Assessing IP Weight Metrics for Cloud Intrusion Detection using Machine Learning Techniques

by

Ruiqi Hu
B.Sc, Peking University, 2013

A Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Assessing IP Weight Metrics for Cloud Intrusion Detection using Machine Learning Techniques

by

Ruiqi Hu
B.Sc, Peking University, 2013

Supervisory Committee

Dr. Issa Traoré, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Samer Moein, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Despite the growing popularity of cloud computing, security is still an important concern of cloud customers and potential adopters. Cloud computing is prone to the same attack vectors as traditional networks, in addition to new attack vectors that are specific to cloud platforms. Intrusion Detection Systems (IDS) deployed in the cloud must take into account the specificity of the underlying threat landscape as well as the architectural and operational constraints of cloud platforms. In this project, an IDS that utilizes IP weight metrics for feature selection is implemented. Additionally, this system is tested with different supervised classification models and evaluated on a cloud intrusion dataset. In comparison with the results under conventional network environment, we conclude that the performance of IDS against cloud intrusions is promising, however, other developments such as unsupervised intrusion detection techniques and extra data preprocessing stages should be researched for the best practice of the system.

Contents

List of Tables

List of Figures

List of Acronyms

IDS Intrusion Detection System

IP Internet Protocol

IPS Intrusion Prevention System

ISOT Information Security and Object Technology

CID Cloud Intrusion Dataset

PCA Principal Component Analysis

DoS Denial of Service

TCP Transmission Control Protocol

UDP User Datagram Protocol

SVM Support Vector Machine

KNN K-Nearest Neighbour

CV Cross Validation

TPR True Positive Rate

FPR False Positive Rate

FNR False Negative Rate

ROC Receiver Operating Characteristic

AUC Area Under roc Curve

DARPA Defense Advanced Research Projects Agency

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor Dr. Traoré, who provided me valuable guidance, insightful advice and numerous knowledge throughout my graduate study.

Besides my supervisor, I would like to thank Abdulazi Aldribi for helping me with my project, Dr. Samer Moein for acting as my supervisory committee.

Lastly, my deepest gratitude goes to my parents and my roommate who constantly support me when I am in need. I'm also grateful to many of my friends who have brought me countless joy, especially Beier Luo, Duo li and Jialing Geng for their generous help.

DEDICATION

To my parents

Chapter 1

Introduction

1.1 Background

With the huge amount of data being transmitted via the internet every day, security has become one of the most critical issues of networking. The goal of network security is to guarantee that any useful data, either shared on networks or stored in connected computers, will not be compromised in terms of confidentiality, integrity, non-repudiation, and availability [1]. Among a variety of techniques that are proposed to protect data, intrusion detection is one major research topic.

An Intrusion Detection System (IDS) is an automated alarm system which scans network traffic and/or system activity data and searches for abnormal activities to notify the administrator. It is usually passive, since it doesn't actively prevent the attacks but only monitors and reports to the system. Under certain circumstances, however, it may be able to take actions against the anomalous traffic such as blocking the users or Internet Protocol (IP) addresses. Systems with such capability are known as Intrusion Prevention System (IPS).

According to the approach it takes to accomplish the task of intrusion detection, IDS can be categorized into 3 groups:

- Signature-based IDS: IDS which observes IP packets in the network traffic and looks for distinct signatures that appeared in past attacks. This is the same procedure many anti-virus programs take to prevent, detect and remove mal-

wares. As accurate as it can be in detecting known threats, this type of IDS performs poorly when dealing with recent attacks whose signatures haven't been uncovered yet.

- Anomaly-based IDS: This type of IDS is based on the assumption that malicious behaviors significantly deviate from the usual ones. It first studies a collection of traffic patterns and establishes some standards for "normal" behaviors, then it scans the network traffic or system logs. A certain network activity is considered as intrusion if it exceeds the threshold of "normal". Anomaly-based IDS is capable of detecting unknown attacks, but has generally high false alarm rate.
- Hybrid IDS: An intrusion detection system that combines anomaly-based and signature-based IDS to make use of multiple techniques.

In real world applications, most of the deployed IDS are signature-based IDS, nevertheless, anomaly-based IDS is gaining increasing attention nowadays [2], especially those that are based on various machine learning models [3]. Current machine learning techniques developed for IDS approximately fall into two broad categories: supervised learning (e.g. classification) as well as unsupervised learning (e.g. clustering and outlier detection):

In supervised learning techniques,

- We train a classifier using data of normal usage and prior attacks,
- The classifier can be retrained to include more known attacks,
- IDS can detect unknown attacks if perfect model of normality is available.

While in unsupervised learning techniques,

- We use unlabeled data as input,
- We attempt to separate the anomalous instances from the regular instances,
- After separating the two classes, we then train a traditional anomaly detection algorithm on the clean data.

As discussed above, the major difference between supervised and unsupervised learning techniques is the label information. When deciding which technique should be

applied to an IDS, we need to consider the specific problem we are handling, in particular:

- How hard it is for us to obtain the labels of the dataset? Analyzing network traffic manually is tedious and error-prone, as a result, the training set is usually quite small compared to all the data available. In some extreme cases, it may even be impractical to assign an explicit label to a data example.
- How well do the labels cover all possible threats? The selection of training data has a huge impact on the ultimate performance of the IDS. As attacks evolve, it is likely that some attack examples didn't appear in the training set, which may cause a cutback in the accuracy of the IDS.

There are many other trade-offs between supervised and unsupervised learning techniques in practical use, which are further investigated in [4]. Unsupervised learning is beyond the scope of this report. For this project, we will only focus on supervised learning models in the IDS.

Recent years have witnessed a growing interest in using cloud computing for various computational needs. The attractiveness of cloud computing lies in the flexible resource model it provides. Despite the enthusiasm for cloud computing, security concerns appear to be a major hurdle for its adoption.

Cloud computing faces the same threats as conventional computing, as well as several new threats that are specific to the underlying architecture and resource model. Therefore it has been advocated to develop protection models and tools that are more appropriate to the cloud threat landscape. In this regard, the goal of the project is to study how some of the models used for intrusion detection in conventional networks fare when applied to cloud intrusion dataset. Specifically, we revisit a feature model named IP weight metrics, that was introduced by Wei Lu in his PhD thesis for conventional network intrusion detection [5]. We apply the aforementioned feature model to a cloud intrusion dataset, and investigate the performance achieved in detecting intrusions for different machine learning models.

1.2 Data Source

The data used in this project is from the Information Security and Object Technology (ISOT) Cloud Intrusion Dataset (CID), collected by the ISOT research lab at the University of Victoria [6], in partnership with Compute Canada [7]. The overall dataset consists of 8TB of data collected in a production cloud. In this project, we use a small subset (16MB) of the dataset to study the aforementioned IP weight feature model.

1.3 Project Contributions

This project aims at building an intrusion detection framework with IP weight metrics, as well as evaluating its performance against cloud intrusions under different machine learning models, regarding both the accuracy and ability to detect attacks. Below are some contributions of the project:

1) Study the effectiveness of the IP weight feature model and several commonly used machine learning models for cloud intrusion detection.

2) Implement an IDS in python and experiment with the aforementioned models.

3) Compare the performance of IDS in cloud computing environment with that in conventional network environment.

1.4 Report Outline

This report is organized as follows: Chapter 2 introduces the IP weight metrics used for feature extraction. Background information on five popular classification models are presented in chapter 3. The implementation of IDS and the evaluation of its performance are included in chapter 4. Chapter 5 makes concluding remarks and discusses future work.

Chapter 2

Feature Selection Using IP Weight Metrics

In this project, we collect and analyze IP packets in the network traffic to filter abnormal behaviors. However, IP packets contain a huge amount of information and it is not feasible to apply them all to the detection system. Instead, we need to extract a group of representative features that best describe its behavior. One popular way to achieve this is the Principal Component Analysis (PCA), which is widely used in areas like facial recognition to transform the raw data into limited primal features. For our problem, we choose to select features using the IP weight metrics proposed by Wei Lu [5]. The IP weight metrics is a set of functions that take in a flow of IP packets and output a sequence of numerical values representing how anomalous the flow is. It not only is simple and fast, but also effectively reduces the original feature space to a four-dimensional space, which is as good as the PCA.

For any given IP packet p, we can obtain many useful information based on its structure. Specifically, the following 13 fields shown in Table 2.1 [5] are used in the IP weight metrics, and thus will be our main focus.

Table 2.1: Basic features for an ip packet

Representation	**Meaning**
t	time stamp corresponding to the appearing time of the packet in a certain time window
dip	destination IP address; this usually corresponds to the address of a host we want to protect
dp	destination port
sip	source IP address
sp	source port
ihl	length of IP header for the IP packet
pktl	packet length including header and data
ident	an integer that identifies the current data in a packet, which can be used to piece together data fragments
fragoff	the offset of the IP packet indicating the position of the fragments data relative to the beginning of the fragment data in the original data
pro	upper-layer program receiving the incoming IP packets after IP processing is complete
thl	length of TCP header
seq	data location of the TCP segment
ack	number of data received by the destination host
traffic_{in}	the appearing frequency of packets flowing to the destination IP address over the observation time window
traffic_{out}	the appearing frequency of packets outgoing from the destination IP address over the observation time window

A packet flow is defined as a sequence of IP packets flowing to a certain destination in a specific period of time. We will denote it by a 6-dimensional vector $f =< g, t, \delta_t, dip, nop, nodp_{max} >$, in which

- g is the group of packets.
- t means the start time of the observation.
- δ_t is the observation time frame.
- dip stands for the destination IP address of the flow.
- nop denotes the number of packets in the flow.
- $nodp_{max}$ refers to the maximum number of incoming packets through any destination port in the flow.

IP weight metrics utilize four functions to map the initial feature set to numerical values along four dimensions called frequency, randomness, structure and load. A network flow's IP weight value is directly proportional to the degree of its abnormality, in other words, the larger the IP weights are, the more irregular the network flow is [5]. For simplicity, in the rest of the chapter, we use ipw_{freq}, ipw_{ran}, ipw_{str}, ipw_{load} to represent these 4 components of IP weight measures respectively.

2.1 Frequency-based IP Weight

Assume the frequency of packets flowing to a destination IP address during a time frame δ_t is x_1, and the maximum frequency of packets across all destination ports during δ_t is x_2, then we have

$$x_1 = \frac{nop}{\delta_t} \tag{2.1}$$

$$x_2 = \frac{nodp_{max}}{\delta_t} \tag{2.2}$$

Both x_1 and x_2 with a large value are considered to be a signal for potential threats. The frequency-based IP weight is defined as follows:

$$ipw_{freq}(f) = (x_1 - x_2 + {x_1}^{-1}{x_2}^2)\frac{x_2}{x_1} \tag{2.3}$$

2.2 Randomness-based IP Weight

Entropy is a widely used measure for assessing the uncertainty associated with a random variable. Given a packet flow f during δ_t, the entropy of its source IP addresses, source ports and destination ports are calculated as below:

$$H_{sip}(f) = -\sum_{sip} p(sip)\log_2 p(sip) \tag{2.4}$$

$$H_{sp}(f) = -\sum_{sp} p(sp) \log_2 p(sp) \tag{2.5}$$

$$H_{dp}(f) = -\sum_{dp} p(dp) \log_2 p(dp) \tag{2.6}$$

Then we simply choose the maximum value among the three as the randomness-based IP weight:

$$ipw_{ran}(f) = \max(H_{sip}(f), H_{sp}(f), H_{dp}(f)) \tag{2.7}$$

2.3 Load-based IP Weight

For a network flow f with destination IP address dip during δ_t, let $\mathit{traffic}_{in}$, $\mathit{traffic}_{out}$ be the frequency of the incoming packets to dip and outgoing packets from dip over δ_t correspondingly. These two values are usually about the same, while under scenarios like Denial of Service (DoS) attacks, there is a significant difference between $\mathit{traffic}_{in}$ and $\mathit{traffic}_{out}$. Load-based IP weight defines the load balance of the host and is evaluated by the ratio of $\mathit{traffic}_{in}$ and $\mathit{traffic}_{out}$:

$$ipw_{load}(f) = \frac{\mathit{traffic}_{in}}{\mathit{traffic}_{out}} \tag{2.8}$$

2.4 Structure-based IP Weight

There are certain structural rules that normal IP packets would satisfy according to computer networks theory. Based on these, Table 2.2 [5] defines a set of basic rules which often hold true between any two packets linked to the same Transmission Control Protocol (TCP)/User Datagram Protocol (UDP) connection:

Table 2.2: Rule-base for packets in the same connection

Index	Rule
1	*if* $ident_1 = ident_2$, *then* $fragoff_1 \neq fragoff_2$
2	*if* $ident_1 > ident_2 + 1$, *then* $seq_1 \geq seq_2$ *and* $ack_1 \geq ack_2$
3	*if* $ident_2 > ident_1 + 1$, *then* $seq_2 \geq seq_1$ *and* $ack_2 \geq ack_1$
4	*if* $ident_1 = ident_2 + 1$, *then* $seq_1 = seq_2 + pktl_2 - ihl_2 - thl_2$ *and* $ack_1 \geq ack_2$
5	*if* $ident_2 = ident_1 + 1$, *then* $seq_2 = seq_1 + pktl_1 - ihl_1 - thl_1$ *and* $ack_2 \geq ack_1$

Given a packet flow f during δ_t and any two packets $p_1, p_2 \in f$, we know for sure that if these two packets are associated with the same connection and contravene the above rules, then the traffic is irregular. In other cases, however, we can't make a decision about the abnormality of the network traffic.

To determine whether a pair of packets are associated with the same connection, we notice the fact that two IP packets in the same TCP/UDP connection definitely have the same source IP addresses, source ports, destination IP addresses, destination ports as well as protocols. This condition is necessary but not sufficient, therefore if two IP packets break the condition, then they must not belong to the same connection. As for those that comply with the condition, it is proven that the difference between the identification number of the two packets is inversely proportional to the possibility that these two packets belong to the same connection, which means that the smaller the difference is, the more likely two packets are to be in the same connection [5].

Taken all of the above discussion into consideration, the structure-based IP weight is defined as:

$$ipw_{str}(f) = \frac{1}{nop(nop-1)} \sum_{p \in f} \sum_{q \in f, p \neq q} \varepsilon_{pq} e^{-|ident(p) - ident(q)|} \tag{2.9}$$

where

- $\varepsilon_{pq} = 1$ if packets p and q satisfy the same connection condition while contravening the rule base.
- $\varepsilon_{pq} = 0$ otherwise.

Chapter 3

Classification Models

In this chapter, we outline five classification algorithms that are widely used in machine learning. The theory of each algorithm as well as its advantages and disadvantages are presented. In the next chapter, we will implement each one of the five algorithms and compare their results.

3.1 Naive Bayes Classifier

Naive Bayes classification is a simple and fast technique based on Bayesian theorem. It is called "naive" because it assumes that the features of data are independent with each other, which means that each feature individually contributes to the possibility that the data sample belongs to a certain class.

In probability theory, Bayesian theorem is used to calculate posterior probability $P(A|B)$ from prior probability $P(A)$, $P(B)$ and $P(B|A)$. The equation is shown below:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.1}$$

Where,

- $P(A|B)$ is the conditional probability of A given B.

- $P(A)$ is the prior probability of A.
- $P(B|A)$ is the conditional probability of B given A.
- $P(B)$ is the prior probability of B.

Applying Bayesian theorem, if we are given a certain data sample X and classes C_1, C_2, C_3, Naive Bayes classifier calculates the posterior probability $P(C_1|X)$, $P(C_2|X)$, $P(C_3|X)$ for every class, then it makes a prediction about the class that has the highest probability. Below is an example inspired from [8]:

Suppose we have a group of data samples, each sample contains a feature "weather" (has 3 possible values: sunny, overcast or rainy) and a label "play" (has 2 possible values: yes or no, referring to whether or not a player should play) as depicted by Table 3.1:

Table 3.1: Sample training set

Weather	**Play**
Overcast	Yes
Sunny	No
Sunny	Yes
Rainy	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Overcast	Yes
Sunny	No
Overcast	Yes
Rainy	No

Now, we have a new sample whose weather feature is sunny and we are trying to decide which class of "play" it should fall into. From the table above, we get $P(Sunny) = \frac{5}{14}, P(Yes) = \frac{9}{14}, P(Sunny|Yes) = \frac{3}{9}$, hence $P(Yes|Sunny) = P(Sunny|Yes) * P(Yes)/P(Sunny) \approx 0.64 > 0.5$, i.e. it should be labeled as "Yes".

Naive Bayes classifier is easy to implement and very powerful, especially for large datasets. Even with the "naive" assumption of independence, it is popular in various

modern applications such as text classification/ spam filtering/ sentiment analysis, recommendation systems and real time prediction. To summarize, Naive Bayes classifier:

- is straightforward and runs fast, and can also handle multi-class prediction;
- outperforms many other complex classifiers when underlying assumption holds, while requiring less training data;
- works better for discrete features than continuous ones, since the normal distribution for continuous features is a very strong assumption and limits the performance of the classifier.

Nevertheless, we should notice that:

- if discrete feature in the test set has a new value that didn't appear in the training set, the classifier won't be able to predict the corresponding data sample since the probability is 0. Smoothing technique can be applied to solve this;
- in reality, the assumption of independence is idealistic. Almost all the features we can obtain are somehow correlated.

3.2 SVM

A Support Vector Machine (SVM) is a classifier that makes predictions based on decision boundaries drawn by hyper-planes. A hyper-plane is a frontier that separates data from different classes. In a two-dimensional feature space, the hyper-plane is simply some line that cuts a plane into distinct parts for each class respectively. SVM first creates a n-dimensional space (where n is the size of feature set) and maps each data to a point in the space, whose coordinates are the values of corresponding features. Then it tries to find the "optimal hyper-planes" that segregate the classes. In SVM, "optimal hyper-planes" are defined as the planes which differentiate the classes with the largest margin possible. In simple terms, they maximize the distances between the nearest data points in all classes.

Suppose we are given some data which have already been labeled as blue triangle or red square. A scatter-plot of the data is shown in Figure 3.1, and our goal is to find

a separating line for the two classes:

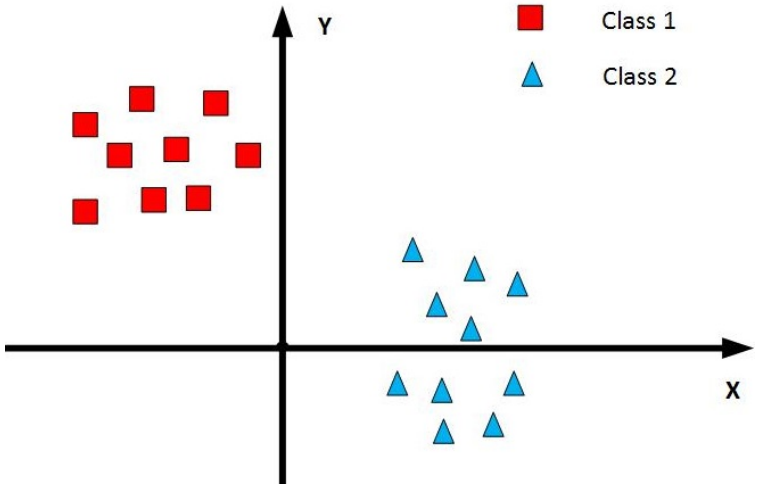

Figure 3.1: Find a line that separates class 1 from class 2

It is pretty obvious that the line should be something similar to the green line depicted in Figure 3.2. Given the separating line and a new data point, if it is on the left side of the line, it should be assigned to class 1, and if it is on the right side, it belongs to class 2.

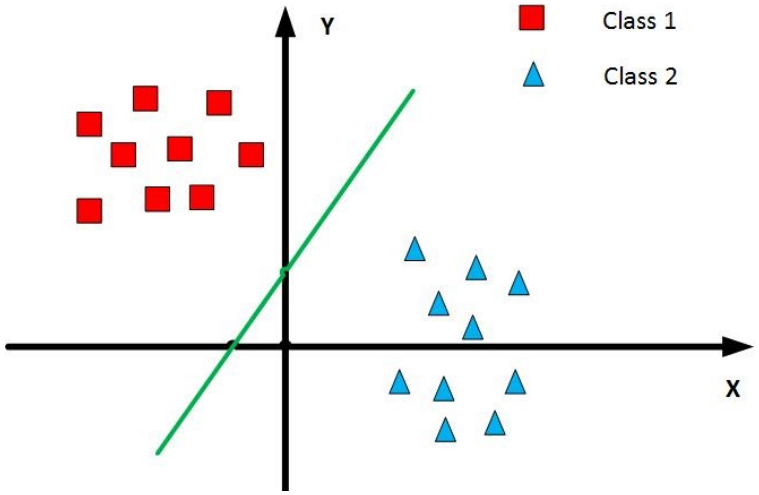

Figure 3.2: A sample partition

However, in many cases, we may not be capable of finding a straight line (linear hyper-plane) that separates the two classes. For example, consider the scenario in Figure 3.3:

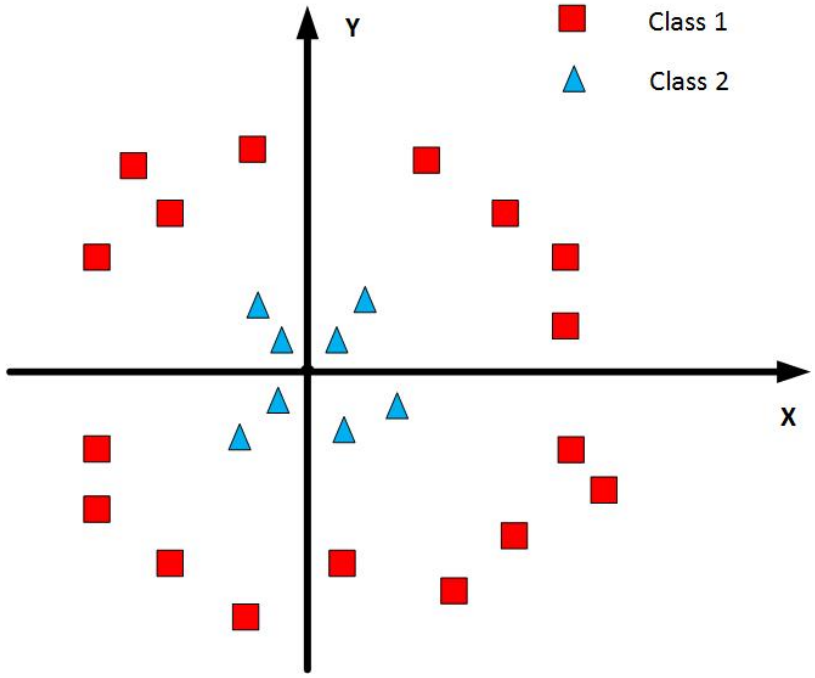

Figure 3.3: No straight line can divide the two classes

We can solve this problem by data transformation. Suppose we introduce another feature $z = x^2 + y^2$ and add a new z-axis as the third dimension. In this case, the new feature may be interpreted as the square of the distance between the data point and origin. Now let's plot in the new dimension, and we'll observe a clear separation as illustrated in Figure 3.4.

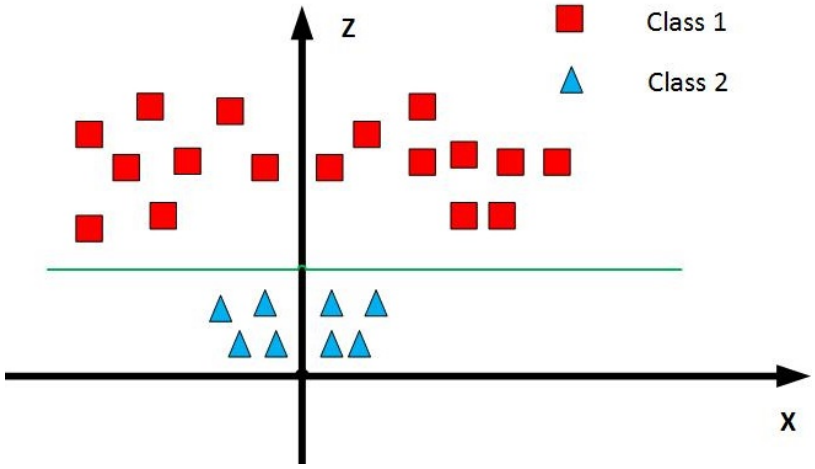

Figure 3.4: Scatter-plot of $z - x$ axis

When we look at the hyper-plane in the original input space, it projects to a round boundary as shown in Figure 3.5. In SVM, we do not need to derive the transformation manually. Instead, there are embedded functions called kernels, which automatically convert the original input space to a higher dimensional space, and therefore make the data separable. SVM is particularly useful in non-linear separation problems, where it does sophisticated transformations and figures out how to separate the labeled data all by itself.

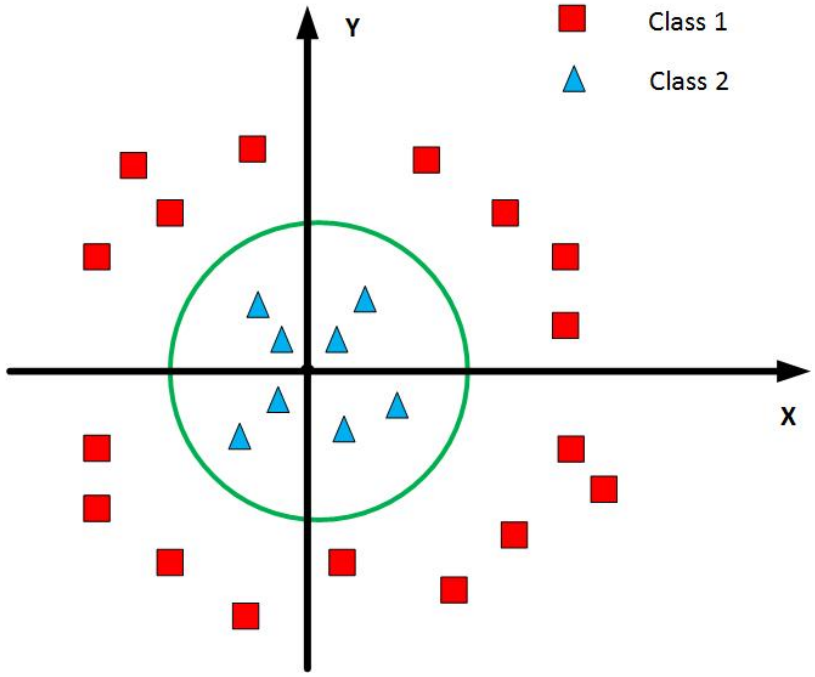

Figure 3.5: Hyper-plane maps to a circle in $x - y$ plane

According to the specific problem we are faced with, it is not uncommon for us to tailor the classifier to best suit our needs. SVM takes in a number of tuning parameters, which can be altered to effectively improve the classifier's accuracy and efficiency. We will introduce 3 important parameters "Kernel", "γ" and "C" in the rest of this section.

- Kernel: Common choices of kernel include linear kernel ("linear"), radial basis function kernel ("rbf"), polynomial kernel ("poly") etc. The last two kernels derive non-linear hyper-planes.

- γ: Defines which part of data samples are more important in calculation for

the optimal separation. Small γ means that the points far away have higher impact, which may cause misclassification on data points close to the hyper-plane. Large γ means that the points closer have higher impact, which may result in over-fitting.

- C: Penalty parameter for misclassifying a data sample. When the cost of error C is high, SVM will settle for a partition with smaller margin to achieve higher accuracy on the training set. In contrast, when C is low, SVM will sacrifice accuracy for larger margin.

In practice, we should try different combinations of the tuning parameters in order to build a SVM with decent precision and reasonable speed while avoiding over-fitting. In conclusion, SVM:

- is very effective when classes are separated with clear margin;
- is efficient in high dimensional space, especially when the dimensionality is higher than the size of data set;
- requires less memory as only a subset of training points are used in the decision function.

In addition, we should consider the following limitations of SVM while choosing our classification model:

- The training time of SVM increases drastically as the size of training set grows;
- Under certain circumstances where the classes overlap with each other, SVM won't be able to produce satisfying results.

3.3 KNN Classifier

The K-Nearest Neighbour (KNN) classifier is a simple classifier based on similarity measure. To predict a new data point, the classifier takes a majority vote of the K nearest surrounding points (K is a positive integer, usually very small compared to the size of the whole data set). If $K = 1$, then the data point is simply assigned to the same class as its closest neighbour.

Let's see an example of KNN classification as depicted by Figure 3.6. Given all the labeled data points, our task is to determine which class the new example (represented by green circle) belongs to. If we choose $K = 3$, then it will be labeled as red square since there are 2 squares as opposed to 1 triangle among the 3 neighbours. Similarly, if we choose $K = 9$, it will be assigned to the first class of blue triangles (6 triangles versus 3 squares inside the circle).

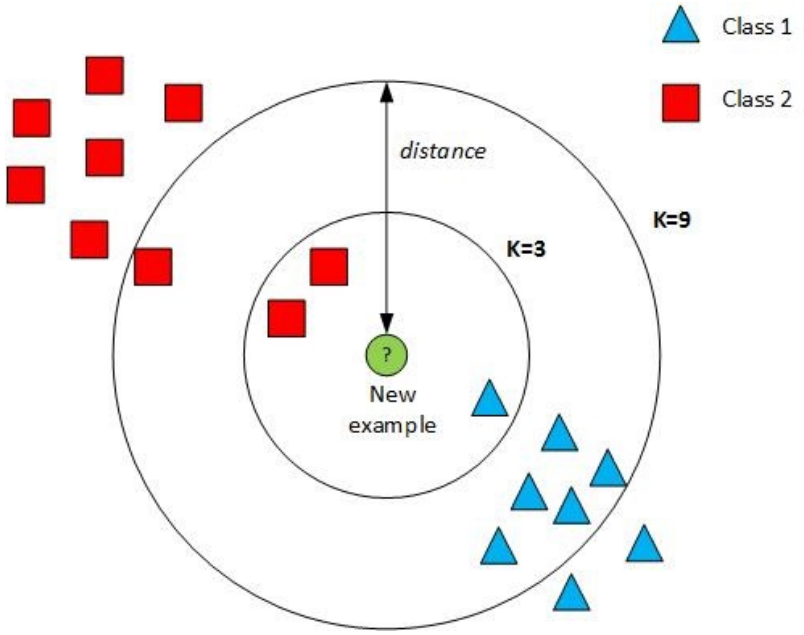

Figure 3.6: KNN classification with different K values

K controls the trade-off between running time and accuracy of the KNN model, and needs to be tuned to find the best value. Besides, if the number of classes is even, we usually choose an odd number of K to avoid a tie and vice versa. Another parameter that can be modified is the distance function. Euclidean distance is commonly used for numerical variables. It is calculated as the square root of the sum of the squared differences between 2 vectors $X =< X_1, X_2, ..., X_n >$ and $Y =< Y_1, Y_2, ..., Y_n >$:

$$Euclidean\ \ Distance(X, Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2} \tag{3.2}$$

Other typical distance functions include:

- Hamming Distance: Mostly used to measure the distance between binary vectors;
- Manhattan Distance (also known as City Block Distance): Takes the sum of the absolute difference between 2 vectors as the distance;
- Minkowski Distance: Generalization of Euclidean and Manhattan distance.

There might be other distance metrics that are more appropriate for certain types of problems. Varying the K value as well as the distance measure together helps us find the best KNN model.

The distance measure is the basis of KNN classifier and it suffers from the "curse of dimensionality". Intuitively, KNN classifier will have the same problem when the feature space is very large. Despite of its competitive performance in low dimensional cases, as the feature space expands, all data points will spread out through the entire space and those who have similar features might end up being far away from each other. The distance function becomes less helpful and the KNN model will inevitably break down.

KNN classification is a competent algorithm frequently used in credit ratings, political science, and image recognition. To sum up, the KNN classifier:

- is robust to outliers in the data set;
- does not require the data to have certain underlying distribution (unlike the Naive Bayes classifier), which is very practical since most of the real-world data don't obey general assumptions;
- is simple and gives highly accurate results.

On the other hand, limited by the fundamental distance measure and the fact that it does not actively "learn" to classify, the KNN classifier:

- is affected by irrelevant features and the scale of feature set;
- is computationally expensive with respect to both time and memory;
- has a negligible training phase but a costly prediction phase.

3.4 Decision Tree Classifier

Decision tree classifier, like the name implies, is built on a tree-shaped model (called decision tree) that shows all possible results from decision chains. A decision tree is formed by 4 basic elements: root node, internal nodes, branches and leaf nodes. Every decision tree must have one and only one root node, acting as the starting line. The root node, along with all internal nodes in the decision tree, contain certain rules to be met, while leaf nodes contain decision information. By connecting nodes with directed paths, the sequence of branches from the root node to a leaf node shows the process of a specific decision being made. Both the root node and internal nodes have at least 2 branches stretching out from it. The number of branches from a node is equal to the total number of possible answers for the problem in that node.

Figure 3.7 illustrates an example of a decision tree designed for credit card applications. For demonstration purposes, we will assume that all middle-aged applicants should be approved, while young applicants will only be approved if they are students, and senior applicants will only be approved if their credit scores are good enough.

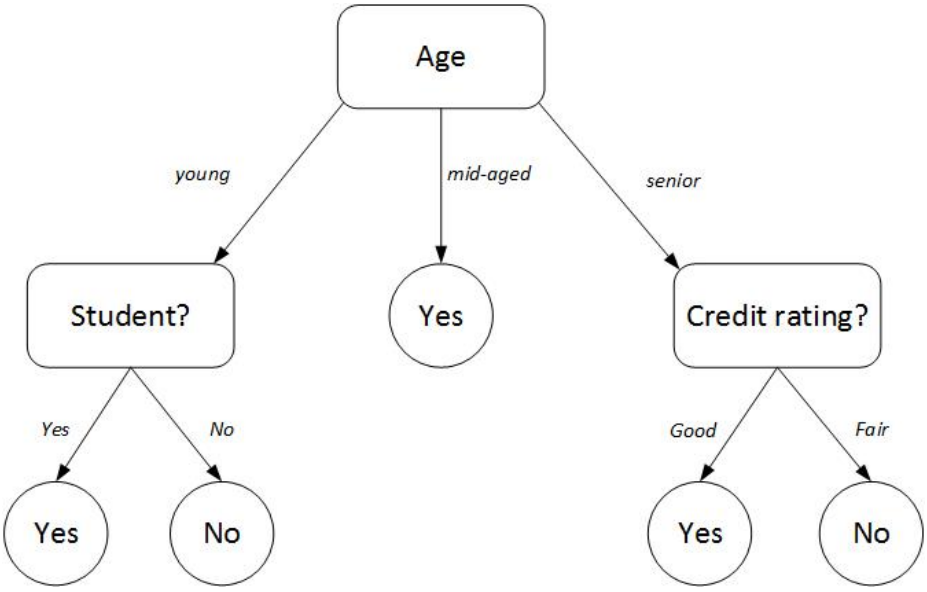

Figure 3.7: Example of a decision tree

Given any application, it is now very easy for us to make a decision on whether it should be approved. Starting from the top, answer a series of splitting questions, and

follow the answer branches all the way down to a leaf node, then the leaf node will tell us the decision.

By creating a decision tree, the decision tree classifier recursively splits the data set into multiple subsets in accordance with the most significant differentiator over all available features, and terminates when either all the subsets are homogeneous (in other words, all data samples in the subset have the same class membership) or certain criteria are fulfilled. Figure 3.8 is a demonstration of a simple decision tree classifier on the scatter-plot for two classes represented by red squares and blue triangles. The classifier first splits the data samples according to their x features. If $x < x_1$, then the sample will be labeled as class 1. For the rest of the samples, it repeatedly splits them based on the y features. If $y < y_1$, the data sample will be classified as the second class, otherwise it will be assigned to the first class.

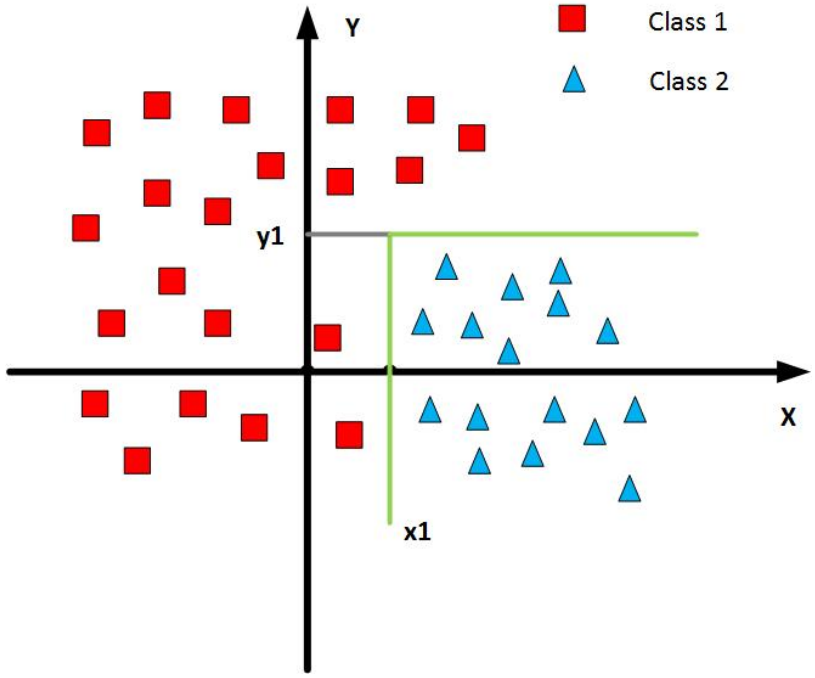

Figure 3.8: Demonstration of a Decision Tree classifier

The strategy of making splits has huge effect on the tree's performance. Decision tree classifier aims at finding the split that generates the most homogeneous subsets. Classic measures for impurity such as gini index, chi-square and entropy are the general metrics for evaluating a split.

The major drawback of a decision tree classifier is its tendency to over-fit when there are no constraints. Worst case scenario, it winds up making a leaf node for every data sample in the training set. There are two ways to prevent a decision tree from becoming overly-complex: tuning parameters and pruning.

Experimenting with different values for tree parameters helps us effectively reduce the complexity of a decision tree. For instance, minimum samples for a node split, minimum samples for a leaf node, maximum depth of tree, maximum number of terminal nodes, maximum features to consider for a split are some popular tuning parameters for a decision tree classifier.

In addition to tuning parameters, pruning is a better way to improve the performance of a decision tree. Pruning suggests that we remove the branches which adopt less important features. Reduced error pruning and cost complexity pruning are two widely used pruning options.

In short, decision tree classifier:

- is easy to understand and visualize;
- is suitable for both discrete and continuous features, as well as multi-class prediction;
- requires minimal data pre-processing since it inherently conducts feature selection during splitting.

Decision tree classifier has a few limitations, such as the fact that:

- it is prone to over-fitting and biased tree;
- it has high variance, i.e. a small change in the data may cause the classifier to create a totally different tree;
- it doesn't ensure that a local optimal tree is also a global optimal tree.

3.5 Random Forest Classifier

Random forest classifier is an ensembled algorithm around the decision tree model. In machine learning, "ensembled" refers to merging two or more models into one to

obtain better results. Random forest classifier, as we may have guessed from its name, constructs a forest with multiple trees, each of which is a decision tree as introduced in the previous section. However, every decision tree is created with only part of the available training set and feature set, both selected at random. For a new data sample, each tree makes a prediction and the classifier aggregates all the predictions and choose the one that is most frequent. Additionally, random forest classifier may be modified to assign weights on each decision tree. With lower weights on less accurate trees and higher weights on more accurate ones, the performance of the classifier can be further improved.

Suppose the size of training data is X, and the dimension of the feature space is K. Random forest classifier works as follows:

1. From the training set, select a subset of x samples at random ($x < X$) to serve as the training set for the decision tree.

2. Define a number $k < K$ as the volume of feature set in the tree and randomly select k features for each of the x training samples. The value of k remains unchanged during the creation of the forest.

3. Use the x training samples with k features to generate a decision tree as illustrated in the previous section.

4. Repeat steps 1 - 3 for n times, and create a forest with n trees.

5. When classifying a new data sample, the classifier combines all results from n trees and choose the majority.

Random forest classifier is based on tree models, thus it naturally has all the tuning parameters associated with decision trees. Besides, the number of trees in the forest can be altered to find the best model.

The underlying principle of ensembled algorithms is that although one single model may be limited under certain circumstances, a group of models tends to be much powerful. While a single decision tree is inclined to over-fitting and high variance, a random forest substantially increases the model stability and is currently adopted in all kinds of applications such as stock market and e-commerce. In short, random forest classifier:

- is robust to over-fitting;
- is capable of handling missing values;
- distinguishes features with higher relevance, hence can be used to reduce dimensionality.

Due to the nature of "ensemble", random forest classifier also has a few drawbacks like:

- It is hard to visualize and computationally expensive;
- It is sophisticated and difficult to implement.

Chapter 4

Evaluation & Result Analysis

In this chapter, we will experiment with different detectors built on the various classifiers introduced in chapter 3, and analyze their results. The detectors are implemented in python (version 2.7.13), together with the powerful machine learning tool scikit-learn (version 0.19.1) [9][10].

4.1 Model Evaluation

4.1.1 Cross Validation

In machine learning, we hope to train the model with as much data as possible to best learn their characteristics, and we also hope to test the model with as much data as possible to evaluate its performance comprehensively. This may seem paradoxical at first, since the dataset is fixed and the size of training set is inversely proportional to the size of testing set. Fortunately, Cross Validation (CV) helps us to solve this dilemma.

In this project, we will use a basic approach in CV called stratified k-fold. In stratified k-fold CV, the original dataset is divided into k subsets. Each subset is "stratified", meaning that the distribution of all the classes in the subset is similar to the distribution of all the classes over the whole dataset. Then we use $k-1$ folds of data to train a model, and test it on the remaining fold of data. This process is repeated k

times, and the performance of the model is measured by the average of the k test scores generated in the iteration.

k-fold CV is computationally expensive, but has the advantage of making use of most of the data while preserving its original features, thus is also widely used in parameter tuning. In our case, all 5 classifiers except for naive bayes classifier have some parameters that need to be adjusted, and this is accomplished by grid search CV. In grid search CV, there is a parameter grid which contains a set of possible values for all the parameters we would like to tune, then the algorithm searches through the grid. For every unique combination of values, the algorithm assigns them to the model and evaluates its performance using k-fold CV. At last, the best model will give us the optimum values for the parameters.

4.1.2 Evaluation Metrics

In statistics, there are all kinds of metrics designed to assess a model in different ways. For a binary classification problem like the one IDS is facing, consider the matrix in Figure 4.1:

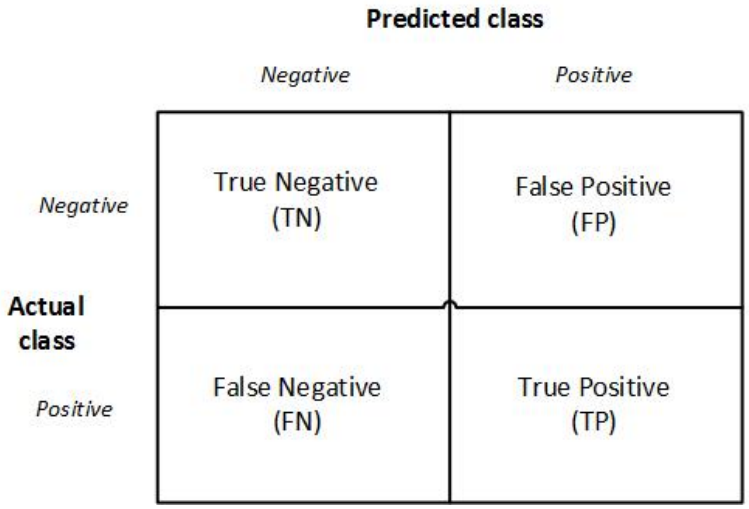

Figure 4.1: Confusion matrix

This is called confusion matrix, which shows all four possible outcomes in the classification. For IDS, the positive class is the actual attack, and the negative class is

normal behavior. The detailed illustration of the confusion matrix for a network IDS (NIDS) is given below:

- True Negative (TN): Network traffic which is normal and predicted as normal by the IDS.
- False Positive (FP): Network traffic which is normal but predicted as malicious by the IDS.
- False Negative (FN): Network traffic which is malicious but predicted as normal by the IDS.
- True Positive (TP): Network traffic which is malicious and predicted as malicious by the IDS.

Given the above confusion matrix, three metrics commonly used to evaluate IDS include True Positive Rate (TPR), False Positive Rate (FPR) and False Negative Rate (FNR), which are defined as follows:

- *TPR = TP/(TP + FN)*: This is also called recall, which tells us how many network attacks among all attacks are correctly identified by the IDS.
- *FPR = FP/(FP + TN)*: This is also called fall-out, which tells us how many normal traffics among all normal traffics are mistakenly treated as attacks by the IDS.
- *FNR = FN/(TP + FN)*: This is equal to *1 - TPR*, which tells us how many network attacks among all attacks are missed by the IDS.

There is a trade-off between TPR and FPR, which is often reflected by drawing a Receiver Operating Characteristic (ROC) curve. With FPR as X-axis and TPR as Y-axis, ROC plots the model's recall against fall-out under different discrimination thresholds. ROC curve is hard to be assessed, instead, we usually use Area Under roc Curve (AUC) metric to evaluate a classifier. AUC transforms ROC curve to a value in the range of $[0.5, 1]$, where the upper bound 1 means it is a perfect classifier and lower bound 0.5 means the classifier just randomly guesses the data. Figure 4.2 gives a demonstration of ROC curve and AUC metric.

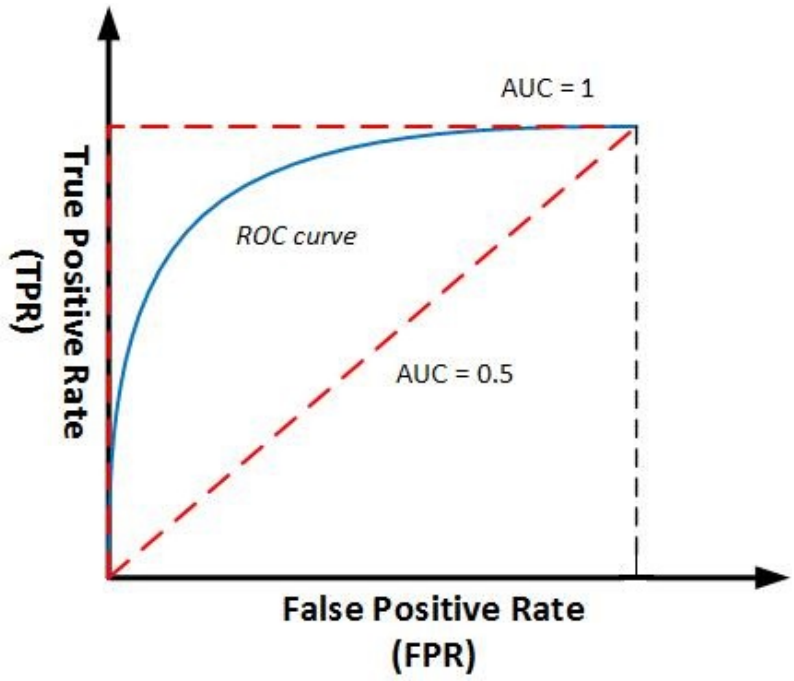

Figure 4.2: ROC curve and AUC metric

All things considered, we choose AUC to be the scoring method in grid search CV. After that, a five-fold CV is used to evaluate the tuned classifier with respect to TPR, FPR, FNR, training time as well as predicting time.

4.2 Experimental Results

In this section, we will explain the implementation of IDS in detail, and present the results obtained under different classification models. The cloud intrusion dataset we used was collected at hypervisor level, and the detailed characteristics are listed in Table 4.1.

Table 4.1: Characteristics of the cloud intrusion dataset

Total IP packets	20903
Time frame for the flow	0.1s
Normal flow	188
Malicious flow	25
Total flow	213

4.2.1 Data Preprocessing

Wireshark [11] is a handy tool for analyzing network traffic, and we will use it in the first step of data preprocessing. As shown in Figure 4.3, we need to select the 13 fields from IP packets:

t	sip	dip	sp	dp	protoc	ihl	thl	pktl	ident	fragoff	seq	ack
0.000000	172.16.1.24	142.104.64.196	35622	22	UDP…	20…	32	1500…	0xb8f0 (47344)…	0,0	1	1
0.000306	172.16.1.24	142.104.64.196	35622	22	UDP…	20…	32	1500…	0xb8f1 (47345)…	0,0	1399	1
0.000312	172.16.1.28	172.16.1.24	22	32900	UDP…	20…	32	1500…	0x8128 (33064)…	0,0	1	1
0.000319	172.16.1.24	142.104.64.196	47174	514	UDP…	20…	32	1500…	0xb8f2 (47346)…	0,0	1	1
0.000337	172.16.1.28	172.16.1.24	22	32900	UDP…	20…	32	1308…	0x8129 (33065)…	0,0	1399	1
0.000390	172.16.1.24	172.16.1.28	32900	22	UDP…	20…	32	102,…	0x8f2f (36655)…	0,0	1	2605
0.000410	142.104.64.196	172.16.1.24	514	47174	UDP…	20…	32	102,…	0xb7e1 (47073)…	0,0	1	1
0.000469	172.16.1.24	142.104.64.196	47174	514	UDP…	20…	32	518,…	0xb8f3 (47347)…	0,0	1399	1
0.000486	172.16.1.24	134.87.154.134	22	59526	UDP…	20…	32	1390…	0xb8f6 (47350)…	0,0	1	1
0.000488	172.16.1.24	134.87.154.134	22	59526	UDP…	20…	32	1390…	0xb8f7 (47351)…	0,0	1289	1

Figure 4.3: Use wireshark to select IP packet fields

4.2.2 Feature Extraction & Labeling

Then we calculate the IP weight features of the flows, and label them according to whether the IP addresses in the flows are trusted. Malicious flow is labeled as 1 (positive class), while normal flow is labeled as 0 (negative class). After we have both the features and labels, we can now feed the data into the classifiers. Figure 4.4 describes the structure of the flow dataset, and Table 4.2 shows the IP statuses.

A	B	C	D	E
ipw_freq	ipw_ran	ipw_load	ipw_str	label
630	1.489934	0.903226	3.27E-52	0
131.2889	0.699843	0.6	5.9E-127	1
500	0.904381	0.420168	0	0
10	0	1	0	1
953.1102	1.241249	2.791045	0	0

Figure 4.4: Features and labels of the flow dataset

Table 4.2: IP statuses of the flow dataset

IPs	Status
172.16.1.28	Normal
172.16.1.20	Normal
142.104.64.196	Normal
172.16.1.24	Normal
172.16.1.27	Normal
134.87.154.134	Normal
210.245.88.110	Malicious

4.2.3 Naive Bayes Classifier

For naive bayes classifier, since the features are continuous, we assume a guassian distribution for the values and there are no parameter to be tuned. The 5-fold CV results are shown in Table 4.3.

Table 4.3: 5-fold CV results for Naive Bayes classifier

Trial	1	2	3	4	5	Average
Recall	100%	100%	100%	60%	60%	84%
FNR	0%	0%	0%	40%	40%	16%
FPR	21.1%	26.3%	34.2%	8.1%	8.1%	19.6%
Training time(ms)	4.0	2.0	3.0	3.0	3.0	3.0
Prediction time(ms)	7.0	7.0	7.0	7.0	7.0	7.0

4.2.4 SVM

For SVM, we choose "kernel" $\in$ { "rbf", "linear", "poly"}, "γ" $\in$ { 1, 10, 100} and "C" $\in$ { 1, 10, 100} as the tuning parameters. The best SVM found by grid search CV is described in Table 4.4, and the results of 5-fold CV are given in Table 4.5.

4.2.5 KNN Classifier

For KNN classifier, we choose the "number of neighbours" $\in$ { 3, 9, 15} as the tuning parameter. Table 4.6 shows the best KNN classifier found. Using the best parameter, the 5-fold CV results are presented in Table 4.7.

Table 4.4: Best SVM found

AUC score	0.958
Parameters	kernel = "linear", $C = 100$, $\gamma = 1$
Training time(ms)	2.3
Prediction time(ms)	19.0
Tuning time(s)	13.7

Table 4.5: 5-fold CV results for SVM

Trial	**1**	**2**	**3**	**4**	**5**	**Average**
Recall	100%	100%	100%	60%	40%	80%
FNR	0%	0%	0%	40%	60%	20%
FPR	2.6%	5.3%	5.3%	5.4%	10.8%	5.9%
Training time(ms)	4.0	3.0	2.0	1.0	2.0	2.4
Prediction time(ms)	8.0	6.0	10.0	12.0	10.0	9.2

Table 4.6: Best KNN classifier found

AUC score	0.909
Parameter	number of neighbours = 3
Training time(ms)	25.0
Prediction time(ms)	40.7
Tuning time(s)	0.543

Table 4.7: 5-fold CV results for KNN

Trial	**1**	**2**	**3**	**4**	**5**	**Average**
Recall	100%	100%	100%	60%	40%	80%
FNR	0%	0%	0%	40%	60%	20%
FPR	0%	2.6%	2.6%	5.4%	10.8%	4.3%
Training time(ms)	5.0	5.0	3.0	3.0	3.0	3.8
Prediction time(ms)	18.0	21.0	48.0	14.0	15.0	23.2

4.2.6 Decision Tree Classifier

For decision tree classifier, we choose the "minimum sample split" $\in \{$ 0.05, 0.1, 0.15, 0.2, 0.25$\}$ as the tuning parameter. The best decision tree classifier is illustrated in Table 4.8. With the best classifier, the 5-fold CV results are presented in Table 4.9:

Table 4.8: Best decision tree classifier found

AUC score	0.848
Parameter	minimum sample split = 0.1
Training time(ms)	2.0
Prediction time(ms)	15.3
Tuning time(s)	0.463

Table 4.9: 5-fold CV results for decision tree classifier

Trial	**1**	**2**	**3**	**4**	**5**	**Average**
Recall	100%	100%	100%	0%	40%	68%
FNR	0%	0%	0%	100%	60%	32%
FPR	5.3%	5.3%	2.6%	2.7%	5.4%	4.3%
Training time(ms)	2.0	3.0	4.0	3.0	3.0	3.8
Prediction time(ms)	11.0	10.0	10.0	11.0	9.0	10.2

4.2.7 Random Forest Classifier

For random forest classifier, we choose the "minimum sample split" $\in \{$ 0.05, 0.1, 0.15, 0.2, 0.25$\}$ and "number of trees" $\in \{$ 5, 10, 15$\}$ as the tuning parameters. The tuned classifier is described in Table 4.10, and Table 4.11 records the 5-fold CV results on it.

Table 4.10: Best random forest classifier found

AUC score	0.914
Parameters	minimum sample split = 0.1, number of trees = 10
Training time(ms)	197
Prediction time(ms)	43.7
Tuning time(s)	8.6

Table 4.11: 5-fold CV results for random forest classifier

Trial	1	2	3	4	5	Average
Recall	100%	100%	100%	40%	40%	76%
FNR	0%	0%	0%	60%	60%	24%
FPR	0%	2.6%	2.6%	2.7%	8.1%	3.2%
Training time(ms)	77.0	64.0	71.0	76.0	78.0	73.2
Prediction time(ms)	10.0	15.0	13.0	11.0	11.0	12.0

4.3 Comparison & Analysis

4.3.1 Comparison Among Different Classification Models

Summarizing the previous results, Table 4.12 provides a straightforward comparison between these five classifiers in terms of recall, FNR, FPR and speed.

Table 4.12: Result comparison among different classifiers

Classifier	Naive Bayes	SVM	KNN	Decision Tree	Random Forest
Recall	84%	80%	80%	68%	76%
FNR	16%	20%	20%	32%	24%
FPR	19.6%	5.9%	4.3%	4.3%	3.2%
AUC score	0.963	0.958	0.909	0.848	0.914
Training time(ms)	3.0	2.4	3.8	3.8	73.2
Prediction time(ms)	7.0	9.2	23.2	10.2	12.0

According to Table 4.12, we see that the results generally agree with the theories of classifiers as explained in chapter 3:

- The naive bayes classifier is extremely fast, and no parameter tuning is necessary. It yields the best detection rate, with a cost of comparably high FPR, which is caused by the underlying assumption of independence. According to IP weight metrics, the IP weight values are proportional to the anomalousness of flow. Consequently, if several IP weight features are large, this flow is likely malicious and the other IP weight features are probably large too. Similar

conclusions can be made for normal traffic as well. This clearly violates the guassian distribution we assumed for the feature set, hence the high FPR;

- KNN classifier offers a competent recall and FPR, but it is memory-consuming and particularly slow in prediction stage, which makes it hard to accomplish time sensitive tasks like intrusion detection;

- Random forest classifier combines multiple decision trees to improve its performance, and generally produces better results than a single decision tree classifier. It has the lowest FPR, but is also computationally expensive. Just like we saw in the table, its training time is the longest among all classifiers;

- Taking everything into account, SVM maybe the best choice for this problem. It is efficient regarding both speed and detection capability. Despite that, it has a few drawbacks such as the tedious parameter tuning and less flexibility than random forest classifier. For instance, if the size of the dataset continues to grow, or if there is much noise in the data, SVM may no longer be suitable.

4.3.2 Comparison Between Different Network Environments

The goal of IDS is to achieve maximum detection rate with high detection speed while generating minimum false alarms, in other words, we expect the IDS to have high recall, low FPR and predicting time. As stated in the previous section, the average recall and FPR of our IDS against network intrusions in the cloud is about 80% and 5% respectively. In Wei Lu's experiment with the 1998 Defense Advanced Research Projects Agency (DARPA) dataset, the IDS provides about 88.79% recall and 0.8% FPR against intrusions in traditional network environment [5]. By analyzing these results, we make the following conclusions:

- Intrusion detection for cloud computing is more challenging than conventional networks, and security techniques especially designed for cloud environment should be researched;

- Although the IP weight metrics perform better against traditional attacks, they are able to provide encouraging preliminary results for cloud intrusion detection;

- As the IDS developed by Wei Lu [5] adopted various techniques including k-means algorithm and outlier detection to obtain better performance, we believe our results can be further improved by combining other models;
- In addition, due to the limitations of IP weight metrics, other approaches more appropriate for identifying single-connection based intrusions should be investigated. Unsupervised machine learning techniques that do not require labeled data should also be studied.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

In this project, we built an anomaly-based cloud IDS using IP weight metrics for feature extraction and a variety of popular models for classification. With IP weight metrics, we manage to reduce the feature space to a four-dimensional space, then we apply different classifiers on the data and compare the results.

According to our experiment, naive bayes classifier is very fast but has higher FPR compared to other classifiers. KNN classifier offers a pretty good recall and FPR, but is memory-consuming and relatively slow in predicting. Decision tree classifier almost always performs worse than the random forest classifier, which ensembles multiple decision tree models to give the final result. Overall, SVM produces the optimal results, however, it requires longer time for parameter tuning and is not as versatile as the random forest classifier.

In comparison with the experiment conducted by Wei Lu, we see that the IP weight metrics work better for conventional network traffic. Nevertheless, the results for cloud traffic are promising, and can be enhanced by adding other data preprocessing stages.

Through the implementation, we get a general idea of the working theory behind IDS. Moreover, we now have a more straightforward understanding about the popular classifiers with respect to both their advantages and disadvantages. The results

presented in this report may be taken into consideration for authorities when deploying real world IDS.

5.2 Future Work

Any classification technique has its own pros and cons, and there exists no universal classifier which is suitable for every problem. For intrusion detection, the two most important goals are real-time analysis and high detection accuracy. Consequently, one approach for this would be combining multiple classifiers. For example, the conjunction of naive bayes classifier and random forest classifier is able to make use of the fast speed of the former and the good recall as well as low FPR of the latter.

Furthermore, due to the difficulty in obtaining labeled data, unsupervised machine learning techniques (like clustering and outlier detection) need to be researched [12]. Except for anomaly detection, signature-based detection and specification-based detection may also be applied to IDS in practice [13][14]. Besides, since IP weight metrics aim to deal with multiple-connection based attacks, tactics for distinguishing single-connection based attacks should be studied as well.

Bibliography

[1] J. Wang, *Computer network security: theory and practice*. Springer Publishing Company, Incorporated, 2009.

[2] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[3] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[4] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: supervised or unsupervised?" *Image Analysis and Processing–ICIAP 2005*, pp. 50–57, 2005.

[5] W. Lu, "An unsupervised anomaly detection framework for multiple-connection based network intrusions," Ph.D. dissertation, University of Victoria, 2005.

[6] ISOT Research Lab, https://www.uvic.ca/engineering/ece/isot/.

[7] Compute Canada, https://www.computecanada.ca/.

[8] S. Ray, "6 easy steps to learn naive bayes algorithm (with codes in python and r)," https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[11] Wireshark, https://www.wireshark.org/.

[12] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.

[13] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[14] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Technical report, Tech. Rep., 2000.