CHAPTER 1

INTRODUCTION

1.1 Introduction

Cloud computing is a paradigm shift following the shift from mainframe to client–server in the early 1980s. Details are abstracted from the users, who no longer have need for expertise in, or control over, the technology infrastructure "in the cloud" that supports them. Cloud computing describes a new supplement, consumption, and delivery model for IT services based on the Internet, and it typically involves over-the-Internet provision of dynamically scalable and often virtualized resources. It is a byproduct and consequence of the ease-of-access to remote computing sites provided by the Internet. It is both a platform and type of application. A cloud computing platform dynamically provisions, configures, reconfigures, and de-provisions servers as needed [2]. Servers in the cloud can be physical machines or virtual machines. Advanced clouds typically include other computing resources such as storage area networks (SANs), network equipment, firewall and other security devices. These cloud applications use large data centers and powerful servers that host Web applications and Web services. Anyone with a suitable Internet connection and a standard browser can access a cloud application. It is Internet-based computing, whereby shared resources, software, and information are provided to computers and other devices on demand, like the electricity grid. This frequently takes the form of web-based tools or applications that users can access and use through a web browser as if it was a program installed locally on their own computer. Clouds often appear as single points of access for all consumers' computing needs [3]. Commercial offerings are generally expected to meet quality of service (QoS) requirements of customers, and typically include SLAs (Service Level Agreement).

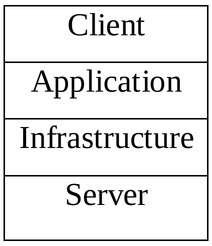


Figure 1.1: A simple generalization of cloud computing system

1.2 Cloud computing characteristics

Cloud computing has five fundamental characteristics as follows:

- **On-demand service:** Cloud customers can demand computing capabilities such as network storage.
- **Broad network access:** The cloud capabilities are available over the network and are accessed by customers using platforms (e.g.: laptop, Smartphone).
- **Resource pooling:** Cloud provider's computing resources are pooled to support multiple users or multitenancy model.
- **Rapid elasticity:** The capabilities can be rapidly and elastically demanded. The capabilities are appeared to be infinitely available to the customers and can be purchased at any time.
- **Measured service:** Cloud system automatically controls and optimizes the resources usage by leveraging metering capability to the specific type of service. (e.g. Bandwidth, storage). Resource usage is controlled, monitored and reported providing transparency for both cloud provider and customer.

1.3 Cloud computing services

Cloud computing services or cloud services are typically categorized into three types namely Software as a Service (SaaS), Platforms as a Service (PaaS) and Infrastructure as a Service (IaaS). These three categories offer different services to cloud customers. Usually, cloud customers can demand on the type of services they require.

1.3.1 Software as a Service (SaaS)

In SaaS, customers are renting complete applications instead of purchasing and installing the applications or software on their computers. SaaS provider hosts the applications and makes the applications available over the network. SaaS applications are multi-tenant applications, which mean that the applications are shared to multiple customers. However the applications are logically unique for each customer. It is the responsibilities of the provider to secure customers information in SaaS. Several examples of SaaS applications are online word processing tools and web content delivery services. Companies that offer SaaS services include Google and Salesforce.com

1.3.2 Platforms as a Service (PaaS)

In PaaS service, cloud provider offers a platform for development environment to the customers to run their applications. The development platform is Application Programming Interface (API) and is configurable remotely. The platform service includes configuration management, deployment platform and development tools. Therefore, customers can run their applications without having specialized administration skills. Further, the customers can build and deploy their web applications without having to install any tools on their computers. Similar to SaaS, PaaS provider is responsible for securing the leased services. PaaS security spans between two software layers:

- Security of PaaS platform itself. For instance: runtime engine that is integrated in PaaS service.
- Security of client applications which are running on PaaS platform. Therefore, the PaaS provider is responsible for securing the platform and the customer's applications. Companies that offer PaaS service include Microsoft Azure and Google App Engine.

1.3.3 Infrastructure as a Service (IaaS)

IaaS service offers virtual machines as well as other abstracted hardware and operating system over the network. By renting IaaS service, the customers can use the latest infrastructure technology and they do not have to concern with updating the technology . Contrast to SaaS and PaaS, customers of IaaS are mainly responsible for securing the leased infrastructure. Companies that offer this service include GoGrid, Flexiscale, and Amazon.

1.4 Cloud Deployment Models

Similar to type of service, cloud may be hosted and deployed in different fashions depending on the use case. Cloud deployment models are as follows:

- **Private cloud**: In private cloud model, the cloud infrastructure is deployed merely for single organization.
- **Community cloud**: In community cloud model, the cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns.
- **Public cloud**: In public cloud model, the cloud infrastructure is made available to public or a large industry group. The cloud is owned by an organization that sells service.
- **Hybrid cloud**: In hybrid model, the cloud infrastructure is a composition of two or more clouds (private, community, or public).

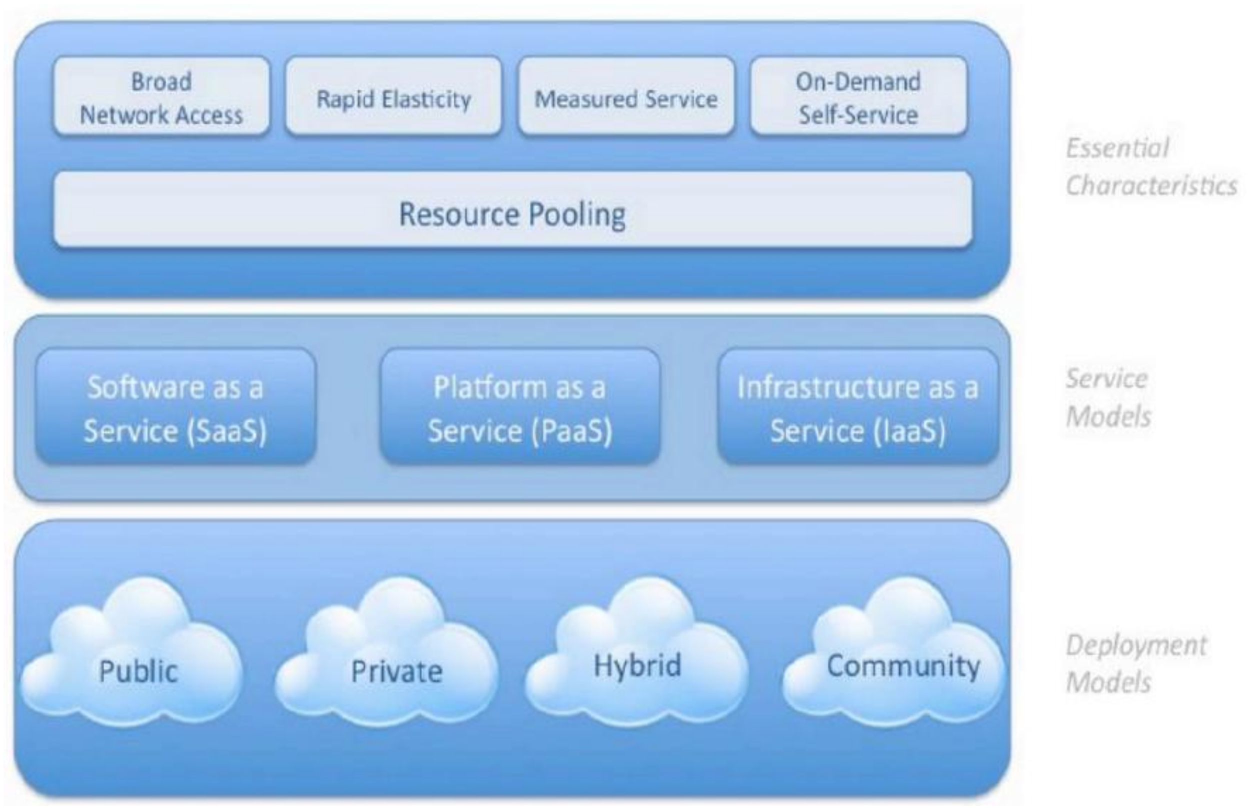


Figure 1.2: Cloud computing overview model (NIST) [4]

1.5 Motivation

Cloud computing is a new technology which is gaining momentum rapidly but there are many security concerns about this technology. Data security, privacy and insider attack are some of the burning issues related to the cloud computing. Maintaining trust of cloud customers to their cloud service is the main hurdle in the rapid expansion of cloud computing. To make cloud computing more secure, SLA with additional security constraints can play a vital role. Using SLA we can make a better user profile of a cloud customer, which will be helpful in detecting an anomaly in the user behavior.

1.6 Objectives

The main Objectives of this work are:

- To create a classification algorithm based on the SLA agreement between users and CSP, which will assign the class to cloud user's logging session.
- To detect the masquerader based on the sum of weight of the *k* temporal activities that is calculated dynamically.
- To find an relation of threshold weight with respect to *k*.

- To analyze and compare the performance Backpropagation algorithm with Support Vector Machine.

1.7 Organization of Thesis

The rest of this thesis is organized as: chapter 2 gives a brief discussion of basic concept related to this work, chapter 3 is a survey of the major existing masquerade detection systems, chapter 4 details the selection of basic input parameters, the implementation of the Backpropagation Algorithm and Support vector Machine, chapter 5 presents the training and testing results, and chapter 6 concludes the thesis, summarizing its achievements and further recommendations.

CHAPTER 2

BACKGROUND AND PROBLEM DEFINITION

2.1 Service Level Agreement

SLA is a written agreement about service levels offered by providers to customers. In the context of this study, SLA is an agreement between cloud providers and cloud customers. Main advantage of SLA is to gain common understanding of various issues including service levels and responsibilities of provider and customer. The stated issues and service levels in SLA depend on negotiation between provider and customer.

According to Chaves et al. SLA defines the "what" and not the "how". It means in regards to information security, SLA states what type of service levels customer should receive. However it does not state how the service levels are achieved. SLA also provides information about responsibilities of both cloud provider and customer towards unexpected events that happen to the service. It is the serious issue to consider as it records a common understanding about services, priorities, responsibilities, guarantees, and warranties between the cloud provider and the costumers. According to SLA information zone [5], a regular SLA usually includes:

Service delivered: It describes the services, how they are delivered and the possible or unexpected disturbance within the time frame or system. This information should be very detailed and accurate so we will get the information about what exactly is going to be delivered.

Performance: Performance of the system is measured by monitoring and measuring the services, which was offered whether it is as per the contract or not.

Problem management: It explained how the unplanned or unexpected incidents can be occurred and how to solve and prevent them from future occurrence of such events.

Customer duties: It explains relationship the customer and provider has and also the responsibilities that the customer has to follow and bear in the service delivery process.

Warrant & remedies: It covers topics such as service quality, third party claims and exclusions.

Security: It is the most critical feature of any SLA where it is defined of which security approaches must be followed and respected.

Disaster recovery: It is usually included in the security section and sometimes also in the problem management area.

Termination: Termination at end of initial term after the contract period expires or if either the customer or provider violates the contract or not satisfied with the performance.

As more and more consumers delegate their tasks to cloud providers, Service Level Agreements (SLA) between consumers and providers emerge as a key aspect. Due to the dynamic nature of the cloud, continuous monitoring on Quality of Service (QoS) attributes is necessary to enforce SLAs [6]. Also numerous other factors such as trust (on the cloud provider) come into consideration, particularly for enterprise customers that may outsource its critical data. This complex nature of the cloud landscape demands SLAs that can ensure QoS and maintain trust of consumers towards their cloud service. In this work a SLA is proposed that contains some additional constraints that will be helpful in detecting masquerader. Such as:

- Allocated User Devices
- Normal Cloud Service Usage Time
- Dummy Weight of events\activities
- Number of events in Temporal Sequence
- Threshold sum of temporal sequence

2.2 Security Concern

Like traditional computing environments, cloud computing brings risks and security concerns to the business that need to be considered appropriately. Such risks and security concerns include challenges in handling privileged user access, ensuring legal and regulatory compliance, ensuring data security, maintaining data recovery, difficulty in investigating illegal activities, and lack of assurance of long-term viability of the cloud provider. Due to these challenges cloud customers therefore need to institute mechanisms to measure and improve security of their information assets operating in the cloud.

2.3 Masquerader

In computer security, a masquerader is an intruder trying to impersonate a legitimate user. A masquerade attack occurs when an illegitimate user tries to impersonate a legitimate user; therefore, the masquerade user gets the privileges from the legitimate user account. The task of detecting masquerade users is not easy since the masquerade user has yield the name and password of a valid user (probably an administrator). However, detecting illegitimate users could be done if information about the behaviour of the impersonate user is taken as a characteristic pattern which is valid only for this user. Early and effective intrusion detection is a critical factor in securing a computer system.

2.4 MAC Address

MAC, Media Access Control, address is a globally unique identifier assigned to network devices, and therefore it is often referred to as hardware or physical address. MAC addresses are 6-byte (48-bits) in length, and are written in MM:MM:MM:SS:SS:SS format [7]. The first 3-bytes are ID number of the manufacturer, which is assigned by an Internet standards body. The second 3-bytes are serial number assigned by the manufacturer.

In this work we are proposing a SLA that includes the specification of devices through which a cloud user accesses his/her cloud service account. MAC address of these devices is used to identify whether a user is logged in with the specified device or not.

2.5 User Behaviour Profiling

Masquerader detection is the keeping track of anomaly in the regular user behaviour. A user profile is a record of user-specific data that define the user's activities. Access control and authentication are not sufficient to prevent potential intrusions from masqueraders which already got the authorization to access system resources by obtaining an authorized user identity illegally. User behaviour profiling can be used for the purpose of classification, future behaviour prediction and masquerader detection. In this work we are creating a user profile of a cloud computing service’s customer based upon the proposed SLA. The user profile will be based upon user’s device, login time and the user activity during the logging session.

2.6 Problem Definition

A user logs into cloud service account from a device that may or may not have been registered in the SLA agreement between that cloud user and the CSP (cloud service provider) .The user's logging time, MAC address is recorded along with temporal sequence activities in a log file.

Let, S = (a_1, $a_{2,}$ a_3, $a_{4,}$ a_5, $a_{6,}$ a_7, a_8............................, a_n) be the sequence of activities of a user during the user session. This user session consists of many temporal sessions of length *k* as shown in figure 2.1. If there are *n* activities in a session then there will be (*n*-k + 1) temporal sessions. Cloud service provider has assigned a dummy weight on each of the activities/events that is agreed on the SLA agreement. Such as:

- Deleting Contacts = 8
- Changing Passwords = 9
- Checking Mails = 5
- Transferring Data =10

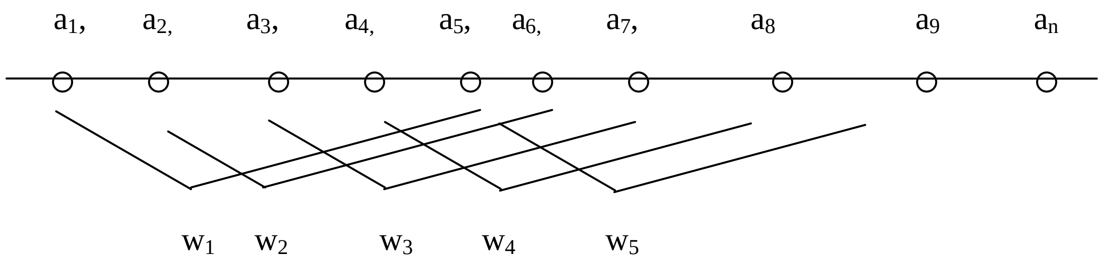


Figure 2.1: User activity graph

The SLA also includes the threshold weight i.e. maximum tolerable weight of a temporal sequence of length *k*, where threshold weight and *k* can vary. Temporal weights of temporal sequences are computed in succession one after another and compared against the threshold weight defined in SLA agreement. Based on the weight calculated in succession, user's logging time and MAC address of the user's logging device; the session is classified as normal or suspicious.

CHAPTER 3

LITERATURE REVIEW

3.1 Literature Review

Cloud Computing is a new paradigm. Many works have been done in this field for intrusion and Masquerader detection. Hisham A.K and Fabrizio Baiardi in [8] Created a Cloud Intrusion Detection Dataset (CIDD) including both knowledge and behaviour based audit data using log analyzer and correlator system. Xijun Cheng and Juanjuan Chen in [9] modeled user interests in cloud for masquerade detection based on the how a user interacts with a computer system. This work is an attempt to detect masquerader on the cloud based on the SLA agreement which is probably first in this field. There has been a lot of research works that dealt the Masquerader Detection in computer and computer network and various methods are proposed.

3.1.1 Information-Theoretic-based Approaches

Schonlau et al. first proposed a rather simple compression-based approach in [10], called the Compression method. It is based on the premise that data from the same user compresses more readily than mixed data from different users. The score of a testing block is defined as the number of additional bytes needed to compress it when appended to the training data. The score of a testing block is defined as the number of additional bytes needed to compress it when appended to the training data. The UNIX command compress was applied, which is based on the popular Lempel-Ziv algorithm [11]. In 2007 Evans et al. proposed a grammar-inference algorithm called MDL compress [12] which uses Minimum Description Length (MDL) principles from the theory of Kolmogorov Complexity and Algorithmic Information Theory to model legitimate user activity and use the resulting grammar to detect masqueraders. This algorithm is reported to produce results very similar to those of Schonlau et al.'s Compression approach [6], though no quantitative results are available to our knowledge.

3.1.2 Time based Inductive machine

TIM (Time –based Inductive machine was originally developed as a general purpose tool with potential applications in many domains. TIM discovers temporal pattern [13] from observations of a temporal process, where the patterns represents highly repetitive activities and can be used for prediction with high accuracy.

3.1.3 Text Mining-based Approaches

Latendresse proposed in 2005 a text-mining approach [14] based on the Sequitur algorithm. The Sequitur algorithm extracts hierarchical structures from a string by constructing a context-free grammar [15]. For each user, a Sequitur grammar is generated to extract the repetitive sequences of commands and their associated frequencies, and each testing block is compared recursively with the legitimate user's profile. This method achieved high detection with low false alarm rates.

3.1.4 HMM-based Approaches

A Hidden Markov Model (HMM) is a statistical model where the modeled system is assumed to be a Markov process with unobserved state. An HMM is defined in as a doubly stochastic process with an underlying stochastic process[16] that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols. A Hidden Markov Model is used to construct user profiles, but the approach is independent of the profile construction method. Each testing block gets a score from every agent and, by means of a voting mechanism and a threshold value, the legitimacy of that block is decided.

3.1.5 Naive Bayesian based Approaches

The Naive Bayesian classifier is one of the most popular classifiers in text classification. They are simple, probabilistic classifiers known for their inherent robustness to noise and their fast learning curve. It is generally used with the "bag of words" model, which profiles documents based on word frequencies, ignoring the sequence information [17]. The Naive Bayesian approach with online profile updating, using the bag of words model, performed remarkably well and achieved one of the best detection results.

3.1.6 SVM-based Approaches

Support Vector Machines (SVMs) are a set of machine learning algorithms used for classification and regression. SVM classifies data by constructing a separating hyperplane in the n-dimensional feature space of training inputs, which maximizes the margin between them [18]. Wang and Stolfo's work in [19]introduced the application of one-class training for masquerader detection.

3.1.7 BackPropagation Based Approach

Ryan used a back propagation neural network NNID (Neural Network Intrusion Detector) to identify users simply by what commands and how often they use, called the ‘print’ of a user [20]. The system administrator runs NNID at the end of each day to see if the user’s sessions match their normal pattern. If not, an investigation can be launched. The NNID model is implemented in a UNIX environment and consists of keeping logs of the commands executed, forming command histograms for each user, and learning the user’s profiles from these histograms

3.2 Classification

Given the example data $\{(x_i, y_i),\ i=1\ldots.n\}$,where the x_i is input vector and the y_i is its associated label or class. Then the classification task is to learn the discriminate function

$$y=f(x),$$

which correctly classify the example data and optimized so that it will make minimal error on the classification of unseen data.

If the label ‘y’ is not discrete as above, then this task is called regression. Based on these examples (x_i, y_i), one is particularly interested to predict the answer for other cases before they are explicitly observed. Hence, learning is not only a question of remembering but also of generalization to unseen cases.

3.3 Multilayer perceptrons

Multilayer perceptrons have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as the error back-propagation algorithm. This algorithm is based on the error correction learning rule. Basically, error back-propagation learning consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, an activity pattern (input vector) is applied to the sensory nodes of the network, and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During the forward pass the synaptic weights of the networks are all fixed. During the backward pass, on the other hand, the synaptic weights are all adjusted in accordance with an error-correction rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is then propagated backward through the network, against the direction of synaptic connections-hence the name "error back-propagation." The synaptic weights are adjusted to make the actual response of the network move closer to the desired response in a statistical sense. The learning process performed with the algorithm is called back-propagation learning.

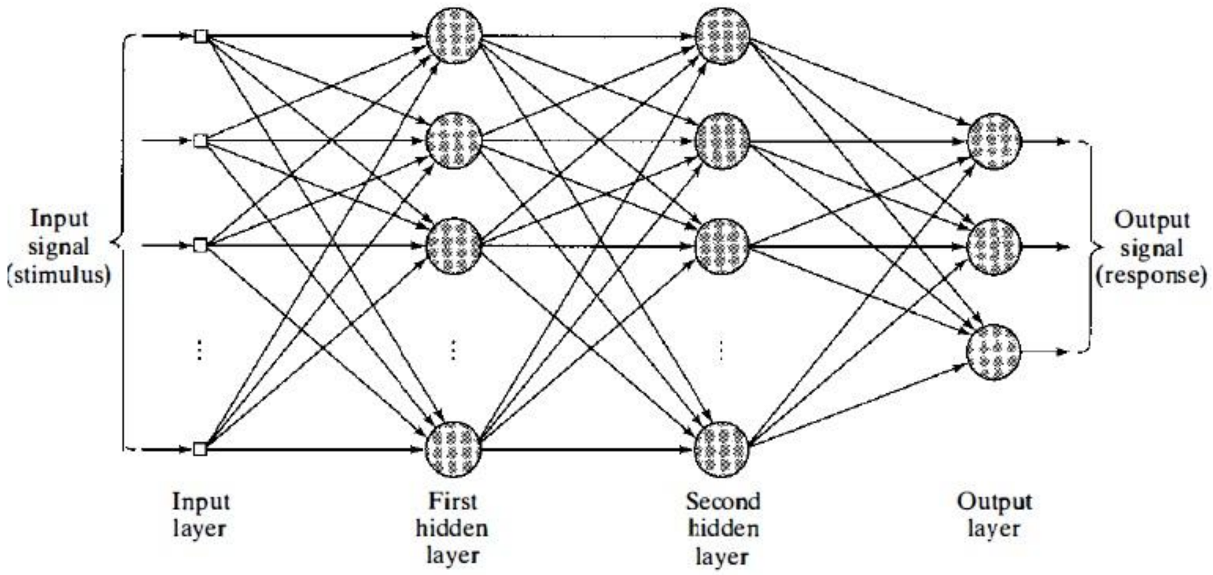


Figure 3.1: A Multilayer Perceptron with two hidden layers [21]

A multilayer perceptron has three distinctive characteristics:

i. The model of each neuron in the network includes a nonlinear activation function. The important point to emphasize here is that the nonlinearity is smooth (i.e., differentiable everywhere). A commonly used form of nonlinearity that satisfies this requirement is a sigmoidal nonlinearity [21]defined by the logistic function:

$$y_j = \frac{1}{1 + \exp(-v_j)} \qquad (3.1)$$

Here vj is the induced local field (i.e., the weighted sum of all synaptic inputs plus the bias) of neuron j, and y_j is the output of the neuron. The presence of nonlinearities is important because otherwise the input-output relation of the network could be reduced to that of a single-layer perception.

ii. The network contains one or more layers of hidden neurons that are not part of the input or output of the network. These hidden neurons enable the network to learn complex tasks by extracting progressively more meaningful features from the input patterns (vectors).

iii. The network exhibits a high degree of connectivity, determined by the synapses of the network. A change in the connectivity of the network requires a change in the population of synaptic connections or their weights.

It is through the combination of these characteristics together with the ability to learn from experience through training that the multilayer perceptron derives it computing power.

3.3.1 BackPropagation Algorithm

A typical Back Propagation Neural Network (BPNN) start as a network of nodes arranged in three layers--the input, hidden, and output layers. The architecture of BPNN is same as MLP and the learning rule applied is BP algorithm. There is no theoretical limit on number of hidden layers but there are just one or two. The input and output layers serve as nodes to buffer input and output for the model, respectively, and the hidden layer serves to provide a means for input relations to be represented in the output.

Before any data has been run through the network, the weights for the nodes are random, which has the effect of making the network much like a newborn's brain--developed but without knowledge. When the connections of the network are going forward the architecture is called Feed Forward neural network.

```
Input:
        D, a data set consisting of the training tuples and their associated target value;
        l, the learning rate;
        network, a multilayer feed forward network
Output: A trained neural network
Method:
        Initialize all weights and biases in network;
        while terminating condition is not satisfied
        {
                for each training sample X in D
                {
                // Propagate the inputs forward:
                for each input layer j
                        $O_j = I_j$; //output of an input unit is its actual input value
                for each hidden or output layer unit j
                {
                        $I_j = \sum w_{ij} O_i + \Theta j$; //compute the net input of unit j with respect to
                        previous input layer, i
                        $O_j = \frac{1}{1+e^{-I_j}}$ ;
                }
                // compute the output of each unit j
                // Backpropagate the errors:
                for each unit j in the output layer
                        $Err_j = Oj(1 - O_j)(T_j - O_j)$; // compute the error
                for each unit j in the hidden layers
                        $Err_j = Oj(1 - O_j) \sum_k Err_k w_{jk}$; // compute the error
                for each weight $w_{ij}$ in network
                {
                        $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
                        $w_{ij} = w_{ij} + \Delta w_{ij}$; // weight update
                }
                for each bias $\Theta j$ in network
                {
                        $\Delta \Theta j = (l) Err_j$ ; // bias increment
                        $\Theta_j = \Theta j + \Delta \Theta_j$; // bias update
                }
}}
```

3.4 Support Vector Machine

This is the supervised machine learning approach that can be used for both classification and regression. In their basic form, SVM construct the hyperplane in input space that correctly separate the example data into two classes. This hyperplane can be used to make the prediction of class for unseen data. The hyperplane exist for the linearly separable data [1]. This can be illustrated with figure 3.2.

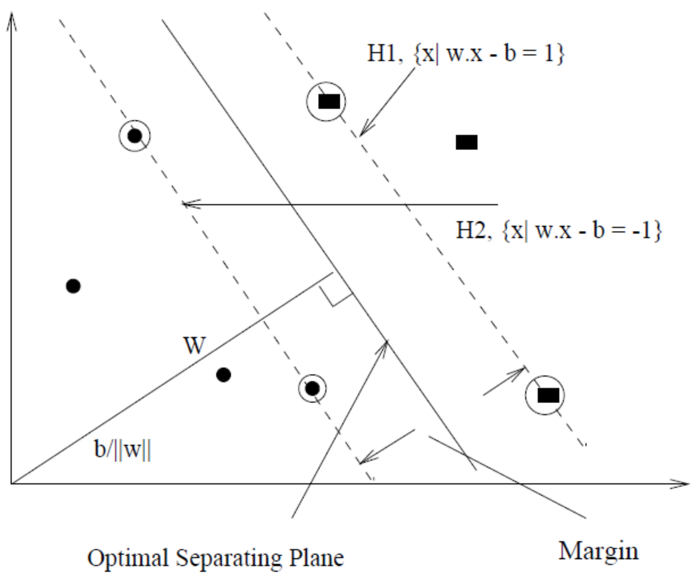


Figure 3.2: Support Vector Machine

The equation for general hyperplane can be written as

$$w.x - b = 0 \tag{3.2}$$

Where x is point vector, w is a weight vector and b is bias. The hyperplane should separate training data{(x_i,y_i), i=1….n and y_i ∈(+1,-1)} in such way that $y_i(w.x_i - b) \geq 1$. The two plane H1 and H2 are supporting hyperplane. We can see that there exist so many hyperplans that can separate the training data correctly but the SVM find one hyperplane that maximize the margin between two supporting hyperplanes. It finds the w and b such that the distance (margin) between H1 and H2 is maximum. This can be formulated as optimization problem as

$$\text{Minimize f} = \frac{|w|^2}{2} \tag{3.3}$$

$$\text{Subject to constraints} \quad y_i(w.x_i - b) \geq 1$$

This can be solved by the variant of quadratic programming technique.

3.4.1 Kernel Trick

To deal with nonlinear separation, the same formulation and techniques as for the linear case are still used. We only transform the input data into another space (usually of a much higher dimension) so that a linear decision boundary can separate positive and negative examples in the transformed space. The transformed space is called the feature space. The original data space is called the input space [1] .

The basic idea is to map the data in the input space X to a feature space F via a nonlinear mapping "ϕ",

$$\phi : X \rightarrow F$$
$$\mathbf{x} \alpha \;\; \phi(\mathbf{x})$$

After the mapping, the original training data set $\{(x_1, y_1), (x_2, y_2), \ldots, (x_r, y_r)\}$ becomes:

$$\{(\phi(x_1), y_1), (\phi(x_2), y_2), \ldots, (\phi(x_r), y_r)\}$$

Then perform linear separation in this feature space. Geometric interpretation is shown in figure 2.2.

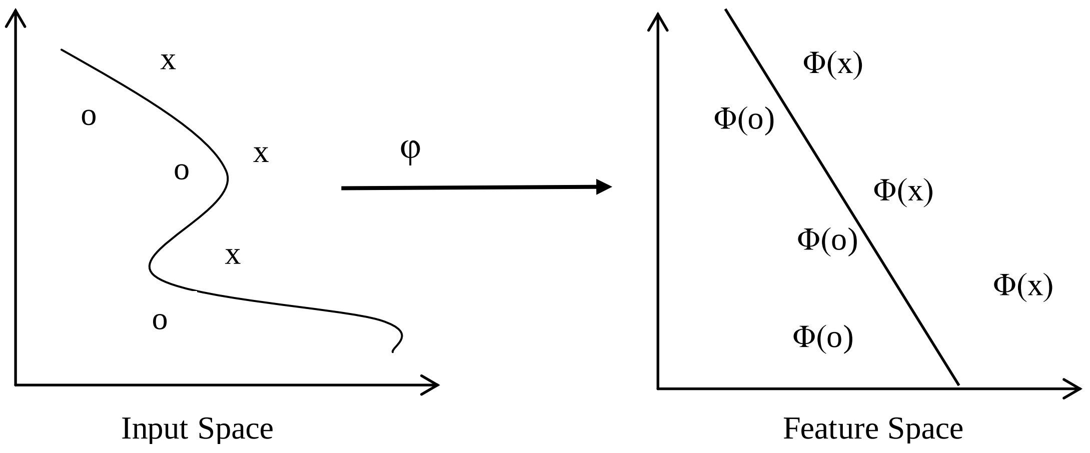


Figure 3.3: Feature Mapping [1]

The potential problem with this explicit data transformation and then applying the linear SVM is that it may suffer from the curse of dimensionality. The number of dimensions in the feature space can be huge with some useful transformations even with reasonable numbers of attributes in the input space. This makes it computationally infeasible to handle. Fortunately, explicit transformation is not needed. In SVM, this is done through the use of kernel functions, denoted by K,

$$K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle$$

For example let us take Polynomial kernel

$$K(x, z) = \langle x \cdot z \rangle^d$$

Let us compute the kernel with degree $d = 2$ in a 2-dimensional space: $x = (x_1, x_2)$ and $z = (z_1, z_2)$.

This shows that the kernel $\langle x \cdot z \rangle^2$ is a dot product in a transformed feature space.

$$\begin{aligned}\langle \mathbf{x}\cdot\mathbf{z}\rangle^2 &= (x_1z_1 + x_2z_2)^2\\ &= x_1^{\,2}z_1^{\,2} + 2x_1z_1x_2z_2 + x_2^{\,2}z_2^{\,2}\\ &= \langle (x_1^{\,2}, x_2^{\,2}, \sqrt{2}x_1x_2)\cdot(z_1^{\,2}, z_2^{\,2}, \sqrt{2}z_1z_2)\rangle\\ &= \langle \phi(\mathbf{x})\cdot\phi(\mathbf{z})\rangle,\end{aligned}$$

3.4.2 Optimization

Many situations arise in machine learning where we would like to optimize the value of some function. It turns out that in the general case, finding the global optimum of a function can be a very difficult task. However, for a special class of optimization problems, known as convex optimization problems, we can efficiently find the global solution in many cases. Here, “efficiently” has a both practical and theoretical connotation: it means that we can solve many real-world problems in a reasonable amount of time, and it means that theoretically we can solve problems in time that depends only polynomially on the problem size.

A convex optimization problem is an optimization problem of the form

minimize f(x)

subject to x ∈ C

where f is a convex function, C is a convex set, and x is the optimization variable. A linearly constrained optimization problem with a quadratic objective function is called a quadratic program (QP). The general quadratic program can be written as

Minimize $f(\mathbf{x}) = \mathbf{cx} + 1/2\ \mathbf{x}^T\mathbf{Q}\ \mathbf{x}$

Subject to $\mathbf{Ax} \le \mathbf{b}$ and $\mathbf{x} \ge \mathbf{0}$

where **c** is an *n*-dimensional row vector describing the coefficients of the linear terms in the objective function, and **Q** is an $(n \times n)$ symmetric matrix describing the coefficients of the quadratic terms. If a constant term exists it is dropped from the model. As in linear programming, the decision variables are denoted by the *n*-dimensional column vector **x**, and the constraints are defined by an $(m\times n)$ **A** matrix and an *m*-dimensional column vector **b** of right-hand-side coefficients. We assume that a feasible solution exists and that the constraint region is bounded.

CHAPTER 4

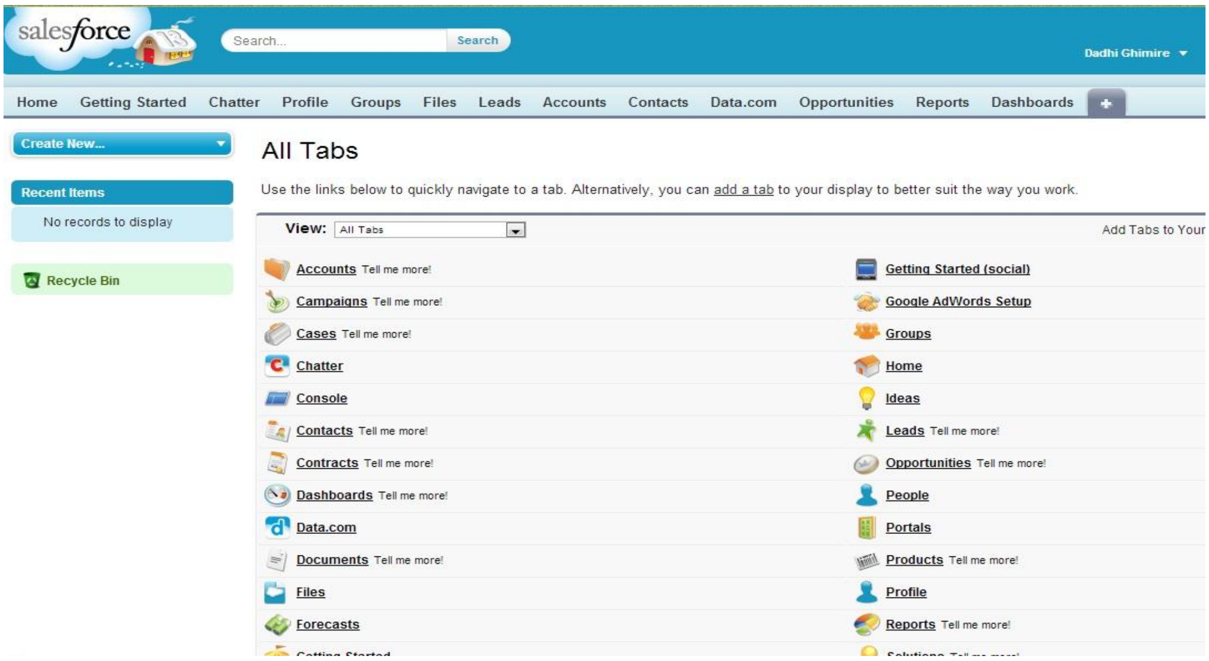
IMPLEMENTATION

4.1 Proposed SLA

In this work we are proposing a SLA with additional constraints than a traditional SLA. These additional constraints are assignment of dummy weight for each activity/event in the cloud service, length of the temporal sequence *k*, threshold weight of the temporal activities, device likely to be used to access cloud service and time frame within which a customer is likely use his cloud service account.

4.1.1 Assignment of Dummy Weight

There are so many activities that a user can do in a cloud service as shown in figure 5.1. In this work we are working with only 16 activities and these activities are categorized into two categories, normal and critical. Each activity in each category is associated with a dummy weight. We have assigned weight 1 to 8 to the activities in the normal category and 9 to 16 to the activities in the critical category.



Use the links below to quickly navigate to a tab. Alternatively, you can add a tab to your display to better suit the way you work.

Figure 4.1: A glimpse of Salesforce.com (Saas)

4.1.2 Length of Temporal Sequence (*k*)

Selecting the length of the temporal sequence is crucial in classifying the user session and to create a user profile. A user session may have more than one temporal sequence. As per our best knowledge, we are first to apply temporal sequence length *k* in a masquerade detection approach. We have chosen *k* = 5; as the minimum length of the temporal sequence and we increased the length of temporal sequence to 6 and 7.

4.1.3 Selection of Threshold weight of Temporal Sequence

Suppose (a_1, a_2, a_3 ….a_n) is a user session with *n* activities in the session. There are (*n*-*k*+1) temporal sequences in a *n* activity session. Choosing the right threshold weight for each temporal sequence is crucial for classifying a user session.

Let (a_1, a_2, a_3 ….a_k) is a temporal sequence and total weight of temporal sequence be W_{tm}

$$W_{tm} = \sum_{i=1}^{k} a_i \qquad (4.1)$$

We have chosen threshold weight, W_{th} as the product of *k* and weight of first activity belonging to the critical category. This threshold weight is less than or equal to the sum of all the combinations involving *k* critical category activities. Comparison between W_{tm} and W_{th} along with other constraints determines whether a session is normal or suspicious.

$$W_{th} = k \text{ x } C_{1st} \qquad (4.2)$$

Where, k is temporal sequence and C_{1st} is the weight of first activity of the critical category.

4.1.4 Device and Time Frame

In this SLA, cloud costumers specify at least two devices, from which they are most likely to log into their cloud service account. MAC address of these devices is recorded in the SLA agreement. We use 1 to simulate a user accessing from the device specified in SLA and 0 for those who are not. Similarly cloud costumers together with CSP agree upon on a time frame, within which frame he/she is most likely to use the cloud service. We use 1 to simulate a user accessing cloud service within the time frame specified in SLA and 0 for those who are not.

4.2 Creation of Training and Testing Data

We have used MSNBC data set to create user profile in this work. MSNBC dataset contains 989818 user sessions, ranging from single activity to 345 activities per session. The dataset is filtered by removing very short and very long sessions. Then sessions are created of uniform length. The resultant dataset is then divided to make user profile of 50 users having 1000 session per user. Sample of MSNBC dataset is given below.

1 1

```
2
3 2 2 4 2 2 2 3 3
5
1
6 7 7 7 6 6 8 8 8 8
6 9 4 4 4 10 3 10 5 10 4 4 4
1 1 1 11 1 1 1
12 12
1 1
8 8 8 8 8 8
6
2
```

4.2.1 BackPropagation propagation Training and Testing Data

Each user profile containing 1000 session is taken and MAC address (0 and 1) and time (0 and 1) is added randomly. The resulting sessions with MAC address and time factor is then fed to the classification algorithm to label the session; 1 for normal session and 0 for suspicious session. The data are then normalized using min-max normalization [22] to keep values between 0 and 1. The sample of training data is shown below.

```
1 0 0.75 0.45 0.75 0.45 0.45 0.75 0.45 0.75 0.45 0.75 0.45 0.75 0.8 0.8 0.8 0.75 0.75 0
1 1 0.15 0.2 0.15 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.15 0.15 0
0 1 0.3 0.3 0.6 0.6 0.6 0.6 0.2 0.2 0.2 0.2 0.6 0.7 0.7 0.6 0.45 0.45 0.45 0
0 0 0.5 0.5 0.5 0.5 0.5 0.75 0.75 0.75 0.75 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0
1 0 0.75 0.75 0.75 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0
1 1 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 1
0 1 0.65 0.15 0.15 0.15 0.2 0.15 0.15 0.35 0.35 0.35 0.35 0.35 0.35 0.15 0.15 0.4 0.4 1
```

4.2.2 SVM Training and Testing Data

Each user profile containing 1000 session is taken and MAC address (0 and 1) and time (0 and 1) is added randomly. The resulting sessions with MAC address and time factor is then fed to the classification algorithm to label the session, 1 for normal and -1 for suspicious session. A feature vector is constructed using the resulting session where there are altogether 18 features. Feature 1 to 16 represents the activities whose value is the number of occurrence of particular activity in a session. Whereas feature 17 and 18 has binary values representing the true or false condition of MAC address and time frame. The training data sample is shown below.

```
-1 13:9 14:6 3:1 17:1 18:0
-1 2:4 7:5 13:6 1:1 17:1 18:1
1 2:3 4:5 7:4 8:4 17:0 18:1
-1 7:1 13:12 14:3 17:0 18:0
-1 1:2 5:2 6:7 15:4 17:1 18:0
1 1:7 2:4 4:1 6:1 11:1 16:1 17:1 18:1
```

4.3 Classification Algorithm based on SLA Agreement

Input:

k, no of temporal activities

$w_{th,}$ threshold weight

n, no of events\activities done by the user done user during the user session.

add, MAC Address of the user's login device

sadd , MAC address of allocated device of particular user in SLA

ltime ,users logging time

stime, time frame of user in SLA

Output: classification of the cloud login session into two classes

Method:

```
    //compute the temporal weights
    For(j=0;j<=n-k+1;j++)
    {
        w=0;
        For(i=0;i<=k-1;i++)
        {
            w = w + $a_w$[j+i];  //events with their respective weight
        }

    // compare time, address and weight to classify session

    if(add = = sadd && ltime = = within(stime) && $w_{tm}$<= $w_{th}$)
        classify as normal session
    else if(add = = sadd && ltime = = within(stime ) && $w_{tm}$ >= $w_{th}$)
        classify as suspicious session
    else if(add != sadd && ltime = = within(stime ) && $w_{tm}$ <= $w_{th}$)
        classify as normal session
    else if(add != sadd && ltime = = within(stime ) && $w_{tm}$ >= $w_{th}$)
        classify as suspicious session
    else if(add = = sadd && ltime != within(stime ) && $w_{tm}$ >= $w_{th}$)
        classify as suspicious session
    else if(add = = sadd && ltime != within(stime ) && $w_{tm}$ <= $w_{th}$)
```

```
            classify as normal session
        else if(add != sadd && ltime != within(stime ) && $w_{tm} <= w_{th}$)
            classify as suspicious session
        else if(add != sadd && ltime != within(stime ) && $w_{tm} >= w_{th}$)
            classify as suspicious session
}
```

4.3 System Architecture for Masquerader Detection in Cloud Based on SLA

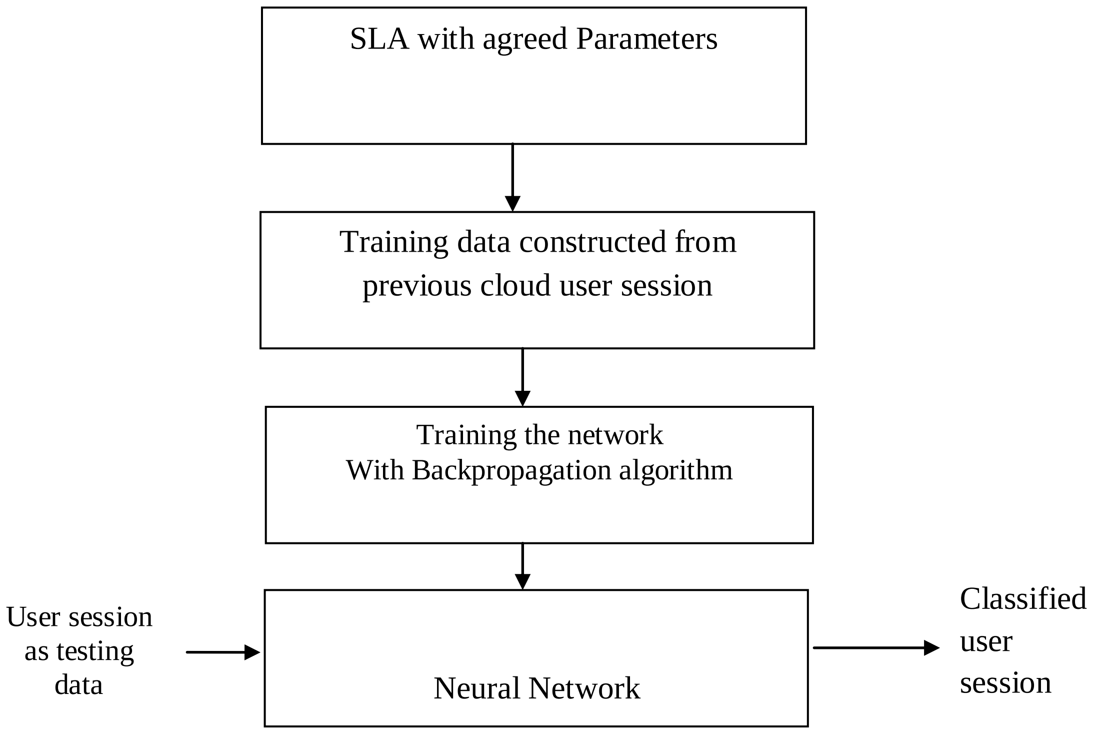

SLA with agreed Parameters
Training data constructed from previous cloud user session
Training the network With Backpropagation algorithm
User session as testing data
Neural Network
Classified user session

Figure 4.2: System Architecture for Masquerader Detection Based on SLA

4.4 ANN Learning: Back Propagation Algorithm

The back propagation algorithm cycles through two distinct passes, a forward pass followed by a backward pass through the layers of the network. The algorithm alternates between these passes several times as it scans the training data.

4.4.1 Forward Pass: Computation of outputs of all the neurons in the network

The algorithm starts with the first hidden layer using as input values the independent variables of a case from the training data set. The neuron outputs are computed for all neurons in the first hidden layer by performing the relevant sum and activation function evaluations. These outputs are the inputs for neurons in the second hidden layer. Again the relevant sum and activation function calculations are performed to compute the outputs of second layer neurons.

4.4.2 Backward pass: Propagation of error and adjustment of weights

This phase begins with the computation of error at each neuron in the output layer. A popular error function is the squared difference between O_k the output of node k and Y_k the target value for that node. The target value is just 1 for the output node corresponding to normal session and 0 for the suspicious session. The new value of the weight w_{jk} of the connection from node j to node k is given by:

$$w_{newjk} = w_{oldjk} + (n)Err_j. \qquad (4.3)$$

Here *n* is an important tuning parameter that is chosen by trial and error by repeated runs on the training data. Typical values for *n* are in the range 0.1 to 0.9 [23]. The backward propagation of weight adjustments along these lines continues until we reach the input layer. At this time we have a new set of weights on which we can make a new forward pass when presented with a training data observation

4.5 Considerations for choosing Network Structure

There are many ways that feed forward neural networks can be constructed. It depends upon the number of neurons in the input, hidden and output layers.

4.5.1 The Input Layer:

The input layer to the neural network is the channel through which the external environment presents a pattern to the neural network. Once a pattern is presented to the input layer of the neural network the output layer will produce another pattern. The input layer represents the condition for which we are training the neural network for. Here we have used a input layer consisting of 18 neurons.

4.5.2 The Output Layer:

The output layer of the neural network actually presents a pattern to the external environment. Whatever patter is presented by the output layer can be directly traced back to the input layer. If the neural network is to be used to classify items into groups, then it is often preferable to have one output neurons for each group that the item is to be assigned into. If the neural network is to perform noise reduction on a signal then it is likely that the number of input neurons will match the number of output neurons [23]. Here we have one output layer whose value is 0 or 1; where 1 signifies the normal session whereas 0 signifies suspicious session.

4.5.3 The Number of Hidden Layers:

Neural networks with two hidden layers can represent functions with any kind of shape [23]. There is currently no theoretical reason to use neural networks with any more than two hidden layers. Further for many practical problems there's no reason to use any more than one hidden layer [23]. To calculate the weight changes in the hidden layer the error in the output layer is back-propagated to these layers according to the connecting weights. This process is repeated for each sample in the training set.

4.6 Parameters to be considered to build BP algorithm

Initial weight range(r): It is the range usually between [-r, r], weights are initialized between these range. These weights are used in calculating bias.

Number of Nodes in Hidden Layer: Selecting the number of hidden layers and the number of nodes is largely a matter of trial and error [23]. Through hit and trial method, we have used 2 hidden layers in this work, where first hidden layer consist of 6 neurons and second hidden layer consists of 5 neurons

Number of Epochs: An epoch is one sweep through all the records in the training set. Increasing this number improves the accuracy of the model, but at the cost of time, and decreasing this number decreases the accuracy, but takes less time [23] .

Step size (Learning rate) for gradient descent: This is the multiplying factor for the error correction during back propagation; it is roughly equivalent to the learning rate for the neural network. A low value produces slow but steady learning; a high value produces rapid but erratic learning. Values for the step size typically range from 0.1 to 0.9 [23].

Error tolerance: The error in a particular iteration is back propagated only if it is greater than the error tolerance. Typically error tolerance is a small value in the range 0 to 1 [23].

Hidden layer sigmoid: The output of every hidden node passes through a sigmoid function. Standard sigmoid function is logistic; the range is between 0 and 1 [23].

Output layer sigmoid: Standard sigmoid function is logistic; the range is between 0 and 1.

Critical error: The desired error (MSE) for stopping network training. If the actual error is equal to or less than this error, the training will be terminated.

4.7 PHP

In this work we have used PHP to implement classification algorithm and BackPropagation algorithm. PHP is an open source server-side scripting language used in Web development to produce dynamic Web pages. It is one of the first developed server-side scripting languages to be embedded into an HTML source document rather than calling an external file to process data. The code is interpreted by a Web server with a PHP processor module which generates the resulting Web page. It has also evolved to include a command-line interface capability and can be used in standalone graphical applications.

4.7 SVM Light Tool

In this work SVM Light tool is used to implement SVM. SVM Light is a C program by Thorsten Joachim's that implements a support vector machine. SVM Light is used for the problem of pattern recognition, for the problem of classification, for the problem of regression and for the problem of ranking the function. It includes two efficient estimation methods for both error rate and precision/recall. One is svm_learn for training and other is svm_classification.

4.8 SVM Algorithm for Training

INPUT: A user profile file containing 1000 session

OUTPUT: Learn a model (SVM model)

1. Read a user session
2. Construct support vector for a session.
3. Do step 1 and step 2 for all session.
4. Use SVM light (SVMlearn.exe) to learn the model.
5. Stop.

4.9 SVM Algorithm for Testing

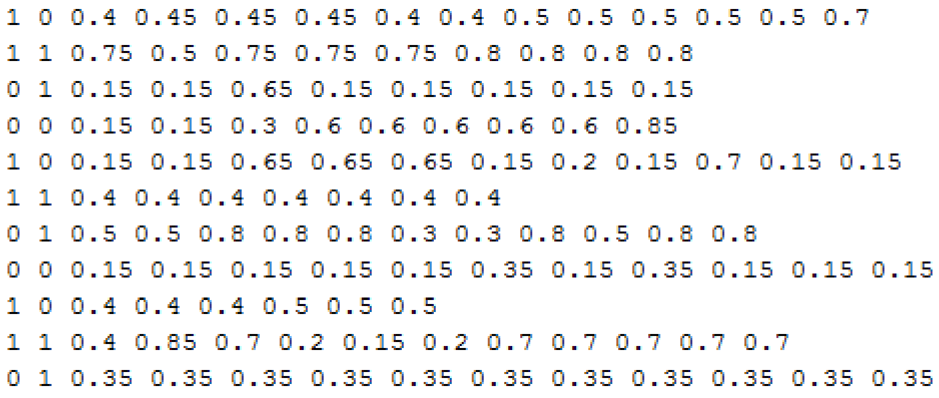
```
1. Read a user session
2. Construct vector for a session.
3. Use SVM light (SVM classify.exe) to classify the session.
4. Stop.
```

CHAPTER 5

TESTING AND ANALYSIS

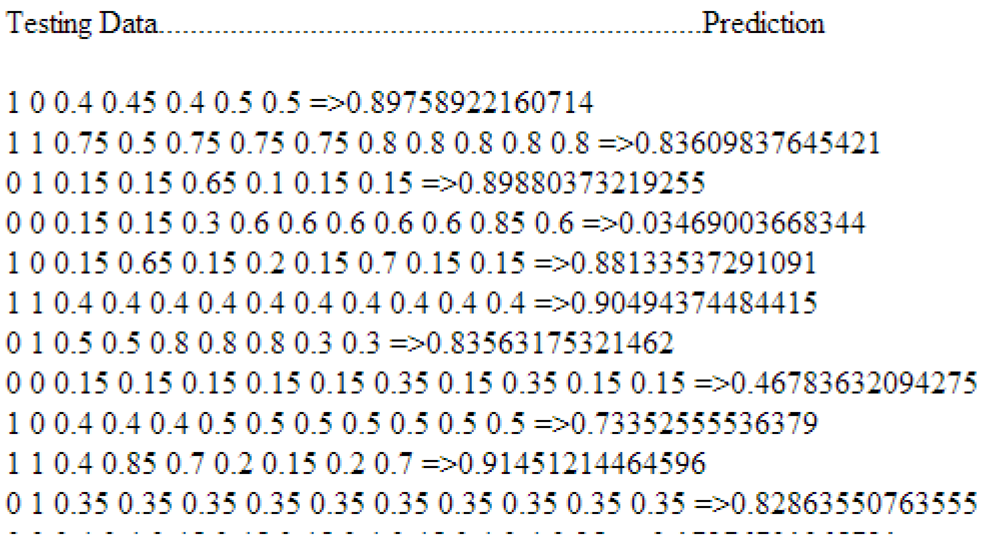
5.1 Backpropagation Algorithm Training and Testing

We prepared testing data containing 100 sessions using same procedures as used in preparing training data. Altogether 50 test data were used; one for each user. Test data contained sessions of various length as shown in the figure 5.1.



```
1 0 0.4 0.45 0.45 0.45 0.4 0.4 0.5 0.5 0.5 0.5 0.5 0.7
1 1 0.75 0.5 0.75 0.75 0.75 0.8 0.8 0.8 0.8
0 1 0.15 0.15 0.65 0.15 0.15 0.15 0.15 0.15
0 0 0.15 0.15 0.3 0.6 0.6 0.6 0.6 0.6 0.85
1 0 0.15 0.15 0.65 0.65 0.65 0.15 0.2 0.15 0.7 0.15 0.15
1 1 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0 1 0.5 0.5 0.8 0.8 0.8 0.3 0.3 0.8 0.5 0.8 0.8
0 0 0.15 0.15 0.15 0.15 0.15 0.35 0.15 0.35 0.15 0.15 0.15
1 0 0.4 0.4 0.4 0.5 0.5 0.5
1 1 0.4 0.85 0.7 0.2 0.15 0.2 0.7 0.7 0.7 0.7 0.7
0 1 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35
```

Figure 5.1: Sample of test data for BP Algorithm



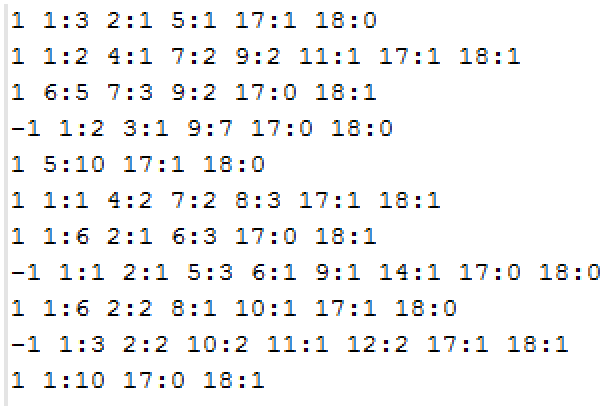
```
Testing Data..........................................................................Prediction

1 0 0.4 0.45 0.4 0.5 0.5 =>0.89758922160714
1 1 0.75 0.5 0.75 0.75 0.75 0.8 0.8 0.8 0.8 0.8 =>0.83609837645421
0 1 0.15 0.15 0.65 0.1 0.15 0.15 =>0.89880373219255
0 0 0.15 0.15 0.3 0.6 0.6 0.6 0.6 0.6 0.85 0.6 =>0.03469003668344
1 0 0.15 0.65 0.15 0.2 0.15 0.7 0.15 0.15 =>0.88133537291091
1 1 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 =>0.90494374484415
0 1 0.5 0.5 0.8 0.8 0.8 0.3 0.3 =>0.83563175321462
0 0 0.15 0.15 0.15 0.15 0.15 0.35 0.15 0.35 0.15 0.15 =>0.46783632094275
1 0 0.4 0.4 0.4 0.5 0.5 0.5 0.5 0.5 0.5 0.5 =>0.73352555536379
1 1 0.4 0.85 0.7 0.2 0.15 0.2 0.7 =>0.91451214464596
0 1 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 0.35 =>0.82863550763555
```

Figure 5.2: Prediction made by BPNN on testing data

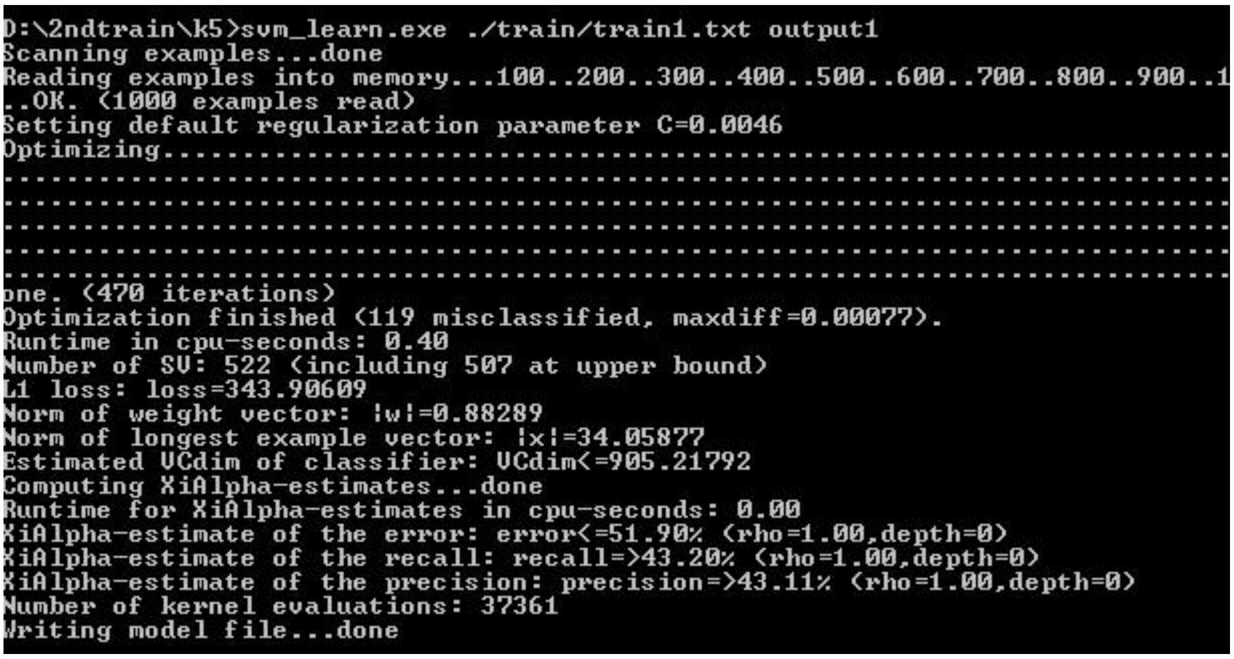
5.2 SVM Training and Testing

We prepared testing data containing 100 sessions using same procedures as used in preparing training data. Altogether 50 test data were used; one for each user. Test data contained sessions of various length as shown in the figure 5.2.



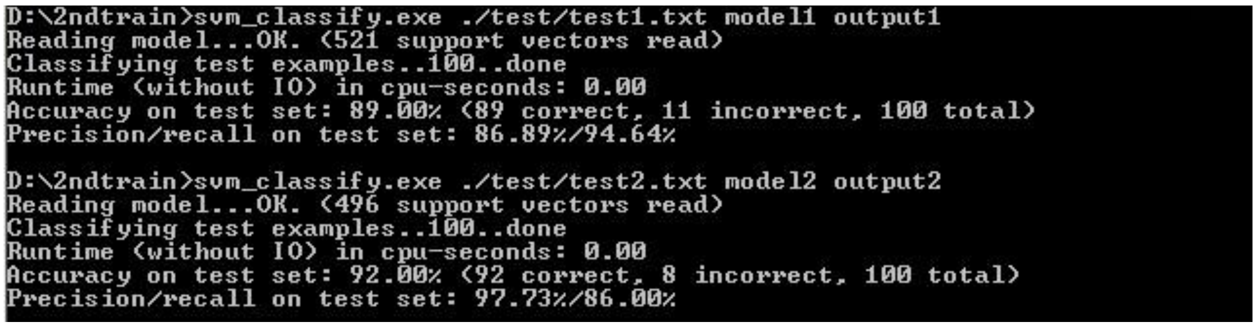
```
1 1:3 2:1 5:1 17:1 18:0
1 1:2 4:1 7:2 9:2 11:1 17:1 18:1
1 6:5 7:3 9:2 17:0 18:1
-1 1:2 3:1 9:7 17:0 18:0
1 5:10 17:1 18:0
1 1:1 4:2 7:2 8:3 17:1 18:1
1 1:6 2:1 6:3 17:0 18:1
-1 1:1 2:1 5:3 6:1 9:1 14:1 17:0 18:0
1 1:6 2:2 8:1 10:1 17:1 18:0
-1 1:3 2:2 10:2 11:1 12:2 17:1 18:1
1 1:10 17:0 18:1
```

Figure 5.3: Sample of testing data for SVM



```
D:\2ndtrain\k5>svm_learn.exe ./train/train1.txt output1
Scanning examples...done
Reading examples into memory...100..200..300..400..500..600..700..800..900..1
..OK. (1000 examples read)
Setting default regularization parameter C=0.0046
Optimizing.............................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
one. (470 iterations)
Optimization finished (119 misclassified, maxdiff=0.00077).
Runtime in cpu-seconds: 0.40
Number of SV: 522 (including 507 at upper bound)
L1 loss: loss=343.90609
Norm of weight vector: |w|=0.88289
Norm of longest example vector: |x|=34.05877
Estimated VCdim of classifier: VCdim<=905.21792
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=51.90% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>43.20% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>43.11% (rho=1.00,depth=0)
Number of kernel evaluations: 37361
Writing model file...done
```

Figure 5.4: SVM learning with Training data



```
D:\2ndtrain>svm_classify.exe ./test/test1.txt model1 output1
Reading model...OK. (521 support vectors read)
Classifying test examples..100..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 89.00% (89 correct, 11 incorrect, 100 total)
Precision/recall on test set: 86.89%/94.64%

D:\2ndtrain>svm_classify.exe ./test/test2.txt model2 output2
Reading model...OK. (496 support vectors read)
Classifying test examples..100..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 92.00% (92 correct, 8 incorrect, 100 total)
Precision/recall on test set: 97.73%/86.00%
```

Figure 5.5: Classifying test data with SVM

5.3 Evaluation Metrics

False Positive Rate: - False positive rate measures the proportion of legitimate session that gets misclassified as suspicious.

$$\text{False Positive Rate} = \frac{Number\ of\ normal\ session\ classified\ as\ suspicious}{Total\ number\ of\ testing\ sessions} \times 100\ \%$$

Detection Rate: - Detection Rate measures the proportion of all sessions that were correctly classified.

$$\text{Detection Rate} = \frac{Total\ number\ of\ correctly\ predicted\ sessions}{Total\ number\ of\ testing\ sessions} \times 100\ \%$$

5.4 Experimental Results

User	BackPropagation Algorithm		SVM	
	Detection rate (%)	False Positive Rate (%)	Detection Positive Rate (%)	False Alarm Rate (%)
User1	80	0	89	8
User2	83	0	92	1
User3	87	1	89	2
User4	90	0	89	3
User5	87	2	87	1
User6	90	7	87	2
User7	90	6	88	2
User8	92	6	89	2
User9	83	5	81	4
User10	82	0	88	5
User11	85	5	92	1
User12	83	0	90	2
User13	86	3	82	2
User14	84	8	91	3
User15	84	0	91	4
User16	89	0	88	2
User17	83	0	89	3
User18	84	0	85	4
User19	84	0	87	0
User20	84	4	82	5
User21	89	0	85	4
User22	82	0	96	4
User23	88	0	93	1
User24	90	2	87	4
User25	86	6	90	2
User26	88	5	92	1
User27	90	6	84	1
User28	91	2	85	2

User29	85	0	93	2
User30	87	1	84	3
User31	85	6	82	3
User32	90	2	87	2
User33	81	0	85	5
User34	86	1	83	3
User35	84	3	91	1
User36	81	0	82	3
User37	85	5	91	3
User38	88	4	82	2
User39	88	0	91	1
User40	87	0	96	3
User41	84	0	85	2
User42	83	3	86	6
User43	87	6	91	4
User44	88	2	86	5
User45	86	0	87	2
User46	82	0	80	6
User47	88	0	88	3
User48	86	0	92	4
User49	91	0	88	1
User50	86	0	92	1
Average	86.04	2.02	87.8	2.8

Table 5.1: Detection rate and False alarm rate of each user (*k*=5)

We increased the value of *k* to 6 and 7and observed the detection rate and false alarm rate, which is presented below in Table 5.2.

k	BackPropagation		SVM	
	Detection Rate (%)	False Positive Rate (%)	Detection Rate (%)	False Positive Rate (%)
5	86.04	2.02	87.8	2.8
6	87.34	0.9	88.96	2.83
7	89.42	2.28	90.06	2.94

Table 5.2: Detection rate and False alarm rate of BP and SVM

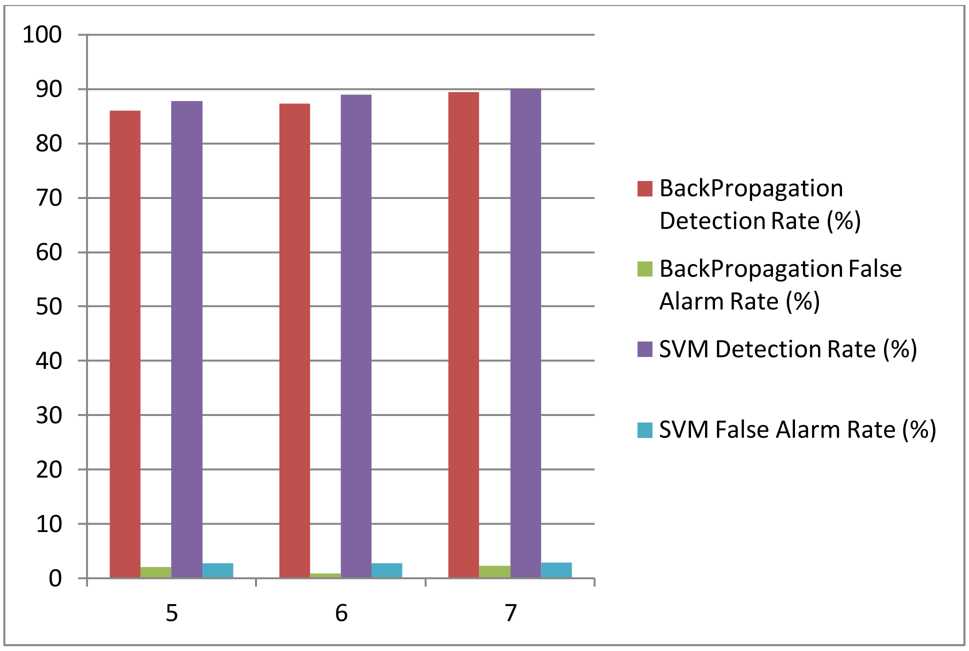


Figure 5.6: Column chart showing performance of SVM and BP Algorithm

CHAPTER 6

CONCLUSION AND RECOMMENDATION

6.1 Conclusion

With Cloud Computing becoming a popular term on the Information Technology (IT) market, security and accountability has become important issues to highlight. In our research we focused on security risks with Cloud Computing and the associated services. The main concern of this work was to assess the risk of cloud computing and to implement a new methodology in detecting masqueraders involving SLA agreement between cloud computing users and CSPs. This work also proposed a new SLA by mutual consent of CSP and cloud costumers, which will be helpful in detecting masquerades, insider attack. We also compared the efficiency of BackPropagation Algorithm with SVM; we found out that SVM has a better detection rate with a higher false alarm rate compared to the BackPropagation Algorithm.

6.2 Future Work

Masquerade detection systems are not cent percent accurate. The classification accuracy of Backpropagation and SVM in this research work can be further improved with better estimation of threshold weight and length of temporal sequence. This research can be extended further with more number of activities in the real world implementation. Furthermore the detection rate of SVM can be increased by constructing feature vector that represents sequence based information more precisely as in string Kernel.

References

[1] J. S. T. N. Christianini, "An Introduction to Support Vector Machines and Other Kernel-based Learning Methods," *Cambridge University press*, 2002.

[2] R. Kaur, "Cloud computing," vol. 2, 2011.

[3] A. G. Christos A. Yfoulis, "Honoring SLAs on cloud computing services: a control perspective," 2009.

[4] "The NIST definition of Cloud," *Special publication by U.S Department of Commerce*, 2011.

[5] *SLA information zone*. Available: http://www.sla-zone.co.uk/

[6] A. R. Pankesh Patel, Amit Sheth, "Service Level Agreement in Cloud Computing," 2010.

[7] *MAC Address*. Available: http://en.wikipedia.org/wiki/MAC_address

[8] F. B. Hisham A.Kholidy, "CIDD: A Cloud Intrusion Detection Dataset For Cloud Computing and Masquerede Attacks," *IEEE:NInth International Conference Of Information Technology-New Generations*, 2012.

[9] X. C. J. Chen, "Modeling User Interests Based on Cloud Model for Masquerade Detection," *IEEE*, 2009.

[10] M. Schonalu, Dumouchel, W.,Karr, "Computer Intrusion:Detectiong masqueraders," pp. 58-74, 2001.

[11] T. A. Welch, "A Technique for high performance data compression. IEEE Computer 17(6)," pp. 8-19, 1984.

[12] S. E. Evans, E. Markham, S. Impson, J. Laczo, "Mdlcompress for intrusion detection: Signature inference and masquerade attack.," *Military Communications Conference*, 2007.

[13] K. C. Henry S.Teng, Stephen C-Y Lu, "Adaptive Real-Time Anamoly Detection Using Inductively Genereated sequential Patterns " 1990.

[14] M. Latendresse, "Masquerade Detection via Customized Grammars," 2005.

[15] I. H. W. Craig G. Nevill-Manning, "Identifying Hierarchical Structure in Sequences:A linear-time algorithm," 1997.

[16] S. B. Bockler, Bateman, "An introduction to hidden markov models.Current protocols in bioinformatics," 2007.

[17] K. N. Andrew McCallum, "A Comparison of Event Models for Naive Bayes Text Classication," 1998.

[18] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition.," pp. 121-167, 1998.

[19] K. W. S. J. Stolfo, "One-Class Training for Masquerade Detection," 2003.

[20] M.-J. L. Jake Ryan, Risto Miikkulainen, "Intrusion Detection with Neural Networks," 1997.

[21] S. Haykin, *NEURAL NETWORKS A Comprehensive Foundation:Second Edition*, pp 201-222.

[22] D. A. S. T.Jayalakshmi, "Statistical Normalization and Back Propagation for Classification," 2011.

[23] Y. P. K. Amit Ganatra, Gaurang Panchal, Chintan Gajjar, "Initial Classification Through Back Propagation In a Neural NetworkFollowing Optimization Through GA to evaluate fitness of an algorithm," 2011.

APPENDIX A

Activities/Events and their respective weight

1. Normal Activities

a.	Checking Notifications	1
b.	Creating Documents	2
c.	Editing Dashboard	3
d.	Updating Products	4
e.	Updating Groups	5
f.	Creating Leads	6
g.	Using Terminal	7
h.	Writing Reports	8

2. Critical Activities

a.	Checking Email	9
b.	Deleting Contacts	10
c.	Editing profiles	11
d.	Editing Files	12
e.	Transferring Files	13
f.	Updating Contracts	14
g.	Database Update	15
h.	Checking Accounts	16

APPENDIX B

Implementation of Classification Algorithm in PHP

```
<?php
$infile = "./traindata/backpro/macntime/k9/td$c.txt";
$myfile = "./traindata/backpro/classified/k9/tr$c.txt";
$handle=fopen($infile,"r");
$fhandle=fopen($myfile,"w");
while(!feof($handle))
{
$line=fgets($handle);
$delet=trim($line);
$arr=(explode(' ',$delet));
$num=count($arr);
print("Number of array elements is:$num");
print ("</br>");
$var=weightcal($arr);
print "</br>";
fwrite($fhandle,$delet.$var);
fwrite($fhandle,"\n");
}
}
function weightcal($arr)
{
$th=45;
$k=5;
$j=0;
$num=count($arr);
```

```
$wt=array();
do
{
            $w=0;
            $l=0;
            $m=0;
            do
            {
            $w=$w+$arr[$j+$l];
            $l++;
            }while($l<=$k-1);
            array_push($wt,$w);
            $j++;
}while($j<($num-$k+1));
$high=max($wt);
//for($a=0;$a<14;$a++)
//{
            if($high>=$th && $arr[0]==1 && $arr[1]==1)
    return " 0";
            else if($high<$th && $arr[0]==1 && $arr[1]==1)
    return " 1";
            else if($high>=$th && $arr[0]==0 && $arr[1]==1)
    return " 0";
            else if($high<$th && $arr[0]==0 && $arr[1]==1)
    return " 1";
            else if($high>=$th && $arr[0]==1 && $arr[1]==0)
    return " 0";
```

```
          else if($high<$th && $arr[0]==1 && $arr[1]==0)
   return " 1";
          else if($high>=$th && $arr[0]==0 && $arr[1]==0)
   return " 0";
          else if($high<$th && $arr[0]==0 && $arr[1]==0)
   return " 0";
          else
          return " 0";
}
?>
```

Implementation of BackPropagation Algorithm

```
<?php
error_reporting(E_STRICT);
define("_RAND_MAX",32767);
class BackPropagation
{
/Output of each neuron
public $output=null;
//delta error value for each neuron
public $delta=null;
//Array of weights for each neuron
public $weight=null;
//Num of layers in the net, including input layer
public $numLayers=null;
//Array num elments containing size for each layer
public $layersSize=null;
//Learning rate
```

```
public $beta=null;
// Momentum
public $alpha=null;
// Storage for weight-change made in previous epoch
public $prevDwt=null;
// Data
public $data=null;
// Test Data
public $testData=null;
// N lines of Data
public $NumPattern=null;
// N columns in Data
public $NumInput=null;
public function __construct($numLayers,$layersSize,$beta,$alpha)
{
        $this->alpha=$alpha;
        $this->beta=$beta;
        // Set no of layers and their sizes
        $this->numLayers=$numLayers;
        $this->layersSize=$layersSize;
        // Seed and assign random weights
        for($i=1;$i<$this->numLayers;$i++)
        {
                for($j=0;$j<$this->layersSize[$i];$j++)
                {
                        for($k=0;$k<$this->layersSize[$i-1]+1;$k++)
                        {
                                $this->weight[$i][$j][$k]=$this->rando();
                        }
                        // bias in the last neuron
                        $this->weight[$i][$j][$this->layersSize[$i-1]]=-1;
                }
        }
        // initialize previous weights to 0 for first iteration
        for($i=1;$i<$this->numLayers;$i++)
        {
                for($j=0;$j<$this->layersSize[$i];$j++)
                {
                        for($k=0;$k<$this->layersSize[$i-1]+1;$k++)
                        {
                                $this->prevDwt[$i][$j][$k]=(double)0.0;
                        }
                }
        }
}
public function rando()
{
        return (double)(rand())/(_RAND_MAX/2) - 1;
}
// sigmoid function
public function sigmoid($inputSource)
{
        return (double)(1.0 / (1.0 + exp(-$inputSource)));
```

```
}
// mean square error
public function mse($target)
{       $mse=0;
        for($i=0;$i<$this->layersSize[$this->numLayers-1];$i++)
        {
                $mse+=($target-$this->output[$this->numLayers-1][$i])*($target-$this
                        ->output[$this->numLayers-1][$i]);
        }
        return $mse/2;
}
// returns i'th outputput of the net
public function Out($i)
{
        return $this->output[$this->numLayers-1][$i];
}
// This function takes the input to the net and finds the output of each neuron
public function ffwd($inputSource)
{
        $sum=0.0;
        // assign content to input layer
        for($i=0;$i<$this->layersSize[0];$i++)
        {
                $this->output[0][$i]=$inputSource[$i];
        }
        // assign output (activation) value to each neuron using sigmoid function
        for($i=1;$i<$this->numLayers;$i++)
```

```
          {
                    for($j=0;$j<$this->layersSize[$i];$j++)


                    {
                              $sum=0.0;
                              for($k=0;$k<$this->layersSize[$i-1];$k++)

                              {
                      $sum+=$this->output[$i-1][$k]*$this->weight[$i][$j][$k];
                              }
                              // Apply bias
                              $sum+=$this->weight[$i][$j][$this->layersSize[$i-1]];
                              // Apply sigmoid function
                              $this->output[$i][$j]=$this->sigmoid($sum);
                    }
          }
}
// Backpropagate errors from outputput layer back till the first hidden layer
public function bpgt($inputSource,$target)
{
          //Update the output values for each neuron
          $this->ffwd($inputSource);
          /// FIND DELTA FOR OUPUT LAYER (Last Layer)
          for($i=0;$i<$this->layersSize[$this->numLayers-1];$i++)
          {
          $this->delta[$this->numLayers-1][$i]=$this->output[$this->numLayers-1][$i]*(1-
$this     ->output[$this->numLayers-1][$i])*($target-$this->output[$this->numLayers-1][$i]);
```

```
	}
	//FIND DELTA FOR HIDDEN LAYERS
	for($i=$this->numLayers-2;$i>0;$i--)
	{
		for($j=0;$j<$this->layersSize[$i];$j++)
		{
			$sum=0.0;
			for($k=0;$k<$this->layersSize[$i+1];$k++)
			{
				$sum+=$this->delta[$i+1][$k]*$this->weight[$i+1][$k][$j];
			}
			$this->delta[$i][$j]=$this->output[$i][$j]*(1-$this-
>output[$i][$j])*$sum;
		}
	}
	//MOMENTUM (Alpha)
	for($i=1;$i<$this->numLayers;$i++)
	{
		for($j=0;$j<$this->layersSize[$i];$j++)
		{
			for($k=0;$k<$this->layersSize[$i-1];$k++)
			{
				$this->weight[$i][$j][$k]+=$this->alpha*$this
					->prevDwt[$i][$j][$k];
			}
			$this->weight[$i][$j][$this->layersSize[$i-1]]+=$this->alpha*$this
				->prevDwt[$i][$j][$this->layersSize[$i-1]];
		}
```

```
        }
        //ADJUST WEIGHTS (Using Steepest Descent)
        for($i=1;$i<$this->numLayers;$i++)
        {
                for($j=0;$j<$this->layersSize[$i];$j++)
                {
                        for($k=0;$k<$this->layersSize[$i-1];$k++)
                        {
                                $this->prevDwt[$i][$j][$k]=$this->beta*$this-
>delta[$i][$j]*$this                                    ->output[$i-1][$k];
                                $this->weight[$i][$j][$k]+=$this->prevDwt[$i][$j][$k];
                        }
                        /* --- Apply the corrections */
                        $this->prevDwt[$i][$j][$this->layersSize[$i-1]]=$this->beta*$this
                                ->delta[$i][$j];
                        $this->weight[$i][$j][$this->layersSize[$i-1]]+=$this
                                ->prevDwt[$i][$j][$this->layersSize[$i-1]];
                }
        }
}
public function Run($data,$testData)
{
        /* --- Threshhold - thresh (value of target mse, training stops once it is achieved) */
        $Thresh =  0.0001;
        $numEpoch = 200000;
        $MSE=0.0;
        $NumPattern=count($data);   // Lines
        $NumInput=count($data[0]); // Columns
        $logfile="./result/k5/log21.txt";

        $lhandle=fopen($logfile,"w");

        /*Start training: looping through epochs and exit when MSE error < Threshold */

        echo  "Network Training Started....";

        echo "</br>";

        for($e=0;$e<$numEpoch;$e++)

        {

                /* -- Backpropagate */

                $this->bpgt($data[$e%$NumPattern],$data[$e%$NumPattern][$NumInput-
    1]);

                $MSE=$this->mse($data[$e%$NumPattern][$NumInput-1]);

                if($e==0)

                {

                        echo "\nFirst epoch Mean Square Error: $MSE";

                        echo "</br>";

                }

                if( $MSE < $Thresh)

                {

       echo "Threshold value achieved in ".$e." iterations.";

                  echo "</br>";

       echo "MSE:  ".$MSE;

                  echo "</br>";

       break;

                  echo "</br>";

      }

        }

        echo "Last epoch Mean Square Error: $MSE";
        echo "</br>";
        echo "</br>";
        echo "Testing Data....................................................................Prediction";
        echo "Testing Data....................................................................Prediction";
        echo "</br>";
        $test=array();
for ($i = 0 ; $i < 100; $i++ )
{
    $this->ffwd($testData[$i]);
                $ii=count($testData[$i]);
    echo "</br>";
                for($j=0;$j<$ii;$j++)
                {
                echo $testData[$i][$j];
                echo " ";
            }
                echo "=>";
                $lg=(double)$this->Out(0);
                echo $lg;
                if($lg>0.5)
                {
                        fwrite($lhandle,1);
                        array_push($test,1);
                        fwrite($lhandle,"\n");
                }
                else
                {
                        fwrite($lhandle,0);

                        array_push($test,0);

                        fwrite($lhandle,"\n");

                }

}

        $count=0;

        $clfile="./testdata/backpro/classified/k5/tr1.txt";

        $chandle=fopen($clfile,"r");

        $carray=array();

        while(!feof($chandle))

        {

                $cline=fgets($chandle);

                $ss=trim($cline);

                $carr=explode(" ",$ss);

                array_push($carray,$carr[12]);

        }

        for($c=0;$c<100;$c++)

        {

                if($test[$c]==$carray[$c])

                {

                $count++;

                }

        }

        echo "</br>";

        $wt="Total Correct Prediction is: $count";

        echo "</br>";

        fwrite($lhandle,$wt);
        echo "$wt";
        echo "</br>";
        echo "Training and Testing Concluded";
}
}
//traindata
$trainfile="./traindata/backpro/final/k5/merge1.txt";
$tandle=fopen($trainfile,"r");
$data=array();
while(!feof($tandle))
{
        $tline=fgets($tandle);
        $arr1=explode(" ",$tline);
        array_push($data,$arr1);
}
//testdata
$testfile="./testdata/backpro/test/final1.txt";
$handle=fopen($testfile,"r");
$testData=array();
while(!feof($handle))
{
        $line=fgets($handle);
        $arr=explode(" ",$line);
        array_push($testData,$arr);
}
$layersSize=array(18,6,5,1);
$numLayers = count($layersSize);
```

```
// Learning rate – beta and momentum beta

$beta = 0.3; $alpha = 0.1;

// Creating the net

bp=new BackPropagation($numLayers,$layersSize,$beta,$alpha);

$bp->Run($data,$testData);
?>
```