Survey of Machine Learning Based IDS and Description of a New Combined Classifier Proposal for Host-Based IDS

Sattyik Kundu
Information Security and Assurance Department,
George Mason University, Fairfax, Virginia

Abstract

Intrusion detection systems (IDS) are software or devices that monitor a network or system for malicious activity or violations of policy. The detected malicious activity and policy violations are then reported to the administrator or collected centrally by a security information and event management (SEIM) system.

For operating systems (OS), protecting its kernel is especially important because if any malicious data successfully intrudes and subverts the kernel; the damage can be very difficult to undo considering the most sensitive files are inside the kernel. In the worst case, the OS would need to be reinstalled.

Regarding the issue of protecting an OS kernel, this research paper surveys different machine learning scheme available for IDS, and proposes a new machine learning based scheme for developing an IDS to detect malicious activities attempting to intrude the OS kernel. In particular, a new approach of a combined classifier using multi-layer perceptron neural net (MLP-NN), Random Forest (RF) and Naive Bayes classifier is proposed. Additionally, an appropriate dataset to develop and test such a system is also described.

This proposed IDS is a host-based IDS that uses anomaly-detection. The IDS would be placed on top of the kernel to read the system calls taking place within the kernel. The combined classifier, trained using ADFA-LD (for a Linux OS) or ADFA-WD (for Windows OS) dataset, used in the IDS will decide normal versus anomalous system behavior in an uninterrupted fashion.

I. Introduction

At first, the details about intrusion detections systems both for networks and systems will be described.

A. Intrusion Detection Systems

A.1: Definitions

As noted, IDSs are software or devices that monitor a network or system for malicious activity and policy violations. The two main categories of IDS are:

- Network Based: Network IDS (NIDS) are placed on a strategic location(s) within a network to monitor traffic from all devices to and from all devices with a network. The monitored traffic is then matched against a library of known attacks to determine if an intrusion has taken pace.
- Host Based: host IDS (HIDS) can only be implemented inside a host device within a network. A HIDS monitors the incoming and outgoing packets on the device for suspicious activity. A HIDS works by taking a snapshot of current system files and compares it against a previous snapshot. From the comparison, if the HIDS notices any changes in critical system files or deviations from system policies, the user/administrator is notified.

Given that the eventual IDS in the proposed system needs to be placed between the User space and Kernel space, the IDS will be host-based. Besides IDSs being categorized based on location, IDSs can also be categorized based on the detection method used. The two main detection methods are signature and anomaly based.

A.2: Signature-based

This method detects attacks by comparing traffic patterns against a database of traffic patterns that could considered malicious. If a traffic pattern was found to match one of the patterns in the database, the IDS would alert and intrusion detection.

Signature detection generally has low false-positive rates and is very effective against known attacks. Hence, this detection method is favored for most IDSs. However, if malicious traffic that is not found in the database passes through, the result is a **zero-day attack**. To mitigate this, the IDS needs to be updated and patched regularly.

A.3: Anomaly-based

This method uses heuristics and rules to determine what traffic and system activity is considered normal or anomalous. If any traffic or system activity operates within the given heuristics, then the activity is *normal*; if the activity is outside the defined rules, the activity is then considered *anomalous*.

An anomaly-based IDS generally needs to be taught how to recognize system activity. This is done over two phases which are the training phase (where a model or profile representing normal or normal/abnormal behavior is created) and the testing phase (where attack vectors are tested against the created profile). Most importantly, training an IDS to

recognize system activity needs a labeled database of normal and abnormal behavior of the system.

The main benefit of an anomaly-based system is that they are better suited for zero-day attack since any attack should theoretically deviate from an established model for normal system behavior. Hence, the proposed IDS will be anomaly-based and implement

A.4: What the Proposed IDS will Use

Given the importance of the OS kernel, the proposed **IDS will be anomaly-based and implemented on the host system** in order to give the OS kernel greater protection from zero-day attacks.

The IDS provides one of the most promising paths towards network/host robustness. IDS are used to detect the several types of malicious activities that can violate the rules and trust of the host systems. As noted, anomaly-based IDS determines normal network behavior like bandwidth range, types of protocols, ports and a device used to connect each other and alerts will be sent to the network administrator or user, when inconsistent traffic is detected. On the other hand, Signature-based IDS monitors packets in the network and compares them with pre-configured and pre-identified attack behaviors, called signatures

B. Survey of Existing IDS Systems

Here is a survey of several different existing IDS. The goal here is to determine which kind of IDS is best suited for intrusion detection inside the host

This section deals the related works on HIDS and the existing classification algorithms **mostly applied on KDD'99 dataset**. Enormous amount of work has been conducted on KDD 99 Intrusion Detection datasets for classification of different attacks [1]. Tavallaee [2] worked with several clustering algorithms for anomaly detection using KDD'99 data set; their scheme tried to identify the unknown attack but could only achieve maximum accuracy of 80%. The IDS designed by Ficco et.al. [3] collects information at several architectural levels and performs complex event correlation based on complex event processing engine. This intrusion detection system implements a comprehensive alert correlation workflow for detection and diagnosis of complex intrusion scenarios in large scale complex infrastructures. Their model could attain a higher accuracy of 91% at the cost of increased complexity. The intrusion model proposed by Ying et.al. [4] enhances the detection of intrusion by combining two detection techniques, namely, log file analysis technique and multi-layer perceptron neural network model technique with accuracy of 93% at cost of longer detection time. Most researchers use C4.5 algorithm (a decision tree algorithm), Naïve Bayes, Random tree and Random forest algorithm (apart from neural net) to train the systems using the features in the training samples of KDD'99 dataset.

The C4.5 Tree classifier produces good results but no combined classifier was used or proposed. Gudadhe et.al. [5] proposed new ensemble decision tree (conceptually Random Forest) for IDS. It reduces the computational cost of the existing methods and achieves 93% accuracy. Patil et.al. [6] proposed an IDS and two algorithms are used, namely, C5.0 (a faster version of C4.5 Decision Tree) and CART. From the statistical analysis it is proved that C5.0 provides better accuracy when compared to the performance of the CART algorithm. The CART algorithm achieves the accuracy of 94%. The researchers contends that the existing works need more computational time (to be useful for a real-time system) and more memory space to store the 42 features of KDD dataset, and proposed CPDT (Correlation based Partial Decision) and CFS (Correlation Feature Selection) to overcome some of the limitations [7].

Although works described so far do not use Random Forest as the mainstay of classification, the works described in [8] uses different variants of a combined classifier using three classification schemes: Random Forest, Random Projection and Support Vector Machines (SVM). The work described in [8] again uses NSL-KDD (an enhanced version of KDD-99) data for NIDS implementation. Their results shows that Random Forest alone performs better than any other combination effectively nullifying SVM type classifier. Random Forest classifier is also used in [9].for NIDS using KDD-99 dataset, and good results are reported. Similar results using Random Forest classifier using KDD-99 and NSL-KDD datasets are reported [10, 11]. A modified version of KDD data set is used to reflect more practical scenarios with two classification schemes: Random Forest and SVM in [12]. There are two main conclusions drawn by this paper: 1) RF produces similar accuracy in a much faster manner, 2) the most important deficiency in the KDD'99 dataset is the huge number of redundant records. The scheme in [12] developed a derived dataset, KDD99Train+ and KDD99Test+, which does not include any redundant records in the training set as well as in the test set, so the classifiers will not be biased towards more frequent records.

A very comprehensive use Artificial Neural Network (ANN)) is discussed in [13]. It should be noted that multi-layer perceptron is one of many possible ANN architectures. In fact ANN is the most used scheme for detecting both HIDS and NIDS [14, 15, 16 17]. The works described used KDD-99 or some proprietary dataset.

A HMM (hidden Markov model) is a doubly stochastic Markovian model. In HMM implementation, exemplars from each class is modeled as one HMM. During classification, incoming data (exemplar) is tested again all models, and the most likely model given by its matching probability is selected as the chosen class. The most attractive aspect of HMM is that it can model dynamically evolving signal. Since HIDS signals evolves as it goes through many changing states, HMM could be a natural fit and is used in implementing IDS [18, 19].

Two problems transpire: 1) KDD-99 databases is old and no longer reflects realistic situations of today and 2) no proper combined classifier has been used. **Thus, I propose in this paper to use a database that is more reflective of realistic scenarios and a combined classifier made of RF, NN and Bayes classifier.**

The paper is organized as follows. Section II discusses database that I propose to use. Section III describes the proposed combined classifier scheme. Section IV discusses the proposed experiment set-up with a good description of the software. Finally, section V contains the conclusions while Section VI has all the references.

II. Data for Training/Testing

Even if the outline of what kind of IDS would be best for OS security has already been decided; testing the IDS via implementation is needed to determine whether the chosen IDS is viable or not. The IDS will need to be trained to determine whether it can recognize and differentiate between normal and malicious data during testing.

Thankfully, training data does not need to be made from scratch as there are two publicly available sets of data that will be useful for training. These data sets are called KDD-99 [24, 25] and ADFA-LD12 [20 - 23]. Another database which is also used sometimes is the UNM (University of New Mexico) database [27]. The UNM dataset is of comparable vintage to the KDD data, and focuses on single processes rather than a whole task of system evaluation.

A. KDD-99

In 1999, this dataset was created during the *3rd International Knowledge Discovery and Data Mining Tools Competition* which was held together with the *5th International Conference on Knowledge Discovery and Data Mining*. The dataset was made so an IDS can create a predictive model capable of differentiating between “bad” connections (a.k.a. intrusions or attacks) and normal connections. An IDS model created from this data set is mainly suited for simulating network intrusions and not host intrusions.

Of the data set, the training data represents 24 out of 38 different attacks present in the test data. The 24 different attacks in the training data represent known attacks whereas the additional 14 attacks not found in the training data represent novel attacks. With the IDS predictive model created from the data of 24 attacks, the 14 novel attacks from the test data would be unknown to the created model; this is meant to create a more realistic simulation of how the simulated IDS would handle detection of known and unknown attacks.

The 38 total attacks can be roughly categorized into 4 main groups which are:

- DOS: denial-of-service, e.g. syn flood;
- R2L: unauthorized access from a remote machine, e.g. guessing password;
- U2R: unauthorized access to local super-user (root) privileges, e.g., various ``buffer overflow'' attacks;
- Probing: surveillance and other probing, e.g., port scanning

B. ADFA

Back in 2013, a graduate student named Gideon Creech from the University of New South Wales spearheaded the creation of the ADFA dataset for the evaluation of HIDS and as a modernization of the antiquated KDD-98 datasets which have been widely used since its creation from 1999. Whereas the KDD-99 dataset was meant to be used to train and evaluate network-based IDS with signature detection, the ADFA data sets are supposed to be used for the evaluation of host-based IDS with anomaly detection. Given that our IDS's outline is host-based and uses anomaly detection, ADFA will be the primary testing data set and the KDD-99 can be used as a control to compare the ADFA dataset against.

The ADFA dataset comes in two variations. There is an ADFA dataset for Linux operating systems; and there is another ADFA dataset for Windows operating systems. Give that Linux and Windows are the two most common operating systems; these ADFA datasets are a much needed and welcome inclusion in HIDS and anomaly-detection research.

B.1: ADFA-LD

ADFA-LD is the dataset made for evaluating HIDS on Linux distributions. The ADFA-LD dataset is a collection of system call traces created on an *Ubuntu Linux 11.04 Host OS* with *Apache 2.2.17* running *PHP 5.3.5*, *FTP*, *SSH*, *MySQL 14.14*, and *TikiWiki*. This was supposed to represent a typical Linux host OS [20, 21]. **As defined, each trace** contains the system calls of a single process in the order in which the kernel received them [22, p.14].

Attack Selection

The table below shows a breakdown of payloads and vectors of attacks represented within the call traces. The creators of this dataset put considerable effort in determining the possible attacks used by contemporary hackers and penetration testers to promote realism.

Table I: Attack Structure

Payload/Effect	**Vector**
Password bruteforce	FTP by Hydra THC
Password bruteforce	SSH by Hydra THC
Add new superuser	Client-side poisoned executable
Java Based Meterpreter	Tiki Wiki vulnerability exploit
Linux Meterpreter Payload	Client-side poisoned executable
C100 Webshell	PHP Remote File Inclusion vulnerability

1. The first two attack vectors represent brute force password guessing attempt on the open services of FTP and SSH. Because FTP and SSH are commonly used services, they are regularly attacked if exposed to external sources [2].

2. The third vector represents an attack via privilege escalation. While privilege escalation attacks are difficult in a Linux environment, one possible way is the creation of a new user with super privileges. In this dataset, this attack is represented via client-side attack. More specifically, a payload encoded into a Linux executable (with Metasploit) was uploaded to the server in a simulation of a social engineering attack. Finally, this payload is then triggered both remotely and locally using a collection of techniques such as local file inclusion and simulated social engineering [21-23].
3. The fourth and fifth attack vectors revolve around the use of Meterpreter (which is part of the Metasploit Framework). Meterpreter is basically an enhanced functionality command shell which facilitates the remote compromise of a targeted system. Meterpreter is available in multiple versions including Java and standalone executables. While Meterpreter presented functionality is the same regardless of the version, the interaction with the underlying OS is different for each version. Hence, this warrants using data to represent a different possible attack vector for each Meterpreter version [21, p.66].
4. The sixth and last attack vector revolves around the C100 Webshell. This is a sophisticated piece of PHP code which provides an illicit GUI interface to the attacker through a web browser and thus allow manipulation of the underlying OS. A PHP-based remote file inclusion vulnerability was used to upload the C100 Webshell which is then used to further compromise the host system and escalate privileges [21-23].

Dataset Structure

The ADFA-LD dataset is a collection of system call traces; and they're divided into three group. Each of the system call traces were collected during normal operation of the selected host (which uses the Ubuntu Linux 11.04 OS as explained earlier); with activities range from web browsing to LaTeX document preparation [20, 21].

The traces were generated using an audited Unix program and then filtered by size. The size of the training data and validation traces range at 300 Bytes–6kB and 300 Bytes–10 kB, respectively. These trace sizes provide a trade-off between data fidelity and unnecessary processing computations [20, 21].

Finally, the below table shows the number of traces in each group of data. The training data is first used to fit the parameters of the IDS's classifier and create a machine model. The validation data is then used to evaluate the fitted model while tuning the model's hyper parameters; this results in a fully-specified and completed model. Finally, the attack data is used to evaluate the model's ability to differentiate between suspicious and normal activity.

Table II: Dataset Structure	
Data Type	**Trace Count**
Normal Training Data	833 Traces
Normal Validation Data	4373 Traces
Attack Data	10 Attacks per Vector(see Table I)

The new ADFA dataset has a much larger degree of similarity between attack data and normal data than the KDD collections. Consequently, the new dataset is much more representative of current cyber-attacks of today, and forms a realistic and relevant metric for IDS performance evaluation. Also, the new ADFA is more challenging for machine learning algorithms because of much larger degree of similarity between attack data and normal data than the KDD collections. The validity of this AFDA test data was examined by evaluating the performance of several well-known IDS DE algorithms: hidden Markov models, the STIDE approach, K-Means clustering, and the K-Nearest Neighbor algorithm. The performance was significantly worse than when evaluated against the KDD98 and 99 datasets [21].

B.2: ADFA-WD

ADFA-LD vs ADFA-WD Datasets

Due to the differences in architecture between Linux and Windows OS systems, the traces used in the ADFA-WD dataset are different from those of the ADFA-LD dataset covered earlier.

For Linux OS distributions, the interaction between programs and the OS kernel is highly linear. This linearity enables systems calls to be easily mapped to various system activities. This consequently allows accurate baselining of what is considered normal system activity as a result [21, p.83].

On the other hand, Windows OS distributions make extensive use of DLL (dynamic linked library) which complicates the interaction between programs and kernel whilst simplifying other aspects of the software engineering cycle. The DLL centric architecture of Windows means that system calls are often accessible through multiple paths with the result that decisions on normalcy cannot be made based on the system call patterns alone. Because of this lack of linearity; it was made difficult to provide a high quality dataset needed to create a high-performing anomaly-based HIDS protection. As a result, many Window vendors resorted to Anti-Virus systems which use signature-detection [21, p.83].

Eventually, because system calls traces could not be used due to dimensionality issues, DLL access requests were used instead for ADFA-WD. The DLL access request is any call by a program to a particular location within a DLL. Once the call has been made, the DLL will automatically involve other aspects of the system architecture to implement the desired action;

probably using system calls and other low-level manipulations. The subsequent actions, however, are fixed and represent the effect on the kernel of calling that particular memory location. As DLL access requests form a linear pattern, dimensionality reduction is performed inherently by this process and any consideration of these requests will provide the linearity required by existing HIDS algorithms [21, p.91].

Software Environment of ADFA-WD

ADFA-WD is the dataset made for evaluating HIDS on Windows distributions. This dataset was collected on a *Windows XP Service Pack 2* (*XP SP2*) OS. The reason XP SP2 was chosen despite being relatively early on the Windows OS development timeline is because it did not have the security controls ASLR (Address Space Layout Randomization) and DEP (Data Execution Prevention); which are found in later Windows OS distribution. This meant HIDS performance can be evaluated in isolation from these additional protections; as thus avoid artificial inflation of performance results due to assistance by security measures other than the HIDS itself [21, p.100]. Additionally, the ADFA-WD dataset still remains relevant for later Window OS distributions because their underlying architecture have not changed significantly since XP SP2.

Additionally, the stateless firewall for XP SP2 was turned on during the dataset collection to block all attacks. Because the firewall was effective against low-level attacks, this forced attacks to be set to a higher level thus increased the relevancy of results generated from the final dataset to current threats [21, p.101].

Lastly, Norton AV 2013 was used to scan certain payloads. Many hackers have bypassed signature-based recognition systems; hence payload obfuscation detection became an important contribution of anomaly-based systems. By including payloads which have successfully bypassed an advanced and well-regarded AV program, subsequent detection by anomaly-based methodologies indicates an important contribution to the overall security posture of the host-based HIDS [21, p.101].

Software Environment Vulnerabilities

When collecting the ADFA-WD dataset, besides setting up the *Windows XP service Pack 2*, stateless *Windows firewall*, and *2013 Norton Anti-Virus* environment; the last added features of the host OS included [21, p.104] :

- The addition of an FTP server
- web server and management tool
- a streaming audio digital radio package provide a generous network-based attack surface

- A mix of wireless and Ethernet networking was used to provide connectivity, with one of the more advanced attacks using a fake wireless access point to provide DNS spoofing and a consequent browser attack upon redirection.

The vulnerabilities listed below were also added to the OS environment to add more realism and represent possible exploits the attackers can used on a host OS secure by *Windows firewall*, *2013 Norton Anti-Virus* environment, and the above. This is all represented in the ADFA-WD dataset.

VID	Vulnerability	Program	Exploit Mechanics
1	CVE: 2006-2961	CesarFTP 0.99g	Reverse Ordinal Payload Injection - custom exploit.
2	EDB-ID: 18367	XAMPP Lite v1.7.3	Xampp_webdav used to upload and execute a malicious payload.
3	CVE: 2004-1561	Icecast v2.0	Metasploit Framework exploit used.
4	CVE: 2009-3843	Tomcat v6.0.20	Metasploit Framework exploit used.
5	CVE: 2008-4250	OS SMB	Metasploit Framework exploit used.
6	CVE: 2010-2729	OS Print Spool	Metasploit Framework exploit used.
7	CVE: 2011-4453	PMWiki v2.2.30	Metasploit Framework exploit used.
8	CVE: 2012-0003	Wireless Karma	Pineapple Router with an active *Karma attack* and DNS Spoofing used to automatically associate and redirect to an attacking computer hosting a browser attack.
9	CVE: 2010-2883	Adobe Reader 9.3.0	Malicious PDF file used to spawn a reverse shell.
10	···	Back-doored Executable	Reverse Inline Shell spawned.
11	CVE: 2010-0806	IE v6.0.2900.2180	Metasploit Framework Exploit used.
12	···	Infectious Media	Bind Shell spawned.

Some notes regarding the exploit mechanics [21, p.108-109]:

1. **VID 1**(Vulnerability ID) used a custom-made exploit to compromise the FTP server. The shellcode for this exploit was generated using the Reverse Ordinal payload included with the Metasploit Framework (MSF) and represents a fairly traditional server-side attack.
2. **VID 2** utilized default credentials in the webdav plug-in to upload and execute a malicious payload and was the first web-based attack against the host.
3. **VID 3** employed a standard MSF exploit to compromise a streaming media server in another server-side attack.
4. **VID 4** compromised the Tomcat Manager application 108 to upload and execute a connect back payload, extending the web-facing attack surface.
5. **VID 5** and **VID 6** attacked inbuilt aspects of the OS SMB implementation, focussing on file sharing and printer sharing.
6. **VID 7** introduced a web application attack, with a multi-stage hack against PMWiki resulting in full system compromise.
7. **VID 8** is a special case and uses a piece of hacking hardware known as the Pineapple Router. This device is designed to simulate any previously joined wireless access points that a host may have used in the past, and when it detects a probe request containing a specific SSID responds as if it was that access point. In doing so, it effectively inserts itself as a man-in-the-middle device and is able to perform traffic manipulation, redirection, and modification.

8. **VID 9** compromised the popular Adobe Reader program with a PDF containing executable malware.
9. **VID 10** was used to create a connect-back shell in the execution flow of an otherwise benign program. When the program in VID 10 was opened by the user, a shell was spawned on the attacker's computer whilst presenting the same interface and functionality to the user.
10. **VID 11** used a browser attack to compromise Internet Explorer when the user visited a malicious web-page, mimicking the increasing use of this vector by hackers "in-the-wild".
11. Lastly, **VID 12** used a compromised autorun procedure on a CD to execute a portbind payload when the CD was inserted. This method is similar to that hypothesized for some of the Stuxnet infections, which used a USB rather than CD but likely exploited the same technique.

ADFA-WD Data Structure

The selected attacks used in the creation of the ADFA-WD dataset can be represented by the vectors chosen to be attacked as well the effects of the attacks on the vectors listed on table III on the right.

Table III

Vectors	
• TCP ports; • Web based vectors;	• Browser attacks; and • Malware attachments.
Effects	
• Bind shell; • Reverse shell; • Exploitation Payload; • Remote operation; • Staging;	• System manipulation; • Privilege escalation; • Data exfiltration; and • Back-door insertion.

While attacks on TCP ports, browser attacks, and malware attachments are common place and well-covered by Anti-virus; web-based vectors highlight a unique importance of ADFA-WD. Traditionally, defense against web-based vectors is extremely difficult for signature-based systems as accessing hosted pages and supplying input to web languages such as PHP or JavaScript are fundamentally the same for normal activities as for attacks. Additionally, the high variance in content for web activity coupled with the ease of obfuscation means that identification of clear signatures for web-vectors attacks is extremely difficult. For example, an organization's website can be a point of entry into the security perimeter for attackers via web attacks. Hence, the importance of anomaly-centric detection of web based attacks is readily apparent [21, p.102]. The table below summarizes the normal and attack data for ADFA-WD.

Data Type	Sampling Runs per VID	Number of Raw Traces	Length 200 Traces
Normal Data for Training	10 runs	356 traces	67,752 traces
Normal Data for Validation and FAR Calculation	100 runs	1,828 traces	590,675 traces
Attack Data for DR Calculation	10 runs	5773 Traces	374,806 traces

III. Proposed Combined Classification Scheme

As stated before, I proposed a combined classifier made up of three distinct classifiers: MLP-NN, Random Forest and Naïve Bayes classifier. In the following, each classifier is described in some details. First, some pattern classification definitions are described.

Supervised learning (classification) uses Supervision: The training data (observations, measurements, exemplars, etc.) are accompanied by labels indicating the class of the observations. New data is classified based on the training set

Unsupervised learning (clustering): The class labels of training data are unknown. Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data.

Classification: Predicts categorical class labels (discrete or nominal), classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data.

Model construction: Each exemplar/sample is assumed to belong to a predefined class, as determined by the class label attribute. The set of exemplars used for model construction is training set. The model is represented as classification rules, decision trees, or mathematical formulae.

Model usage: Trained models are used for classifying future or unknown objects. To estimate accuracy of the model, the known label of test sample is compared with the classified result from the model. Accuracy rate is the percentage of test set samples that are correctly classified by the model. Test set is independent of training set (otherwise overfitting). If the accuracy is acceptable, use the model to classify new data.

Artificial Neural Net [28]

An Artificial Neural Network (ANN) is a computational model that is inspired by the way biological neural networks in the human brain process information. Artificial Neural Networks have generated a lot of excitement in Machine Learning research and industry, thanks to many breakthrough results in speech recognition, computer vision and text processing. Here, we will try to develop an understanding of a particular type of Artificial Neural Network called the Multi-Layer Perceptron **(MLP-NN).**

A feedforward neural network can consist of three types of nodes:

1. **Input Nodes –** The Input nodes provide information from the outside world to the network and are together referred to as the “Input Layer”. No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.

2. **Hidden Nodes –**The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.
3. **Output Nodes –**The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network [28]. This property of feed forward networks is different from Recurrent Neural Networks in which the connections between the nodes form a cycle.

Two examples of feedforward networks are given below:

1. **Single-Layer Perceptron**– This is the simplest feedforward neural network and does not contain any hidden layer.

2. **Multi-Layer Perceptron**– A Multi-Layer Perceptron has one or more hidden layers. I will only discuss Multi-Layer Perceptrons (MP-NN) below since they are widely useful for practical applications today.

Multi-Layer Perceptron

Figure 1 shows a multi-layer perceptron with a single hidden layer. Note that all connections have weights associated with them, but only three weights (w0, w1, w2) are shown in the figure.

Input Layer: The Input layer has three nodes. The Bias node has a value of 1.The other two nodes take X1 and X2 as external inputs (which are numerical values depending upon the input dataset). As discussed above, no computation is performed in the Input layer, so the outputs from nodes in the Input layer are 1, X1 and X2 respectively, which are fed into the Hidden Layer.

Hidden Layer: The Hidden layer also has three nodes with the Bias node having an output of 1. The output of the other two nodes in the Hidden layer depends on the outputs from the Input layer (1, X1, X2) as well as the weights associated with the connections (edges). Figure 1 shows the output calculation for one of the hidden nodes (highlighted). Similarly, the output from other hidden node can be calculated. Remember that "***f" r***efers to the activation function. These outputs are then fed to the nodes in the Output layer.

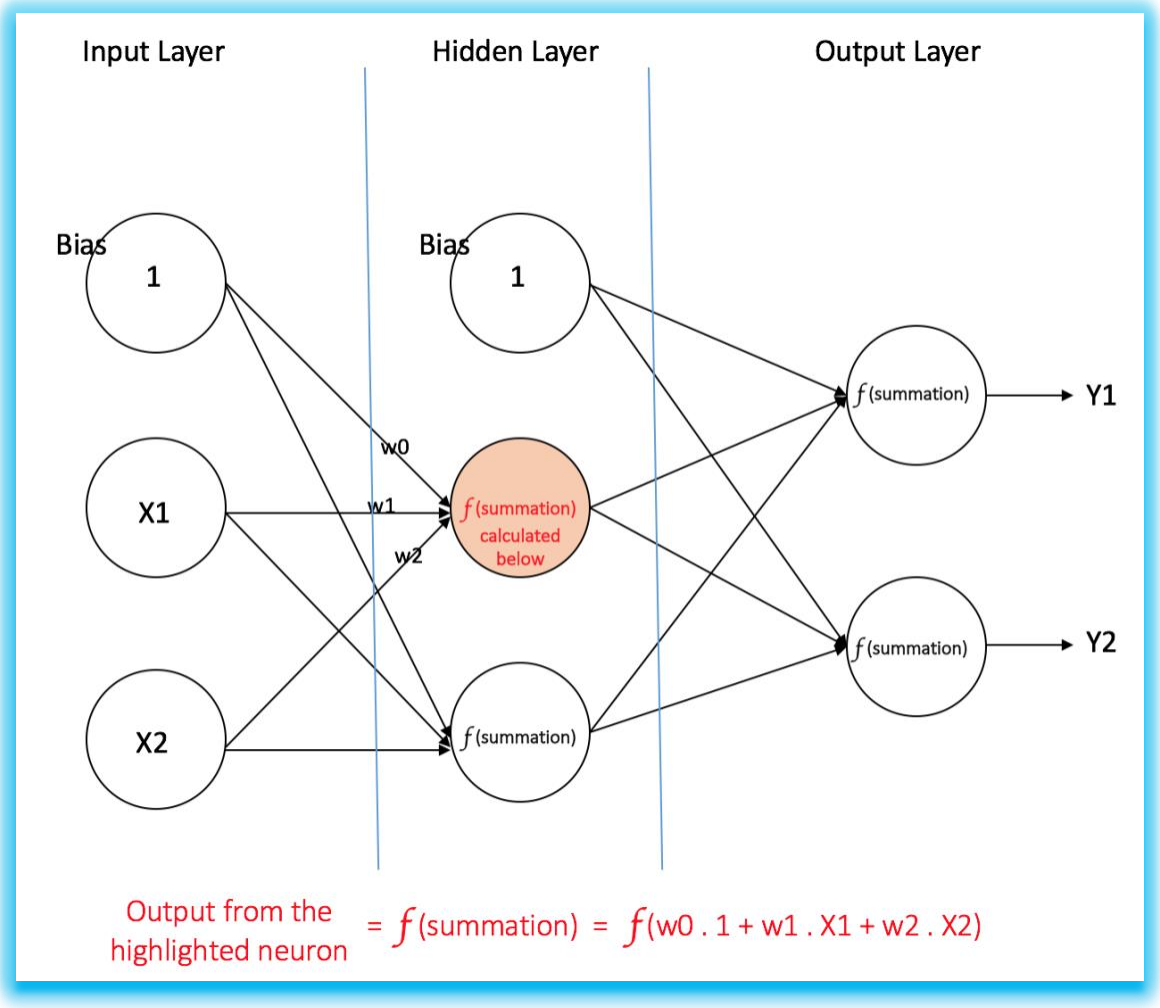


Figure 1: A multi-layer perceptron having one hidden layer

Output Layer: The Output layer has two nodes which take inputs from the Hidden layer and perform similar computations as shown for the highlighted hidden node. The values calculated (Y1 and Y2) as a result of these computations act as outputs of the Multi-Layer Perceptron.

Given a set of features **X = (x1, x2, ...)** and a target **y**, a Multi-Layer Perceptron can learn the relationship between the features and the target, for either classification or regression.

A Single Neuron

The basic unit of computation in a neural network is the **neuron**, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated **weight** (w), which is assigned on the basis of its relative importance to

other inputs. The node applies a function ***f*** (defined below) to the weighted sum of its inputs as shown in Figure 1 below:

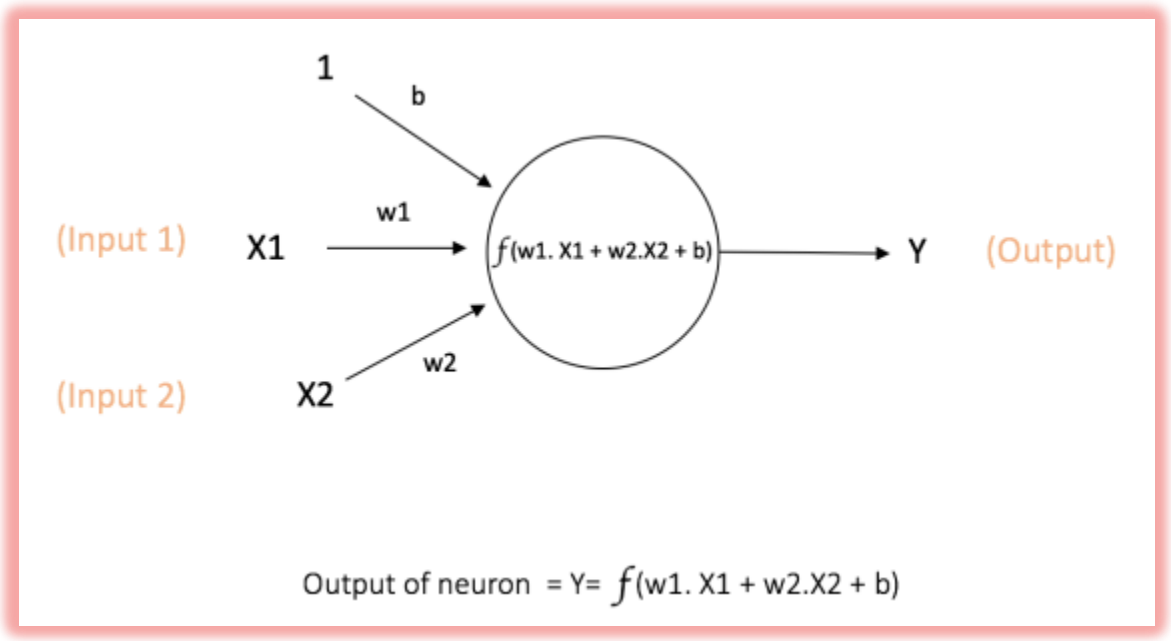


Figure 2: A single neuron

The above network takes numerical inputs **X1** and **X2** and has weights **w1** and **w2** associated with those inputs. Additionally, there is another input **1** with weight **b** (called the **Bias**) associated with it. More details will be covered about role of the bias later.

The output **Y** from the neuron is computed as shown in the Figure 1. The function ***f i***s non-linear and is called the **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data is nonlinear and neurons should *learn* these nonlinear representations.

Every activation function (or *non-linearity*) takes a single number and performs a certain fixed mathematical operation on it [28]. There are several activation functions you may encounter in practice:

- **Sigmoid:** takes a real-valued input and squashes it to range between 0 and 1

$$\sigma(x) = 1 / (1 + \exp(-x))$$

- **tanh:** takes a real-valued input and squashes it to the range [-1, 1]

$$\tanh(x) = 2\sigma(2x) - 1$$

- **ReLU**: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero)

$$f(x) = \max(0, x)$$

The below Figures3 show each of the above activation functions.

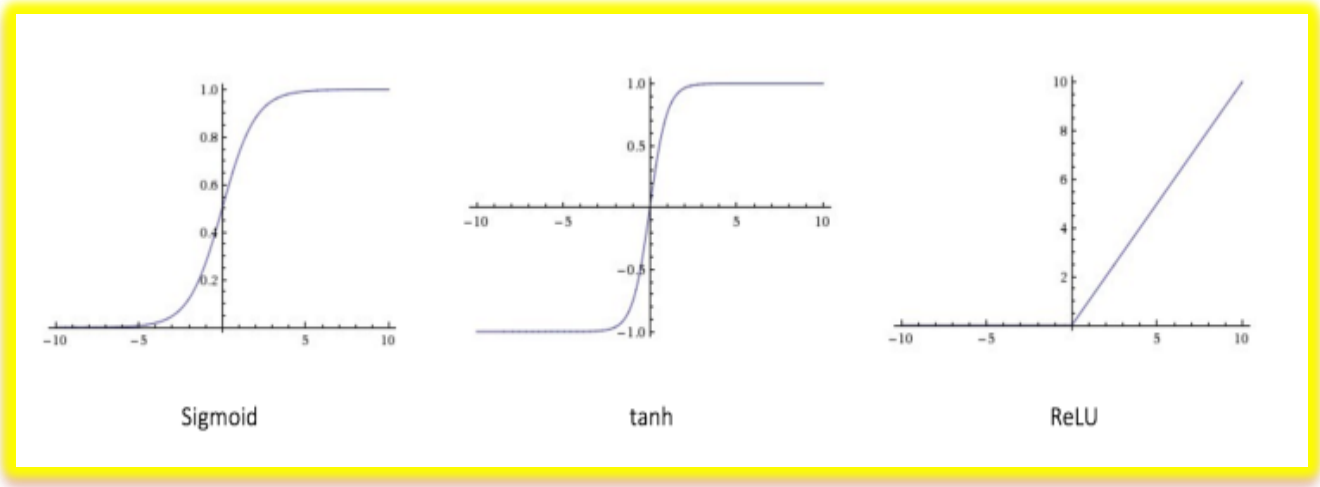


Figure 3: Different activation functions

Importance of Bias: The main function of Bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives) to offset any possible constant bias in the data.

BackProp Algorithm:

Backward Prorogation of Errors, often abbreviated as BackProp, is one of the several ways in which an artificial neural network (ANN) can be trained. It is a supervised training scheme, which means, it learns from labeled training data (there is a supervisor, to guide its learning). To put in simple terms, BackProp is like "learning from mistakes". The supervisor corrects the ANN whenever it makes mistakes.

An ANN consists of nodes in different layers; input layer, intermediate hidden layer(s) and the output layer. The connections between nodes of adjacent layers have "weights" associated with them. The goal of learning is to assign correct weights for these edges. Given an input vector, these weights determine what the output vector is. In supervised learning, the training set is labeled. This means, for some given inputs, we know (label) the desired/expected output.

Initially all the edge weights are randomly assigned. For every input in the training dataset, the ANN is activated and its output is observed. This output is compared with the desired output that we already know, and the error is "propagated" back to the previous layer. This error is noted and the weights are "adjusted" accordingly. This process is repeated until the output error is below a predetermined threshold.

Once the above algorithm terminates, we have a "learned" ANN, which we consider is ready to work with "new" inputs. This ANN is said to have learned from several exemplars (labeled data) and from its mistakes (error propagation). The BackProp algorithm gives no guarantee of global convergence to 'optimal' solution; however, more often than not, it provided good experimental performance

Random Forest [29,30]

Suppose we have the following Training data set: buys_computer

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Figure 4: buys_computer data set

We can construct a tree classifier based on this data as follows.

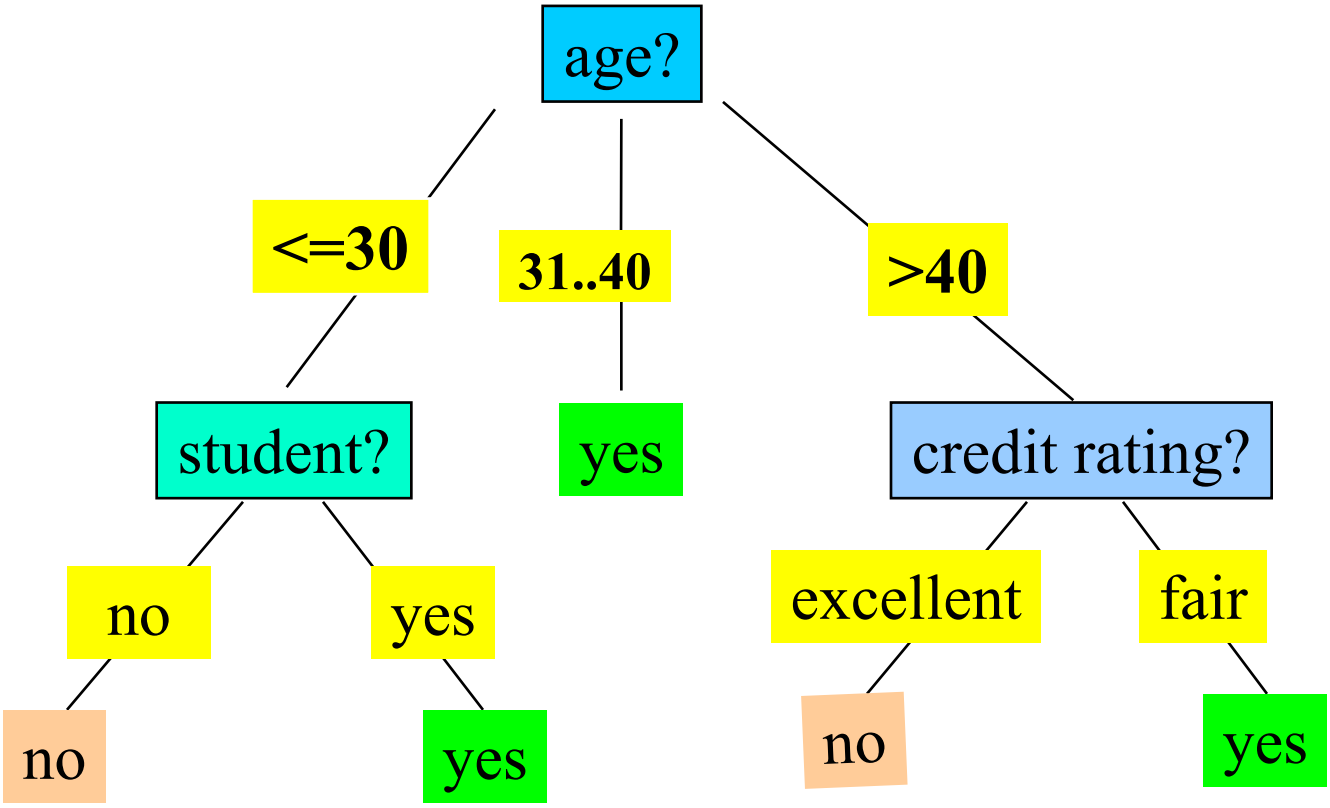


Figure 5: A Tree Classifier of buys_computer Data

Random Forests are an ensemble of k (an integer) Decision Trees (trees with only a root node) with M (an integer) bootstrap samples (k and M do not have to be the same though often are same) trained using a variant of feature bagging method. Note the method of training random forests is not quite as straightforward as applying bagging to a bunch of individual decision trees and then simply aggregating the output. Bootstrap sample simply means a portion of all exemplars are selected at random replacing each exemplar after it is drawn from the sample.

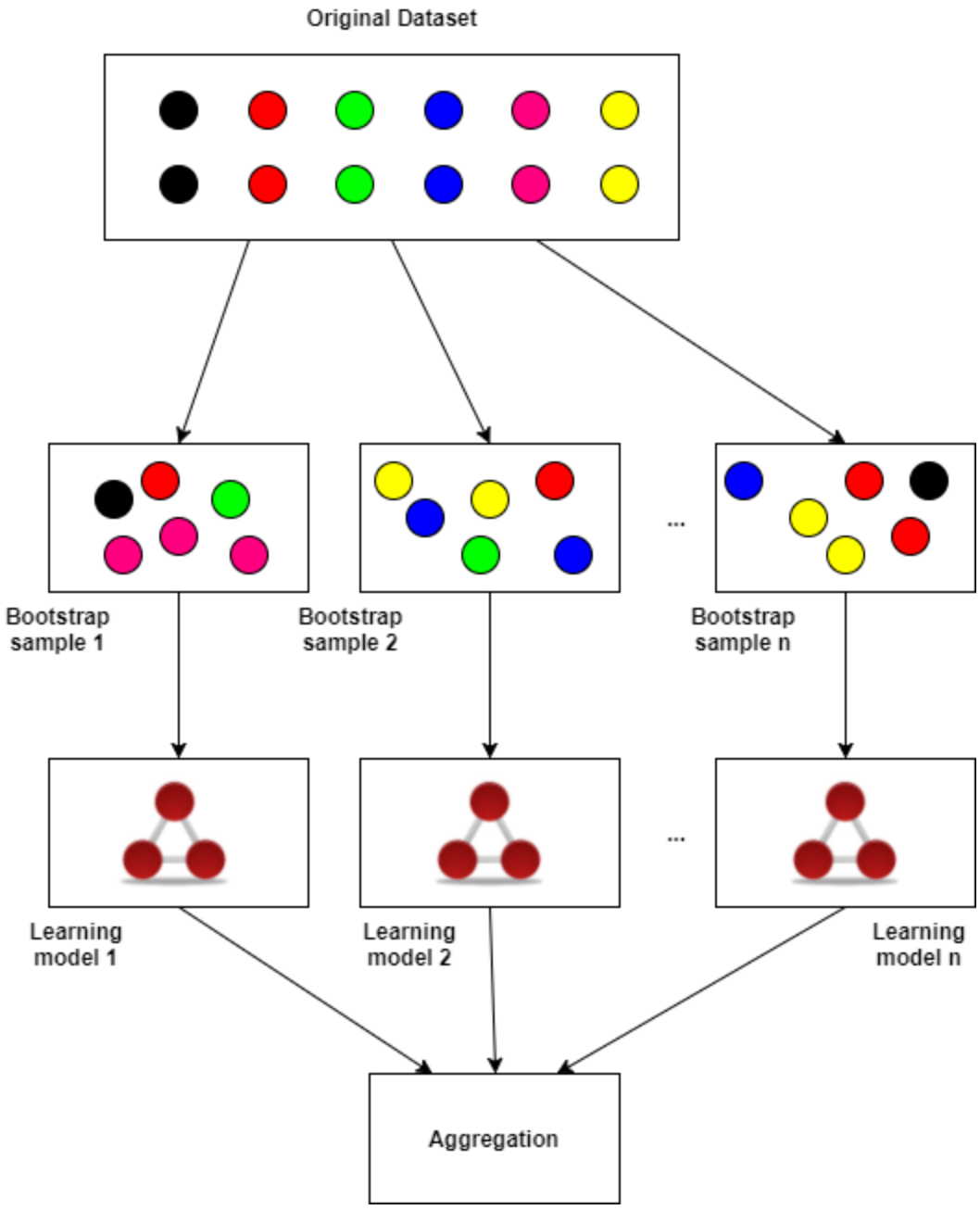


Figure 6: A schematic view of Random Forest Classifier.

Random Forests Training:

1 Create at least M bootstrap sample to create M decision trees of the forest.

2. At the current node (of any tree), randomly select p features from available features D. The number of features p is usually much smaller than the total number of features D.

3. Compute the best split point for tree k using the specified splitting metric (Gini Impurity, Information Gain, etc.) and split the current node into daughter nodes and reduce the number of features D from this node on.

4. Repeat steps 2 to 3 until either a maximum tree depth has been reached or the splitting metric reaches some extrema.

5. Repeat steps 2 o 4 for each tree k in the forest.

6. Vote or aggregate on the output of each tree in the forest.

Compared with single decision trees, random forests split by selecting multiple feature variables instead of single features variables at each split point. Intuitively, the variable selection properties of decision trees can be drastically improved using this feature bagging procedure. Typically, the number of trees k is large, on the order of hundreds to thousands for large datasets with many features. This feature makes Random Forest very attractive for Big Data Analytics. Random forests can be used for robust classification, regression and feature selection analysis.

Naïve Bayes

The Naive Bayes Classifier technique is based on the so-called Bayesian theorem and is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods.

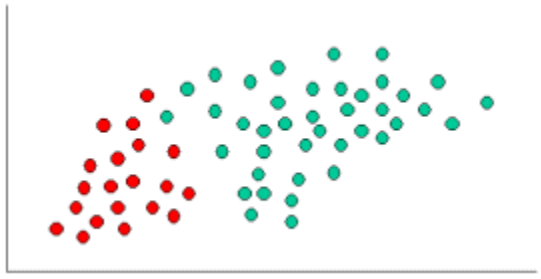


Figure 7: Distribution of data of two classes

To demonstrate the concept of Naïve Bayes Classification, consider the example displayed in the illustration above. As indicated, the objects can be classified as either GREEN or RED. Our task is to classify new cases as they arrive, i.e., decide to which class label they belong, based on the currently exiting objects.

Since there are twice as many GREEN objects as RED, it is reasonable to believe that a new case (which hasn't been observed yet) is twice as likely to have membership GREEN rather

than RED. In the Bayesian analysis, this belief is known as the prior probability. Prior probabilities are based on previous experience, in this case the percentage of GREEN and RED objects, and often used to predict outcomes before they actually happen.

Thus, we can write:

$$\text{Prior probability for GREEN} \propto \frac{\text{Number of GREEN objects}}{\text{Total number of objects}}$$

$$\text{Prior probability for RED} \propto \frac{\text{Number of RED objects}}{\text{Total number of objects}}$$

Since there is a total of 60 objects, 40 of which are GREEN and 20 RED, our prior probabilities for class membership are:

$$\text{Prior probability for GREEN} \propto \frac{40}{60}$$

$$\text{Prior probability for RED} \propto \frac{20}{60}$$

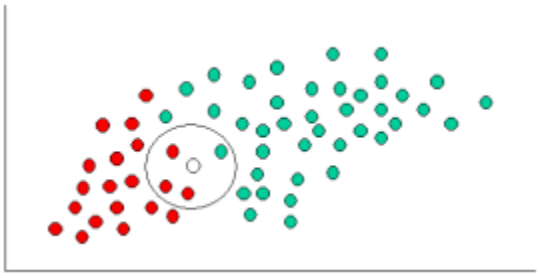


Figure 8: Computing likehood within a region

Having formulated our prior probability, we are now ready to classify a new object (WHITE circle). Since the objects are well clustered, it is reasonable to assume that the more GREEN (or RED) objects in the vicinity of X, the more likely that the new cases belong to that particular color. To measure this likelihood, we draw a circle around X which encompasses a number (to be chosen a priori) of points irrespective of their class labels. Then we calculate the number of points in the circle belonging to each class label. From this we calculate the likelihood:

$$Likelihood\ of\ X\ given\ GREEN \propto \frac{Number\ of\ GREEN\ in\ the\ vicinity\ of\ X}{Total\ number\ of\ GREEN\ cases}$$

$$Likelihood\ of\ X\ given\ RED \propto \frac{Number\ of\ RED\ in\ the\ vicinity\ of\ X}{Total\ number\ of\ RED\ cases}$$

From the illustration above, it is clear that Likelihood of X given GREEN is smaller than Likelihood of X given RED, since the circle encompasses 1 GREEN object and 3 RED ones. Thus:

$$Probability\ of\ X\ given\ GREEN \propto \frac{1}{40}$$

$$Probability\ of\ X\ given\ RED \propto \frac{3}{20}$$

Although the prior probabilities indicate that X may belong to GREEN (given that there are twice as many GREEN compared to RED) the likelihood indicates otherwise; that the class membership of X is RED (given that there are more RED objects in the vicinity of X than GREEN). In the Bayesian analysis, the final classification is produced by combining both sources of information, i.e., **the prior and the likelihood**, to form a posterior probability using the so-called Bayes' rule (named after Rev. Thomas Bayes 1702-1761).

$$Posterior\ probability\ of\ X\ being\ GREEN \propto$$
$$Prior\ probability\ of\ GREEN \times Likelihood\ of\ X\ given\ GREEN$$
$$= \frac{4}{6} \times \frac{1}{40} = \frac{1}{60}$$
$$Posterior\ probability\ of\ X\ being\ RED \propto$$
$$Prior\ probability\ of\ RED \times Likelihood\ of\ X\ given\ RED$$
$$= \frac{2}{6} \times \frac{3}{20} = \frac{1}{20}$$

Finally, we classify X as RED since its class membership achieves the largest posterior probability.

Naive Bayes classifiers can handle an arbitrary number of independent variables whether continuous or categorical. Given a set of variables, X = {x1,x2,..,xd}, we want to construct the posterior probability for the event Cj among a set of possible outcomes C = {c1,c2,c...,cd}. In a more familiar language, X is the predictors and C is the set of categorical levels present in the dependent variable. Using Bayes' rule:

$$p(C_j \mid x_1, x_2, \ldots, x_d) \propto p(x_1, x_2, \ldots, x_d \mid C_j) p(C_j)$$

where p(Cj | x1,x2,x...,xd) is the posterior probability of class membership, i.e., the probability that X belongs to Cj. Since Naive Bayes assumes that the conditional probabilities of the independent variables **are statistically independent** we can decompose the likelihood to a product of terms:

$$p(X \mid C_j) \propto \prod_{k=1}^{d} p(x_k \mid C_j)$$

and rewrite the posterior as:

$$p(C_j \mid X) \propto p(C_j) \prod_{k=1}^{d} p(x_k \mid C_j)$$

Using Bayes' rule above, we label a new case X with a class level Cj that achieves the highest posterior probability.

Although the assumption that the predictor (independent) variables are independent is not always accurate, it does simplify the classification task dramatically, since it allows the class conditional densities p(xk | Cj) to be calculated separately for each variable, i.e., it reduces a multidimensional task to a number of one-dimensional ones. In effect, Naive Bayes reduces a high-dimensional density estimation task to a one-dimensional kernel density estimation. Furthermore, the assumption does not seem to greatly affect the posterior probabilities, especially in regions near decision boundaries, thus, leaving the classification task unaffected.

Underlying distribution of each feature of Naive Bayes can be modeled in several different ways including discrete, normal, lognormal, gamma and Poisson density functions:

Why are we interested in Naïve Bayes? As stated, ADFA-LD dataset shows a great similarity between normal data and attack data. Moreover, it is supposed to favor 'normal' case as it has much more exemplars in the training phase. Thus Naïve Bayes is expected to perform poorly for ADFA-LD data; in other words, it is likely to be an overall weak classifier. However, if it favors any classification as an attack, it is likely to be more robust and can be trusted when other classifiers provide much lesser confidence of an 'attack'.

Combined Classification Methods

Combined Classification Methods use a combination of models to increase accuracy. The scheme combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved composite model M*. **The key to such a scheme is to choose individual classifiers who are very different in nature.**

The popular methods of combination are: 1) Bagging: averaging the prediction/classification over a collection of classifiers**, 2) Boosting**: weighted vote with a collection of classifiers **and 3) Ensemble**: combining a set of heterogeneous classifiers**.**

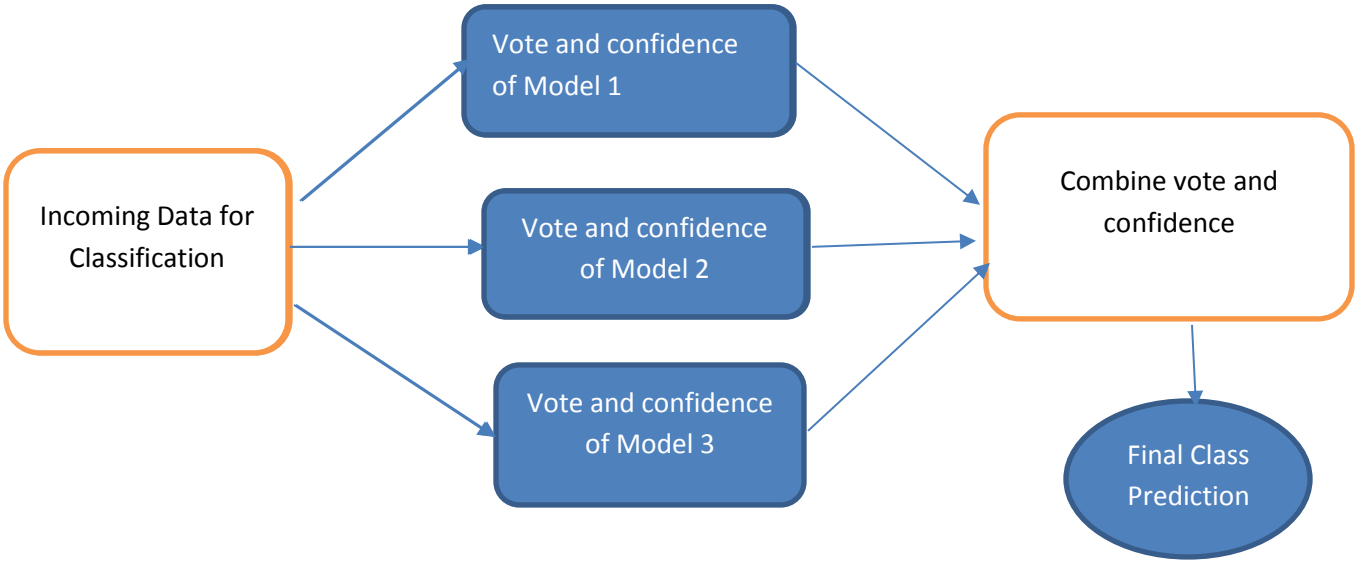


Figure 9: A Combined Classification Scheme

A **Combined Classifier** is similar to diagnosis based on multiple doctors' majority vote. In brief,

- **Training**
  - Each classifier is trained separately.
- **Classificatio**n: classify an unknown sample **X**
  - Each classifier returns its class prediction and confidence value
  - The bagged classifier M* counts the votes as well as confidence and assigns the class with the most votes/confidence to **X**
  - Often provides significantly better performance than that of a single classifier
  - For noisy data in particular provides more robust results

IV. Software and Implementation of Proposed Scheme

Software: Weka is a collection of machine learning algorithms for data mining tasks. **The software package is publicly available for free.** The algorithms can either be applied directly to

a dataset or called from Java code (very suitable for Linux). Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes [29.30]. Appendix-B shows screen shots of Weka to demonstrate it versatility..

Weka also has tools for preprocessing of data as well as feature selection. It can handle missing data as well as a combination of many data types, such as numeric, strings etc. as features. Weka expects input feature vector of each class to be presented using a specific format call *'arff'* format which is quite easy. The plan is to use Weka software. It has been used by researchers and developers all over the world with great performances.

Feature Extraction: Here I will follow the guidelines given in [20, 21, and 31] for ADFA dataset. As already stated [31], the ADFA data set is labeled, and an updated labeled ADFA-LD information is given below. I need cross validation to train and test each classifier. This stage is highly experiment dependent.

Trace Type	Number	Label
Training	833	normal
Validation	4373	normal
Hydra-FTP	162	attack
Hydra-SSH	148	attack
Adduser	91	attack
Java-Meterpreter	125	attack
Meterpreter	75	attack
Webshell	118	attack

Table IV. Summary of ADFA-LD dataset with labels, normal vs. attack

Cross-validation (*k*-fold, where k = 10 is most popular)

1. Randomly partition the data into *k mutually exclusive* subsets , each approximately equal size
2. At *i*-th iteration, use i-th subset D_i as test set and others together as training set

Combination Scheme: At this time, I propose a simple combination scheme. This scheme could change after experimentation. The idea is to illustrate a 'decision mechanism' of a combined classifier. In this scheme, I favor Naïve Bayes for attack as it is likely to favor 'normal' state as there are 5 times more data for the 'normal' scenario. Another important point is I proposing to implement a 2-class system while many anomaly based system

implements its classification as 1-class versus its outlier. At this time, I prefer 2-class modality as 'attack data', though fewer, are available, **and likely to perform better**. The system is continuously monitoring, and gets its observation every *'t'* unit of time (to be determined).

Number of classes: 2 (Attack vs. Non-Attack)

AP = Attack probability (or likelihood) from a classifier (0—1 given by Weka)

NP = Normal probability (or likelihood) from a classifier (0—1 given by Weka)

If Naïve Bayes has higher probability for attack compared to its normal behavior

Attack probability = 0.8 *(AP of Naïve Bayes) + 0.1 * (AP of RF) + 0.1 * (AP of NN)

Normal probability = 0.2 *(AP of Naïve Bayes) + 0.4 * (AP of RF) + 0.4 * (AP of NN)

Else

Attack probability = 0.5 *(AP of RF) + 0.5 * (AP of NN)

Normal probability = 0.1 *(AP of Naïve Bayes) + 0.45 * (AP of RF) + 0.45 * (AP of NN)

Done

If Attack probability > Normal probability,

Decision: Attack is found and System Notified

Else Decision: No Attack is found

Classification Metrics: I need to report performance of the proposed combined classifier. The classification metrics and implementation mode (10 folds cross-validation) are described below.

FP – False Positive

FN – False Negative

TP- True Positive

TN – True Negative

Precision: exactness – what % of tuples that the classifier labeled as positive are actually positive

Precision= TP/(TP+FP)

Recall: completeness – what percentage of positive tuples did the classifier label as positive?

Recall = TP/(TP+FN)

Perfect score is 1.0. There is an inverse relationship between precision & recall

Accuracy: Percentage of exemplars correctly classified

Accuracy= (TP+TN)/(TP+TN+FP+FN)

F-Measure:

F = (2*Precision*Recall)/(Precision+Recall)

IV. Conclusions

The following conclusion are made based on what have been written so far.

1. I have surveyed many papers on HIDS and NIDS that use machine learning.
2. I have surveyed publicly available database for implementing any new machine learning scheme for HIDS. I have concluded that AFDA-LD data base is the optimal choice though KDD-99 can help with training and gaining some insights.
3. I have proposed to use WEKA publicly available machine learning software to implement the combined classifier scheme.
4. I have chosen 3 classifiers, namely, Radom Forest, MLP-NN and Naïve Bayes, to implement the combined classification scheme.
5. I have outlined a preliminary combination scheme utilizing evidence from each classifier.
6. I have discussed how to extract features and difficulty associated with it.

However, I am very optimistic that the proposed scheme is sound and will bear fruits.

References

[1] Brown C, Cowperthwaite A, Hijazi A, SoMayaji A. Analysis of the 1999 DARPA/Lincoln Laboratory IDS evaluation data with NetADHICT. In: IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009. pp.1-7

[2] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defense Applications 2009.

[3] Ficco, M., & Romano, L. (2011, June). A generic intrusion detection and diagnoser system based on complex event processing. In Data Compression, Communications and Processing (CCP), 2011 First International Conference on (pp. 275-284). IEEE.

[4] Ying, L., Yan, Z., & Yang-jia, O. (2010, April). The design and implementation of host-based intrusion detection system. In Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on (pp. 595-598). IEEE.

[5] Gudadhe, M, Prasad, P., & Wankhade, K. (2010, September). A new data mining based network intrusion detection model. In Computer and Communication Technology (ICCCT), 2010 International Conference on (pp. 731-735). IEEE.

[6] Patil, N., Lathi, R., & Chitre, V. (2012, June). Comparison of C5. 0 & CART classification algorithms using pruning technique. In International Journal of Engineering Research and Technology (Vol. 1, No. 4 (June-2012)). ESRSA Publications.

[7] Chen, Y., Hwang, K., & Ku, W. S. (2007). Collaborative detection of DDoS attacks over multiple network domains. Parallel and Distributed Systems, IEEE Transactions on, 18(12), 1649-1662.

[8] Susan Rose Johnson and Anurag Jain, An Improved Intrusion Detection System using Random Forest and Random Projection, International Journal of Scientific & Engineering Research, Volume 7, Issue 10, October-2016)

[9] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque, Random-Forests-Based Network Intrusion Detection Systems, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS, VOL. 38, NO. 5, SEPTEMBER 2008 649

[10] Prakash Chandra, Umesh Kumar Lilhore, & Nitin Agrawal NETWORK INTRUSION DETECTION SYSTEM BASED ON MODIFIED RANDOM FOREST CLASSIFIERS FOR KDD CUP-99 AND NSL-KDD DATASET, International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056, Volume: 04 Issue: 08 | Aug -2017

[11] Ningxin Shi, Xiaohong Yuan, William Nick , Semi-supervised Random Forest for Intrusion Detection Network, MAICS 2017 : The 28th Modern Artificial Intelligence and Cognitive Science Conference, pp. 181-185

[12] Md. Al Mehedi Hasan, Mohammed Nasser, Biprodip Pal1, Shamim Ahmad, Journal of Intelligent Learning Systems and Applications, 2014, 6, 45-52, http://www.scirp.org/journal/jilsa

[13] Christian Bitter, David A. Elizondo and Tim Watson, Application of Artificial Neural Networks and Related Techniques to Intrusion Detection), The 2010 International Joint Conference on Neural Networks (IJCNN), July 2010, place: Barcelona, Spain,

[14] A. Iftikhar, A. B. Abdullah, and A. S. Alghamdi, Application of artificial neural network in detection of dos attacks, Proceedings of the 2nd international conference on Security of information and networks (SIN '09), New York, NY, USA: ACM, 2009, pp. 229-234.

[15] D. Stopel, Z. Boger, R. Moskovitch, Y. Shahar, and Y. Elovici,"Application of artificial neural networks techniques to computer worm detection," in International Joint Conference on Neural Networks (IJCNN), 2006, pp. 2362-2369.

[16] J. Bai, Y. Wu, G. Wang, S. X. Yang, and W. Qiu, "A novel intrusion detection model based on multi-layer self-organizing maps and principal component analysis," in Advances in Neural Networks - ISNN 2006, Lecture Notes in Computer Science. Springer Berlin Heidelberg, May 2006, pp. 255-260.

[17] Moradi M, Zulkernine M.A. Neural network based system for intrusion detection and classification of attacks, Proceedings of the 2004 IEEE international conference on advances in intelligent system-theory and applications, 2004; p. 142-152.

[18] Jiankun Hu, Xinghuo Yu, D. Qiu and Hsiao-Hwa Chen, A Simple and Efficient Hidden Markov Model Scheme for Host-Based Anomaly Intrusion Detection, IEEE Network Magazine, January/February 2009, pp. 42-247

[19] Wael Khreich, Eric Granger, Robert Sabourin and Ali Miri, Combining Hidden Markov Models for ImprovedAnomaly Detection, IEEE ICC 2009 proceedings

[20]G. Creech and J. Hu, 'ADFA IDS Dataset (Readme)', University of Arizona Artificial Intelligence Lab, 2016. [Online]. Available: https://s3-us-west-2.amazonaws.com/adfa-ids/readme.txt

[21]G. Creech and J. Hu, 'Generation of a New IDS Test Dataset: Time to Retire the KDD Collection', IEEE Wireless Communications and Networking Conference (WCNC): Services and Applications, 2013, pp. 4487-4482. Available: http://ieeexplore.ieee.org/document/6555301/

[22] G. Creech, 'Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks', Ph. D. Thesis, The University of New South Wales, 2013. [Online]. Available: http://unsworks.unsw.edu.au/fapi/datastream/unsworks:11913/SOURCE02?view=true

[23] G. Creech and J. Hu.A Semantic Approach to Host-based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns. IEEE Transactions on Computers, Volume: 63 Issue: 4, pp. 807 – 819

[24] KDD Cup 1999 Data, The UCI KDD Archive, Information and Computer Science, University of California, Irvine, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.htm, Accessed 12/07/2017

[25] Adetunmbi A.Olusola, Adeola S.Oladele and Daramola O.Abosede, Analysis of KDD '99 Intrusion Detection Dataset for Selection of Relevance Features, Proceedings of the World Congress on Engineering and Computer Science (WCECS-2010), Vol I, October 20-22, 2010, San Francisco, USA

[26] Murtaza S. S, Khreich W, Hamou-Lhadj A, Couture M. A Host-based Anomaly Detection Approach by Representing System Calls as States of Kernel Modules. In:24th International Symposium on Software Reliability Engineering, Pasadena, 2013, p.431-440.

[27] Computer Science Department, "University of New Mexico Intrusion Detection Dataset. [Online]. Available: http://www.cs.unm.edu/_immsec/systemcalls.htm. Also, https://www.cs.unm.edu/~immsec/data-sets.htm . Accessed 12/07/2017.

[28] R Lippmann , An introduction to computing with neural nets, IEEE Assp magazine, 1987

[29] WEKA software, Machine Learning, http://www.cs.waikato.ac.nz/ml/weka/, The University of Waikato, Hamilton, New Zealand.

[30] Leo Breiman and Adele Cutler, Random forests, http://statwww.berkeley.edu/users/breiman/RandomForests/cc_home.htm, University of California, Berkeley, CA, USA.

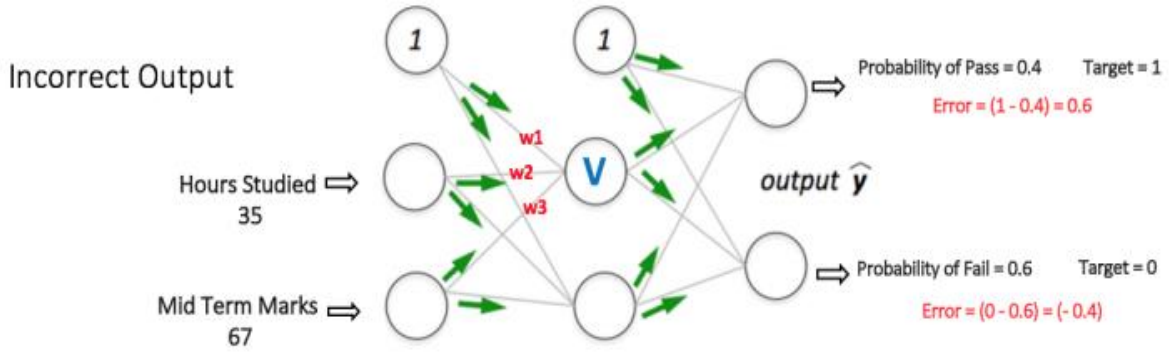
[31] Xie M, Hu J, Slay. Evaluating Host-based Anomaly Detection System: Application of The One-class SVM Algorithm to ADFA-LD,". [Online]. Available:http://icnc-fskd.xmu.edu.cn/doc/invitedSession/FSKD_5_Miao.pdf. Accessed, November 5, 2014.

Appendix-A: Neural Net Back Prop

There are two input – Hours Studied and Mid Term Marks, and two output – Probability of Pass and Probability of Fail

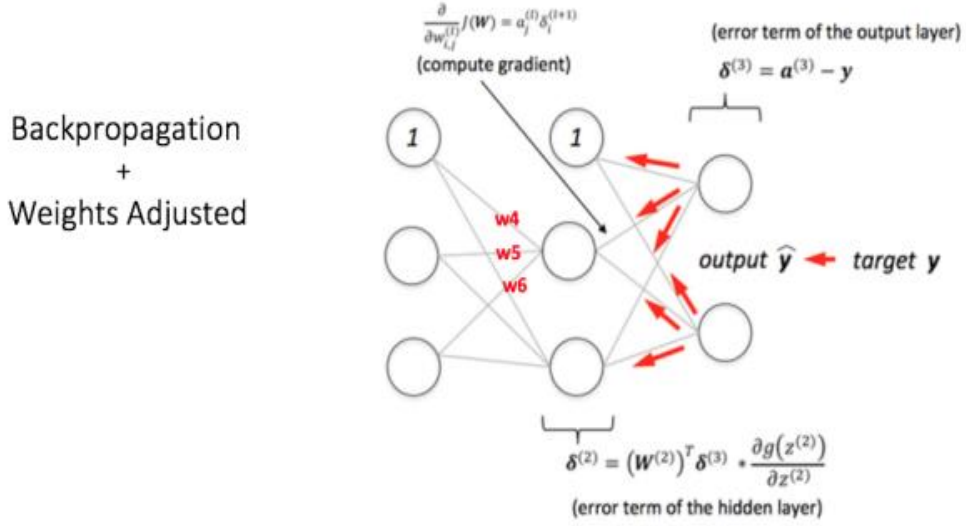
Forward Pass

This pass does not update weights, just computes value at each node and propagated to output (see forward green arrows). Finally, I have values at output nodes which may or may not match the expected values at output.



Backward Pass

Errors computed at output nodes (expected value – computed value) are propagated back to update weights of all layers -- input layer to first hidden layer, hidden layer to hidden layer if there are more than 1 hidden layer, and last hidden layer to output layer (see backward red arrows).



Appendix-B

The WEKA classifier is trained on five classes of images. This is a simple experiment to show WEKA's capability, and is not part of my proposal. For each image, 140 HoG (histogram of gradients) features are computed. There are 100 images per class, and 10 ten-fold cross validation is used. The screen shot shows Weka's performance for these images. WEKA software can also help selecting the most important features.

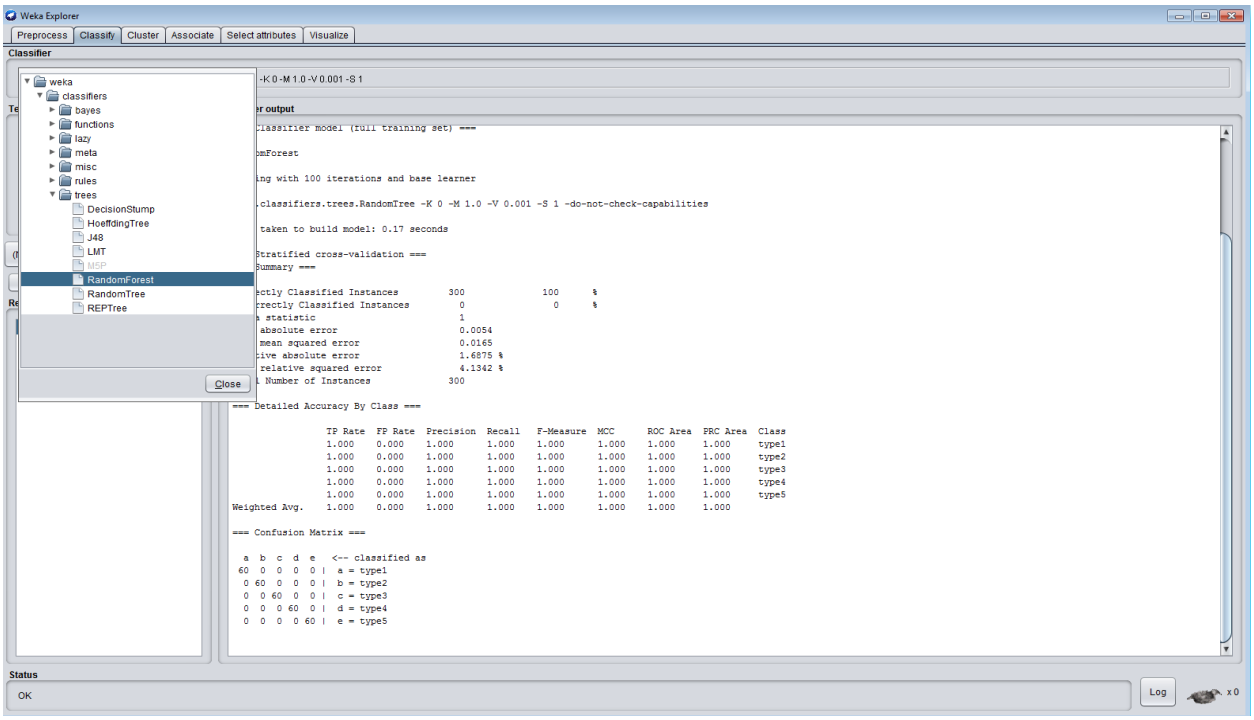


Figure Appendix-B-1: Screen snap shot of Weka classifiers showing performance metrics

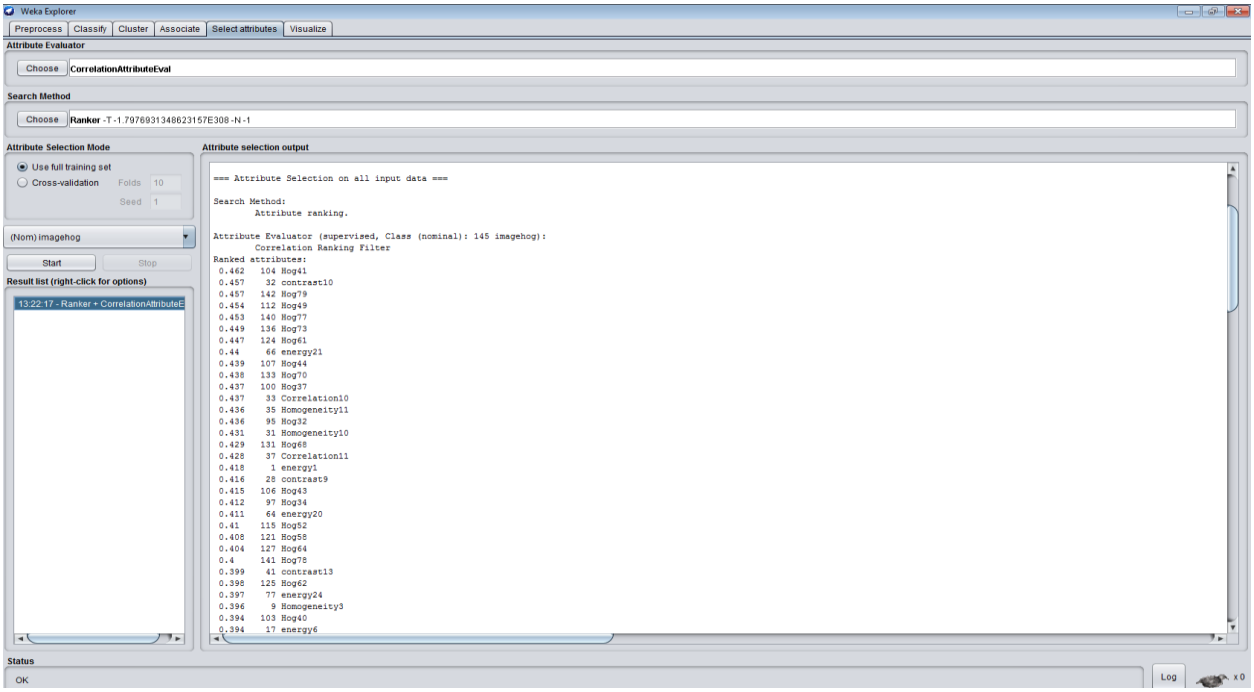


Figure Appendix-B-2: WEKA ranks features for selecting important features (from 140 features with name)