will be explained in the following sections. This also makes the training much faster.

In short, our classifier is lightweight, and does not require many parameters. Additionally, we know from our previous research that it does not require a lot of training data [18-19]. Lastly, it leverages several techniques to make training much faster. Figure 1 shows a graphical representation of previous research, including our own. The current research position is highlighted in the figure.

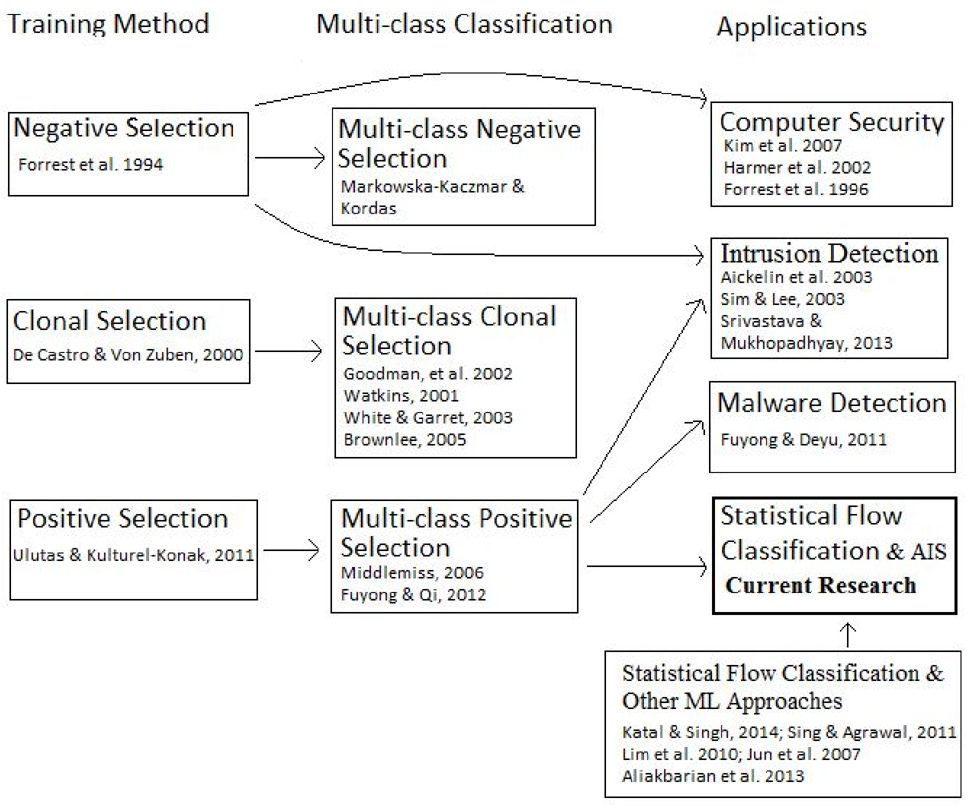


Fig 1. Recent Related Literature and Research Position

2.7 AIS Algorithms and Optimization

The idea of creating a more efficient Artificial Immune System algorithm is first presented by Elberfeld and Textor in [33], in which they show how r-contiguous and r-chunk based set of detectors could be trained and used in recognition in a more efficient manner. Through the use of a specialized technique, the authors are able to compress the set of detectors. The compression scheme used is simply using a single pattern to describe a set of several detectors. By using the patterns instead of the detectors themselves in the matching steps of the training and classification algorithms, the time complexity is lowered. The worst case time complexity of the original algorithm is exponential, but it becomes polynomial with the new technique. The results of this paper show that storing all of the detectors is unnecessary, and by compressing the set a substantial speedup is achieved.

In [34], Wang, Yibo, and Dong propose a faster training and classification methods for Negative Selection algorithms. The technique they show uses "neighborhoods" in the feature space to represent both detectors and samples to be classified. They also introduce a method to improve the matching operation between detectors and samples that improves the performance, especially in high dimensions.

In [35], the authors show a similar approach to the one used in [34]. The algorithm is called GF-RNSA, a Negative Selection algorithm which uses the concept of grid cells to speed up execution. The feature space is separated into grid cells, and detectors are generated separately for each cell, only comparing the candidate detectors with the self samples that are in the same grid cell, instead of the whole set. This approach also manages to eliminate the exponential time complexity of the NSA. This publication also showed experimental results.

Lastly, a novel detector generation method is proposed in [36]. The method was developed by Ji and Dasgupta and is called V-detector. The strategy involves the statistical analysis of the data in order to improve the amount of non-self space that is covered while also minimizing the number of detectors needed to do so. The detector generation process also takes into account the boundary of the classes in the data set to improve the quality of the set of detectors. The detectors are allowed to be of variable size, as well. These techniques allow the algorithm to be very efficient. The scheme is also applicable across many different detector types.

3. Methodology

3.1 Problem Statement

The supervised learning classification problem is defined in this section. Given a set of training data consisting of pairs of inputs and outputs :

$$S = \{(X_1, y_1), (X_2, y_2),(X_3, y_3), \dots , (X_s, y_s)\} \tag{1}$$

where X_i is a vector of parameter values in d-dimensional space:

$$X_i \in R^d \tag{2}$$

A classifier implements a function that maps every vector Xi in vector space to a class in set Y, where:

$$y_i \in Y \tag{3}$$

and for binary problems:

$$Y = \{1, -1\} \tag{4}$$

The goal of a learning algorithm is to build a function h that approximates a function g, where g is the true function that classifies an input vector in X to a category in Y, described like this:

$$g(x): X \rightarrow Y \tag{5}$$

To test a model of the data we use a set of test data T, which is of the same structure as S:

$$T = \{(X_1, y_1), (X_2, y_2),(X_3, y_3), \dots , (X_t,y_t)\} \tag{6}$$

where y_i is the true classification of each vector X_i. The quality of the predictions of a binary classifier can be measured in this way:

$$\sum_{i=1}^{|T|}\left(\frac{I(h(X_i)=y_i)}{|T|}\right) \tag{7}$$

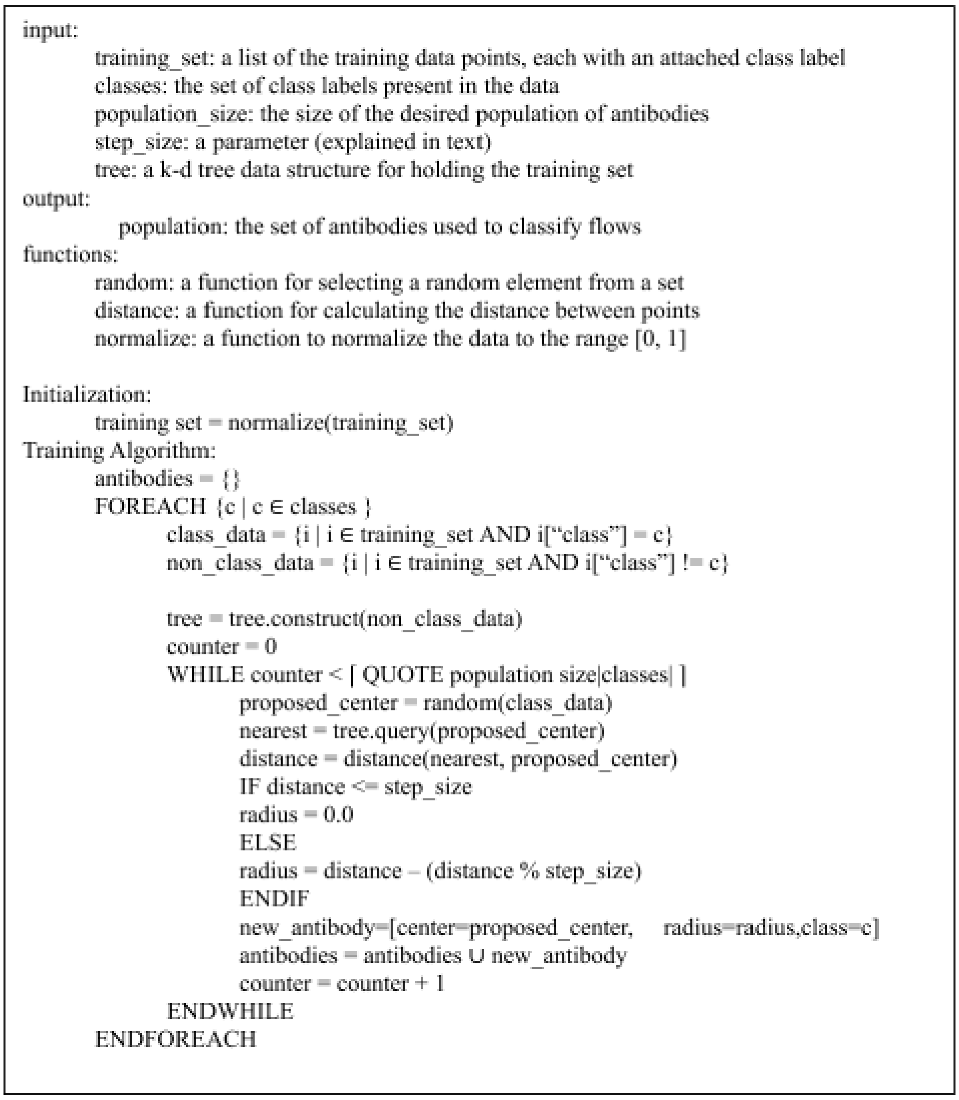
where I is the indicator function, which is equal to 1 when the statement is true and 0 when it isn't. The goal of a training algorithm is to produce an accurate model, so that, when used to make predictions, it will maximize the above function.

The previous definition introduced binary classification, in which the number of possible categories an entity can belong to is limited to two. Multi-class classification is a natural extension of the binary classification problem. Very simply, the definition above can be extended by allowing the set of classes Y to hold more than two classes.

3.2 Algorithm Description

The original algorithm described and tested in [18-19] will be improved in several aspects in this paper. The original algorithm created a set of hyper spheres by randomly sampling the training set with replacement, each hyper sphere centered on one training point, this process allows more than one hyper-sphere to be centered on the same point. Each hyper sphere is also labeled with the class label of the element from the training set from which it is created. Each hyper sphere is called an "antibody" in the parlance of AIS, and used as a simple classifier.

The original algorithm works within a normalized feature space, meaning that all data is normalized to be in [0,1]. This simplifies the code. This is possible since every feature in the data set used for testing is real-valued.



```
input:
    training_set: a list of the training data points, each with an attached class label
    classes: the set of class labels present in the data
    population_size: the size of the desired population of antibodies
    step_size: a parameter (explained in text)
    tree: a k-d tree data structure for holding the training set
output:
      population: the set of antibodies used to classify flows
functions:
    random: a function for selecting a random element from a set
    distance: a function for calculating the distance between points
    normalize: a function to normalize the data to the range [0, 1]

Initialization:
    training set = normalize(training_set)
Training Algorithm:
    antibodies = {}
    FOREACH {c | c ∈ classes }
        class_data = {i | i ∈ training_set AND i["class"] = c}
        non_class_data = {i | i ∈ training_set AND i["class"] != c}

        tree = tree.construct(non_class_data)
        counter = 0
        WHILE counter < ⌈ QUOTE population size|classes| ⌉
            proposed_center = random(class_data)
            nearest = tree.query(proposed_center)
            distance = distance(nearest, proposed_center)
            IF distance <= step_size
            radius = 0.0
            ELSE
            radius = distance – (distance % step_size)
            ENDIF
            new_antibody=[center=proposed_center,     radius=radius,class=c]
            antibodies = antibodies ∪ new_antibody
            counter = counter + 1
        ENDWHILE
    ENDFOREACH
```

Fig 2. Pseudo Code for the Training Algorithm

When newly created, the hyper spheres have a radius of 0, and are expanded iteratively. The training algorithm expands each antibody until an element of the training set is misclassified, then it contracts the size of the antibody slightly. The size by which the algorithms expands and contracts each antibody in every iteration is called the "step size," and is given as a parameter. Each class in the multi-class training set was given the same number of antibodies in the population, no matter how unbalanced the training data is. The original training algorithm's pseudo code is listed in Figure 2.

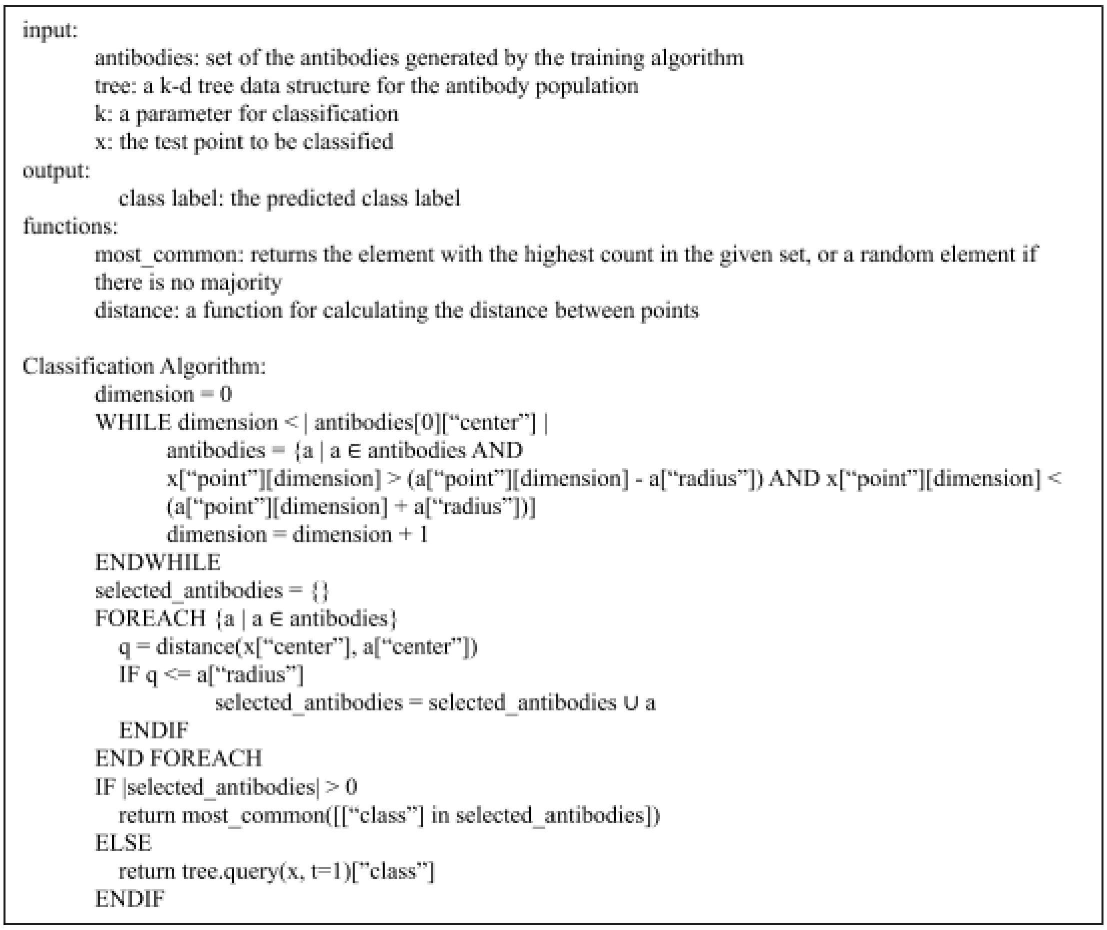
The classification of a pattern happens very simply. The original algorithm iterates through all of the antibodies in the population, checking whether the test pattern falls within the hyper sphere defined by any one of the antibodies. The pattern is classified as belonging to the class of the antibody in which it falls. If the point does not fall within any antibody, then the *k*-Nearest Neighbors algorithm is performed with the center of the nearest antibody, and k is given as a parameter.

The training algorithm takes two parameters: the number of antibodies required and the step size which is the amount by which an antibody expands and contracts at each iteration of the training process. The step size also determines the factor by which the antibody falls back when it misclassifies a member of the training set. The classification algorithm only takes one parameter: *k*, which is the number of antibodies that are used to classify a test point when the point does not fall within any antibodies. The original classification algorithm's pseudo code is listed in Figure 3.

We performed tests on the original algorithm and found that the performance of the training algorithm, as expected, is linear on the size of the training data set and the size of the antibody population that is required. The classification algorithm's execution is linear on the size of the antibody population. We will be trying several techniques in this paper to improve on this performance as well as on the classification performance of the algorithm.

The original algorithm approximates the class boundaries using two methods: sampling the training set to find good centers for the hyper-spheres, and calculating a radius for each hyper-sphere that will not misclassify any points in the training set. These two techniques for building the classifier are not similar to anything found in the field of AIS. This is different from the Negative Selection and Positive Selection algorithms.

Based on previous work [18-19], we have found the original algorithm to be insensitive to the use of kernels, and is able to deal with non-linearly separable data sets easily. Since the original algorithm does not require kernel functions, parameters for them do not need to be determined, speeding up the training process. However, in the tests done for the previous publications, it is obvious that the SVM algorithm is much faster, both in the training and classification portions.


```
input:
        antibodies: set of the antibodies generated by the training algorithm
        tree: a k-d tree data structure for the antibody population
        k: a parameter for classification
        x: the test point to be classified
output:
          class label: the predicted class label
functions:
        most_common: returns the element with the highest count in the given set, or a random element if
        there is no majority
        distance: a function for calculating the distance between points

Classification Algorithm:
        dimension = 0
        WHILE dimension < | antibodies[0]["center"] |
                antibodies = {a | a ∈ antibodies AND
                x["point"][dimension] > (a["point"][dimension] - a["radius"]) AND x["point"][dimension] <
                (a["point"][dimension] + a["radius"])}
                dimension = dimension + 1
        ENDWHILE
        selected_antibodies = {}
        FOREACH {a | a ∈ antibodies}
          q = distance(x["center"], a["center"])
          IF q <= a["radius"]
                    selected_antibodies = selected_antibodies ∪ a
          ENDIF
        END FOREACH
        IF |selected_antibodies| > 0
          return most_common([["class"] in selected_antibodies])
        ELSE
          return tree.query(x, t=1)["class"]
        ENDIF
```

Fig 3. Pseudo Code for Classification Algorithm.

3.3 Changes to the Training Algorithm

Several changes to the original algorithm have been proposed by us. A description of each one follows. The purpose of the changes is to optimize the performance of the algorithm, both to make it faster and more accurate.

The training algorithm has been changed with the addition of the *k-d* tree data structure. *K-d* trees are data structures first seen in [37], and are used for partitioning multi-dimensional spaces for searching. They are used in this context to search in a set of points for the nearest point to a query point. *K-d* trees work by arranging a set of points in a binary tree data structure, allowing them to be searched with much less processing time. Instead of having to search the entire set of points for the closest point to a query point, the *k-d* tree is traversed in logarithmic time.

Like the original training algorithm, the new training algorithm generates antibodies by randomly sampling the training data set, and then setting the radius so that the antibody does not misclassify any points in the training set. However, the old training algorithm used a very CPU-intensive loop to iteratively expand the radius of the current antibody being generated, until it misclassifies a training data point.

The new training algorithm iterates through each class present in the data set, creating a subpopulation of antibodies for each class. For every class, the training algorithm separates the training data set into two subsets: a self-class training set and a non-self class training set. It then generates a *k-d* tree data structure with the non-self class data set. To generate a single antibody for a given class, the algorithm selects a data point randomly from the self-class training set, and makes the coordinates the center of the

```
Definitions:
training_set: a list of the training data points, each with an attached class label
population: the set of antibodies used to classify flows
classes: the set of class labels present in the data
population_size: the size of the desired population of antibodies
tree: a k-d tree data structure for holding the training set
random: a function for selecting a random element from a set
distance: a function for calculating the distance between points
normalize: a function to normalize the data to the range [0, 1]
step_size: a parameter (explained in text)

1. Initialization:
      training set = normalize(training_set)
2. Training Algorithm:
antibodies = {}
FOREACH {c | c ∈ classes }
class_data = {i | i ∈ training_set AND i["class"] = c}
non_class_data = {i | i ∈ training_set AND i["class"] != c}

tree = tree.construct(non_class_data)
counter = 0
WHILE counter < ⌈ QUOTE population size|classes| ⌉
proposed_center = random(class_data)
nearest = tree.query(proposed_center)
distance = distance(nearest, proposed_center)

IF distance <= step_size
radius = 0.0
ELSE
radius = distance – (distance % step_size)
ENDIF
new_antibody=[center=proposed_center,    radius=radius,class=c]
      antibodies = antibodies ∪ new_antibody
counter = counter + 1
ENDWHILE
ENDFOR
```

Fig 4. Pseudo code for the training algorithm

new antibody, setting the class identifier of the new antibody to the class it is working with. The algorithm then queries the *k-d* tree created in the earlier step and finds the closest non-self training point to the center of the antibody, setting the radius so as not to misclassify this training point using the step size parameter. In this manner, the radius is set in one step, instead of iteratively, as before. The pseudo code for the new training algorithm is given in Figure 4. In total, there are |*C*| *k-d* trees created, where *C* is the set of classes present in the dataset. In the pseudo code, the list of classes found in the data set is given to the algorithm. The list of classes present in the data set is given to the training algorithm. In the pseudo code, i["class"] denotes the class that a member of the training set belongs to.

An illustration of the training algorithm working in two dimensions is found in Figure 5. The figure shows three classes graphed in two dimensions, with an antibody for class A being trained. The hyper-sphere is centered on a point belonging to class A, and the radius is set so that the hyper-sphere does not contain any points of other classes. In this case, a point in class B is the closest. The radius of the antibody is set by finding the closest point to the center that is not of the same class as the center point. This process is accomplished using *k-d* trees.

When dealing with running time of machine learning algorithms, the curse of dimensionality is often found to limit what is possible to compute, since adding even one dimension can have a noticeable effect on the performance of the algorithm. The training algorithm deals with this by using *k-d* trees, which are data structures that are able to search for points in a set in less time. Whereas naively finding the nearest point in a set to another point is done by calculating the distance function for every point in the set, *k-d* trees are able to do this in log p steps, where p is the number of points in the data set belonging to the non-self. The training algorithm described in this section uses *k-d* trees for this reason.

Lastly, the optimized and unoptimized training algorithms are functionally identical, creating identical sets of antibodies when all conditions are the same (accounting for the random selection of antibody centers).

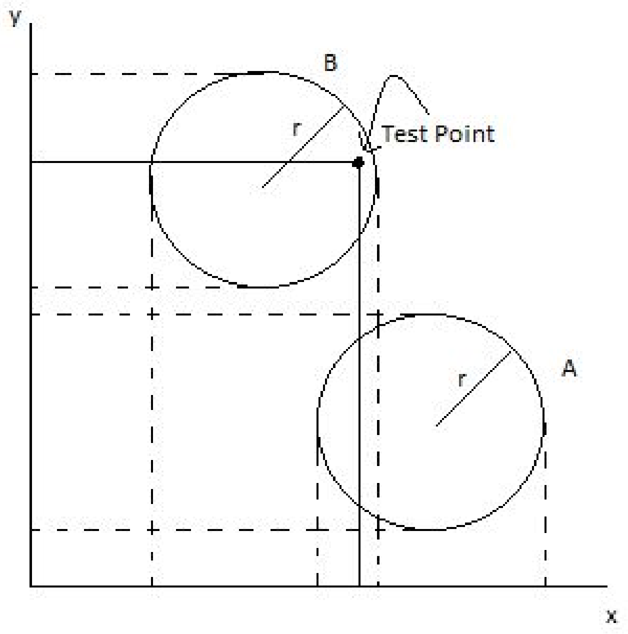


Fig 5. Illustration of the Training Algorithm

3.4 Changes to the Classification Algorithm

The classification algorithm has been changed extensively. The original classification algorithm was a simple loop which compared every antibody with the data point to be classified, and returned the class label of the first antibody which contained the test point. This loop was very computationally expensive, since it calculated the distance between the pattern to be classified and every antibody center.

The new algorithm cuts down on the execution time by "filtering" the set of antibodies so that only the antibodies that could contain the test data point remain in the set of antibodies. This process happens in two stages. The first stage iterates through the features of the dataset, selecting only the antibodies that contain the point to be classified in that feature, and removing the rest. After each dimension in processed, the set of antibodies is smaller, making the process go faster. This also negates the need to calculate a distance function, but only in this stage. We call the first stage "primary" filtering. The first stage of filtering does not deal with the fact the hyper spheres are “round.” The first stage of filtering only removes antibodies that are outside the hyper cube that contains the hyper sphere that is each individual antibody

Figure 6 shows a depiction of primary filtering in 2 dimensions. The test point falls within the range of both antibodies in the x dimension, so it would not be filtered out when that dimension is processed by the primary filtering. However, it does not fall within the range of the A antibody in the y dimension, meaning that that antibody would be filtered out when that dimension is processed.

A second stage of filtering is necessary since, as mentioned earlier, the first stage of filtering does not deal with the fact the hyper spheres are “round.” This is fixed by calculating the distance function on the remaining antibodies, and removing the antibodies that do not contain the point, just as in the original algorithm. We call this “secondary filtering.”

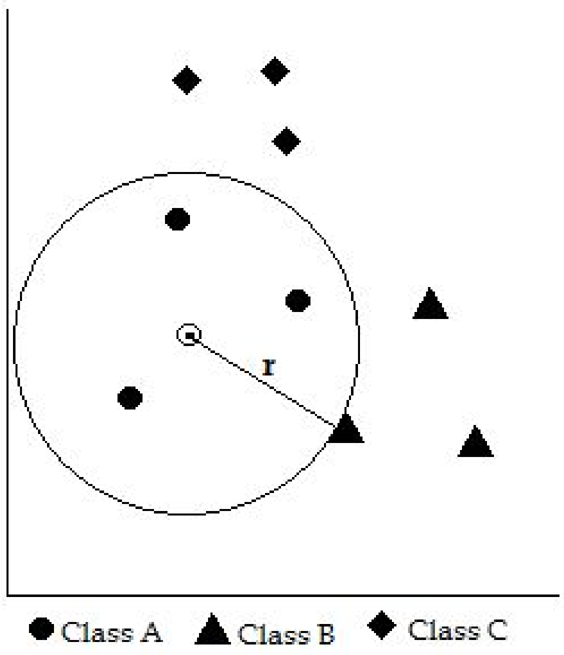


Fig 6. Explanation of primary filtering scheme

Lastly, the new classification algorithm still handles test data points that do not fall within any antibody by performing k-NN classification using the antibody population. A point is classified in k-NN classification by finding the *k* closest points to it. The class is then assigned by a majority vote of its neighbors. The point is assigned the class that is most common among its nearest *k* neighbors. The “NN” in the name of the algorithm stands for “Nearest Neighbor”. This is done in the new algorithm with a *k-d* tree.

Although, the *k-d* tree does cut down slightly in the classification time, it is not used in a large portion of the classifications, and its removal would not affect the classification time greatly. The pseudo code for the new classification algorithm is given in Figure 7. In the pseudo code, a[“center”] denotes the coordinates of the center point of the hyper-sphere that makes up the antibody and a[“center”][dimension] references one of the dimensions of that vector. Further, x[“point”] references the coordinates of the point to be classified. Lastly, a[“radius”] references the radius of the hyper-sphere.

We also changed the classification scheme to deal with a weakness that we failed to find in our previous work. The old classification algorithm did not deal with the fact that antibodies in the population are allowed to overlap. The problem with the previous approach arises because the first antibody found in the population that matched the test point was returned as the point's class, even though there is a chance that the first antibody found would misclassify the test point. This problem is dealt with by concept of “majority voting” in the current classification algorithm. The algorithm creates the set of the antibodies that contain the test point with filtering, counts the antibodies, and then returns the majority class to determine the predicted class of the test data point. If there is no majority found, the algorithm returns a random class of the ones found in the antibodies that remain. The classification algorithm takes one parameter: k, which determines the number of antibodies used when performing k-NN classification.

The simplest manner to determine whether a point lies within a hyper-sphere is to calculate the distance between the center of the hyper-sphere to the point and comparing the distance against the radius of the hyper-sphere. This calculation is subject to the curse of dimensionality because a calculation is performed on every dimension of both points. The classification algorithm minimizes the effect of the curse by minimizing the size of the set of hyper-spheres through primary filtering, as described above.

In our literature review we did find some efforts to optimize the execution of AIS algorithms, but we did not find any techniques used that are similar to the ones explored in this paper.

4. Theory and Analysis

This section deals with the analysis of the complexity of the optimized training and classification algorithms. We show the big O complexity of the optimized algorithm, as well as the expected classification performance of the algorithm.

4.1 Analysis of the Training Algorithm

The training algorithm is a one-shot algorithm that is dominated by the cost of creating and querying *k-d* trees. Like the original, unoptimized algorithm, the optimized algorithm is still a linear algorithm, although the constants are much smaller than the constants in the original algorithm.

For the creation of the *k-d* trees, the data set must be divided into subsets, this is linear on the size of the training set:

$$N \tag{8}$$

where N is the size of the training set.

The algorithm must first initialize one *k-d* tree for each class in the training set, therefore:

$$O(|Y| * en \log n) \tag{9}$$

where Y is the set of classes present in the training set, e is the number of dimensions in the dataset, and n is the total number of samples in the training set that are not in the current class.

Once the *k-d* tree is built for each class, the algorithm queries one *k-d* tree one time for each antibody created in order to determine the correct radius for each antibody. The complexity is now dominated by the number of antibodies created:

$$O(p \log n) \tag{10}$$

where p is the number of antibodies required, and n is the number of samples in the training set not in the class of the antibody being generated. A query from one of the *k-d* trees created in the previous step requires $O(\log n)$ time.

Lastly, since the classification falls back to *k*-NN classification if there is no antibody that contains the test point, we must create a *k-d* tree that contains all of the antibody centers. This takes:

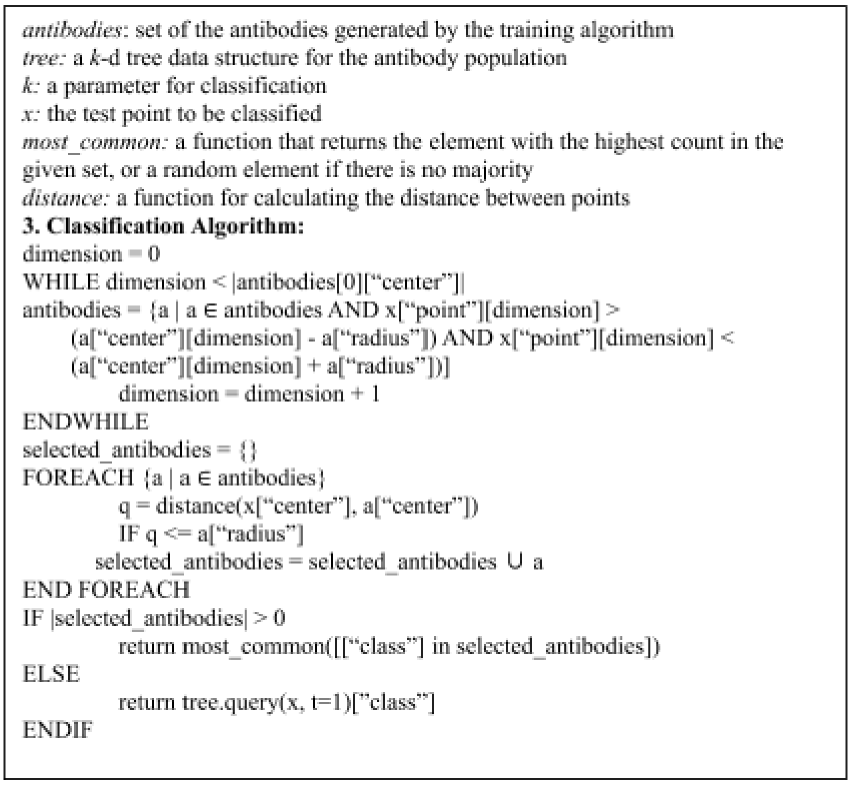
$$O(e\mathrm{p} \log p) \tag{11}$$

where e is the number of dimensions in the dataset, and p is the total number of antibodies created.

Putting these terms together, the training algorithm has a big O complexity of:

$$O(\mathrm{N} + [|\mathrm{Y}| * (en \log n)] + [p \log n] + [ep \log \mathrm{p}]) \tag{12}$$

which accounts for both the cost of creating the *k-d* trees and querying them, as well as creating the *k-d* tree for fallback classification.The new training algorithm works faster and more efficiently than the previous version of the algorithm. The previous training algorithm handled the comparison between a proposed antibody and all training samples naively, performing a comparison between each sample and the proposed antibody. This comparison required the calculation of the distance function, and was performed even when the sample was guaranteed to not fall into the proposed antibody.



```
antibodies: set of the antibodies generated by the training algorithm
tree: a k-d tree data structure for the antibody population
k: a parameter for classification
x: the test point to be classified
most_common: a function that returns the element with the highest count in the
given set, or a random element if there is no majority
distance: a function for calculating the distance between points
3. Classification Algorithm:
dimension = 0
WHILE dimension < |antibodies[0][“center”]|
antibodies = {a | a ∈ antibodies AND x[“point”][dimension] >
      (a[“center”][dimension] - a[“radius”]) AND x[“point”][dimension] <
      (a[“center”][dimension] + a[“radius”])}
            dimension = dimension + 1
ENDWHILE
selected_antibodies = {}
FOREACH {a | a ∈ antibodies}
            q = distance(x[“center”], a[“center”])
            IF q <= a[“radius”]
         selected_antibodies = selected_antibodies ∪ a
END FOREACH
IF |selected_antibodies| > 0
            return most_common([[“class”] in selected_antibodies])
ELSE
            return tree.query(x, t=1)[”class”]
ENDIF
```

Fig 7. Pseudo code for the classification algorithm.

4.2 Analysis of the Classification Algorithm

The classification algorithm is linear on the number of antibodies present in the population. We split up the analysis into two parts. The primary filtering process proceeds dimension by dimension, therefore:

$$O(\, d * |A_{primary}| \,) \tag{13}$$

where d is the number of dimensions of the antibody centers, and $A_{primary}$ is the set of antibodies present in the population before primary filtering begins. Furthermore, the size of the antibody population decreases with each dimension processed, meaning that the primary filtering takes less and less time as it executes. This is one of the reasons why it is much faster than the original algorithm, even though the filtering process is still linear, like the original algorithm.

The secondary filtering process is linear on the number of antibodies remaining in the population after the primary filtering is done:

$$O(\, e * |A_{secondary}| \,) \tag{14}$$

where e is the number of dimensions of the dataset, and $A_{secondary}$ is the set of antibodies present in the population before secondary filtering begins and after primary filtering is done.

Lastly, the assignment of the class to the test point is done in two ways. If the point is within one or more antibodies, then majority voting is performed, otherwise a query to a *k-d* tree is done. The time taken for final classification is described by the function h, parametrized by the set of antibodies A_{final}, and the size of the original set of antibodies *N*. Final classification takes:

$$h(A_{final}, M) = \begin{cases} |A_{final}| \; if \; \exists a \; \{a | a \in A_{final}, d(a[center], x) \le a[radius]\} \\ \log M \qquad\qquad\qquad\qquad otherwise \end{cases} \tag{15}$$

where A_{final} is the set of antibodies remaining in the set after secondary filtering is completed, and *M* is the size of the original antibody population. The distance function used in the classification is shown as d(), the sample to be classified is shown as x. The majority voting step of the algorithm is linear on the number of antibodies left in the population after the filtering steps are finished.

Combining the three costs, we get the big O complexity of the optimized classification algorithm:

$$O([\, d * |A_{primary}| \,] + e * |A_{secondary}| + h(A_{final}, M)) \tag{16}$$

As we expected, the algorithm is still linear on the number of antibodies in the population, and any speedup present is due to the efficient implementation of the filtering, which is usually faster than the previous implementation of the classification algorithm. However, the new classification algorithm will be faster than the previous classification algorithm in almost all cases.

4.3 Analysis of the Memory Requirements of the Algorithm

The memory requirements of the algorithm can be broken down into four categories: the memory required to store the data set, the memory required to store the antibody population, the memory required by the training algorithm to perform its calculations, and the memory required by the classification algorithm to make a prediction. In this section, the variable i is the number of bytes required to store an integer, k is the number of dimensions present in the data set, and f is the number of bytes required to store a floating point number.

The memory required to store the data set used during the training algorithm can be calculated as:

$$|S| * [\, i + (e * f) \,] + (|S| * 2 * p) \tag{17}$$

where S is the data set. This amount of memory is required when the class label is encoded as an integer, which simplifies the calculation. Lastly, the amount of memory used for pointers is calculated, where each node in the tree contains two pointers and the amount of memory required to store a pointer is p.

The antibody population memory requirement is very simple to calculate and depends on the size of the population and the number of dimensions of the data set:

$$(|A| * [\, i + (e * f) + f \,]) \tag{18}$$

where A is the set of antibodies. A floating point number is used to store the radius of the hyper-sphere. Again, this amount of memory is required only when the class label is encoded as an integer.

The initialization step of the algorithm involves a simple normalization step, which can be done in-place, requiring no more memory than it takes to store the highest and lowest values of each variable in the data set:

$$e * 2f \tag{19}$$

where e is the number of dimensions of the data set.

The training algorithm, which generates the population of antibodies requires no memory to run, since the memory it uses is already accounted for in the above analysis of the antibody population.

The classification algorithm requires memory to perform the filtering steps. However, by encoding the structure of the *k-d* tree used for fallback classification on the array storing the population of antibodies, it is possible to avoid storing the antibody population twice. The amount of memory required to perform primary filtering never exceeds:

$$|A_{primary}| * [\, i + (e * f) + f \,] \tag{20}$$

where $A_{primary}$ is the set of antibodies present in the population before primary filtering starts. This is due to the fact that the antibody population is copied every time primary filtering is performed on one dimension. However, the filtering seeks to remove portions of the population at each step, so the amount of memory required is always guaranteed to be equal or smaller than the original antibody population.

Secondary filtering can be done in place, and does not require any memory, the copied antibody population is guaranteed to never exceed:

$$|A_{secondary}| * [\, i + (e * f) + f \,] \tag{21}$$

where $A_{secondary}$ is the set of antibodies present in the population before secondary filtering starts, but after primary filtering is done. The majority voting step only requires a set of counter variables, which keep track of the number of times each class appears in the population that remains after primary and secondary filtering is done:

$$|Y| * i \tag{22}$$

where Y is the set of classes present in the data set.

4.4 The VC-dimension of Hyper-sphere Classifiers

Vapnik-Chervonenkis theory was developed by Vladimir Vapnik and Alexey Chervonenkis between 1960 and 1990. The theory aims to explain learning from a rigorous mathematical and statistical point of view. Through the use of Vapnik-Chervonenkis theory it is possible to quantify the descriptive power of a model. This is done by measuring the VC-dimensionality of the functions that define the model with the concept of a shattering set. Through the use of VC-dimensionality, it is also possible to provide a bound on the generalization error of an algorithm. It is applied to the current problem to explain the ability of the algorithm developed in this work to describe class boundaries, as well as to make it easier to derive bounds. To lay down a foundation that more complicated proofs can be built on, a proof for the VC-dimension of a single hyper-sphere classifier is given here. A more thorough description of VC-dimension is found in [38].

A hyper-sphere of radius r is defined as:

$$S^f = \{ x \in R^f,\ c \in R^f \mid q(c,\ x) = r \} \tag{23}$$

where r is the radius of the hyper-sphere, f is the number of dimensions that the hyper-sphere is defined in, c is the center of the hyper-sphere and q is the distance function being used. Given that q is a distance metric, it can only be a positive real number. The distance function need not be the Euclidian distance function.

To accomplish classification using a hyper-sphere, a function must be defined that maps members of X to members of Y, as defined in section IV. Therefore, a formalization of a hyper-sphere classifier is given by defining a function that can be used as a classifier:

$$f(x) = \begin{cases} 1 & if\ q(c,x) \le r \\ -1 & otherwise \end{cases} \tag{24}$$

A single hyper-sphere classifier is a simple classifier that defines every point within its radius to be of class 1, and everything outside of it to be of class -1. A class of functions denoted as OHSC, where:

$oh_i \in$ OHSC (for "one hyper-sphere classifiers") (25)

is defined by two parameters:

$$oh_i = [c_i, r_i] \tag{26}$$

The center of the hyper-sphere is the vector c, and the radius r is a scalar. Both defined to be:

$$c = R^d \tag{27}$$

$$r = R \tag{28}$$

and the center vector c_i has d dimensions.

For the first proof, it is useful to remind ourselves of Radon's theorem. The theorem states that any set of $d + 2$ points in R, d can be divided into two disjoint sets whose convex hulls intersect. In other words, there always exists a way to partition a set of points so that the convex hulls of the subsets have at least one point in common. The convex hull of a set of points can be visualized as the set of points that correspond to a string stretched around the points (in two dimensions).

The proof for Theorem 1 shows that sets of size d+2 cannot be shattered by hyper-spheres by proving that the VC dimensionality of hyper-spheres is the same as the VC dimensionality of hyper-planes in the same number of dimensions. The proof can be better understood by visualizing a hyper-sphere with a radius equal to infinity. Such a hyper-sphere would behave in the same way as a hyper-plane. Theorem 1 is important because every other proof within this section depends on the VC-dimensionality of a single hyper-sphere. The second part of the proof relies on the fact that half-spaces are proven to have a VC-dimensionality of d+1. The proof for Theorem 1 first appeared in the unpublished manuscript [39]. To the best of our knowledge, there is no other publication describing the VC-dimensionality of hyper-spheres.

Theorem 1: VC dimension of hyper-spheres in R^d is equal to d+1, where d is the number of dimensions of the hyper-sphere. Stated in another way, VC_{dim}(OHSC) = d+1.

Theorem 1 establishes the VC-dimensionality of the class of functions OHSC. The proof is done in two parts. First, it is proven that a set of points of size d+1 can be shattered by the class OHSC, then it is proven that no set of points of size d+2 can be shattered by the class OHSC. This is enough to prove that the VC-dimensionality of the class of functions is equal to *d*+1, according to the definition of VC-dimensionality. Lemma 1 and 2 contain the two parts of the proof.

Lemma 1: *A set of d+1 points consisting of the unit vectors and the origin can be shattered by hyper-spheres of OHSC.* Suppose *A* is a subset of the *d*+1 points. The center a_0 of a hyper-sphere will be the sum of the vectors in *A*. For every unit vector in set *A*, its distance to the center a_0 will be $\sqrt{|A|-1}$ and for every unit vector outside *A*, its distance to the center a_0 will be $\sqrt{|A|+1}$. The distance of the origin to the center is $\sqrt{|A|}$. Thus it is easy to see that we can choose the radius so that precisely the points in *A* are in the hyper-sphere.

Lemma 2: *No set of points of size d+2 can be shattered by class OHSC.* Proceeding by contradiction. Suppose that there exists a set S with *d* + 2 points in it that can be shattered by a hyper-sphere of *d* dimensions. Applying Radon's theorem, for any partition of set S into subsets A_1 and A_2, there exist hyper-spheres B_1 and B_2 such that $S \cap B_1 = A_1$ and $S \cap B_2 = A_2$. B_1 and B_2 may intersect, but assume w.l.o.g. there is no point belonging to S in their intersection. Because no points in S are allowed to exist in the intersection of B_1 and B_2, it is easy to see then that there is a hyper plane with all of A_1 on one side and all of A_2 on the other. This implies that half-spaces are able to shatter the set S which is of size *d*+2, which is a contradiction since half-spaces are proven to have a VC-dimensionality of d+1. Therefore, no set of *d*+2 points can be shattered by a hyper-sphere.

Corollary 1: *All classifiers that implement a function of the class OHSC will have a VC-dimensionality of d+1.* This follows from the fact that the VC-dimensionality of a single hyper-sphere is equal to *d*+1.

Corollary 1 shows that all one hyper-sphere classifiers will have a VC-dimensionality of *d*+1, no matter how they are parametrized. These results will be used for all of the remaining proofs in this section.

A definition is now given for a classifier made up of multiple hyper-spheres. As before, each hyper-sphere is parametrized by:

$$h_i = [c_i, r_i] \tag{29}$$

The set of hyper-spheres that makes up the classifier is then defined to be:

$$HS = \{ h_1, h_2, h_3, \dots , h_M\} \tag{30}$$

Where the size of the set is:

$$M = |HS| \tag{31}$$

The classification function then becomes:

$$f(x) = \begin{cases} 1 & if\ \exists h \{ h \mid h \in HS,\ d(h[c], x) \le h[r] \} \\ -1 & otherwise \end{cases} \tag{32}$$

Using the definition above, a class of functions is defined:

$$mh_i \in \text{MHSC} \quad \text{(for “multi hyper-sphere classifiers”)} \tag{33}$$

where each function mh_i within MHSC is defined by many pairs of parameters, each pair representing one hyper-sphere:

$$mh_i = \{ [c_1, r_1]_1, [c_2, r_2]_2, [c_3, r_3]_3, \dots , [c_M, r_M]_M \} \tag{34}$$

where each center ci is a vector, and each radius r_i is a scalar, defined to be in:

$$c_i = R^d \tag{35}$$

$$r_i = R \tag{36}$$

and the center vector c_i has *d* dimensions.

The lower bound for the VC-dimensionality of the class of functions MHSC is now given through two corollaries. These corollaries are possible because it can easily be seen that the class of functions MHSC contains all possible one hyper-sphere classifiers, which are described in the class of functions OHSC.

Corollary 2: *The VC-dimensionality of the class of functions MHSC when M=1 is d+1, where d is the number of dimensions.* Stated in another way, $VC_{dim}(\text{MHSC}) = d+1$ when $M=1$.

This corollary is easily seen from Theorem 1, since when M=1 the function in MHSC would be equivalent to a function in OHSC.

Corollary 3: *The lowest possible value of the VC-dimensionality of the class of functions MHSC when M≥2 is d+1, where d is the number of dimensions.* Stated in another way, VC_{dim}(MHSC)≥d+1 when M ≥ 2.

Corollary 3 can easily be seen when visualizing the class of functions MHSC. The lowest possible VC dimensionality of the class is achieved when it equals the VC dimensionality of a single hyper-sphere (a classifier that is a member of OHSC). This happens when all hyper-spheres in the set HS have the same radius and the same center. The hyper-spheres behave as one and the classifier behaves as a member of OHSC even though it belongs to MHSC. It can also be seen that the VC dimensionality of the classifier can only increase if the radii and centers of the hyper-spheres do not match, since a single hyper-sphere is the simplest class boundary representable by MHSC.

Corollary 4: *The upper bound of the VC-dimensionality of the class of function MHSC is M(d+1), where d is the number of dimensions and M is the number of hyper-spheres in the set.* Stated in another way, VC_{dim}(MHSC) ≤ M(d+1) if M ∈ {1,2,3, …, N}. This applies when M is bounded by an integer N.

As an example, suppose that there exists a set of points of size (M(d+1))+1 that can be shattered by a function in the class MHSC. This is not possible, since the maximum number of points that an individual hyper-sphere can shatter is d+1, meaning that the maximum number of points a set of hyper-spheres can shatter is M(d+1).

The maximum VC dimensionality of the class MHSC is achieved when each hyper-sphere in the set HS shatters d+1 points in the set. In aggregation, then, the set of hyper-spheres would shatter M(d+1) points. Since it is enough to show that there is an arrangement of points in a set of a certain size to achieve a certain VC dimensionality, it is easy to imagine a set of points on which it is possible to place M hyper-spheres, each hyper-sphere shattering a subset of points of size d+1, and achieve a VC dimensionality of M(d+1). Furthermore it is easy to see that it would be possible to place an additional point in the set so that the set of hyper-spheres would not be able to shatter the set, making a new set of size (M(d+1))+1. Therefore, the VC-dimensionality of the class of functions MHSC can never be more than M(d+1).

Theorem 2: *The VC dimensionality of MHSC is infinite if the number of hyper-spheres in the set is unbounded.* More formally: VC_{dim}(MHSC) = ∞ if M ∈ {1,2,3, …}. This theorem applies when M is unbounded.

Proof: Proceeding by contradiction, assume that there does not exist a function in class MHSC that can shatter a set of points D. But a function can be easily built by centering a hyper-sphere on every element of D_1, setting the radius to be equal to 0. Then proceed to set the radius of every hyper-sphere in the classifier so that each hyper-sphere does not contain any element in D_{-1}. This is a contradiction, since such a function would be able to shatter the set D and is in MHSC. Therefore, the VC dimension of the class of functions MHSC is infinite because it is able to shatter a set of points of any size.

Corollary 5: *The VC-dimensionality of all multi hyper-sphere classifiers is equal to the VC-dimensionality of the class of functions MHSC.* This also follows easily from the observation that all multi-hypersphere classifiers implement a function in the class MHSC. It can be seen that the maximum descriptive power of a classifier that uses a function from the class MHSC grows along with the number of hyper-sphere.

Theorem 2 is important because it shows the ability of the classifier to distinguish non-linearly separable data sets, given enough hyper-spheres. This also means that the training set error will be 0%, if enough hyper-spheres are used in the classifier. However, the only way to guarantee that this will hold true is if the number of hyper-spheres used in the classifier is equal to the number of elements in the training set with class label equal to 1. In contrast, SVMs cannot separate non-linearly separable data sets without giving a training error of more than 0%.

4.5 Bound for the Generalization Error of the Classifier

Because the set of functions in MHSC has a VC-dimensionality equal to infinity, the bound on the generalization error of the classifier is based on the concept of the margin of the ensemble classifier. The case of function classes with infinite VC-dimensionality is specifically excluded from the better known work on generalization error bound by Vapnik [40]. It is possible, however, to use the work of Breiman and Shapire [41] since they define the generalization error bound based only on the VC-dimensionality of the base classifiers used in an ensemble classifier. Specifically, it can be shown that their bound applies directly to the classifier described in this paper, because it is a voting ensemble classifier.

An analysis is given here for the expected classification performance of our algorithm. To accomplish this, the margin of an ensemble classifier and margin distributions are described and linked to the current work. The key insight in this section is the fact that the positive selection AIS algorithm that is being examined is, in fact, an ensemble classifier, using many hyper-spheres to classify a test example. The hyper-spheres are base classifiers.

The way that the current classifier being analyzed is built has some similarity to bagging, which was first described in [42] by Breiman. Bagging is used to randomly create many data sets out of the original data set, and train one "base" classifier with each created data set. The outputs of the base classifiers are then combined to make a prediction. Bagging is used to diminish the variability of the data set and can often give good results. Subagging is a variation of bagging in which the randomly created datasets are smaller than the original data set.

The work by Breiman in [43] is also useful to us, since it shows an analysis of the generalization error bound for random forests, which are a type of bagging classifier. The research links the generalization error of the ensemble classifier to the strength of individual trees and the correlation between them, two concepts that are defined thoroughly in the publication. This analysis is valuable because of the similarity of both AIS and random forest approaches, both being ensemble classifiers. There has also been a lot of research into the generalization error bound for the Adaboost algorithm, which uses the concept of boosting, itself is first described in [44]. The classification error bound of AdaBoost is first described in [41], where the concept of the margin is first introduced as an explanation of the success of ensemble classifiers.

The problem is couched in the following definitions. The definitions are taken from Cai, Chang, and Peng's work in [45].

Let *X* be the feature space, and let *Y* be the set of class labels. The data set *D* is made up of (*x*, *y*) pairs, with each *x* and *y* being members of *X* and *Y* respectively. Therefore, it can be seen that the set *D* is a subset of *X* x *Y*, and follows an unknown underlying distribution.

A classifier is the product of a learning process, and is implemented as a mapping function from $X \rightarrow Y$, which seeks to minimize the generalization error between the predictions of the classifier and the true underlying distribution, but accomplishes this through the minimization of the error on the training set. The classifier is written as:

$$h(x;\ \theta) :\ X \rightarrow Y \tag{37}$$

where *x* stands for the input vector which is a member of *X*, θ is a vector of parameters, and the output is a member of *Y*. The classifiers can be thought of as existing within a classifier space which is defined to be Θ, which contains all possible classifiers, with $\theta \in \Theta$.

The voting margin is a concept related to majority voting ensemble classifiers, such as the one described in this paper. The voting margin is defined as follows:

$$mg(x,y;\ \theta_1,\ \theta_2,\ \dots\ \theta_k) = 1/k(\sum_{i=1}^{k} I(h(x;\theta_i) = y) - \sum_{i=1}^{k} I(h(x,\ \theta i) = j)\ ) \tag{38}$$

where there are k base classifiers in the ensemble classifier, each defined by a set of parameters θ, y is the correct class label of x, and I is the indicator function. The margin function mg has a range of [-1, 1]. The sign of the output of mg signifies the accuracy and confidence of the classification. If the sign is negative the classification is incorrect, if it is positive it is correct. The absolute value of the margin function indicates the confidence that the classifier has about the prediction.

According to Shepiro et. al. [46], with high probability, the bound on the generalization error of a voting classifier is:

$$[mg(x,y;\ \theta_1,\ \theta_2,\ \dots\ \theta_k)]\ + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right) \tag{39}$$

where R_v refers to the risk of ensemble voting classifiers, which is the same as the generalization error of the classifiers. Here, $\hat{P}r$ [mg(x,y; θ_1, θ_2, … θ_k)] is the probability distribution of the margin function, d is the VC- dimensionality of the base classifiers, and m is the size of the data set. This bound holds for any $\theta > 0$. The proof for the above bound is found in [41].

By the margin explanation of ensemble methods, the best classifiers have a large margin. In the previous version of the algorithm, the margin was related to the value of the step_size parameter, which was the same for all base classifiers. In the new classifier, the margin is determined for each individual base classifiers, and is also not given as a parameter.

Since the algorithm used in this work also uses the k-Nearest Neighbors classification algorithm, the generalization error bound is shown here for it as well. The bound for the generalization error of the k-Nearest Neighbor was first given in [47]. A stronger bound for the k-NN algorithm is found in [48] and is reproduced here:

$$R_k \le R^*(1 + \frac{\gamma}{\sqrt{k}}(1 + O(k^{-1/6}))) \tag{40}$$

where R_k is the risk of the k-NN classifier, which is the same as the generalization error of the classifier. Here, γ is a constant equal to 0.33994241 and the O notation refers to the limit as $k \rightarrow \infty$. R* is the risk obtained by an optimal decision strategy, known as the Bayes error.

Since the algorithm proposed in this research is also a voting classifier, the bound given in the previous section also applies to it. However, not every base classifier in the ensemble classifier is allowed to cast a vote. As seen in previous sections, only hyper-spheres that contain the example are used to make a prediction. Because of these differences, the margin function is similar, but not the same as in other ensemble classifiers. This work is an extension of [46].

The margin of the classifier is calculated for multi-class classification, but is otherwise the same as in previous research. Each hyper-sphere that contains the test point votes for its own class. Therefore, the margin function becomes:

$$mg(x,y;\theta_1,\ \theta_2,\ldots\theta_k) = 1/k\ (\sum_{i=1}^{k} I\,(hs\,(x;\theta_i) = y) - \sum_{i=1}^{k} I\,(hs\,(x,\ \theta_i) = j)\ ) \tag{41}$$

where hs is a hyper-sphere in the set HS which defines the classification function, and each hyper-sphere hs_i is parametrized by a set of parameters θ_i. As previously mentioned, the number of hyper-spheres that cast a vote (k, above) is not necessarily equal to the size of the set HS, since only hyper-spheres that contain the test point are allowed to cast a vote.

Described more informally, the margin function for the classifier under study is calculated by subtracting the number of incorrect votes made by the hyper-spheres from the number of correct votes made by the hyper-spheres. This is multiplied by one divided by the total number of votes cast, which causes the margin to be between [-1, 1].

Applying the margin function described above to the margin drawn by Shapiro is simple, now that the margin function is defined for it. The VC-dimension of the base classifier is (d+1), as proven in the previous section. The data set size is easily determined, and the margin function is defined above. Therefore the following bound on the generalization error of voting classifiers also holds for the current algorithm under study:

$$\text{Generalization Error} \leq \begin{cases} if\,\exists h\{h|h \in HS, d(h[c],x) \leq h[r]\} & \widehat{\Pr}[mg(x,y;\theta_1,\theta_2,\ldots\theta_k)] + \tilde{O}(\sqrt{\frac{d}{m\theta^2}}) \\ otherwise & R^*(1+\frac{\gamma}{\sqrt{k}}(1+O(k^{-1/6}))) \end{cases} \tag{42}$$

The bound is a combination of the k-NN and voting classifier bounds already defined. When the point to be classified is found within one of the hyper-spheres that makes up the classifier, then the voting classifier bound is applied. Otherwise, the k-NN bound is applied.

5. Experimental Setup

Model validation was done using 10-fold cross validation. To this end, the dataset is split evenly into 10 subsets. The split between training, testing, and validation sets is 80%/10%/10%, respectively, and is the same in every test. Stratification is used as well, which means that the 10 subsets are sampled so that each class is evenly represented in each subset. The results were averaged and graphed in Figures 8 through 14. The averages were calculated from 10 runs of the algorithm. In this paper we will use the same data set as in our previous publications [18-19]. The data set contains data about the statistical properties of traffic flows as well as a class label denoting the type of application to which it belongs. The data set is found in [4]. The data set has 249 features for each flow, but that number is reduced to 11 features in our tests, according to the feature reduction analysis also done in [4]. The features used are listed and described in Table II. The FTP application class listed in the table is separated into three different classes within the dataset, encompassing control, passive and data FTP flows respectively. The total number of classes in the data set equal to 12. The class labels and corresponding applications are listed in Table III. These features are used throughout the tests shown in the next section, however they are not used in the tests performed to show the curse of dimensionality.

Since the algorithm requires parameters, we set aside one subset in every test run to determine the best values for these parameters. To accomplish this, a grid search is performed on the validation set, with the objective of finding the parameters which maximize the accuracy of the algorithm. Two parameters are varied: k is varied between 1 and 12, and the step size is varied between 0.01 and 0.99 in 0.01 increments. Through experience, we have found that these ranges are big enough to find the best values for these parameters, without taking too long to finish the search. However, we do not determine the size of the population required for maximum accuracy, since we are varying this value and graphing it along with accuracy.

TABLE II. FEATURE NAMES AND DESCRIPTIONS[4]

Feature	Description
Port, *server*	Port Number at server
Number of pushed data packets, *server->client*	# of packets with the PUSH bit set in the TCP header
Initial window bytes, *client->server*	# of bytes in the initial window
Initial window bytes, *server->client*	# of bytes in the initial window
Average segment size, *server->client*	average segment size
IP data bytes median, *client->server*	Median of total bytes in IP packets
Actual data packets, *client->server*	# of packets with at least a byte of TCP data payload
Data bytes in the wire variance, *server->client*	Variance of # of bytes in Ethernet packet
Minimum segment size, *client->server*	minimum segment size
RTT samples, *client->server*	The total number of Round Trip Time (RTT) samples.
Pushed data packets, *client->server*	# of packets with the PUSH bit set in the TCP header

We performed four sets of experiments to be able to fully explore the capabilities of the algorithm, and compare them to other work. Four claims will be tested in the experiments dealing with the execution time, classification time, and classification performance of the algorithm. Our claims about the algorithm are these:

1. The new and optimized training algorithm is faster than the old training algorithm.

2. The new and optimized classification algorithm is faster than the old classification algorithm.

3. The new and optimized training and classification algorithms deal with the curse of dimensionality well.

4. The classification performance of the old algorithm might be improved with different distance measures.

We tested these claims by performing a total of 9 experiments, the results of which are graphed in the figures in the next section. In order to test the time required by the new training algorithm, we varied the size of the training set and the size of the antibody population, while measuring the amount of time required to train the population of antibodies.

To test the time taken by the new classification algorithm, we varied the size of the antibody population while measuring the time required to classify the test set, since the antibody population size is the only value that affects the performance of the classification algorithm.

The classification algorithm was tested in four different configurations. First, the original training algorithm, unchanged, second, the original training algorithm with majority voting. Thirdly and fourthly, the optimized training algorithm with and without secondary filtering. These tests were used in order to show the performance of the algorithm in many different configurations.

The curse of dimensionality is an important concern in machine learning. The effects of the curse of dimensionality on the performance of the algorithm are tested by increasing the number of dimensions used to train the antibody population. By

TABLE III. CLASS LABELS AND APPLICATIONS[4]

Class Label	Applications
FTP-CONTROL, FTP-PASV, FTP-DATA	FTP
DATABASE	Postgres, Sqlnet, Oracle
INTERACTIVE	SSH, klogin, rlogin, telnet
MAIL	IMAP, POP2/3, SMTP
SERVICES	X11, DNS, LDAP, NTP
WWW	WWW
P2P	KaZaA, BitTorrent
ATTACK	Worm and virus attacks
GAMES	Half-Life, etc.
MULTIMEDIA	Windows Media Player

extension, the number of dimensions that the hyper-spheres making up the population of antibodies are defined in is also increased. The features of the data set chosen for these tests are not the same as are used for all other tests, since we are not trying to maximize the accuracy of the algorithm the features chosen are not important. They are chosen by their order in the files making up the data set, the first 25 features being chosen for these tests.

In this research we also tried several different distance functions, as described in [49], [50], and [51]. We tested the Euclidian distance, Manhattan distance, dot product, cosine distance, and the Chebyshev distance. We also tested several kernel functions in our previous research with the Euclidian distance and found them to not be very useful, therefore we did not test kernel functions with the new algorithm.

In order to compare the optimized and unoptimized training algorithms as fairly as possible, in every test done, both the optimized and unoptimized training algorithms were trained using the exact same dataset. For the same reasons, in every test done on the classification algorithm, both optimized and unoptimized algorithms worked with the exact same population of antibodies. This was done so that we could compare both versions of the algorithm without worrying about randomness affecting the results.

The tests were performed on an Intel i5 processor running at 1.80 GHz, with 4 GB of memory. The operating system used was 64-bit Windows 8.1. The programming language used for implementation was Python and the interpreter version was 3.5.

6. Results and Discussion

This section explains the results of our tests, and the validity of the four claims we made in section VII.

The relationship between the training time and the data set size is shown in Figure 8. The antibody population size is held constant at 1000, and the data set size increases from 200 to 1000. While the new training algorithm remains linear, it is still much faster than the old training algorithm.

The relationship between the training time and the antibody population size is shown in Figure 9. The data set size is held constant at 1000, and the antibody population size increases from 200 to 1000. The new training algorithm is also linear in this chart and is also much faster than the old training algorithm.

The relationship between the training time and the number of features in the data set is shown in Figure 10. The data set size is held constant at 1000, and the antibody population size is held constant at 100. The number of dimensions increases from 5 to 25. Figures 8 and 9 support claim 1, and Figure 10 supports claim 3.

The SVM algorithm is tested on the same data sets as the optimized and unoptimized AIS algorithms in Figures 8, 9, and 10. However, the SVM algorithm is still faster than the AIS algorithm, and its line can be seen along the bottom of the three figures.

The three tests performed on the classification portion of the algorithm are unrelated to the data set size used to train the population of antibodies, even when performing k-NN classification as a fallback, since only the antibody population is used. Therefore, we do not vary the data set size, or include the value in any graph.

The relationship between the prediction time and the antibody population size is shown in Figure 11. The antibody population size is increased from 200 to 1000. We included several versions of the prediction function to be able to highlight the differences in performance. The original prediction function is also implemented with majority voting (which was not implemented in the original algorithm) to show the slower performance of this technique. The optimized prediction function is implemented with and without secondary filtering. While the optimized classification algorithm remains linear, it is still much faster than the old classification algorithm. The graph shows that, as expected, the original classification algorithm with majority voting is the slowest version.

The relationship between the prediction time and the number of dimensions of the antibody population is shown in Figure 12. The antibody population size is held constant at 100, and the number of dimensions is increased from 5 to 25. The results shown in Figure 12 are similar to the results in Figure 11. Figure 11 supports claim 2, and Figure 12 supports claim 3.

To demonstrate that the accuracy of the optimized algorithm did not decrease as a result of the changes implemented and that the prediction algorithm remains functionally the same, we graphed the accuracy of the four prediction functions in Figure 13. The size of the population of antibodies is varied from 200 to 1000, and the size of the data set is fixed at 1000. The prediction functions worked with the exact same population of antibodies. Figure 13 shows the relationship between the antibody population size and the accuracy of the classification algorithm.

The accuracy remains the same, except for the version of the algorithm without secondary filtering, which decreased accuracy. We expected the accuracy to decrease, and this graph illustrates that the secondary filtering is absolutely essential for the algorithm to provide accurate predictions.

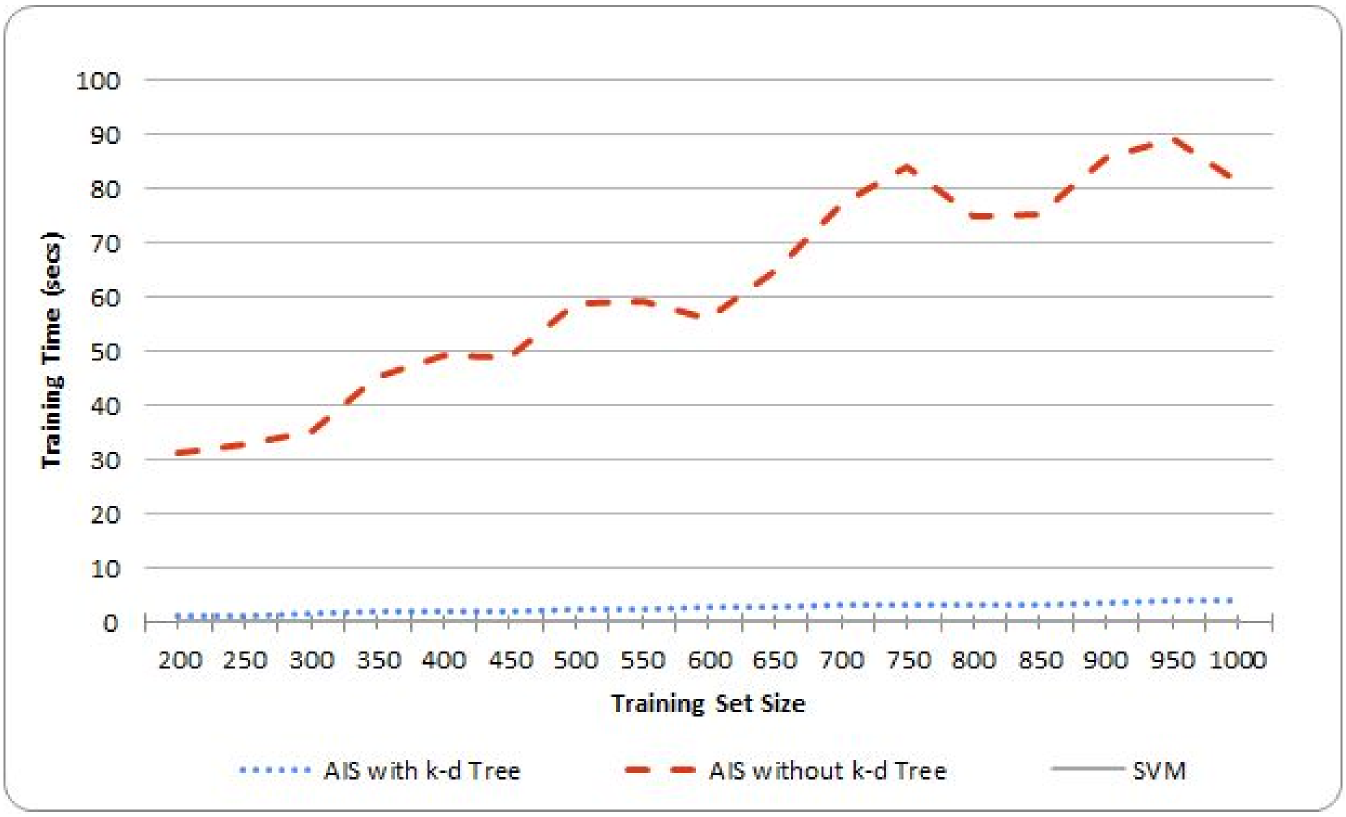


Fig 8. Data Set Size and Training Time

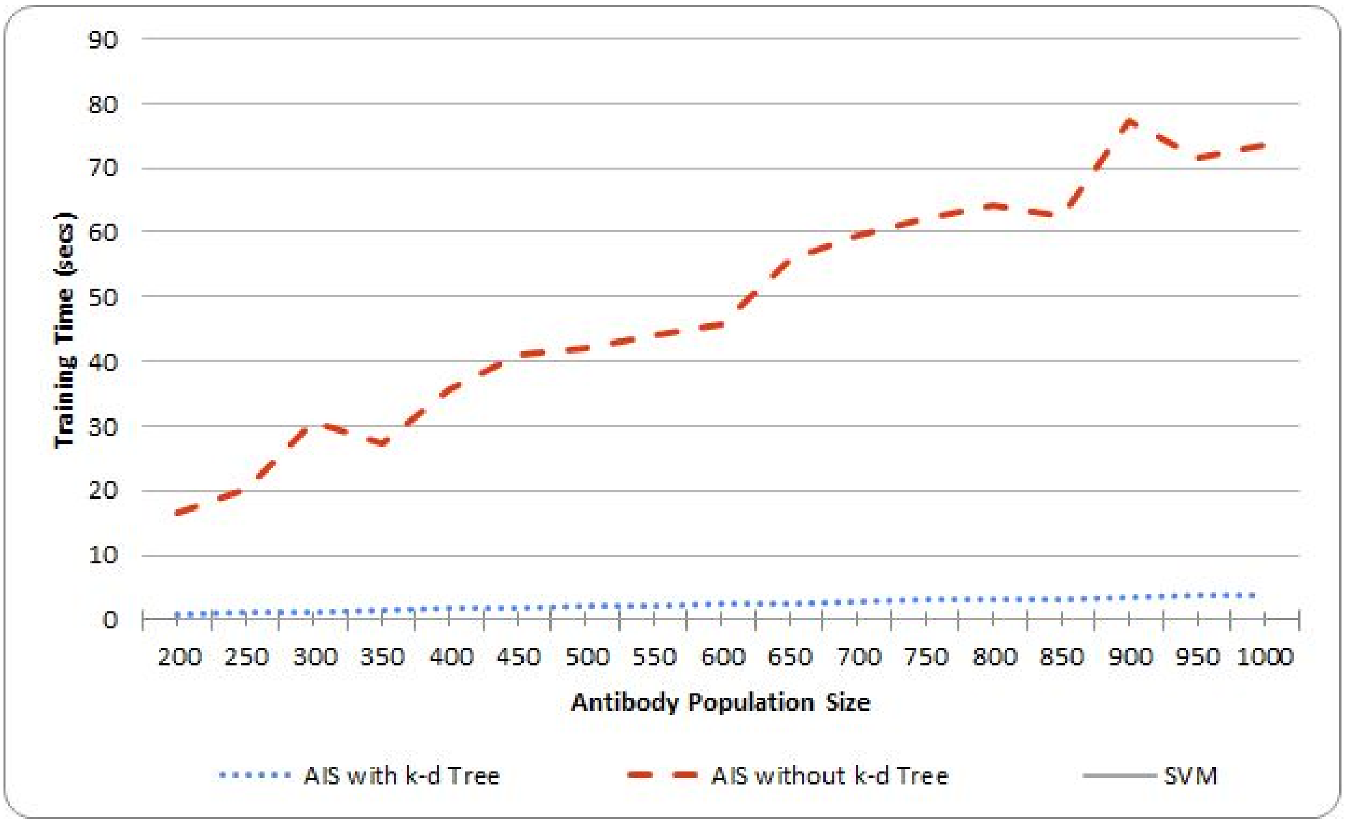


Fig 9. Antibody Population Size and Training Time

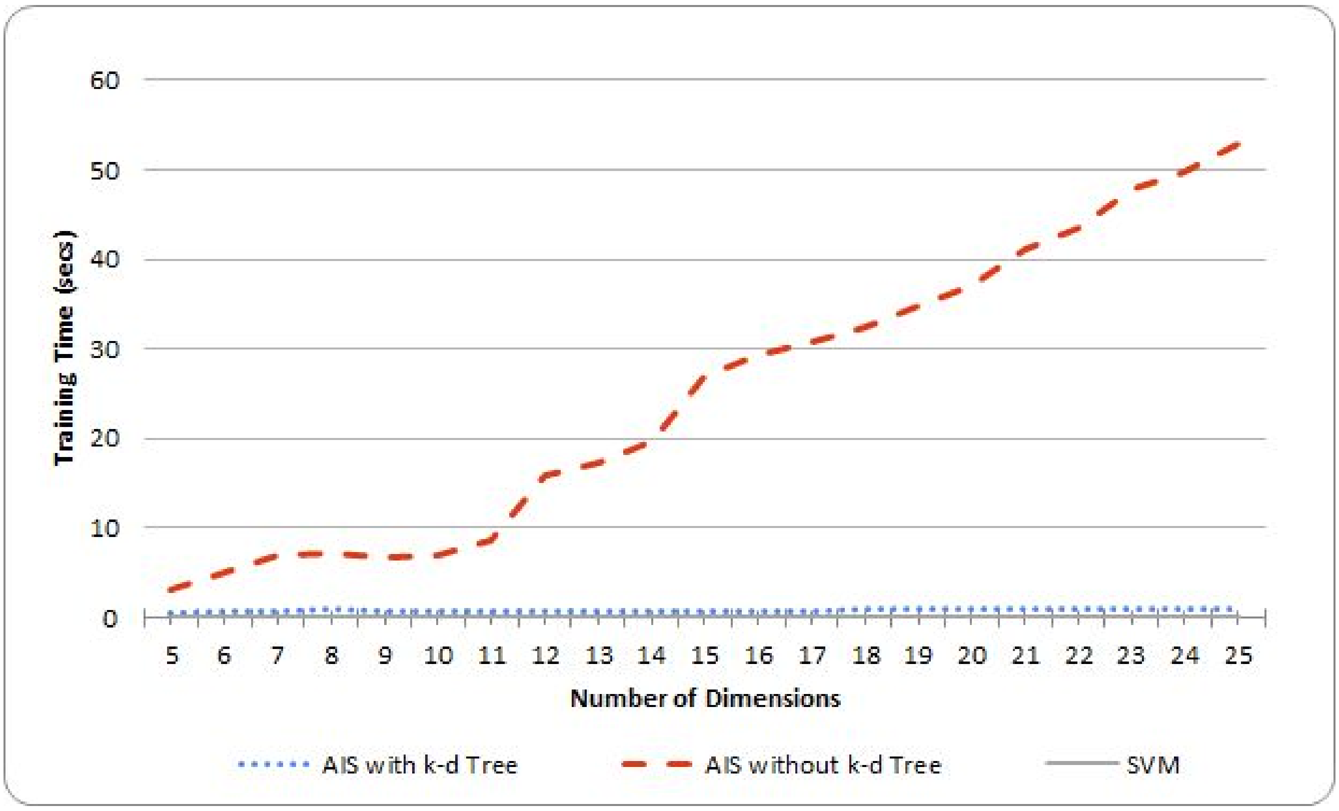


Fig 10. Number of Dimensions in Data Set and Training Time

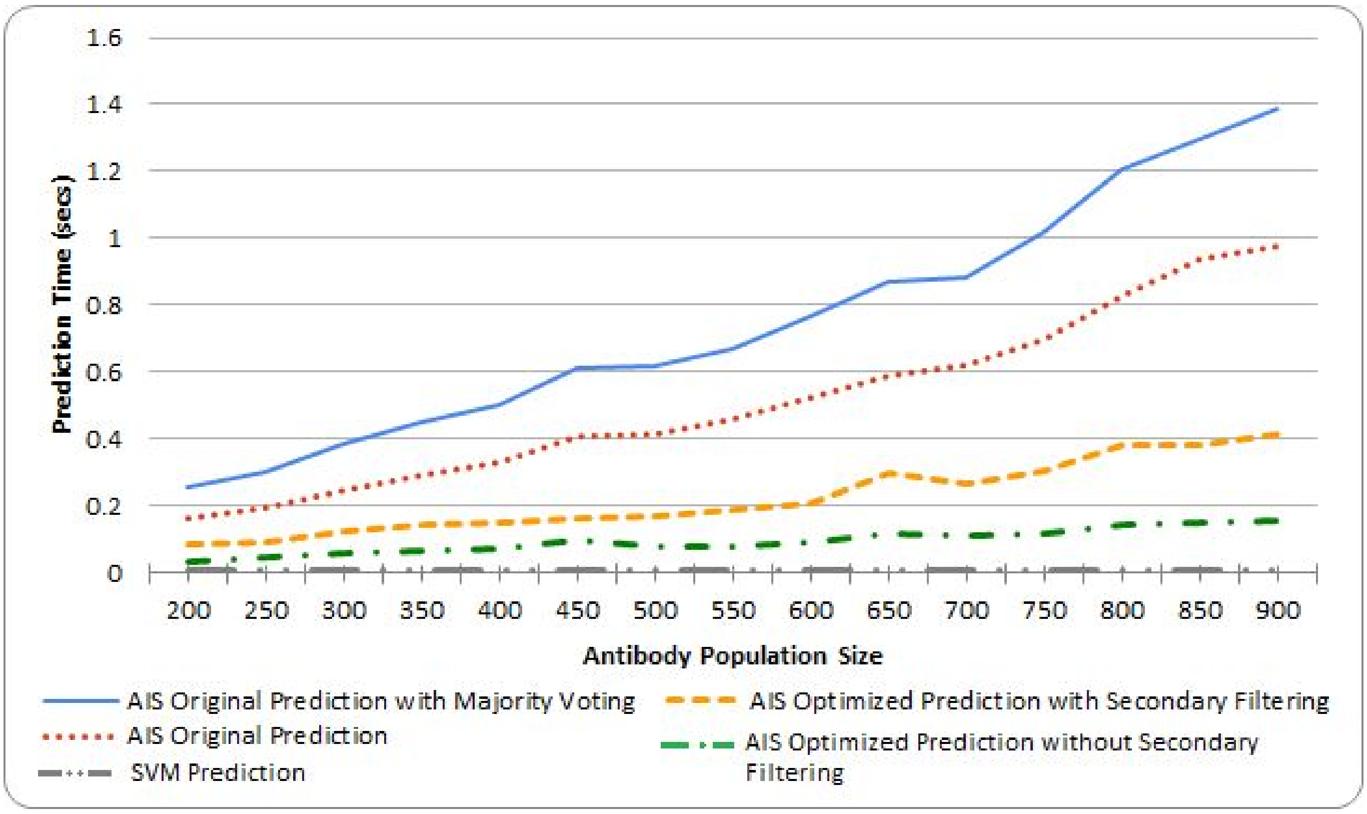


Fig 11. Antibody Population Size and Prediction Time

The SVM prediction algorithm is graphed along with the AIS algorithms in Figures 11, 12, and 13. It can be seen in Figures 11 and 12 that the SVM algorithm is faster in making predictions than any AIS algorithm. However, Figure 13 shows that the AIS algorithms have higher accuracy than SVM, for the given data set size.

By combining knowledge gained from Figures 11, and 12 we can see that removing secondary filtering improved the classification time significantly, but it proves to be useless, since it gives very low classification accuracy, as shown in Figure 13. Figure 11 shows that primary filtering provides most of the speedup gained by the optimized algorithm.

Figure 13 shows that the resulting antibody population did not change in any way from the original training algorithm, and the classification performance remains the same, even as the training and classification algorithms are faster. However, the new classification algorithm does require the creation of a *k-d* tree from the antibody population. With the current implementation it doubles the memory needed for the population of antibodies. This is currently an implementation detail of the algorithm, it is not necessary to store the antibody population twice, since the structure of the tree can be encoded into the data structure holding the antibody population.

For Figures 8 through 13 the algorithm was tested using the Euclidian distance, since it is the most commonly used distance measure in AIS research. However, we are also interested in improving the classification accuracy of the algorithm in this application, so the next set of tests and figures deal with several different distance measures.

The remaining figures deal with claim 4 from the previous section, which is about the classification performance of several different distance measures. Figure 14 shows the relationship between the antibody population size and the accuracy of the classification. The size of the data set is fixed at 1000 for this test. Although it is not always the most accurate distance measure, we can see that the Manhattan distance is the most accurate throughout most of the range. The maximum accuracy achieved is 94.77% by the Manhattan distance measure.

A confidence interval was calculated using data from Figure 14 for the difference in the accuracy between the algorithm using the Euclidian distance and the algorithm using the Manhattan distance functions. The difference was calculated by subtracting the

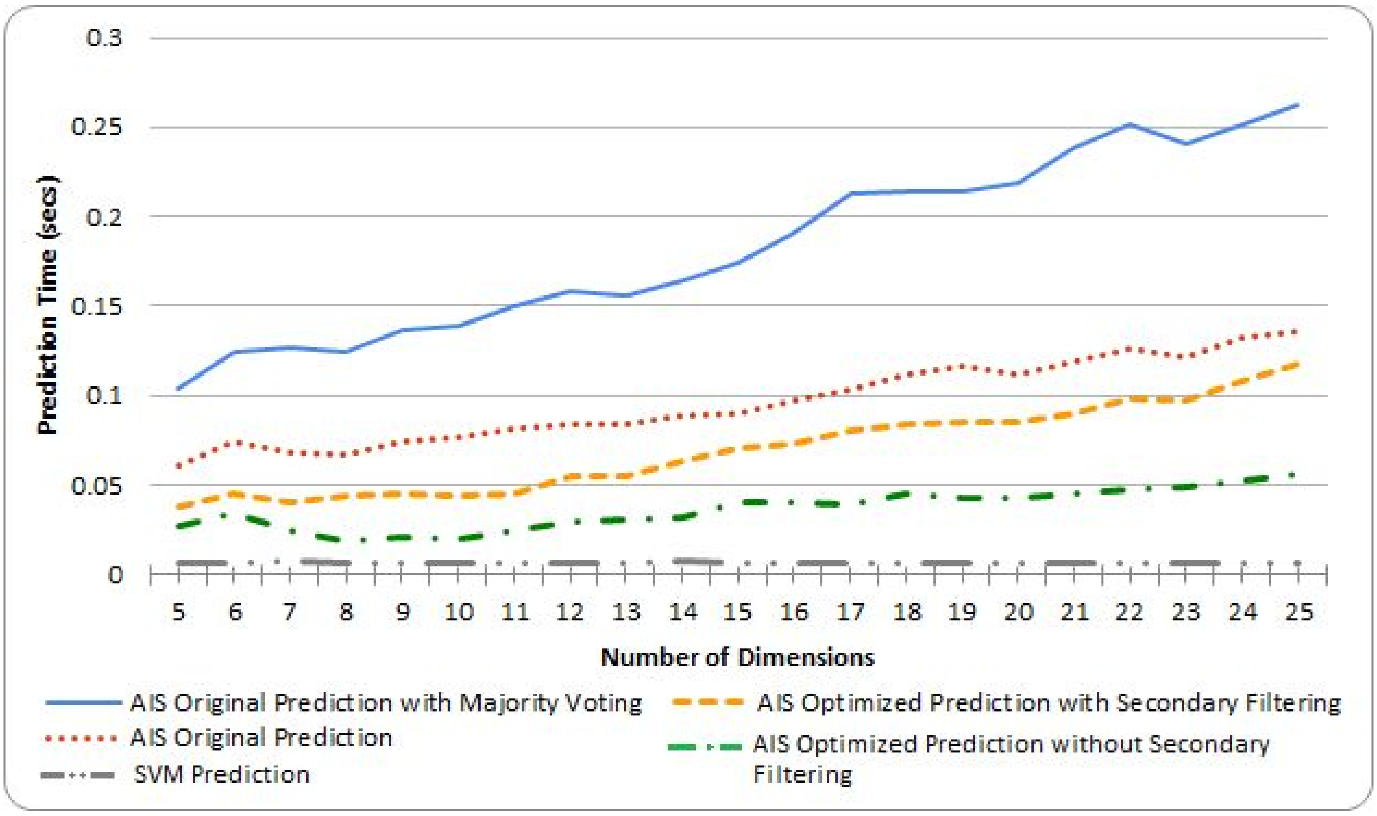


Fig 12. Number of Dimensions of Antibodies and Classification Time

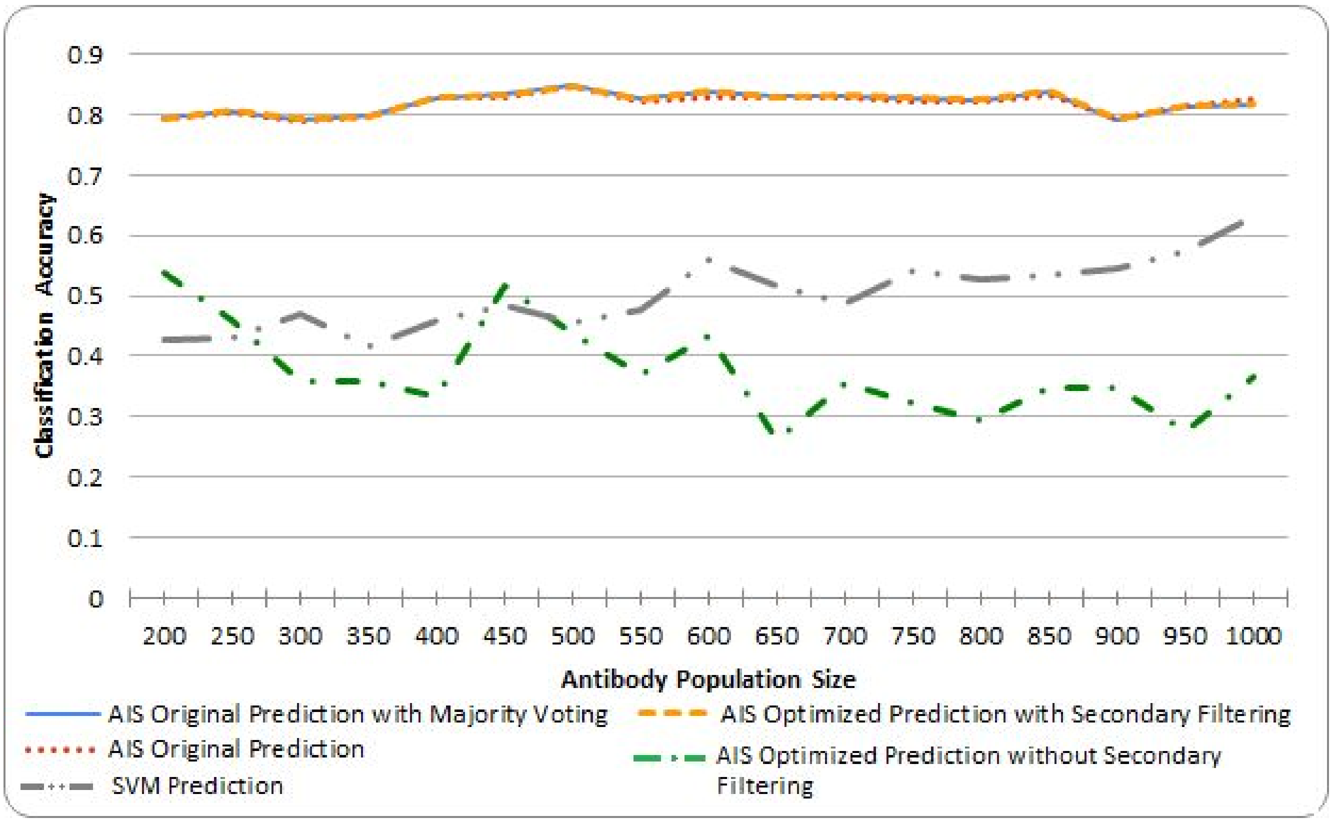


Fig 13. Population Size and Accuracy

accuracy of the Manhattan distance and the accuracy of the Euclidian distance, this made the difference positive. With these values a confidence interval was calculated at the 95% confidence level. The average difference in the accuracy was calculated to be between 1.9 and 1.16 percentage points. This shows that using the Manhattan distance gives higher prediction accuracy, and that it is not due to chance but is statistically significant.

Figure 15 shows the relationship between the data set size used and the accuracy. The size of the antibody population is fixed at 1000 for this test. This chart also shows the Manhattan distance giving the highest accuracy, with a maximum of 95.99% accuracy.

A confidence interval was calculated using data from Figure 15 for the difference in the accuracy between the Euclidian distance and the Manhattan distance functions. The difference was calculated by subtracting the accuracy of the algorithm using the Manhattan distance and the accuracy of the algorithm using the Euclidian distance, this made the difference positive. With these values a confidence interval was calculated at the 95% confidence level. The average difference in the accuracy was calculated to be between 1.45 and 0.86 percentage points. This shows that using the Manhattan distance gives higher prediction accuracy, and that it is not due to chance but is statistically significant.

Figure 16 shows the relationship between the data set size and the F-measure. The F-measure of a classifier is calculated on a class-by-class basis and is the weighted average of precision and recall. It has a range of [0, 1], with 1 being the best performance. The size of the antibody set is fixed at 1000 for this test. The Chebyshev and Manhattan distances are the highest performing distance measures tested.

The data set we have used for testing has one characteristic that really affects the performance of our algorithm: it is very unbalanced. The most common class of flow in the data is the “WWW” class, which makes up 86.9% of the flows, with 328,092 flows present in the data set. The least common class is “GAMES” with only 8 flows present, and right with it is “INTERACTIVE” with only 110 flows. Even the combined “FTP” classes contain an order of magnitude less than the “WWW” class. As with many things in the real world, the data follows a Pareto distribution, with many flows available from a few classes, and few flows available from the remaining classes.

The unbalanced nature of the data set affected the algorithm we tested in the previous publications [18-19], with the “GAMES” and “INTERACTIVE” classes having a very low F-measure score in the tests performed. This can be seen in these tests in Figure 16, where the Euclidian as well as the Cosine distances are found at the bottom of the chart. However, we have been surprised by the performance of the Manhattan and Chebyshev distances which have provided very good performance, in spite of the unbalanced data set.

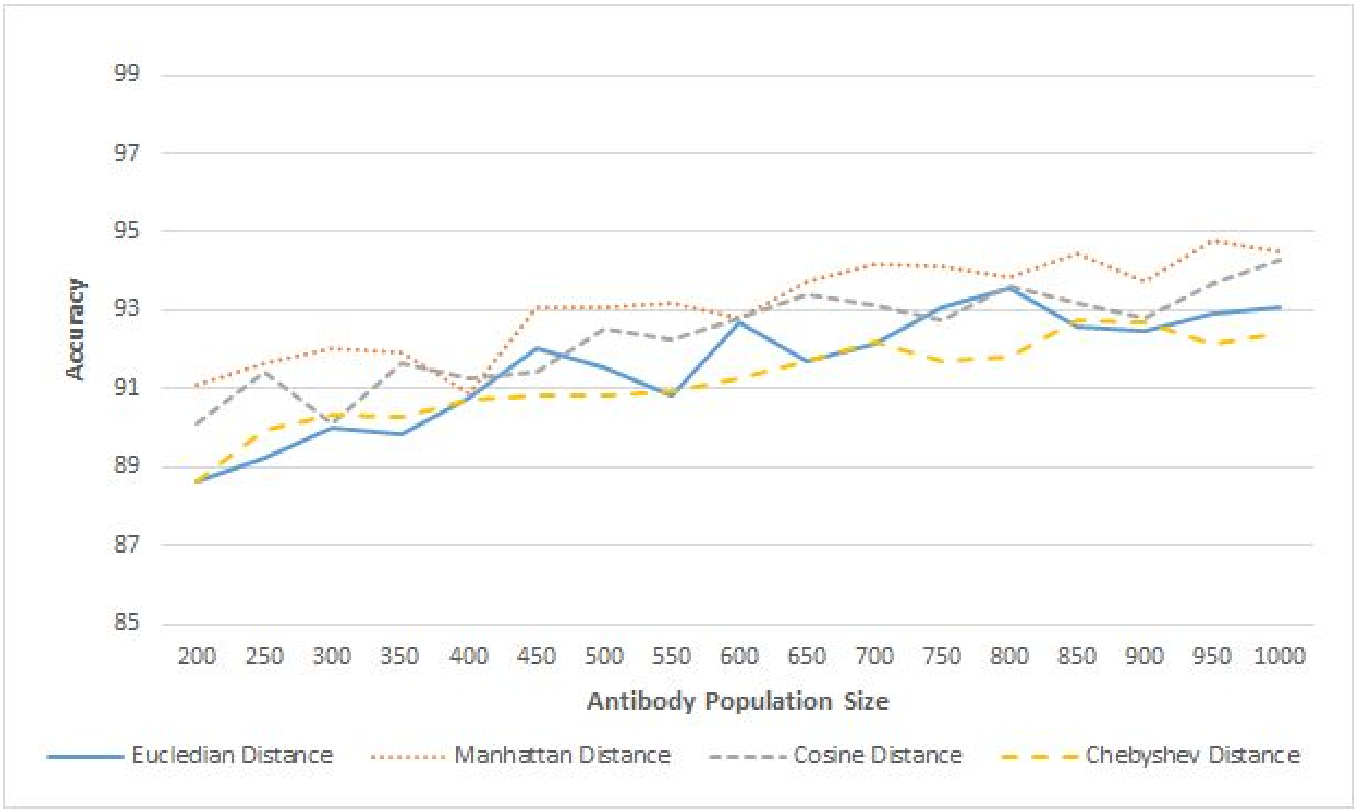


Fig 14. Antibody Population and Accuracy

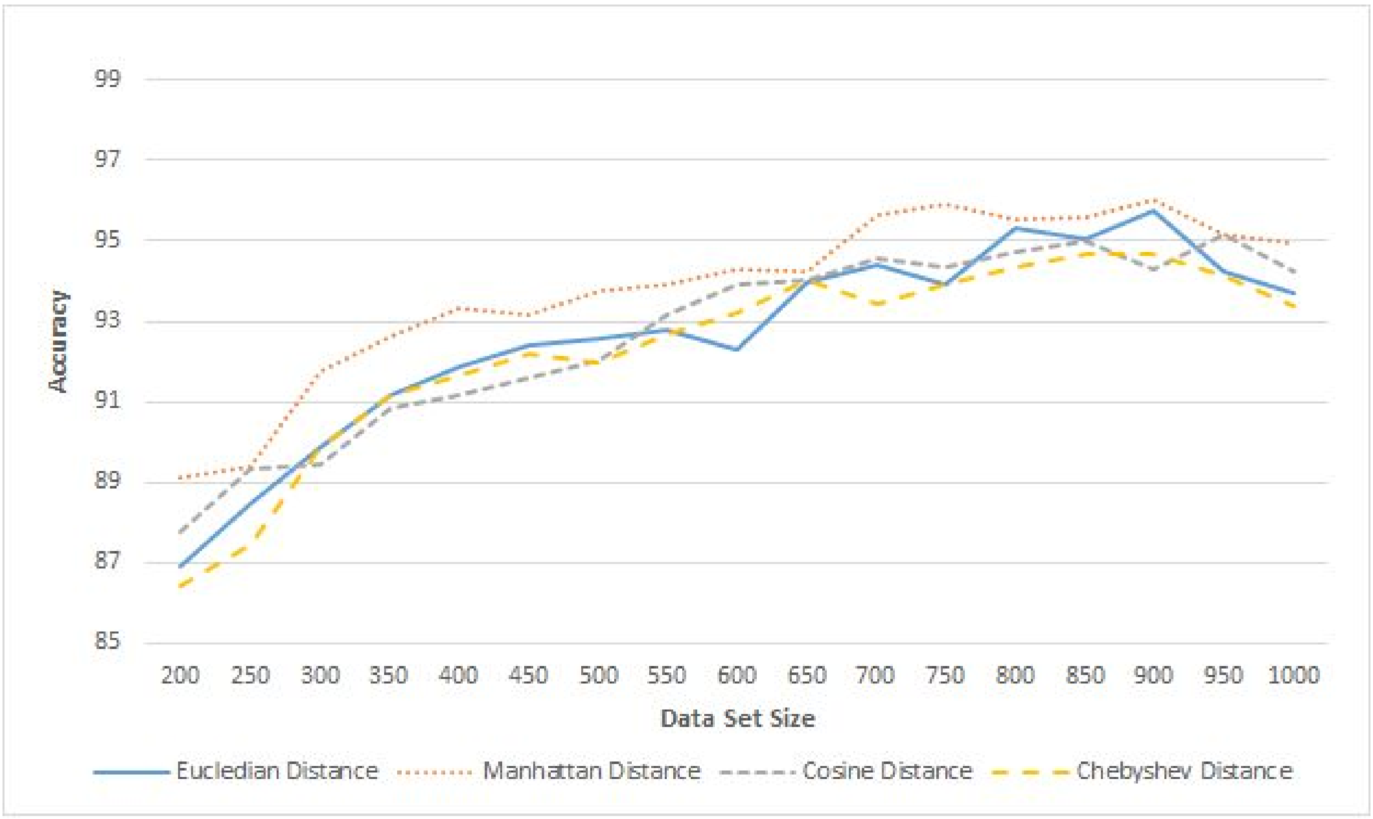


Fig 15. Data Set Size and Accuracy

Lastly, we see that even though the “P2P,” “ATTACK,” and “DATABASE” classes have a comparatively small number of flows in the data set, the F-measure for these classes is not as affected as the “GAMES” and “INTERACTIVE” classes.

Although we did implement and test the dot product as a distance function, the resulting accuracy was very low and did not merit inclusion in any of the figures.

7. Conclusions and Future Work

This work has shown that it is indeed possible to optimize the performance of an AIS-inspired algorithm. We have also seen that the algorithm is functionally identical to the previous unoptimized version. Additionally, we were able to modify the original algorithm to make it more predictable without sacrificing any accuracy. Moreover, we drawn a bound on the generalization error of the algorithm, basing it on previous work in the field of ensemble classifiers. We have not found any other research in the application of bounds to artificial immune system classification algorithms.

We have also shown that the most common distance measure used, the Euclidian distance measure, does not give the best performance for this problem. We have shown that the Manhattan distance measure gives better classification performance, however, we believe that this is problem dependent and there is no good way to predict which distance measure will maximize the accuracy achieved by the algorithm. All distance measures must be tested to find the best one.

We have found that the algorithm performed as well as other algorithms, and was insensitive to input parameters. It is also very simple to implement, and generalizes well from small training data sets. We also give the asymptotic complexity of the optimized algorithm as well as draw a bound on the generalization error of the algorithm. In this paper we have shown how the changes and optimizations applied to the original algorithm do not functionally change the original algorithm, while making its execution much faster.

The AIS-inspired algorithm we designed and tested is based on the positive selection AIS algorithm, which is itself very similar to the well-known Negative Selection AIS algorithm. In future research it would be very simple to implement the same optimization in a negative selection algorithm, and get the same speedups. Based on a literature review, we have not found the techniques used in this paper anywhere else in the literature of optimized AIS algorithms.

Upon examination of the raw data in the data structure holding the antibodies, we found a few antibodies that have a very small radius. There antibodies are not very useful in classification, since they do not generalize well, and are unlikely to assist in the classification of a test point. A simple test in the training algorithm could be implemented so that only antibodies that are bigger than a certain radius are accepted into the population. However, the performance of the classifier would not be guaranteed to increase, and more tests would be needed to confirm that it is a valuable change to the algorithm.

In the analysis section of this paper, we have shown how the quality of the margin of each individual classifier affects the final performance of an ensemble classifier. Our classifier is very naïve in its approach to selecting the placement of it base classifiers in the feature space, using a very simple strategy. This is done this way to make the training faster, and we have gotten good classification performance in spite of this simplicity. In future research we can investigate different ways to optimize the margin of each base classifier during the training portion of the algorithm by optimizing the placement of its center as well as it radius. Furthermore, the scheme used to set the radius of each antibody is also very simple, since it is an approximation of the class boundary and does not take into account the generalization ability of the algorithm. It might be interesting to try other ways of finding an optimal antibody radius which allow an antibody to misclassify examples, but increase the generalization ability of the classifier.

Since beginning work on this AIS-inspired algorithm we have found it to be particularly insensitive to the choice in parameters. Unlike other classification algorithms, this algorithm does not seem need to have its parameters set up perfectly to give good performance. An open question for us is: how insensitive is the algorithm to the values of the parameters given to it? Another interesting research direction would be to apply a similar bound to the negative selection algorithm, using hyper-spheres as base classifiers. Furthermore, a bound could be drawn on this and other artificial immune system algorithms that use base classifiers that are not hyper-spheres.

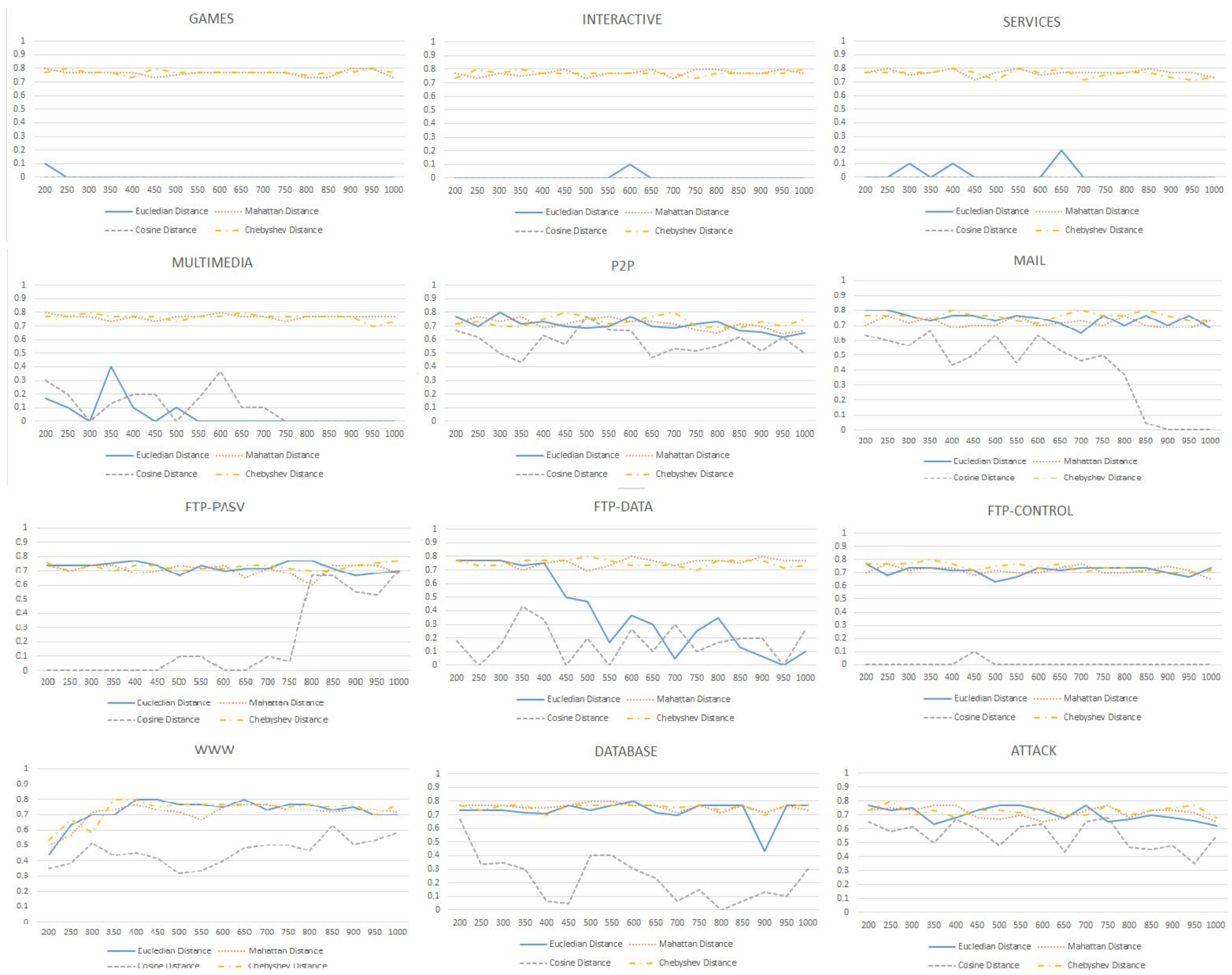


Fig 16. F-measure and Training Set Size

8. Vitae

Brian Schmidt received his Bachelor's in Computer Science from Andrews University in 2009, and a Master's in Software Engineering in 2011 from the same university. He is currently a graduate student in Computer Science in the Computer Science department of Western Michigan University. His research interests include Machine Learning, Artificial Intelligence, Biologically Inspired Computing, and Soft Computing.

Ala Al-Fuqaha (S'00-M'04-SM'09) received his M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Missouri-Columbia and the University of Missouri-Kansas City, in 1999 and 2004, respectively. Currently, he is a Professor and director of NEST Research Lab at the Computer Science Department of Western Michigan University. His research interests include Wireless Vehicular Networks (VANETs), cooperation and spectrum access etiquettes in cognitive radio networks, smart services in support of the Internet of Things, management and planning of software defined networks (SDN), intelligent services for the blind and the visually impaired, QoS routing in optical and wireless networks, and performance analysis and evaluation of high-speed computer and telecommunication networks. He is currently serving on the editorial board for John Wiley's Security and Communication Networks Journal, John Wiley's Wireless Communications and Mobile Computing Journal, EAI Transactions on Industrial Networks and Intelligent Systems, and International Journal of Computing and Digital Systems. He is a senior member of the IEEE and has served as Technical Program Committee member and reviewer of many international conferences and journals.

Ajay Gupta is a Professor of Computer Science at Western Michigan University and the TCPP-Chair of IEEE-CS. From 1998 to 2002, he was the Chairman of the Computer Science Department at Western Michigan University. Dr. Gupta received his Ph.D. in Computer Science from the Purdue University in 1989, his M.S. in Mathematics and Statistics from the University of Cincinnati in 1984, and his B.E. (Honors) in Electrical and Electronics Engineering from the Birla Institute of Technology and Sciences, Pilani, India in 1982. Dr. Gupta's research interests include sensor systems, cloud computing, mobile computing, web technologies, computer networks, evolutionary computation, scientific computing, and design and analysis of parallel and distributed algorithms. He has published numerous technical papers and book chapters in refereed conferences and journals in these areas. He is a senior member of the IEEE and member of the IEEE Computer Society, the IEEE Communications Society, the ASEE and the ACM.

Dionysios Kountanis received his Ph.D. degree in Computer Science from the University of Pennsylvania in 1977. Currently, he is an Associate Professor at the Computer Science Department of Western Michigan University. Prior experience includes service on the faculties of Rutgers University and Temple University and as a systems programmer for Dun and Bradstreet. His research interests include: graph algorithms, computational geometry, computer architectures, artificial intelligence, mobile ad hoc networks, and telecommunication networks.

9. References

[1] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," Proceedings of the 2008 ACM CoNEXT conference, pp. 11, December 2008. DOI: 10.1145/1544012.1544023

[2] S. Katal, and A. Singh. "A Survey of Machine Learning Algorithm in Network Traffic Classification," In International Journal of Computer Trends and Technology (IJCTT), Volume 9 number 6, 2014. DOI: 10.1109/SURV.2008.080406

[3] S. A. Hofmeyer, and S. Forrest, "Immunity by design: An artificial immune system," in Proc. of the Genetic and Evolutionary Computation Conf., Orlando, FL, 1999, pp. 1289-1296.

[4] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," In ACM SIGMETRICS Performance Evaluation Review, Vol. 33, No. 1, pp. 50-60, 2005. DOI: 10.1145/1064212.1064220

[5] T. T. Nguyen, and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," In Communications Surveys & Tutorials, 10(4), pp. 56-76, 2008. DOI: 10.1109/SURV.2008.080406

[6] R. Alshammari and A. N. Zincir-Heywood, "Machine learning based encrypted traffic classification: Identifying ssh and skype," In IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA2009), pp. 1-8, 2009. DOI: 10.1109/CISDA.2009.5356534

[7] K. Singh and S. Agrawal, "Comparative analysis of five machine learning algorithms for IP traffic classification," In 2011 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC), pp. 33-38, April 2011. DOI: 10.1109/ETNCC.2011.5958481

[8] L. Jun, Z. Shunyi, L. Shidong, and X. Ye. "P2P traffic identification technique," In 2007 International Conference on Computational Intelligence and Security, pp. 37-41. IEEE, 2007. DOI: 10.1109/CIS.2007.81

[9] Y. S. Lim, H. C. Kim, J. Jeong, C. K. Kim, T. T. Kwon and Y. Choi, "Internet traffic classification demystified: on the sources of the discriminative power," In Proceedings of the 6th International Conference, pp. 9, November 2010. DOI: 10.1145/1921168.1921180

[10] T. S. Tabatabaei, F. Karray, and M. Kamel, "Early internet traffic recognition based on machine learning methods," In 25th IEEE Canadian Conference on Electrical & Computer Engineering (CCECE), pp. 1-5. IEEE, 2012. DOI: 10.1109/CCECE.2012.6335034

[11] M. S. Aliakbarian, A. Fanian, F. S. Saleh, and T. A. Gulliver. "Optimal supervised feature extraction in internet traffic classification," In 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 102-107. IEEE, 2013. DOI: 10.1109/PACRIM.2013.6625457

[12] F. Ertam, and E. Avci "Classification with Intelligent Systems for Internet Traffic in Enterprise Networks," In International Journal of Computing, Communications & Instrumentation Engineering (IJCCIE), Vol. 3, pp. 2349-1469, 2013.

[13] F. Ertam, and E. Avcı, "Network Traffic Classification via Kernel Based Extreme Learning Machine", International Journal of Intelligent Systems and Applications in Engineering (IJISAE), Vol 4 (Special Issue), pp. 109–113, 2016 .

[14] R. Raveendran, and R. R. Menon, "A novel aggregated statistical feature based accurate classification for internet traffic," In International Conference on Data Mining and Advanced Computing (SAPIENCE), pp. 225-232, IEEE, 2016.

[15] P. Casas, A. D'Alconzo, P. Fiadino, and C. Callegari, “Detecting and diagnosing anomalies in cellular networks using Random Neural Networks,” In International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 351-356, IEEE, 2016.

[16] M. Shafiq, X. Yu, and A. A. Laghari, “WeChat Text Messages Service Flow Traffic Classification Using Machine Learning Technique”, In 6th International Conference on IT Convergence and Security (ICITCS), pp. 1-5, IEEE, 2016.

[17] S. V. Westlinder, “Video Traffic Classification: A Machine Learning approach with Packet Based Features using Support Vector Machine”, Master’s Thesis, Karlstad University, Faculty of Health, Science and Technology, Department of Mathematics and Computer Science, 2016

[18] B. Schmidt, D. Kountanis, A. Al-Fuqaha, “A Biologically-Inspired Approach to Network Traffic Classification for Resource-Constrained Systems,” In International Symposium on Big Data Computing (BDC 2014), London, UK, December. 2014.DOI: 10.1109/BDC.2014.13

[19] B. Schmidt, D. Kountanis, A. Al-Fuqaha, “Artificial Immune System Inspired Algorithm for Flow-Based Internet Traffic Classification,” In CloudCom 2014, Singapore, December, 2014. DOI: 10.1109/CloudCom.2014.108

[20] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri. "Self-nonself discrimination in a computer," In Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 202-212. IEEE Computer Society, 1994. DOI: 10.1109/RISP.1994.296580

[21] L. N. De Castro, and F. J. Von Zuben, “Learning and optimization using the clonal selection principle,” In IEEE Transactions on Evolutionary Computation, 6(3), pp. 239-251, 2000. DOI: 10.1109/TEVC.2002.1011539

[22] M. Burnet, "The clonal selection theory of acquired immunity." The Glonal Selection Theory of Acquired Immunity. (1959). DOI: 10.5962/bhl.title.8281

[23] D. E. Goodman, L. Boggess, and A. Watkins, “Artificial immune system classification of multiple-class problems,” In Proceedings of the artificial neural networks in engineering, vol. 2, pp. 179-183, 2002. doi>10.1023/B:GENP.0000030197.83685.94

[24] A. Watkins, “AIRS: A Resource Limited Artificial Immune Classifier,” M.S. thesis, Department of Computer Science. Mississippi State University, MS, 2001. DOI: 10.1109/CEC.2002.1004472

[25] J. A. White and S. M. Garrett, “Improved pattern recognition with artificial clonal selection?,” In Artificial Immune Systems, pp. 181-193, 2003. DOI: 10.1007/978-3-540-45192-1_18

[26] J. Brownlee, "Clonal Selection Algorithms," Technical Report 070209A, Complex Intelligent Systems Laboratory (CIS), Centre for Information Technology Research (CITR), Faculty of Information and Communication Technologies (ICT), Swinburne University of Technology, 2007

[27] U. Markowska-Kaczmar and B. Kordas, “Multi-class iteratively refined negative selection classifier,” In Applied Soft Computing, pp. 972-984, 2008. DOI: 10.1016/j.asoc.2007.07.012

[28] U. Markowska-Kaczmar and B. Kordas, “Negative Selection based method for Multi-Class problem classification,” In Sixth International Conference on Intelligent Systems Design and Applications, Vol. 2, pp. 1165-1170, October 2006. DOI: 10.1016/j.asoc.2007.07.012

[29] K. B. Sim, and D. W. Lee. "Modeling of positive selection for the development of a computer immune system and a self-recognition algorithm," In International Journal of Control, Automation, and systems,1(4), pp. 453-458, 2003

[30] M. Middlemiss, "Positive and negative selection in a multilayer artificial immune system," University of Otago, Otago, New Zealand, 2006.

[31] Z. Fuyong, and Q. Deyu, "Run-time malware detection based on positive selection," In Journal in Computer Virology 7, no. 4, pp. 267-277, 2011. DOI: 10.1007/s11416-011-0154-8

[32] Z. Fuyong, and D. Qi. "A Positive Selection Algorithm for Classification," In Journal of Computational Information Systems 8, no. 1, pp. 207-215, 2012

[33] M. Elberfeld, and J. Textor, “Efficient algorithms for string-based negative selection,” in Artificial Immune Systems, Springer Berlin Heidelberg, 2009, 109-121. DOI:10.1016/j.tcs.2010.09.022

[34] D. Wang, Y. Xue, and D. Yingfei, “Anomaly Detection Using Neighborhood Negative Selection,” Intelligent Automation & Soft Computing, 17(5), 595-605. DOI: 10.1080/10798587.2011.10643173

[35] T. Yang, H. L. Deng, W. Chen, and Z. Wang, “GF-NSA: A Negative Selection Algorithm Based on Self Grid File,” Applied Mechanics and Materials, 44, 3200-3203. DOI: 10.4028/www.scientific.net/AMM.44-47.3200

[36] Z. Ji, and D. Dasgupta, “V-detector: An efficient negative selection algorithm with “probably adequate” detector coverage,” Information sciences, vol. 179, no. 10, 2009, 1390-1406. DOI: 10.1016/j.ins.2008.12.015

[37] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” In Communications of the ACM, 18(9), 509-517, 1975. DOI: 10.1145/361002.361007

[38] V. Vapnik, “Statistical learning theory,” Vol. 1. New York: Wiley, 1998. DOI: 10.1007/978-1-4757-3264-1

[39] From the lecture notes of CS4850, Section 10, Spring 2010, “Mathematical Foundations for the Information Age,” Cornell University, http://www.cs.cornell.edu/Courses/cs4850/2010sp/Course %20Notes/Chap%206%20Learning-march_9_2010.pdf

[40] V. Vapnik, “The nature of statistical learning theory,” Springer, 2000. DOI:10.1080/00401706.1996.10484565

[41] R. Schapire, Y. Freund, P. Bartlett, and W.S. Lee, “Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods,” in Annals of Statistics, vol. 26, no. 5, pp. 1651-1686, 1998.

[42] L. Breiman, "Bagging predictors." Machine learning, vol. 24, no. 2, pp. 123-140, 1996. DOI: 10.1023/A:1018054314350

[43] L. Breiman, "Random forests," in Machine learning, no. 45, vol. 1, pp. 5-32, 2001. DOI: 10.1023/A:1010933404324

[44] R. Schapire, “The strength of weak learnability,” in Machine Learning, no. 5, vol. 2, pp. 197–227, 1990. DOI: 10.1023/A:1022648800760

[45] Q. Cai, Z. Changshui, and P. Chunyi, "Analysis of classification margin for classification accuracy with applications," Neurocomputing, vol. 72, no. 7, pp. 1960-1968, 2000. DOI: 10.1016/j.neucom.2008.03.015

[46] Y. Freund, R. Schapire, and N. Abe. "A short introduction to boosting," in Journal-Japanese Society for Artificial Intelligence no. 14, pp. 771-780, 1999

[47] T. Cover, and P. Hart. "Nearest neighbor pattern classification," in IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21-27, 1967. DOI: 10.1109/TIT.1967.1053964

[48] L. Devroye, "On the almost everywhere convergence of nonparametric regression function estimates," in The Annals of Statistics, vol. 9, no. 6, pp. 1310-1319, 1981

[49] J. S. Hamaker, and L. Boggess, “Non-euclidean distance measures in AIRS, an artificial immune classification system,” In Congress on evolutionary computation, 2004. DOI: 10.1109/CEC.2004.1330980

[50] T. S. Guzella, T. A. Mota-Santos &W. M. Caminhas, “Artificial immune systems and kernel methods,” In Artificial Immune Systems, pp. 303-315, Springer Berlin Heidelberg, 2008. DOI 10.1007/978-3-540-85072-4_27

[51] A. A. Freitas, and J. Timmis “Revisiting the foundations of artificial immune systems for data mining,” In Evolutionary Computation, IEEE Transactions on, 11(4), 521-540l, 2007. DOI: 10.1109/TEVC.2006.884042