Cortical Learning Algorithm to Predict Anomalous Web Services Usage

By

Ali Tanveer

A thesis submitted in conformity with the requirements for the degree of *Master of Science* in Information Security

Department of Information Security
Military College of Signals (MCS), NUST
National University of Sciences and Technology (NUST)

June 2018

Cortical Learning Algorithm to Predict Anomalous Web Services Usage

By

Ali Tanveer

00000101992

Supervisor

Brig Imran Rashid, PhD

Committee Members

AP Mian Muhammad Waseem Iqbal

Dr. Hammad Afzal

Department of Information Security

Military College of Signals (MCS), NUST

Declaration

I, *Ali Tanveer* declare that this thesis titled "Cortical Learning Algorithm to Predict Anomalous Web Services Usage" and the work presented in it are my own and has been generated by me as a result of my own original research.
I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST

2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated

3. Where I have consulted the published work of others, this is always clearly attributed

4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work

5. I have acknowledged all main sources of help

6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

Copyright Notice

This thesis is dedicated to *my beloved parents*.

Abstract

Cyber Security has become a significant concern in corporate web usage, which includes risks of both internal and external computer attacks. The purpose of this study is to introduce and investigate an application of the Cortical Learning Algorithm (CLA) in minimizing potential threats to a web server using machine learning based anomaly detection. The results are compared with 2 major and 3 minor state-of-the-art conventional algorithms. Approximately 38,000 web usage samples were collected over a 6-month time period. This data was organized as an input to CLA through encoders to standardize individual data formats. The output of each encoder is assigned priority weights based on the nature and significance of data. CLA performed exceptionally with an almost 99% rate of anomaly detection, checked manually on unsupervised data. It also helped in processing data as quickly as it arrives in a continuous fashion unlike conventional methods that store and process it offline. This spontaneous and effective approach of CLA proves its potential in enhancing security against computer attacks in the corporate sector.

Keywords: *Cyber Security, Web Services, Cortical Learning Algorithm, Machine Learning*

Acknowledgments

I would like to express the deepest appreciation to my committee chair Brig. Imran Rashid, PhD, who has the attitude and the substance of a genius: he continually and convincingly conveyed a spirit of adventure in regard to research and scholarship, and an excitement in regard to teaching. Without his guidance and persistent help this dissertation would not have been possible.

I would like to thank my committee members, Dr. Hammad Afzal and Mian Muhammad Waseem Iqbal, whose work demonstrated to me that concern for global affairs supported by an 'engagement' in comparative literature and modern technology, should always transcend academia and provide a quest for our times.

In addition, thanks to Mr. Hameer Abbasi for his insights into machine learning and valuable assistance with implementing the algorithm.

Contents

List of Figures

List of Tables

List of Abbreviations

AIS	Artificial Immune System
CLA	Cortical Learning Algorithm
URL	Uniform Resource Locator HTM Hierarchical Temporal Memory
OSV	One-class Support Vevtor Machines
HMM	Hidden Markov Model
KOAD	Kernel-based Online Anomaly Detection
SDR	Sparse Distribution Representation
SP	Spatial Pooling
TM	Temporal Memory
HTTP	Hyper Text Transfer Protocol
XML	eXtensible Markup Language
SOAP	Simple Object Access Protocol
SAD	Web Session Anomaly Detection
TLD	Top Level Domain
Enc	Encoder

Chapter 1

Introduction

1.1 Description

Computer network security in the contemporary era [1] depends mostly on efficient techniques to detect anomalies in the routine usage, which preferably should not hinder commercial or personal activities of the user [2]. Currently implemented rule or history-based anomaly detection systems are becoming less effective as attack patterns are getting more accurate, customized and adaptive. The same aforementioned nature of attack vectors also needs to be made part of attack-preventing techniques. Adaptive and dynamic techniques need to be considered if computer usage, security has to move with the same pace as improvements in attacking methods or even better. This purpose can be fulfilled by enhancing anomaly detection to a level where an attack's potential is predicted beforehand [3].

If it was just a question to handle anomalies in computer usage like the outlier's detection in an offline manner, it seems greatly answered by Cluster Analysis, OSV Machines and other Encoder-based methods [4].

The factor that differentiates this graduate thesis from already implemented algorithms is that it suggests and anomaly detection technique that works on sequences of data being fed as a stream on-the-go. It suggests that online detection of anomalies is possible while each new entry is passed at least once for sequence

processing [5].

Commercial or corporate networks usually carry web usage data in huge log files which later becomes sequences as users take action with the time. On such servers, huge traffic or large amount of data carries a constraint of storing it and processing online, therefore it is processed offline. The time delay linked to offline processing adds to the limitations of web usage security. Hence there needs to be an online scheme or algorithm to detect and even predict anomalies beforehand.

This specific constraint has not been given much importance in existing literature. Recent developments in Hierarchal Temporal Memory with the addition of Cortical Learning Algorithm have opened the door for future considerations in Computer Networks Security. This thesis covers analysis of Cortical Learning Algorithm as an application to predict anomalous behavior by users of a web server.

1.2 Problem Statement, Objectives and Scope

This Graduate Thesis tries to emphasize the application of a Cortical Learning Algorithm in anomaly detection; predicting behavior of a user visiting some particular website. A temporarily sequence of URLs is ordered as an input to categorize activities as anomalous or normal behavior. These sequences are passed in the form of streams, making sure online processing is carried out with least possible resources consumed.

This application of the CLA covers following properties: hostnames and subdomains visited, referrer domain, frequency and recency taken as frecency, time of the visit and the type of visit made. Most of the times, a URL is possibly a very large string which for resource management purpose has been reduced to only the hostname and the subdomain. On the same grounds, for a web server, log files are very huge that their storing and instant processing (to analyze normal or anomalous behavior) is not easily possible. CLA has a promising nature to learn and decide in parallel algorithms. Its performance is enhanced when used with the Hierarchical Temporal Memories (HTM) [8]. The Major purpose of this research

is to introduce Cortical Learning Algorithm in Cyber Security. Initially, a user's web usage pattern has been monitored for a reliable period of time. It is then passed to unsupervised anomaly detection in sequences through the CLA. As the results are achieved after close fine tuning of several factors (mentioned in later parts) in the algorithm, they are then compared with some traditional algorithms in order to make sure our efforts are placed in the right direction. In the later part of this thesis, emphasis has been made on the future potential of CLA.

This algorithm has been highly customized in this research to cater all resource constraints and privacy considerations. For a local testing a generic user web usage history is used while for an efficient and precise result, a web server's history can do wonders.

1.3 Outline

This thesis comprises of 5 comprehensive chapters, each carrying its own significance, gradually elaborating the idea of introducing CLA into detecting anomalous web service usage. Chapter 1 already covered introduction and brief description of the research work without much technical consideration. Chapter 2 covers background knowledge and previously carried out research on algorithms being suitable for anomaly detection. Chapter 3 elaborates all the considerations made while implementation of the algorithm. Chapter 4 covers all important results which are also compared with state-of-the-art traditional anomaly detection methods. Chapter 5 being final chapter to this research work has all my inferences and interpretations of results from chapter 4.

Chapter 2

Literature Review

2.1 Related Work

We do not see any particular applications of CLA in Web usage, security, but there are few other methods currently in action at large [2]. These methods focus on anomaly detection via the nature of an outlier; based on set rules, learned history or supervised training. There are very few that offer unsupervised machine learning for anomaly detection, but those either consume extra resources or offer an offline solution [6]. In this section of Related Work, we will look into all possible variants of what we are going to implement through this research.

2.1.1 General Anomaly Detection

With any given dataset, any known machine learning algorithm can do the trick of getting customized into detecting an out-lier, the point that does not follow static repartition of its dataset. This anomaly detection is based on Non-temporal techniques where a whole dataset is analyzed and global parameters are set for the cause. The most common examples of this general anomaly detection are a deep approach used in OCSVM with both supervised and unsupervised learning methods [7]. KOAD is an important Online anomaly detection technique that uses dynamic dictionaries in order to approximate a common state for a given

dataset.

2.1.2 Sequential Anomaly Detection

This is more effective method of detecting anomaly in context of web service usage. The dataset is precisely divided into sequences; windowed and fixed-length inputs, and later used with aforementioned machine learning anomaly detection techniques. But such methods only generate results with simple sequenced datasets and are less reliable when it comes to temporal dependencies as the state space grows exponentially.

Hidden Markov Model is specifically designed to cater sequences in temporal memory, but this HMM does not cover the online part of anomaly detection as it requires to work on dataset as a whole [9]. This technique also performs process several times, thus adding latency to the cause [10]. For this purpose, t-stide has been chosen as a competitor with its ability of having a Window-based technique and HMM with its online compatibility.

2.2 Cortical Learning Algorithm Fundamentals

HTM basically comes from deep knowledge of neuroscience in biological brain. This memory technique has been utilized in the software domain by Numenta; thus this algorithm is being used under their Open-Source license code name Nupic. High Order Sequences can be generated by storing, learning and inferring inputs using learning methods devised in HTM. One of those methods is Cortical Learning Algorithm that suits its application in web services security with several customizations. Having an almost real biological model of human neuron, CLA has drastic results due to its SDRs (Sparse Distribution Representations).

CLA comprises of 2 core steps- generation of an SDR via SP (Spatial Pooling) and then learning to predict from complex sequences over generated Sparse Distributed Representations. A fixed number of minicolumns are active against any particular

input and there is almost a zero chance for a perfect overall against two inputs if column length and state space are chosen appropriately. In every next step, each already existing multicolumn becomes some previous state. Thus a cell can be in an inactive, active or predictive state. In the SDR generation phase, similar patterns of the dataset inputs bear similar or close overlapping thus making it feasible for the algorithm to decide its predictions.

2.2.1 Is it an Artificial Neural Network?

ANN are quite different from CLA in a way that ANNs follow a very few synapses in order to conclude a prediction. Unlike a biological neuron, an ANN does not carry any dendrites structure, whereas CLA has dendrites as core elements of the algorithm [11]. CLA also has the capacity to accommodate thousands of synapses without any delays. ANNs learn with the variations in the weights of synapses while CLA adopts/ molds itself to the growth of fresh synapses throughout the process. CLA also offers active cells in multi columns to recognize thousands of patterns individually.

Therefore, Cortical Learning Algorithm being closer to a Neocortical Neuron or the real neuron is more feature enriched and performance oriented than Artificial Neural Networks.

2.2.2 Hierarchical Temporal Memory and CLA

Recent developments in Hierarchical Temporal Mempory have shown some remarkable and promising experiences, but its worth has not yet been brought out in the Cyber Security domain. Successful usage of HTM in few practical life based scenarios of pattern recognition has led me carry out tests using webservices usage data.

A simple structure of an HTM can be divided into two adjacent blocks: Pattern and Transitional Memory. All the learning from Spatial Pooler (SP) takes place under the label of Pattern Memory while the transitional memory learns on

transient patterns and later categorizes them on SP patterns.

2.2.2.1 Spatial Pooler Pattern Memory

This step has a significant impact on the Cortical Learning Algorithm as it creates and works over the SDR (Sparse Distribution Representation). Pattern Memory consists of a set of Boolean inputs with approximately around 2

Each SDR is actually a unique and random representation of an input from the synapse. A set of synapses from neurons further combine together to form the Pattern Memory. The algorithm has a couple of parameters such as permanence value and certain threshold for the synapse. The default value of the former is either 0 or 1 associated with sensitivity of the input while the latter has a general value of 0.2 which can later be varied according to the conditions to where that synapse is linked. Values for both of the above parameters are adjusted with excessive trainings. There is only a chance of a neuron being active if a set number of its synapses are connected to the input and thus it intrigues nearby neurons to propagate to the next level. In this way of propagations, patterns are learned according to the sparse nature over an SDR.

We use brains to understand how we use our brains to understand how we use our brains. It seems complicated, and it actually is. This paragraph will cover elaboration of Spatial Pooling as a significant subprocess of CLA. Once the input space is ready, a new spatial Pooler is instantiated, it randomly sets up its columns to be connected to that input space. This is divided into two steps: activation of columns by calculating the columns' overlaps of the input representation with the input space and how each of those columns learns to represent specific spatial characteristics of that data that's been encoded in the input space over time. Each of the column has an entirely specific relationship with input space compared to its previous and next column's relationship. Selection of the right column to be activated is done by their overall score, which means how many of the cells overlap in input space. As more than one columns happen to overlap, they are graded based on rankings (under number of cell overlaps). This representation finds its roots in global inhibition area where every column is a neighbor of another column.

For an instance, generally, top 40 ranked columns are considered in that particular compute cycle. For implementation details, in case of tiebreakers, that is if there are a bunch that have same overlap, random selection is done.

Each of the individual connections of cells from spatial Pooler to input space is called a dendritic segment. There are three dendritic segments based on the algorithm: actual, possible and impossible cells.

Moving to the learning in Spatial Pooling, let's consider a timestamp at a particular cell. First of all, none of the columns that are inactive will learn anything, no state changes happen to columns that have not been activated so the only learning that goes on happens in these above mentioned 40 columns. In this case, these are all going to increment and decrement the permanence values based on how many and what connections they have in the input space in this timestamp. Increment or decrement in the permanence values means that those particular connections will become stronger or weaker. And further the increase in permanence values is learning about the connections; something that column is to recognize for future references. Any connections that fall outside of the input for an activated column, those permanence values will be decremented. Connections are calculated simply whether they are above a certain threshold of the permanence value for that cell. In a simpler way, as the permanence value goes up and down, connections can be created or destroyed.

So, as it learns from each compute cycle, some of these connections will go away, some of them will appear and some of them will go away and reappear, depending upon the input space of the random initialized state of the system, what it learned so far and how its connections are. If it sees a lot of input over connections that it has well-established, those are going to get reinforced. This is an overview of how learning works and carries very simple learning rules. Considering this in the implementation phase, how much the values are incremented and decremented and what should be a threshold for certain case are also parameters to the spatial Pooler. If we carry out Spatial Pooling to generate SDRs on the same input spacing with Random SP Active Columns and Learning SP Active Columns, it

is realized that one with the learning one produces SDRs that match previous SDRs unlike random SDRs in the other method that is without learning when appropriate.

2.2.2.2 Temporal Memory

Outputs from the aforementioned step are then fed into the Transition Memory Process. With the activation of a column due to the output of the SP, there are two chances to take place: either few of the columns occurred being in a predictive phase or none of the columns happened in predictive phase thus turning on all columns then. With at least one dendrite to the neuron being active, it gets its neuron into a predictive state, meaning thereby it predicts the chances of this neuron to be the next sample are higher in percentage, compared to the rest possible states.

In this way, the algorithm forms blocks carrying boolean function: either the columns are active or the columns are expected to turn on in the next step.

As the Spatial Pooling translates columns from an input space to a normalized representation with a fixed sparsity that contains all the semantic information of the original encoding. We will now look into the temporal patterns and how the temporal memory algorithm CLA recognizes sequences of spatial patterns over time.

The temporal memory algorithm does two things [12]:

- Learns sequences of active columns from Spatial Pooler over time
- Makes predictions about upcoming pattern based on the temporal context of each input.

It does this by activating individual cells within many columns. Once the active cell space has been identified, the second phase is to choose a set of cells to put into a predictive state which means that these cells will be primed to fire on the next timestamp..

We can notice that with this first timestamp in the sequence every cell within every active column is active because it has never seen this spatial input within

any context before. It has no predictive cells to activate. The other reason is cells within active columns activate is because they have been put into a predictive state by some previous context of this spatial input. Lets take example of sequenced alphabets, Spatial Pool can start with any alphabet with its SDR.

Predicting next alphabet in this out of context sequence is impossible, therefore all cells become active as each cell has equal probability of occurrence. The second alphabet is exactly the next alphabet of the previous one in English order. Third cell before instantiation has some cells bursting as it may be the next alphabet from English order and roots of certain context have been imagined by the algorithm.

It is the first phase of temporal memory, deciding which cells to become active in predictive mode: either they all become active in a column because there are no cells in a predictive state that means we are sort of kicking off this sequence and seeing it for the first time and the other is that there are cells in the predictive state within a column and those will be switched to active if they were correctly predictive because for the current set of active columns it basically validates that prediction.

The second phase of the temporal memory algorithm covers how these cells become predictive. After we have identified which cells are currently active in this timestamp based upon their predictive states within the active columns, we will look closer into an HTM-CLA neuron and learn more about it in its distal segment later.

The comparison of the biological neuron to the HTM-CLA neuron shows a feed-forward input, which is the proximal dendritic input from the input space on both sides. The distal input is from the lateral connections to other cells within the space and context is the identified root of the sequence. An HTM neuron shows that it can have more than one distal connections. These are the distal segments where each one of these segments could potentially have one or more synapses or connections to other cells within the HTM Structure. Each one of those cells may be in an off or on state. So that any time a cell has to decide whether it should go into predictive state or not, it can look at all of its segments and its connections

across all of their synapses. If one of these synapses breach some threshold which is configurable, then that cell goes into a predictive state based upon its contextual connections to the other cells within the structure.

Each one of these coincidence detectors could potentially cause the cell to fire in the next timestamp if it is right. In a simple attempt of explanation, all the specific cells connected to a cell are in active state, there are greater chances of that new cell turning into busting more and then becoming active if it is right.

2.3 Conventional Algorithms

Following are few of the conventional algorithms implemented in network security that may come handy in deciding the percentage of anomalies being accurate or false positive:

2.3.1 The Stide Model

The stide model algorithm uses a pretrained model that finds anomalies according to the dataset provided earlier. It actually works on sequences like the cortical learning algorithm and if any sequence happens to be anomalous compared to the learnt patterns then the stide model generates an alarm. The sequences as number of steps are fixed beforehand. This model works basically on the hamming distance in order to find mismatching anomaly or pattern. The implementation and processing of this algorithm will be elaborated in the coming chapters of this thesis. This thesis will also cover the importance of cortical algorithm when it is compared with his tied model.

2.3.2 HMM

It is the sequences that make HMM or hidden Markov model significant enough to compete the results of the cortical learning algorithm. Only difference of hidden Markov model from the above Stide model is its nature of being trained online. In

this thesis, I will compare HMM on the basis of its prediction on forward algorithm before training. This model works with the parameters such as hidden state and total number of possible states. It has a limitation [13] that the URL of given dataset is only considered as one state or observation.

In a further step the possibility of a sequence to occur is compared to the average of previous sequences in log space to find its likelihood of occurrence. Once the percentage is ready, it is then mapped on a threshold to make it one of the normal behaviors or the anomalies.

2.3.3 Analysis of HTTP Requests

Quite recently this approach has been adopted where all the http logs are collected from normal day use at both the server and user end. A fresh dataset is used to train the algorithm to understand them as normal http request which does not carry any attacks. Once all the data from web service usage has been learned by the algorithm, several anomaly detection methods are implemented to classify all the Learn Data as normal user usage. After the learning process this model is implemented in the network at the online mode. It basically works on the deviation of normal traffic from the learnt set. Although it has promising results, but it also carries some limitations such as it does not learn about the online data, but just compare the online data with the patterns it is already sequenced.

2.3.4 XML-Structured SOAP

XML structure anomaly detection uses tree based Association to gain knowledge from the training data set which is then compared with a live data set of network traffic [14]. It is a high rate of anomaly detection and very low for longer, but it does not provide online learning.

Another limitation to examine structured soap messages is that reduces only soap messages. So basically this technique is a way of mining knowledge about the normal behavior of soap messages.

2.3.5 SAD: Web Session Anomaly Detection

SAD is an estimation technique to detect anomaly in web sessions, it uses profile building technique from normal usage[15] and plots it against the frequencies. Later these frequencies are used to expect the outputs of Forward algorithm that has the potential of detecting new possible attacks on the integrity infrastructure, but it is not precise enough to differentiate between a new person's usage pattern or a fresh anomaly.

Chapter 3

Methodology

3.1 Data: Web Services Usage Log

3.1.1 Log Stream Content

The dataset used for this Cortical Learning Algorithm based anomaly detection is the stream of URLs along with their attributes visited by a particular person in the test. Typically, a web server logs following details of web service usage:

- Date of Visit
- Time of Visit
- Referrer Website
- Reffered Website
- Protocol Details
- Request Size
- Size of the Resposnse
- Frecency
- Last Visit Made

- Visit Type

Most of the information mentioned above has been used by different research projects and technical papers in sets such that similar or relevant data is used according to the needs of the anomaly type. Such as request size and type can be used to identify SQL attacks, referrer website can be used to identify if the culprit was a previous move made by the user etc.

This thesis being based on web service usage caters all the factors that add to the behavior of the user while he makes a request. The behavior is most likely a subjective term, but the time stamp of the visit, its last visit time and frecency add to the digital analysis of the activities being made, on the other hand referred and referring URLs and visit types compile a string input to the algorithm.

In this way, aforementioned five factors of information are supposed to help in making credible predictions about the anomalies expected to occur in forward algorithm.

3.1.2 Specific problem constraints

Real time learning and control algorithm preparation and implementation should cater for extensive data size, possible delay, data stream updating and more. Therefore, following are the major constraints that Cortical Learning Algorithm will try to cut-down in all possible attempts of fine tuning.

3.1.2.1 Unsupervised Learning

First and foremost constraint for any future algorithm is that the usage supervision is not ensured, meaning thereby algorithm will be working on data streams which are not in any way categorized as anomalous or normal, beforehand. Therefore, the suggested algorithm should learn and monitor in parallel. As the algorithm learns with every passing second, it needs to create a statistical profile of the data set elements and sequences.

3.1.2.2 Online Learning Once the problem of Unsupervised Learning is acknowledged, it needs powerful equipment to process logs online. These logs range

from thousands of URLs to even millions for just a few days. Therefore, the data stream being too large, suggested algorithm should not involve or add any delays or even processing data streams in batches, as concurrent algorithms do, is not an option. Hundreds of URLs must be processed in the earliest possible.

3.1.2.3 Incremental Complexity

A freshly implemented algorithm does not need much data to learn and process them into categories, but with every passing day, data gets added to the learning phase. This ultimately results in an excessive data to process online, making the system slower and slower gradually. The suggested algorithm should have enough efficiency to maintain stable compatibility against any added level of complexity. It should work without any hindrance for above mentioned constraints.

3.1.3 DataSet Preprocessing

The original stream of data taken in the raw form contains all the succession of websites visited by the user. It is as if all the information is stored to process and prioritize according to the needs of the algorithm.

As a server is taking thousands of requests with high level of attributes it seems impossible for the server to preprocess and then feed it to the algorithm. Therefore, we are making an assumption that the data is being processed at a different level. It is not that similar requests are grouped into a single sequence, but requests at the same time can be regrouped based on the factors mentioned in the chapter. Keeping it simple algorithm to decide normal and anomalous behavior in the real time and what delays are added.

At times there is a possibility that a URL is very wrong. In that case this particular URL consumes enough time and resources. This happens to be a constraint from the list of constraints mentioned in the above section. Therefore, this thesis already suggests controls to reduce time delays and other factors into a negligible influence.

3.2 DataSet Attributes

A huge chunk of traffic has been arranged from the usage pattern of a subject user. Whereas due to security and privacy reasons the data file of the user will not be made public under the domain of this thesis. Even if the data is very large, the algorithm designed for this purpose has been modified in a way that it does not create any un-affordable delays. Another way of generating data is through Marco methods which produce a generalized ETA and give more realistic conditions to formally place anomalies. Therefore, we will be choosing natural behavior-based data from the real life scenario of the user. It contains both normal uses and Anomalous usage by the user, for example, visiting a social media website or platform early in the morning is the behavior of the user, visiting the same website in the late hours of the evening can be considered an anomaly. If an office has working hours of daytime, use of the Internet Services during the night is an anomaly which may generate an alarm. One of the significant attribute of this algorithm is that it also caters percentage of chances when a user may visit any website in any particular time of the day, day of the week, or anytime that may appear anomalous to conventional cyber security measures.

The data stream in context consists of several rows with each row containing preferably a unique or similar visit made by the user. The stream of rows is ordered or sorted based on the time of the visit. As this is a dataset generated from user's live visits, anomaly detection will also be live and potentially fast. There is no such hard and fast rule that some data being trained is a normal while the data to be tested on the forward algorithm will have anomalies. The fact is that this data in itself contains normal traffic as well as otherwise. The attributes of the data sets have been detailed below:

- It has more than 38000 URL visits
- Time duration of visits is 1 year
- URL lengths have been cut-down to sub-domains level to save time and resources

- Frecency of each visit is also considered as it defines a factor based on both the frequency of the visit as well as its recency.
- around 0.0156% of the visits are anomalies.

3.3 CLA in action on Data Stream

Once the data are raised in an order just same as mentioned above, it is then passed to the cortical learning algorithm. This algorithm works on each URL in the data stream in following particular order:

- First of all deactivate all neurons to bring a fresh state
- Convert string URL into an SDR
- Once the SDR stream is ready, input it in to htm
- Machine learning algorithm runs repeatedly
- CLA keeps learning and predicting for the forward algorithm
- It also generates percentage occurrence of new URL for an existing URL in the prediction phase without calling it an anomaly.

At the end, if the percentage is below threshold, the sample is considered as an anomaly. But if the percentages are above the set threshold for an anomaly, such sample is considered as normal traffic.

The categorization on the basis of set threshold is carried out by the classifier maintaining the prediction and anomaly scores, segregating anomalies from the normal traffic usage.

3.3.1 CLA Encoders

All the inputs from the dataset in the raw form cannot be fitted into the CLA [16]. Therefore, this algorithm needs a middle state of inputs called the sparse

distributed representation or simply SDR. Any input can be converted into an SDR by passing it through an encoder. Following is the list of encoder is provided by numenta:

- Scalar Encoder
- Date Encoder
- Category Encoder
- Coordinate Encoder
- Pass Through Encoder
- Multi Encoder

3.3.1.1 Scalar Encoder

Scalar encoded as linear encoding and converts numeric float point value to an array of binaries. Most of the weights are converted into 0s while a continuous block of 1s represents unique SDR [17] for a particular input. Scalar encoder's input depends on several factors:

- **w** - width of on bits (Odd value helps in avoiding middle value problems)
- **minval** - lower input boundary
- **maxval** - upper input boundary
- **periodic** - wrapping around values
- **n** - total number of SDR output
- **radius** - Threshold for two non-overlapping representations
- **resolution** - minimum difference of two different representations
- **name** - Optional name for encoder as in description
- **clipInput** - trimming down any non-periodic input to a standard length.

- **forced** - skipping default safety checks on the input

3.3.1.2 Date Encoder

Edit encoder converts date into a tangible form for the cortical learning algorithm. It is actually a concatenation generated by different sub-encodings dependent on aspect of the day and time.

- **season** - Season of the year
- **dayOfWeek** - Day of the week
- **weekend** - Decide if it is weekend or not.
- **timeOfday** - Time of day
- **customDays** - Custom operation for specific days
- **forced** - skipping default safety checks on the input

3.3.1.3 Category Encoder

String values are encoded through a category encoder. Most of the times, the string inputs are in mixed formats that are not linked to each other in an apparent way. Therefore, we need to adopt an encoder that deals this particular case. The followings are significant parameters for a category encoder:

- **categoryList** - all possible categories for the string inputs
- **forced** - skipping default safety checks on the input

3.3.1.4 Coordinate Encoder

If the input space is N-dimensional, depending on the radius, we use coordinate encoder to represent an SDR position for this. Followings are significant parameters for a coordinate encoder:

- **scale** - demostrating distance between 2 points on map

- **timestep** - Time between units
- **longitude** - Position Longitude
- **latitude** - Position Latitude
- **altitude** - Position Altitude

3.3.1.5 Pass Through Encoder

Once the data have been encoded through different encoders, it is then passed through this encoder to get a standard state of Sparse Distributed Representation. Parameters for Pass Through Encoder are:

- **n** - total number of bits
- **w** - width of on bits to normalize sparsity
- **forced** - skipping default safety checks on the input

3.3.1.6 Multi Encoder

Multi Encoder comprises of several encoders, each of them working of different aspect of the same data. It actually forms a dictionary and later compares multiple components accordingly. Multi encoders are the simplest of all and comprise of one major parameter:

- **encoderDefinitions** - a dictionary of dictionaries, mapping field names against their field parameters.

3.3.2 Encoding of the URLs

The cortical learning algorithm does not take urls in their native form, therefore for the processing of urls in the machine learning algorithm, it is needed to convert the urls into CLA tangible form. Each input for the CLA is encoded in an appropriate encoder suggested by numenta under their nupic platform. Initially the URL is

divided into the hostname, subdomain and the TLD.. Each portion of this division is then passed individually to the relevant encoder.

URLs have been divided into visiting hostname, subdomain and referring address, while removing all the long url strings to avoid unwanted delays and processing overload. A user who visits a normal subdomain is supposed to stay in the normal usage category. Following factors play significant role in fine tuning the encoding of URLs:

- w: 33
- categoryList: raw[‘category’].unique()
- fieldname: ‘stringInput’
- verbosity: 0
- forced: False
- type: CategoryEncoder

3.3.3 Encoding of the date

Encoding of date has been divided into two encodings named as timestamp and weekend. The Major reason behind this division is that weekend and time of visit cannot have same effective weights in the algorithm.

Here is the list of parameters set for Date time encoding using a date encoder:

- enc_timestamp: fieldname: ’timestamp’ name: enc_time

  timeOfDay: [21, 1]

  type: DateEncoder

- enc_weekend:

  fieldname: ’timestamp’

  name: enc_weekend

weekend: 21

type: DateEncoder

3.3.4 Encoding of Frecency

Frecency is one of the major components in our algorithm implementation as it gives an effective support to prediction. Frecency comes from the words Frequency and Recency of that particular visit. If a user has stopped visiting a URL, his frecency starts dropping with each passing day, This helps the machine learning process to weight if very low if the user has stopped visiting it.

As frecency is a scalar quantity, I have considered Randomly Distributed Scalar Encoder with following attributes:

- enc_Frecency:

  fieldname: 'Frecency'

  name: enc_Frecency

  categoryList: raw['Frecency'].unique()

  resolution: 0.25

  seed: 20

  type: RandomDistributedScalarEncoder

3.3.5 Encoding of VisitType

Visit type is also a string, hence we will use the string encoder name as Category Encoder. This encoder takes various types of inputs and normalize them into similar SDRs.

Following parameters have been suggested for Encoding of VisitType using Category Encoder of strings:

- w: 21

categoryList: raw['VisitType'].unique()

fieldname: 'VisitType'

verbosity: 0

forced: False

type: CategoryEncoder

3.3.6 Encoding of Referrer

Slight importance has been suggested for the referrer of some hostname visit as it may be the reason of an anomaly. Such case is also a string, therefore Category Encoder will be implemented. Referrer encoding has the same parameters as the URL Encoder:

- enc_Referrer:

  w: 31

  categoryList: raw['Referrer'].unique()

  fieldname: 'Referrer'

  verbosity: 0

  forced: False

  type: CategoryEncoder

3.3.7 CLA Anomaly Measurement

Once the output of Transition and Spatial Memory is then passed into measurement phase to predict the percentage of possible expectation of that URL visit. This prediction is done in 2 ways: 1 step anomaly detection and 5-steps anomaly detection. The 1step anomaly detection compares the nature of URL visited at time t to the nature of the visit made at time t-1 or in simple words, the current visit being compared to the most immediate previous visit.

This type of 1-step anomaly detection has been elaborated with undermentioned formula: FORMULA INSERTION where: This anomaly measurement formula produces an output that ranges from 0 to 1 depending on the probability of non-predictive nature of the event occurred. In this way, such anomaly can be extended to a broader perspective of sequences just as in this thesis it is raised to 5 steps of URL visits.

3.4 CLA Model Parameters

Cortical Learning Algorithm follows some scenariobased parameters to analyze all the input vectors from a given column and produce a pattern against particular scenario's active bits. These parameters are placed under the 'Compute' Class. Here is the list of all 15 parameters that are specified depending on the subject scenario:

3.4.1 Parameters

- **inputDimensions** - Input Vector Dimensions. Format: (height, width, depth,...)
- **columnDimensions** - Column Dimensions. Format: (height, width, depth,...)
- **potentialRadius** - Defines extent of input bit or the global coverage of the input space. Format: Any integer
- **potentialPct** - Defines column's potential pool. Format: 0 to 1 i-e 0.5 means half the bits are being considered.
- **globalInhibition** - Selection of winning columns on basis of being most active or being related to neighbors. Format: Boolean
- **localAreaDensity** - Desired density of active columns. Format: Real value
- **numActiveColumnsPerInhArea** - Alternative way of controlling active columns' density. Format: Real Value

- **stimulusThreshold** - Minimum number of active synapses in an active column. Format: Integer Value
- **synPermInactiveDec** - Parameter to decrement number of inactive bits in each round. Format: Real Value
- **synPermActiveInc** - Parameter to increment active synapse in each round
- **synPermConnected** - Potential of synapse to contribute to cell's firing strength. Format: Real Value
- **minPctOverlapDutyCycle** - A floor value when reached, that particular column gets boosted again. Format: 0 to 1.0 (Default=0.001)
- **dutyCyclePeriod** - Higher or shorter duty cycles either slowing down or making system unstable respectively. Format: Real Number(Default = 1000)
- **boostStrength** - Boosting encourages columns to have similar activeDuty-Cycles as their neighbors. Format: Real value
- **seed** - initiation for the pseudo-number generator. Default = -1
- **spVerbosity** - Verbosity level. Format: 0 to 3
- **wrapAround** - Determines if inputs at the beginning and end of an input dimension should be considered neighbors when mapping columns to inputs. Format: Boolean

3.4.2 Adopted CLA Parameters

After a rigorous testing of inputs with different CLA Spatial Pooling Parameters, I reached this below mentioned set of spatial pooling parametric values 3.1 for our special case of anomaly detection in web services usage:

And table 3.2 delivers list of Temporal Memory Parameters bearing values specific to our case study:

Parameter	Adopted Value	Range	Default
inputDimensions	(27, 32)	-	(32,32)
columnDimensions	(2048, 32)	-	(64,64)
potentialRadius	16	-	16
potentialPct	0.5	0-1	0.5
globalInhibition	False	Boolean	False
localAreaDensity	-1.0	-	-1.0
numActiveColumnsPerInhArea	40	-	10.0
stimulusThreshold	0	-	0
synPermInactiveDec	0.02	0-1	0.008
synPermActiveInc	0.03	0-1	0.5
synPermConnected	0.650	0-1	0.1
minPctOverlapDutyCycle	0.005	0-1	0.001
dutyCyclePeriod	500	-	1000
activeDutyCycles	0.01	=>0	0.0
wrapAround	True	Boolean	True

Table 3.1: Spatial Pooling Parameters

Parameter	Adopted Value
verbosity	0
columnCount	2048
cellsPerColumn	3
inputWidth	2048
seed	1960
temporalImp	cpp
newSynapseCount	20
initialPerm	0.5
permanenceInc	0.250
permanenceDec	0.01
maxAge	0
globalDecay	0.0
maxSynapsesPerSegment	32
maxSegmentsPerCell	128
minThreshold	4
activationThreshold	9
outputType	normal
pamLength	1

Table 3.2: Temporal Memory Parameters

In the end, there are Cortical Learning Parameters that adjust learning according to set methods and do the mathematics of prediction based on Spatial Pooling and Temporal Memory. Values for clParams in our case study as given in table 4.1.

Parameter	**Adopted Value** [0.5ex]
verbosity	0
regionName	SDRClassifierRegion
alpha	0.075
steps	'1,5'
maxCategoryCount	1000
implementation	cpp

Table 3.3: Cortical Learning Parameters

3.5 Comparison of the Model

Most suitable algorithm for comparison is the Sequence Time Delay Embedding Algorithm that detects anomaly in the patterns online.

3.5.1 The Stide(Sequence Time-Delay Embedding) Algorithm

Stide works closely in methodology with Cortical Learning Algorithm but it needs to be trained first. It is trained on a dataset comprising of normal sequences. Later, this normal learning is compared to sub-sequences, keeping the input length same. Meeting these conditions, stide produces a list of all possible mismatches.

CLA, in the first comparison does not require any prior learning with a normal dataset, takes lead by working with inputs of different lengths and adjusts them to a standard in case needed.

Stide is also not applicable to our problem because it involves a pregenerated dataset, while we do not offer any such data set to CLA for learning.

3.5.2 Hidden Markov Model(Baun-Welch Forward Algorithm)

Hidden Markov Model has also been considered in this scenario under the Forward Algorithm in Baun-Welch Algorithm. This approach seems suitable because it uses an online approach to the case. A deep analysis of this method suggests that HMM Baun-Welch Algorithm slows down and even may stop processing with the increase in the number of state samples [10]. As our sequence dataset may rise more than 50000 samples in a month in the corporate sector, HMM does not seem to accommodate such a huge number.

3.6 Performance Measures

Final results from the algorithm should divide a dataset into two classes: normal or anomaly. In our dataset, anomaly data are less than 1%, while normal data ranges above 99%, therefore a plain accuracy of boolean decision is not a suitable approach. It is one of the reasons for choosing CLA over other conventional algorithms that it caters the results under a threshold to differentiate anomaly from normal URLs. There is one drawback to any algorithm (including CLA) we implement in our scenario that behavior will be poor at first, therefore dataset adaptation is necessary in this case. Discarding initial 2000 sequences appear to be a way to avoid misleading results and declared as the training period of the algorithm. The rest of the sequences are considered in performance measures.

The time complexity is also one of the significant factors when it comes to analysis of big datasets for any machine learning algorithm. For example HMM is one of the algorithms that may slow down and fall behind once the dataset approaches HMM threshold. HTM can readjust its internal states as the learning gets bigger.

Chapter 4

Results

I could not find any algorithm than HTM-CLA that best matches our case scenario of web usage security under given constraints:

- Anomaly Detection is done online
- No major time delay is expected in processing
- Dataset can be huge
- No normal dataset is available for prelearning phase
- Incorporate 6 significant factors of an input pattern

4.1 HTM-CLA

The results are mostly generated by weight-changing method applied to all the significant factors of data in Cortical Learning Algorithm. Initially we will look into the specifications of giving data stream.

4.1.1 Data Attributes

Following are the major data attributes considered in my case scenario:

- timestamp

- visitType
- Frecency
- Referrer
- hostname
- tld
- subdomain

We will look into all of them and analyse their apparent behavior. Later this behavior is passed to the CLA for anomaly detection.

4.1.1.1 timestamp

First of all, the timestamp is in general detailed form as given below:
9/2/2017 15 : 55
It is then divided into day of the week, day of month, weekend and hour of day to visualize the general pattern followed in Internet service usage.

- Day of the Week: From graphics given in figures 4.1 and 4.2, we can surely say that most of the internet service is used on Sunday followed by Monday and the rest of days every week in our samples. Therefore, the probability of an anomalous usage is the highest on a Tuesday and the lowest on a Sunday, but this is just the beginning.
- Day of the Month: From graphics given in figures 4.3 and 4.4, Internet service usage increased in the mid and end of each month. While at the closer end of month, figures show that this is the time usage can hit an anomaly. Our anomaly rating is increased if it is a Tuesday and closer to the end of the month. Throughout this analysis of data attributes, we will look into an apparent anomaly from visualizations.
- Hour of the Day: From graphics given in figures 4.5 and 4.6, most web service usage happens in the afternoon till sunset and least usage in the

early hours of the day. It shows that the user checks most of his internet services during late hours of the day. We can safely expand out assumption based on visualization of the data that anomaly is most likely to occur in the early hours of the last Tuesday each month.

- Is it a weekend?: From graphics given in figures 4.7 and 4.8, on a weekly average, web service usage is higher on weekends compared to the average of the other days. We can safely expand out assumption based on visualization of the data that anomaly is most likely to occur in the early hours of the last Tuesday each month.

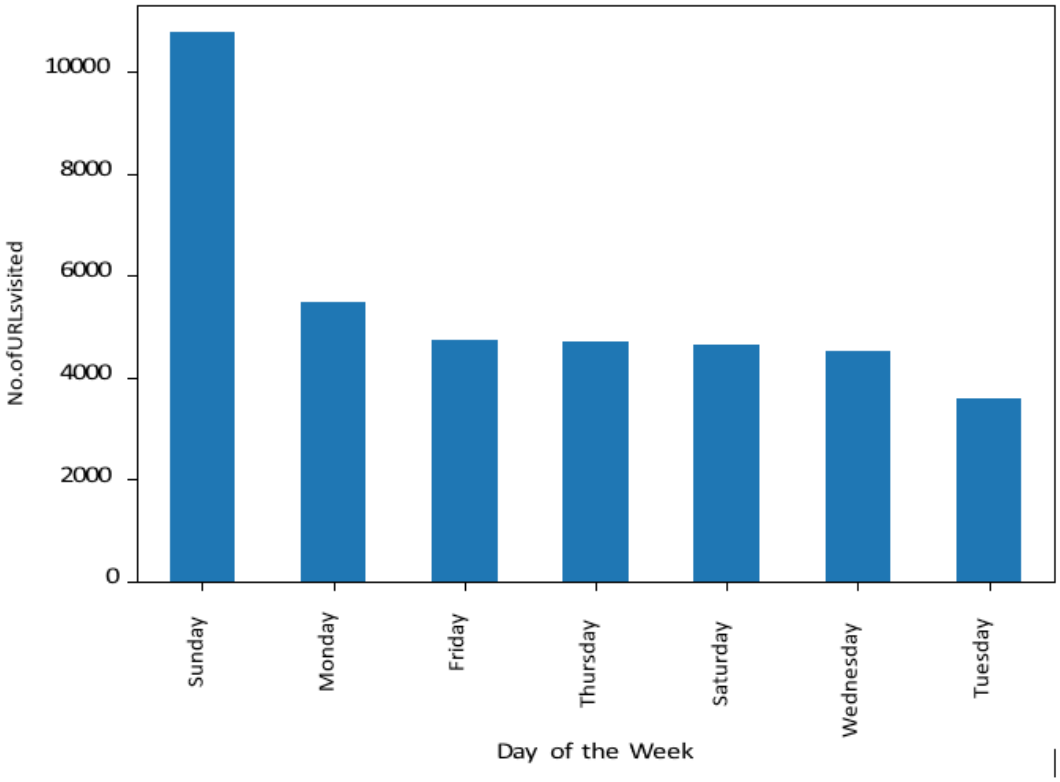

Figure 4.1: Day of the Week

4.1.1.2 visitType

VisitType as defined in chapter 3 is the root of every URL visited by the subject. From the data we analyzed, it has been divided into following categories:

- **Link**: It is a general form of link that may have appeared from the option

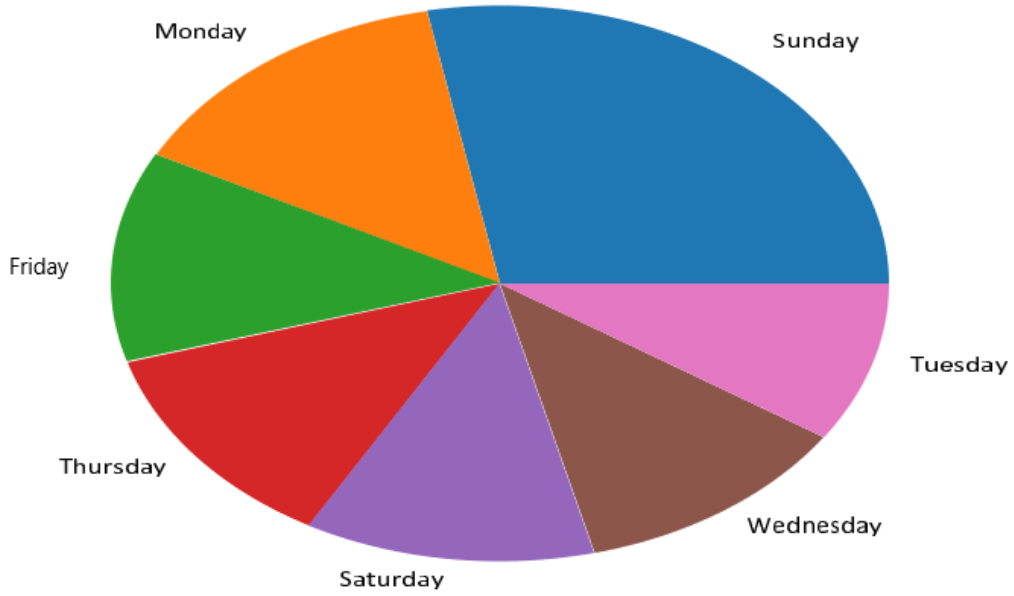

Figure 4.2: Pie: Day of the Week

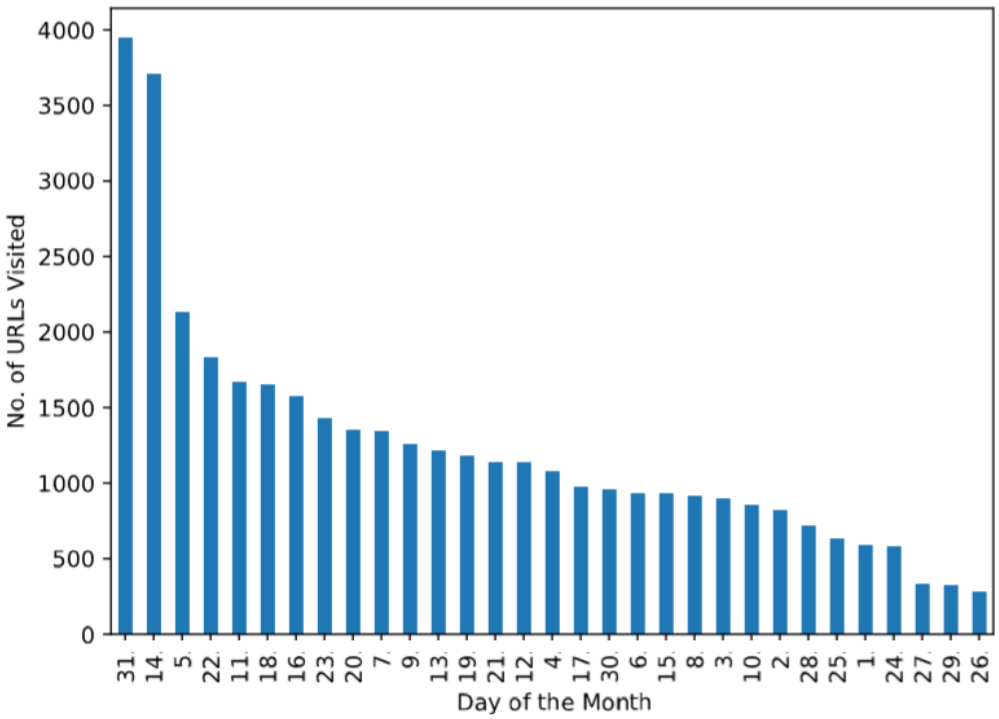

Figure 4.3: Day of the Month

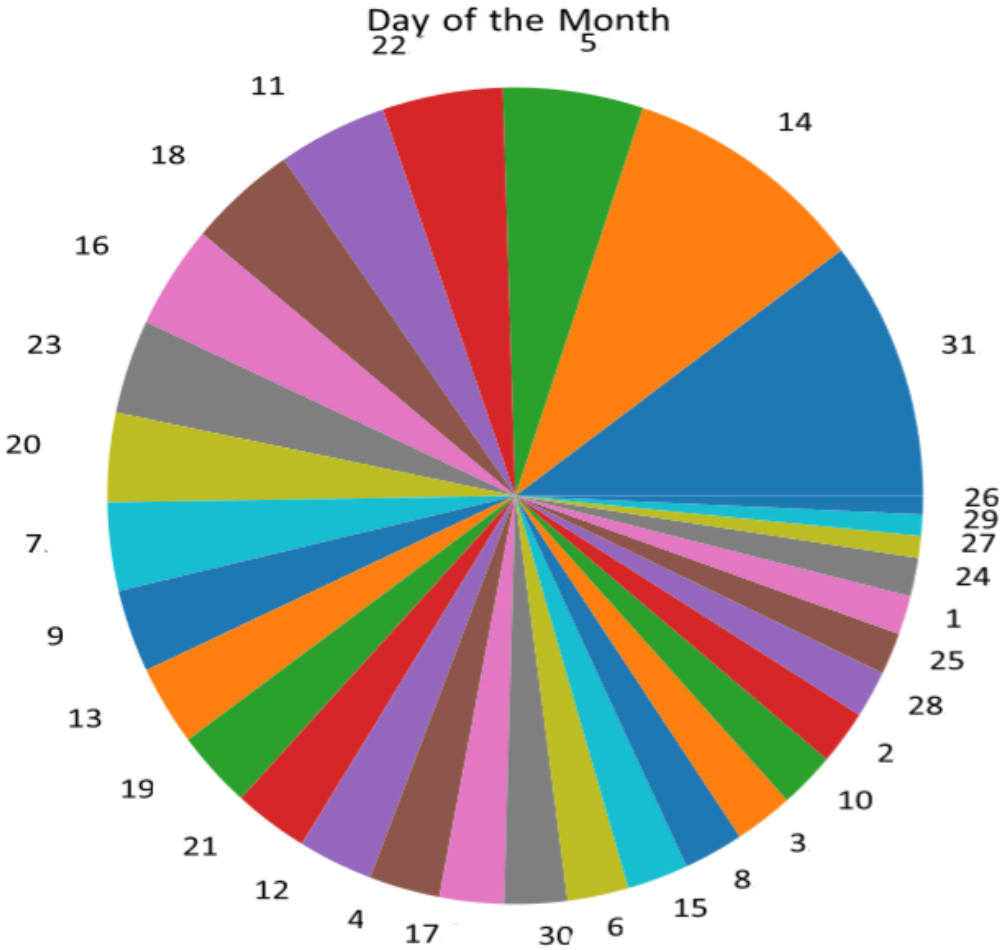

Figure 4.4: Pie: Day of the Month

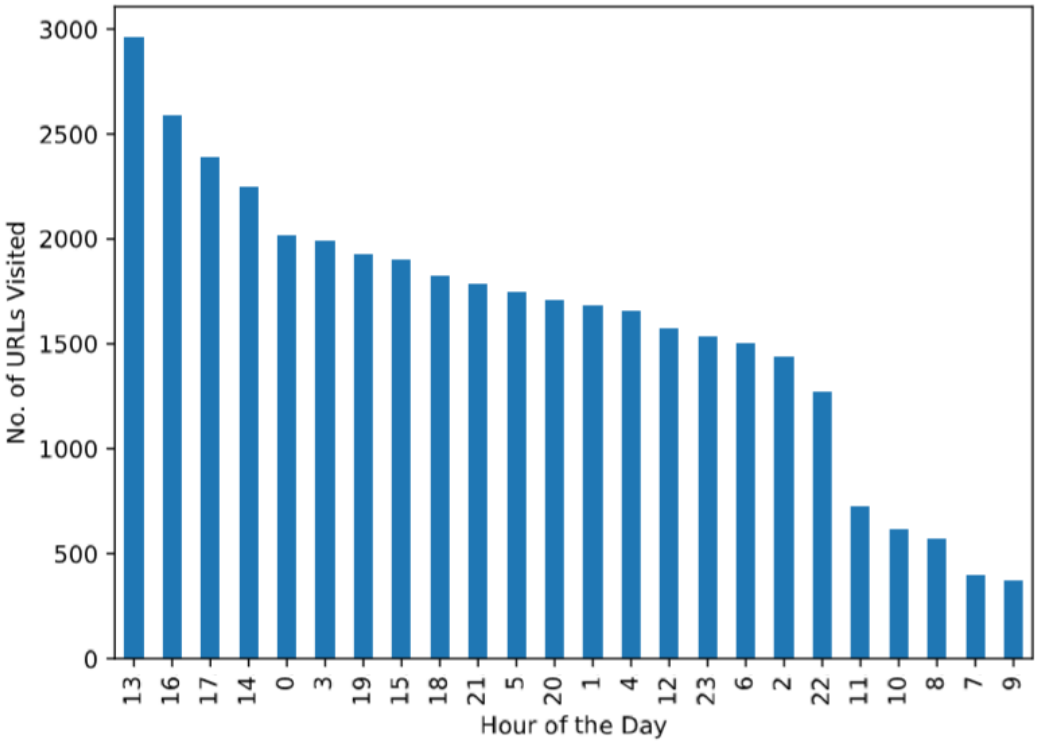

Figure 4.5: Hour of the day

of 'Open in a New Tab', History, Speed visit, an auto-complete suggestion or a visit type that was not properly resolved.

- **TemporaryRedirect**: It is the form of visit when one URL temporarily redirects to another URL in order to reach the permanent redirection.
- **Permanent Redirect**: Permanent landing of a webpage from another webpage
- **Download**: Any link that leads the usage to downloading a file from its webpage.
- **Bookmark**: Loading a saved page in a web browser for future reference is call a bookmark visit. from its webpage.
- **TypedURL**: Visiting a webpage by typing its alphabetical URL is called Typed URL Visit Type of each visit has been elaborated in figure 4.9.

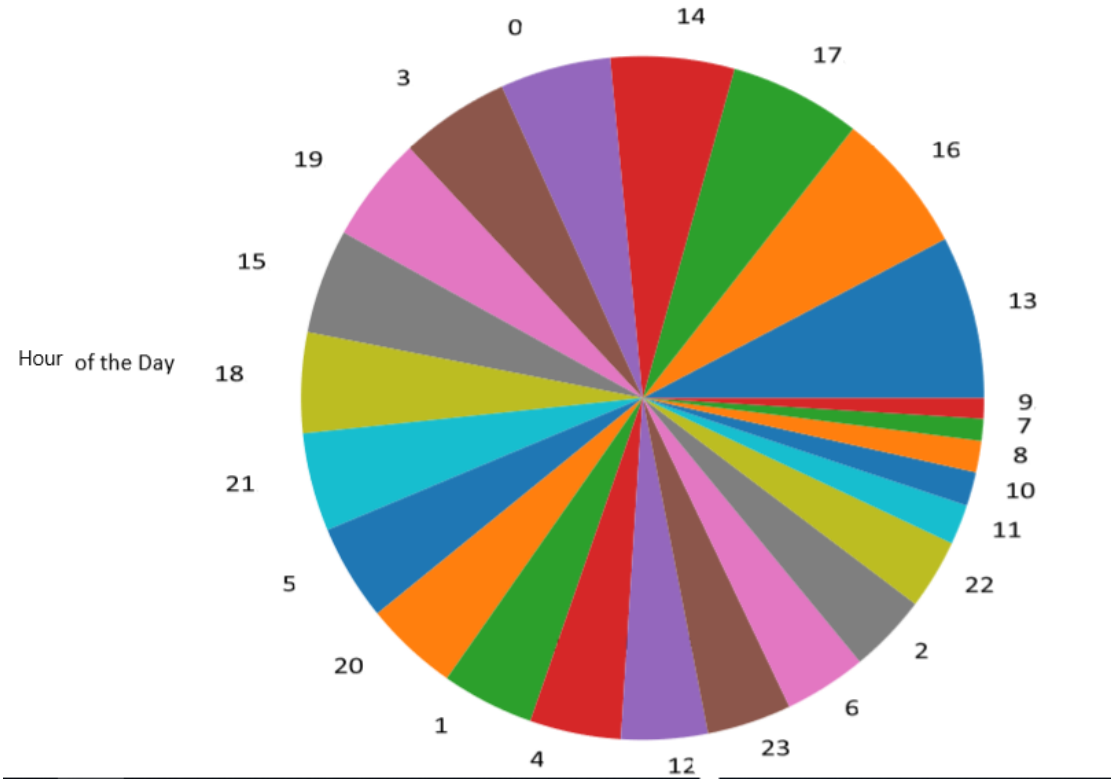

Figure 4.6: Pie: Hour of the day

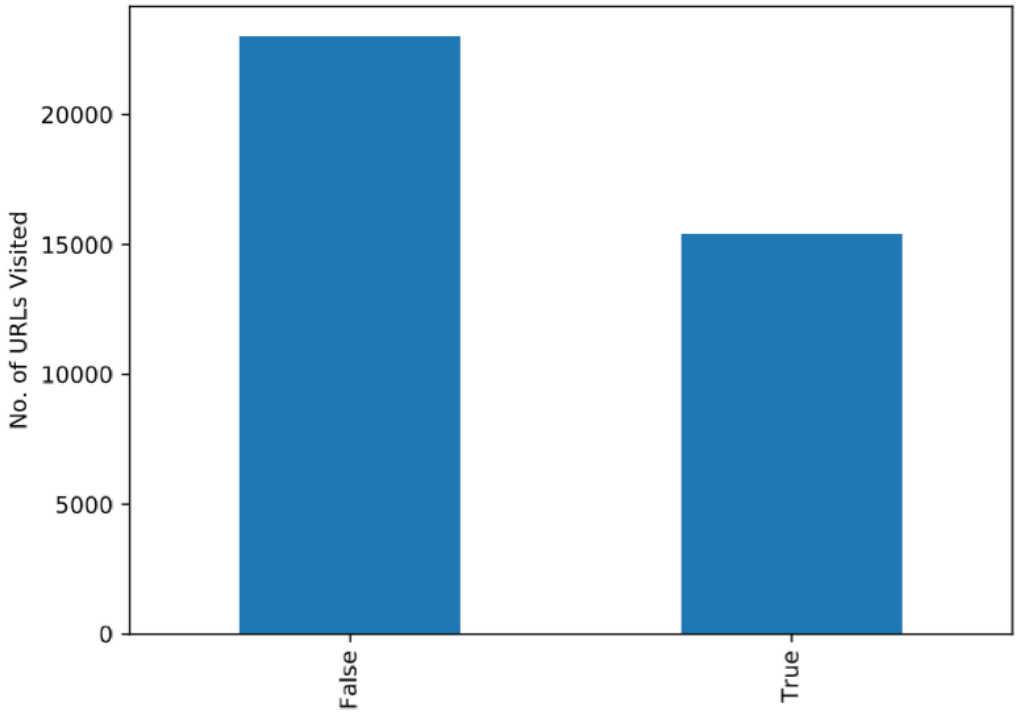

Figure 4.7: Is it a weekend?

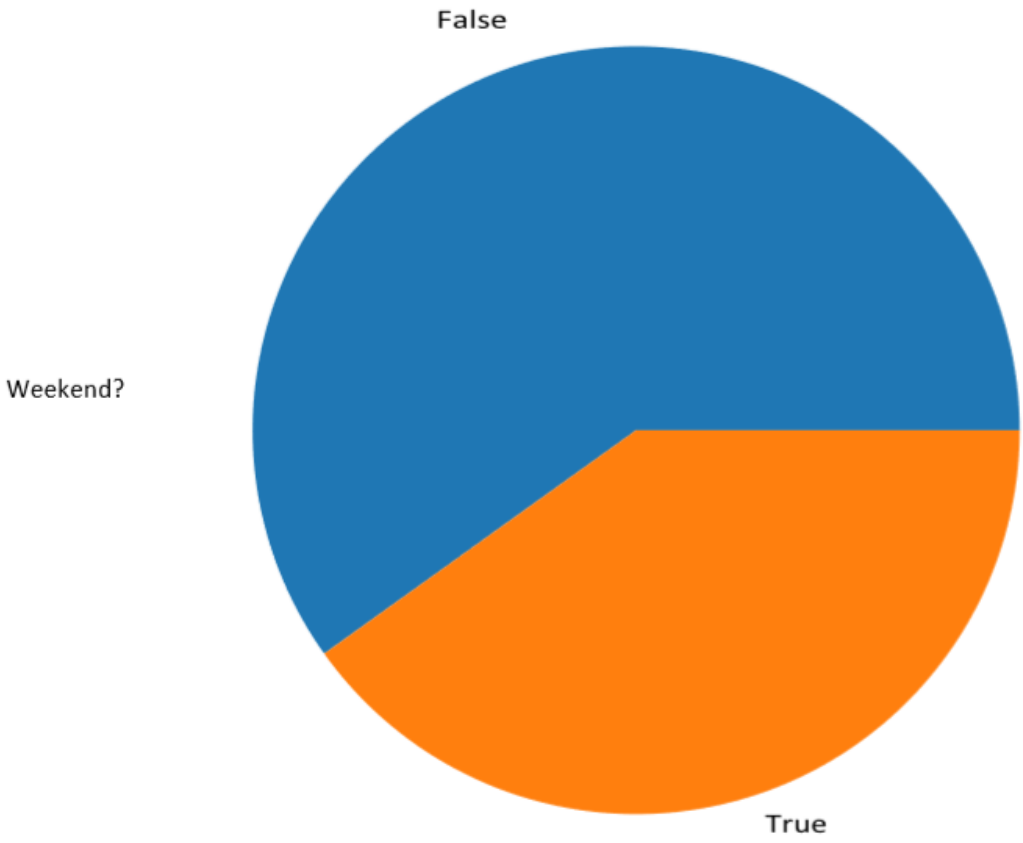

Figure 4.8: Pie: Is it a weekend?

From this graphic, on a weekly average, web service usage is higher on weekends compared to the average of the other days. We can safely expand out assumption based on visualization of the data that anomaly is most likely to occur in the early hours of the last Tuesday each month.

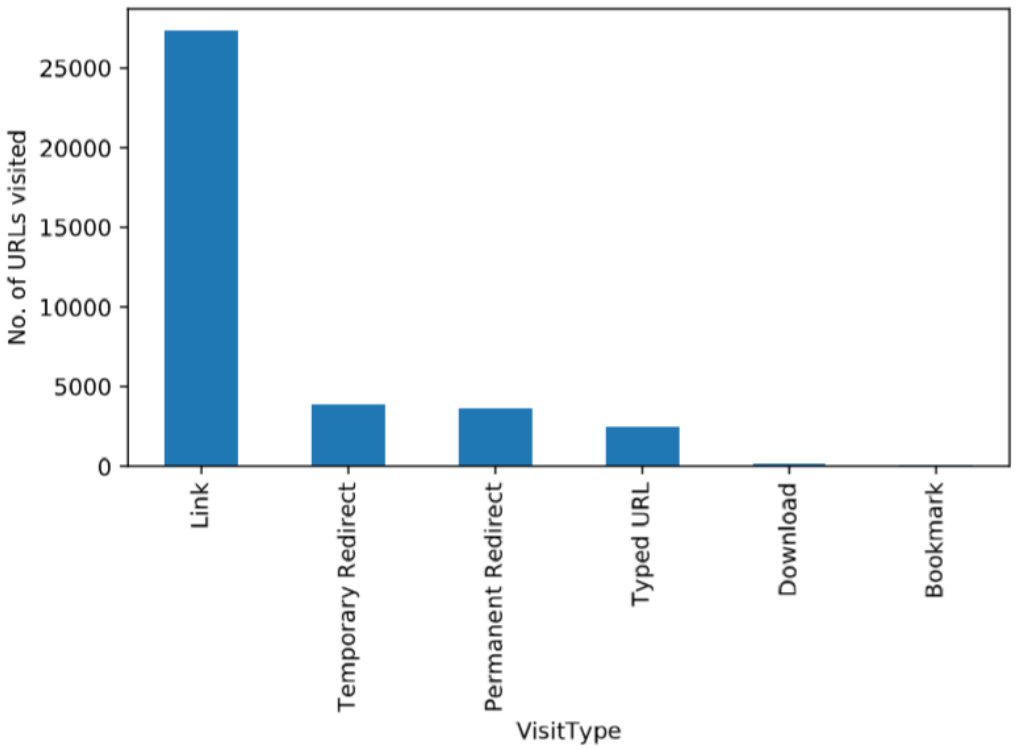

Figure 4.9: Pie: Is it a weekend?

4.1.1.3 Referrer

A referrer is the page which redirects a user from one web service to another through either a temporary redirect or a permanent redirect. Most of the times, it becomes a general practice to visit some particular webservice redirected from a native page that the user does not bother using another method.

In such a case, if a user loads a web page from a different referrer in the early hours of the last Tuesday of the month, then this is an apparent anomaly so far. On the basis of our subject's internet usage, figure 4.10 shows referrers from highest probability to the lowest. For simplicity purposes, least possible referrers have been overlooked.

4.1.1.4 hostname

It is the main page under consideration where a subject happens to land from any means; a temporary or permanent redirect or simple URL is typed. This is the page where a user is supposed to take any normal or anomalous action. Figure

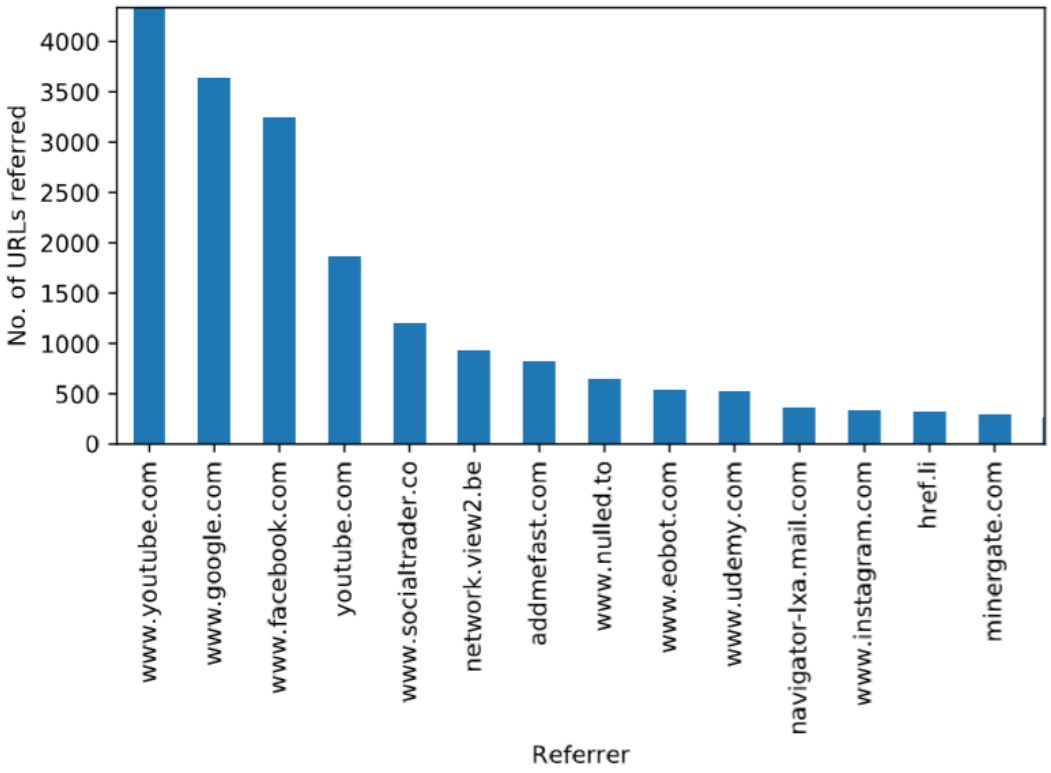

Figure 4.10: Referrer

A referrer is the page which redirects a user from one web service to another through either a temporary redirect or a permanent redirect. Most of the times, it becomes a general practice to visit some particular webservice redirected from a native page that the user does not bother using another method.

4.11 depicts few of the most visited hostnames by the subject.

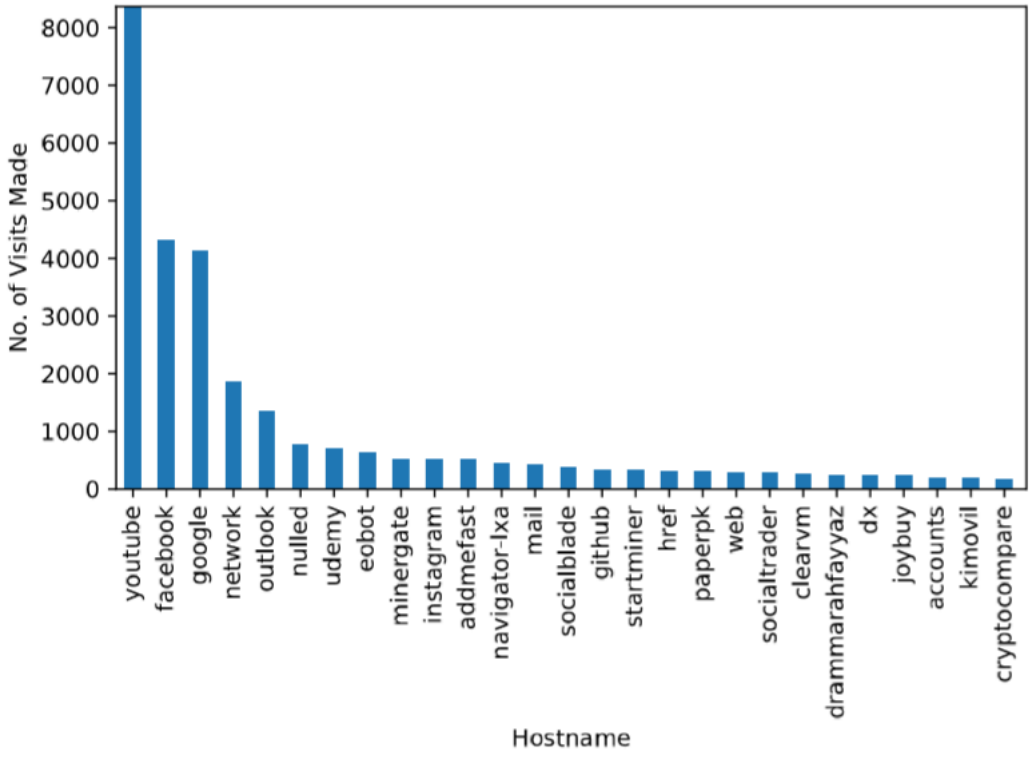

Figure 4.11: hostname

It is the main page under consideration where a subject happens to land from any means; a temporary or permanent redirect or simple URL is typed. This is the page where a user is supposed to take any normal or anomalous action. This figure depicts few of the most visited hostnames by the subject.

4.1.1.5 tld

tld also known as the Top-Level Domain is a hierarchical name of the URL being visited. It is usually a short length set of alphabets. The most common tld is .com. For an instance if the subject under consideration usually visits a domain with a com tld and all of a sudden he lands on the same domain name with a different tld, an alarm should notify his action to be judged. This action is then compared to all the probabilities based on all other factors.

textbf4.1.1.6 subdomain

It is the subpart of hostname being visited and does not affect much as if the

hostname is justified as from a normal web usage category, there is very rare chance of an anomaly in its subdomain. CLA caters for such cases as well because it looks into minute percentages of probability without negligence of being rarely possible.

Figure 4.12 depicts few of the most visited subdomains by the subject. where a user is supposed to take any normal or anomalous action. Figure 4.11 depicts few of the most visited hostnames by the subject.

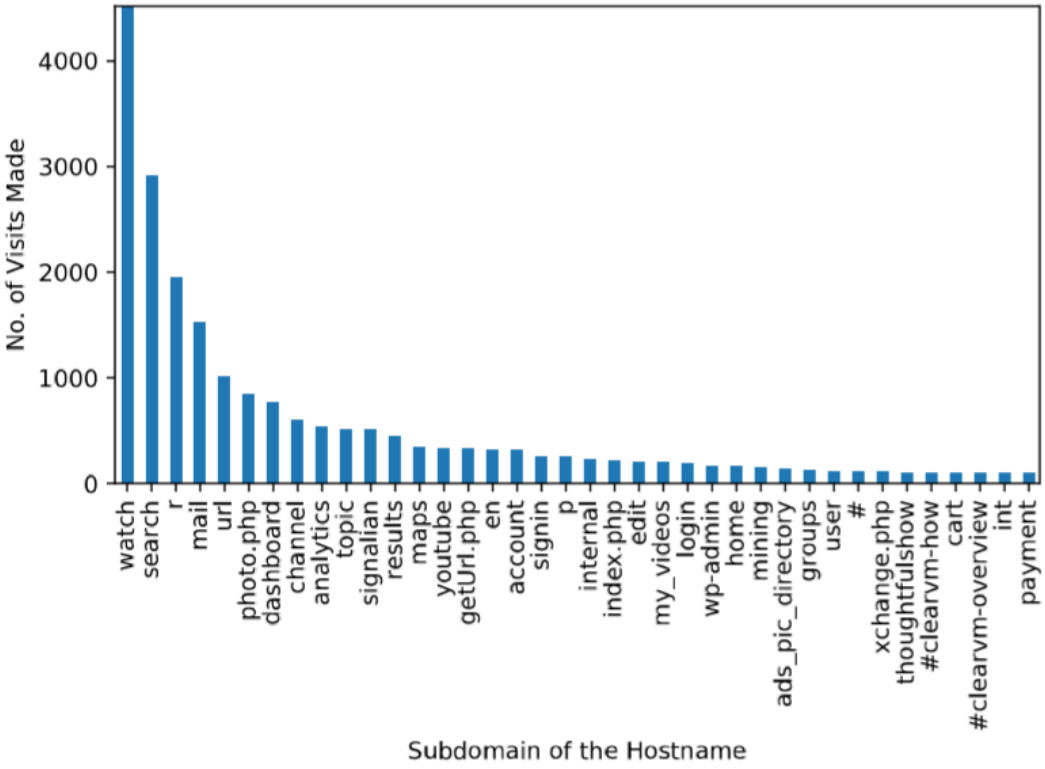

Figure 4.12: subdomain

It is the subpart of hostname being visited and does not affect much as if the hostname is justified as from a normal web usage category, there is very rare chance of an anomaly in its subdomain. CLA caters for such cases as well because it looks into minute percentages of probability without negligence of being rarely possible.

4.1.2 Analysis of Data Segments

Analysis of data attributes show that the most effective factors influencing results were:

- timestamp

  Day of the week

  Hour of the day

  Weekend

  hostname

  Referrer

During the trial period, weight of hostname and timestamp have been varied from 21 to 43(Keeping odd values considered) which changed number of possible anomalies and processing time over 38000 samples, presented in table 4.1 below:

Therefore, it seems really impressive to keep weight of the hostname at 27 which gives an appropriate number of anomalies under short time. Therefore, we will proceed with w(hostname) as 27 and vary weight for the timestamp similarly from 21 to 43 in table 4.2:

The tables 4.1 and 4.2 imply that considering our particular case, w = 27 is perfect weight for both timestamp and hostname which has also been demonstrated in graphs 4.13 and 4.14. It can be observed that on a normal computer, Hierarchical Temporal Memory is performing extra-ordinary with a real-life dataset extracted from usage history of our subject with more than 38000 samples. It also caters for length of data sequences unlike the Hidden Markov Model which becomes slower as the length and even number of data sequences increases. We also see that processing time can be monitored and managed based on case scenario requirements.

w(hostname)	Processting Time(s)	Anomalies
21	193	15
23	206	15
25	210	16
27	211	24
29	216	24
31	221	24
33	226	24
35	231	24
37	240	22
39	246	24
41	251	24
43	260	24

Table 4.1: Varying weight w of hostname

w(timestamp)	Processting Time(s)	Anomalies	w(hostname)
21	210	23	27
23	212	23	27
25	211	24	27
27	210	24	27
29	211	24	27
31	214	24	27
33	212	24	27
35	217	24	27
37	209	22	27
39	214	24	27
41	219	25	27
43	211	24	27

Table 4.2: Varying weight w of timestamp

Considering our particular case, w = 27 is perfect weight for both timestamp and hostname which has also been demonstrated in graphs

Figure 4.13: Weight to processing time graph

Figure 4.13: Weight to Processing Time Graph

The new algorithm tested in thi work, CLA,seems to handle this better, reaching precision scores around 70%. This is still not production-ready performance, but the models were not ne-tuned, and better results can certainly be reached.

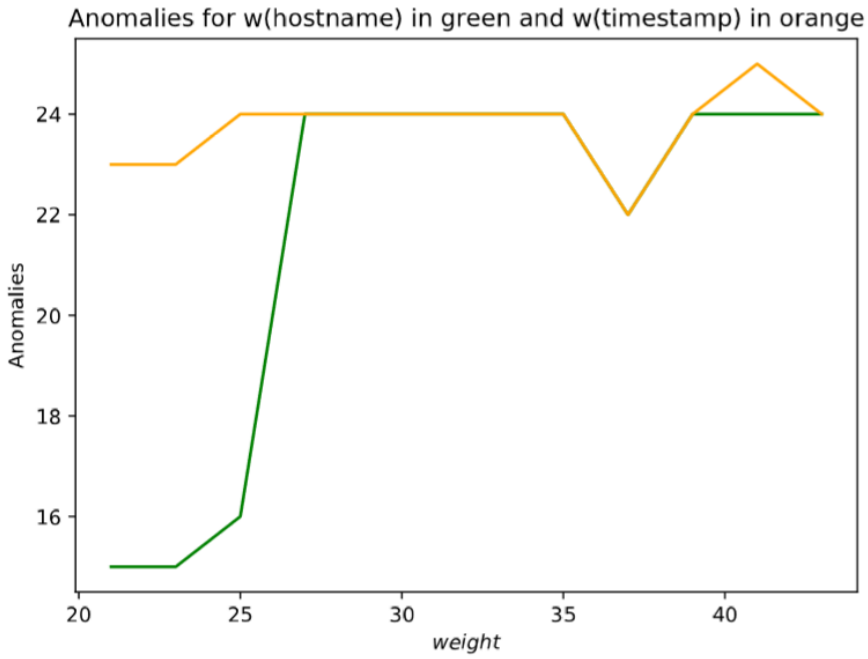

Figure 4.14: Weight to anomalies graph

Figure 4.14: Weight to Processing Time Graph

It can be observed that on a normal computer, Hierarchical Temporal Memory is performing extra-ordinary with a real-life dataset extracted from usage history of our subject with more than 38000 samples. It also caters for length of data sequences unlike the Hidden Markov Model which becomes slower as the length and even number of data sequences increases. We also see that processing time can be monitored and managed based on case scenario requirements.

4.2 Interpretation of Results

I observed that classic state-of-the-art temporal anomaly detection techniques - HMMs and t-stide - cannot reliably handle the problem. It is likely explained by the data characteristics described above: it is composed of short sequences over a very large alphabet. Both methods appear to handle this case very poorly. They were developed for different scenarios, where sequences are relatively long compared to the size of the alphabet. It is very apparent for the HMM, for which the size of the hidden state does not seem to have any impact on the performance. Suggesting the model is completely incapable of learning any temporal pattern on this data (as even an hidden state of size 1 gives roughly the same performance). Basic tests on HMMs with varying sequence lengths and alphabet size seems to confirm that this class of models struggles to learn when the size of the alphabet gets significantly larger that the average size of the sequence. However, I could not find any literature sources to conform or contradict this observation. The new algorithm tested in this work, CLA, seems to handle this better, reaching precision scores over 99% once the program is fine-tuned with hit-and-trial methods and manual inspection of results.

Chapter 5

Conclusions and future work

5.1 Conclusion

Cortical Learning Algorithm as proposed has promising potential when it comes to securing a web-based computer network in corporate level. This algorithm is already being used commercially as well as open source by Numenta where they predict stock market values. Numenta has also implemented it in Natural Language Processing Schemes, performing to its best possible. Bringing this neocortical algorithm into web services security can help network administrators deal with the complex nature of network traffic. In comparison with conventional state-of-the art methods like Hidden Markov Model and t-stide, Cortical Learning Algortihm performs better with increasing complexity in nature of data sequences at temporal level. CLA also has the capacity to learn from the real-time stream of data sequences, making it possible to predict any possible near-future anomaly.

This work, as graduate thesis, tried to highlight major aspects if CLA in Cyber Security covering few of the significant aspects that highly effect the way anomaly detection is improved. This algorithm depends on a lot of such aspects which were not an easy task to evaluate in single thesis to obtain best results. Such a massive evaluation needs a high-end computer and some parameter based tuning algorithms to be carried out thoroughly; Bayesian and Genetic Algorithms belong to such catalytic processes.

CLA also features out in its effectiveness in a way that it is first of its kind algorithm to be implemented in Web usage security because others, such as t-stide and Hidden Markov Model do not perfectly fit in the scenario.

It can be concluded that Numenta CLA is a reliable resource in order to detect attacks on web services by categorizing such behaviors as anomalies; enhancing efficiency of the cyber security wing in a corporate sector.

5.2 Future Work

Work on CLA in Cyber Security can be extended to implementing it on Network level where IP Addresses and Port numbers associated with the source and destination of every packet is analyzed to check if some irrelevant external or internal user makes any changes to the web services [18]. CLA also has the potential to categorize traffic on the basis of packet sequences, packet flags, data segment sizes and frame sizes on the network layer.

Extending the same approach, we can implement CLA on traffic routers that may monitor complete network traffic in both corporate and private/public computer networks. Numenta recently launched its opensource program under the label of NuPic which helps developers, researchers and students to carry out learning and research-based work using this extraordinary algorithm.

For research reference, Numenta also maintains an ongoing YouTube Video Lecture series under the label 'HTM School'. This series offers an impressive learning content which can be used to conduct research and interrogate possibilities of CLA in the real-world case scenarios.

References

[1] E. L. Witzke, 'Computer network security: Then and now', 2016 IEEE International Carnahan Conference on Security Technology (ICCST), 2016. DOI: 10.1109/ ccst.2016.7815710.

[2] M. Ahmed, A. N. Mahmood, and J. Hu, 'A survey of network anomaly detection techniques', Journal of Network and Computer Applications, vol. 60, pp. 19-31, 2016. DOI: 10.1016/j.jnca.2015.11.016.

[3] 'A survey on attack detection techniques in delay tolerant network', International Journal of Modern Trends in Engineering & Research, vol. 4, no. 2, pp. 101-109, 2017. DOI: 10.21884/ijmter.2017.4060.dknlc.

[4] W. Zhang, S. Qin, B. Yi, and P. Tian, 'Study on learning effect prediction models based on principal component analysis in moocs', Cluster Computing, Oct. 2018. DOI: 10.1007/s10586-01825940.

[5] 'Stream mining', Encyclopedia of Machine Learning and Data Mining, pp. 1199-1200, 2017. DOI: 10.1007/978-1-4899-7687-1_789.

[6] T. C. Silva and L. Zhao, 'Case study of network-based unsupervised learning: Stochastic competitive learning in networks', Machine Learning in Complex Networks, pp. 241-290, 2016. DOI: 10.1007/978-3-319-17290-3_9.

[7] F. Dufrenois and J. Noyer, 'One class proximal support vector machines', Pattern Recognition, vol. 52, pp. 96-112, 2016. DOI: 10.1016/j.patcog.2015.09.036.

[8] Kneller, A. and Thornton, J. (2015). Distal dendrite feedback in hierarchical temporal memory. 2015 International Joint Conference on Neural Networks (IJCNN).

[9] Roongroj Nopsuwanchai and Povey, D. (n.d.). Discriminative training for HMM-based offline handwritten character recognition. Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.

[10] Ahmad, S., Shakil, A. and Balbed, M. (2008). Study on the effect of number of training samples on HMM based offline and online signature verification systems. 2008 International Symposium on Information Technology.

[11] Cui, Y., Surpur, C., Ahmad, S. and Hawkins, J. (2016). A comparative study of HTM and other neural network models for online sequence learning with streaming data. 2016 International Joint Conference on Neural Networks (IJCNN)

[12] Li, W. and Franzon, P. (2016). Hardware implementation of Hierarchical Temporal Memory algorithm. 2016 29th IEEE International System-on-Chip Conference (SOCC).

[13] Zhang Jie, Huang Zhitong and Wang Xiaolan (n.d.). Selection and analysis of HMM's state-number in speech recognition. ICSP '98. 1998 Fourth International Conference on Signal Processing (Cat. No.98TH8344).

[14] Wei Lu, Chiu, K. and Gannon, D. (n.d.). Building a Generic SOAP Framework over Binary XML. 2006 15th IEEE International Conference on High Performance Distributed Computing.

[15] Gutiérrez, M., Pongilupi, J. and LLinàs, M. (2010). Web Sessions Anomaly Detection in Dynamic Environments. ISSE 2009 Securing Electronic Business Processes, pp.216-220.

[16] HTM Forum. (2018). NuPIC Encoders. [online] Available at: https://discourse.numenta.org/t/nupic-encoders/2153 [Accessed 3 Jun. 2018].

[17] Obst, O. (2014). Distributed machine learning and sparse representations. Neurocomputing, 124, p.1.

[18] Mopari, I., Pukale, S. and Dhore, M. (2008). Detection and defense against DDoS attack with IP spoofing. 2008 International Conference on Computing, Communication and Networking.

[19] Agrawal, P. and Franklin, S. (2014). Multi-layer Cortical Learning Algorithms. 2014 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB).

[20] Yan, F., Jian-Wen, Y. and Lin, C. (2015). Computer Network Security and Technology Research. 2015 Seventh International Conference on Measuring Technology and Mechatronics Automation.