ALEXANDRU IOAN CUZA" UNIVERSITY OF IASI

FACULTY OF COMPUTER SCIENCE

BACHELOR PAPER

Preventing web-based attacks through Hidden Markov models

proposed by

Alexandru Martiniuc

Session: July, 2019

Scientific coordinator

Assoc. Prof. PhD Gavriluț Teodor Dragoș

ALEXANDRU IOAN CUZA" UNIVERSITY OF IASI

FACULTY OF COMPUTER SCIENCE

Preventing web-based attacks through Hidden Markov models

Alexandru Martiniuc

Session: July, 2019

Scientific coordinator

Assoc. Prof. PhD Gavriluț Teodor Dragoș

Avizat,

Îndrumător lucrare de licență,

Assoc. Prof. PhD Gavriluț Teodor Dragoș.

Data: .... Semnătura: ....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Martiniuc Alexandru** domiciliat în **România, jud. Botoșani, mun. Botoșani, str Mihail Kogalniceanu, nr. 57, sc. A, et. 4, ap. 18**, născut la data de **16 iunie 1997**, identificat prin CNP **1970616070052**, absolvent al Universităţii "Alexandru Ioan Cuza" din Iaşi, FACULTY OF COMPUTER SCIENCE, **FACULTY OF COMPUTER SCIENCE** specializarea **informatică în limba engleză**, promoția 2019, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Preventing web-based attacks through Hidden Markov models** elaborată sub îndrumarea domnului **Assoc. Prof. PhD Gavriluț Teodor Dragoș**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată , ____________________ Semnătură ____________________

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Preventing web-based attacks through Hidden Markov models**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultatţi de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea "Alexandru Ioan Cuza", să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Dată , ________________ Semnătură ________________

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil şi sursă, să aparţină autorului prezentei lucrări, Martiniuc Alexandru

Încheierea acestui acord este necesară din următoarele motive:

Procurarea, analiza si procesarea de date au fost realizate în colaborare cu firma Bitdefender, compania furnizând fondurile şi tehnologiile de actualitate necesare în realizarea acestora.

Iaşi, Data ________________

Decan **Iftene Adrian** Absolvent **Alexandru Martiniuc**

Semnătura: .... Semnătura: ....

Table of Contents

Chapter 1

Introduction

1.1 Motivation

It is obvious that the world we live is evolving to an unprecedented level in history due to the emergence and development of the World Wide Web.

Now more and more people are open to the idea of being exposed in the virtual space through social networking, media networks, automation of any type of payment, generally almost anything related to material assets and private aspects of an individual's life , as these systems are designed to help us become much faster, are easier to use, and can access more and more information.

Yet, the cybersecurity landscape unfolds to us a large variety of threats: malware, spamming , phishing, social engineering, exploits , zero-day vulnerabilities and many others, which makes millions of people become victims of theft and causes billions of dollar losses every year.

In the past 10-15 years the spread of this threats has increased significantly. This made the researchers to look for new ways to identify and stop malware sources, because classic methods such as blacklisting , dont have the abillity to detect new malware URLs.

Our research will try to come up with a solution to this situation using the probabilistic methods of the Hidden Markov Models in an attempt to create a proactive system, capable of detecting malicious URLs which unfolds in front of us just with a simple click.

1.2 Preliminary and related work

The global address of resources in the World Wide Web is the Uniform Resource Locator (from now we will refer it with URL). A URL is composed from many parts, some mandatory and some optional such as : the protocol identifier , which indicates what protocol to use, the subdomain, the main domain , top level domain and others. An URL follows the next structure:

$$protocol : //domain - name.top - level - domain/path$$

URLs are a big part of todays cybersecurity landscape due to the speed and ease of access and return of information. This is the reason why malware writers use them intensely in any possible context. Some of the technologies related to accessing resources on the Internet, that are now used in malicious ways, were in the beginning created with a totaly different purpose.

For example, Microsoft Office templates, a starting point for a newly created document, for the purpose of automating repetitive and common tasks, saving time for more consistent and important tasks, have been used with malicious intent. Bad intentioned people have abused this technology to conceal malicious payloads (that can be used as a scenario for connecting to a command and control server , CC server, or connect , download and execute content from various IPs) inside a document, to be executed when run (because in the beginning the macros were executed automatically). In recent version , macros are disabled by default. Because of this, malware writers are using fake names and different error messages misleading people to activate, and thus, run the template script.

This method is also used by phishing and spam emails, the are sending unwanted messages repeatedly . According to a 2018 survey from F-secure, 9 out of every 10 infections have been done by malware email content.

Hidden Markov models are some kind of stochastic finite automatas which where described from early 1960 by Leonard Esau Baum and improved by Ruslan L. Stratonovich, who described the forward-backward algorithm (i.e. algorithm that computes the marginal distribution of the internal unobservable states of a system using the and observed sequence of emissions) in a problem close related to the current one, namely the optimal nonliniar filtering problem, which is a mathematical model used in fields related to signal processing.

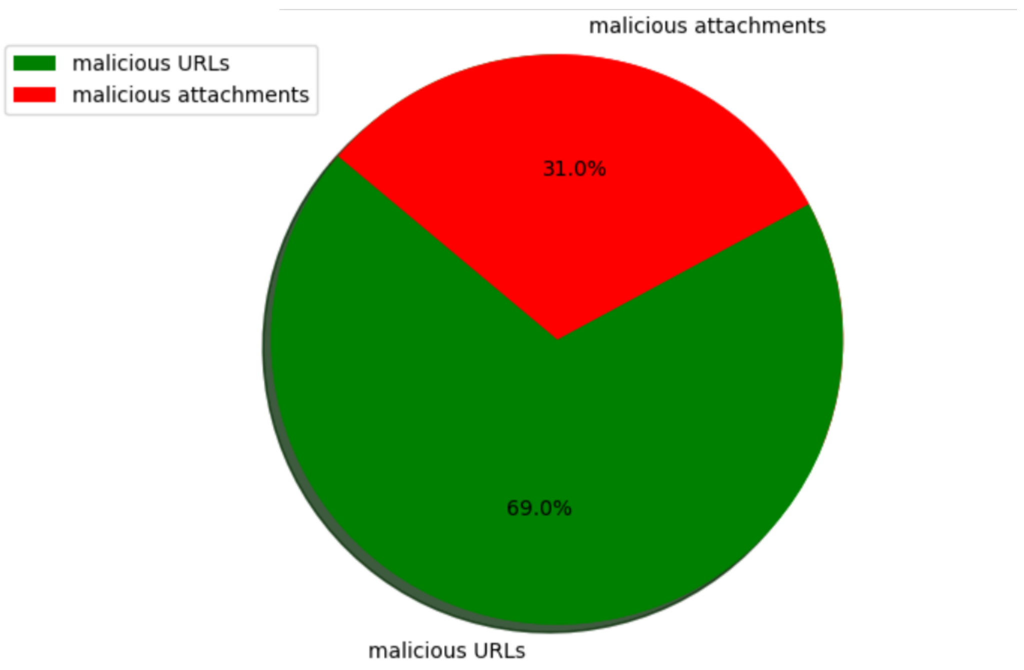

Figure 1.1:

Hidden Markov models are generally used in various fields from pattern analysis like biological sequence analysis , speech recognition and protein modeling . Researche done in the fields of hidden Markov models show that this kind of models are useful for detecting some type of malware families, using features extracted that reflect the behaviour of the malicious content.

In the field of probabilistic heuristics for malware detection this type of system was used to identify many different cyber threats that don't only focus on malicious URLs detection's.

Many researches have shown that Hidden Markov models are useful for detecting different classes of malware. For example [3] presents an investigation on metamorphic malware generators and several compilers, because of the similarity between compilers and generators using HMMs. Also [4] is a survey on Profile Hidden Markov model applied to malware files behaviour (the sequence of system calls triggered) [1]

1.3 Contribution

Considering the landscape of the possible threats ,this paper describes our approach to the problem of malicious URLs classification using the hidden Markov model to create a system capable of clustering this kind of input.

[1] a type of HMM capable of identifying similarities between different sequences)

Given a dataset of URLs, initializing the HMM and use the core algorithms (that will be presented in the theoretical background) create a score that will be used to classify the input in 2 categories: benign and malware.

The score creation will be based only on the lexicograph features of the URL that will be normalized and the formated to fit the template of an n-gram - those will represent the observations given to the system, that will describe the underlying complex processes that are taking place, called hidden states.

The classification part will take place using the k-means algorithm, where the set to be classified will be made up of the previous verdicts. This step will give us the centroids positions that will help us to classify the input.

Chapter 2

Theoretical background

2.1 Markov Model

Before defining what a Markov Model is , we have to present what is a stochastic process. A stochastic process is a family of random variables

$$(X(i))_x \in I \tag{2.1}$$

defined over a space with probability, where each variable:

$$X_i = X(i) : \Omega \to R \tag{2.2}$$

represents a step of the process A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

A process satisfies the Markov property (sometimes characterized as "memorylessness") if one can make predictions for the future of the process based solely on its present state just as well as one could knowing the process's full history, hence independently from such history; i.e., conditional on the present state of the system, its future and past states are independent.

A discrete-time Markov chain with a finite number of states is a stochastic process

$$(X_n)_{n\geq 1}, \text{ where } X(i) : \Omega \to S = x_1, x_2, ..., x_n \tag{2.3}$$

having the Markov property, namely that the probability of moving to the next state depends only on the present state and not on the previous states:

$$P(X_n + 1 = x|X_1 = x_1, X_2 = x_2, ...X_n = x_n) = P(X_n + 1 = x|X_n = x_n) \tag{2.4}$$

and both parts of the equality are well defined, meaning they satisfy the inequality :

$$P(X_1 = x_1, X_2 = x_2, ...X_n = x_n) > 0 \tag{2.5}$$

A Markov Chain is called stationary if the following equality is true:

$$P(X_n + 1 = x_j | X_n = x_i) = P(X_n = x_j | X_{n-1} = x_i) \forall n \geq 2, s_i, s_j \in S \tag{2.6}$$

In the current paper we will only consider discrete Markov chain (non-stationary) having a finite number of states as we will see later on.

An example of a Markov Model or Markov chain is the following state diagram:

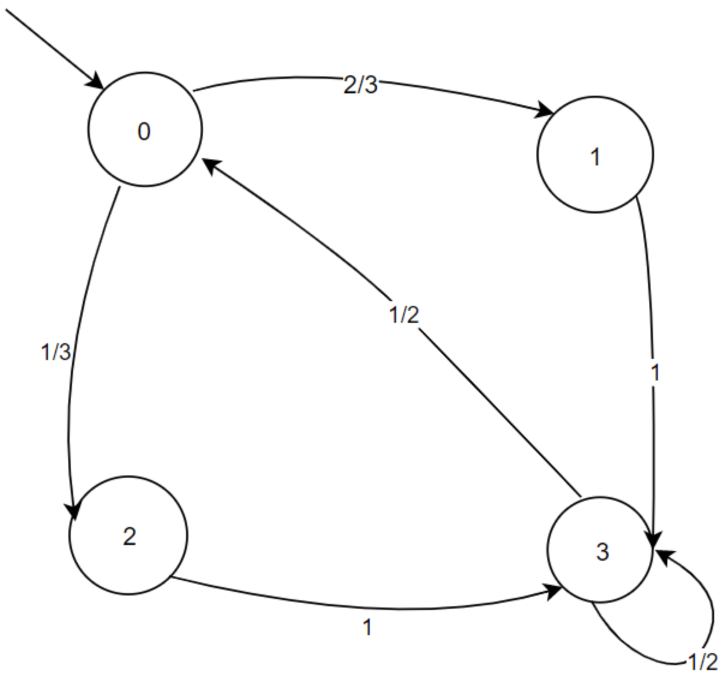

Figure 2.1: Markov model graph example

This model can be described using the transition matrix of the system:

$$\begin{Bmatrix} 0 & 2/3 & 1/3 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1/2 & 0 & 0 & 1/2 \end{Bmatrix}$$

The transition matrix is a stochastic matrix because all of its elements are probabilities and the sum on every row is equal to 1.

The above model could also be called an observable Markov model since the output of the process is a series of states that represent an observable event at each instance of time. As we will see later on there exists another type of Markov model with states that arent obsevable which is of interest for us, this being the core of this paper.

2.2 Hidden Markov Model

A Hidden Markov model (HMM) is a statistical Markov model that has the power to model a system that we assumed is a Markov process but with states that are not observable (we will refer to those with hidden states).

The urn model

The best way to present and extend the ideas of the HMM is considering the following scenario. You have N urns, each containing a big number of balls with M different colors. According to an unknown internal random process, a ball is randomly took from one of the urns (also random choosed) and the color of the ball will represent the observation. We record this event and then put the ball back. In the same manner a random urn is again choosen, and the ball selection is repeated, obtaining a finite sequence of observations.

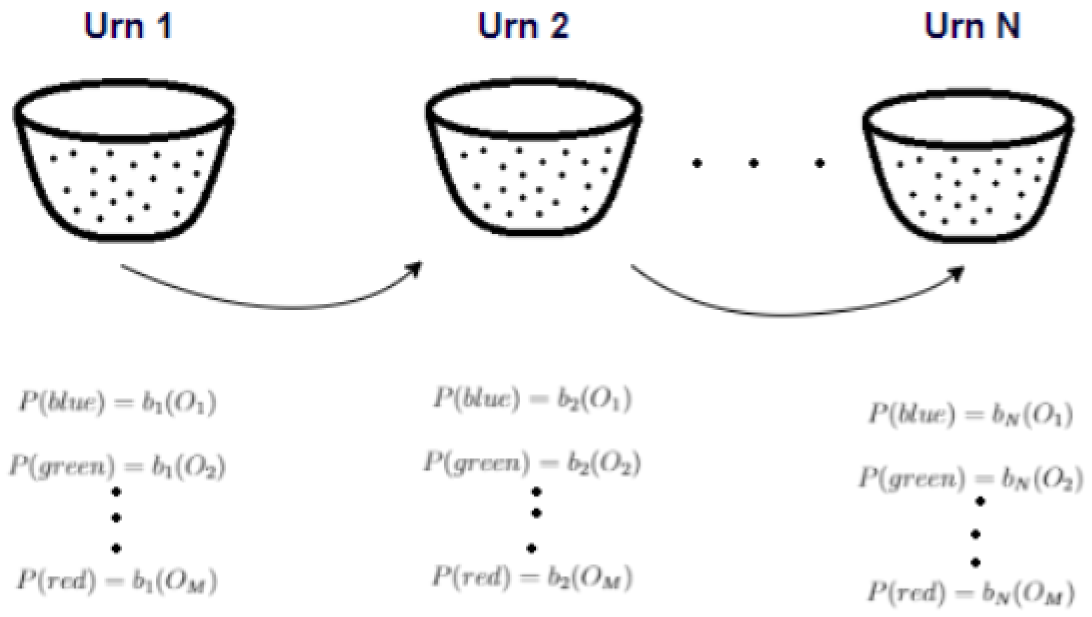

Figure 2.2: Urn model

The HMM that would model such situation will be represented in the following manner: the hidden states will be represented by the urns, the colors of the balls will be the observations (output from the system), and the choice of urns will be the transition matrix.

Elements of an HMM The above mental exercise gives us an idea of what an HMM is. We will now define elements of an HMM and explain how the model generates observatios.

We denote an HMM as $\lambda = (A, B, \pi)$ thus having the folowing notations:

1). N - the number of hidden states in the model (very often a fixed number when use in practice)

2). M - the number of observation (the alphabet)

3). T - length of the observation sequence (discrete time)

4). π - 1 x N row vector, initial state distribution

5). A - N x N matrix, the state transition matrix

6). B - N x M matrix, the observation probability matrix

7). O_i - observation, where i = $\overline{1, T}$

A visual representation of an HMM with its elements will be given by the following Trellis graph [1] :

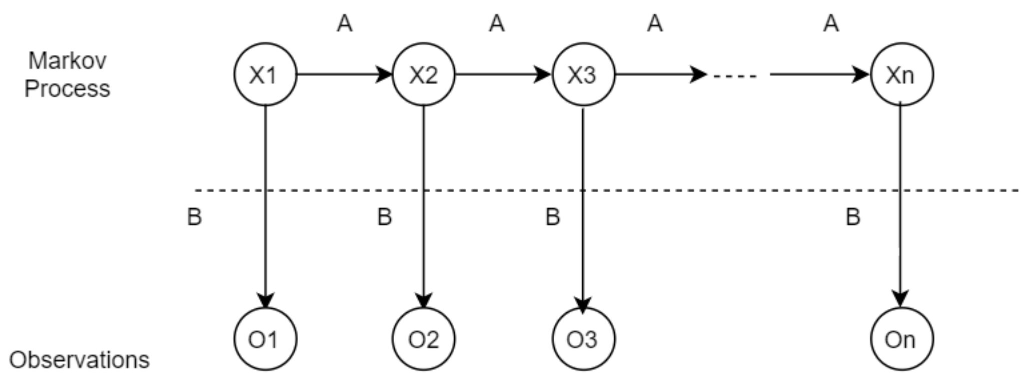

Figure 2.3: HMM Trellis graph representation

[1]A trellis is a graph whose nodes are ordered into vertical slices (time) with each node at each time connected to at least one node at an earlier and at least one node at a later time

2.3 The 3 basic problems for HMM

With the basic elements of the constructed HMM been presented we are capable to resolve real-world situations founded in the following 3 problems[1]:

Problem 1: Given a model λ = (A, B, π) and an observation sequence O, we can determine the probability $P(O|\lambda)$. That is, an observation sequence can be scored to see how well it fits a given model.

Problem 2: Given a model λ = (A, B, π)and an observation sequence O, we can determine an optimal state sequence for the Markov model.

Problem 3: Given an observation sequence O and the parameter N (the number of hidden states) we can determine a model λ that maximizes probability of O. That is, we can train a model to best fit an observation sequence.

2.4 Solutions triad

2.4.1 Solution to problem 1 - The Forward-Backward Algorithm

Given the model λ we want to calculate the probability of the observation sequence O that being, given $O_1, O_2, ..., O_T$ calculate $P(O|\lambda)$.

Considering a sequence of hidden states q_i with $i = \overline{1,T}$. For simplificity we will the denote this sequence with Q. We can see that the probability $P(O|\lambda)$ can be written as :

$$P(O|\lambda) = \sum_{allQ} P(O, Q|\lambda) \tag{2.7}$$

Lets see how we can make this ecuation simpler. The observation probability of the sequence, knowing that the initial state is q_1 will be:

$$P(O|Q, \lambda) = P(O_1|q_1, \lambda) \times P(O_2|q_2, \lambda) \times \cdots \times P(O_T|q_T, \lambda) = \prod_{i=1}^{T} P(O_i|q_i, \lambda) \tag{2.8}$$

If we assume that the observations are disjoint :

$$O_1 \perp O_2 \perp \ldots \perp O_T = \perp_{i=1}^{T} O_i \tag{2.9}$$

then using the multiplication rule [2], we have the following:

$$P(O|Q, \lambda) = b_{q_1}(O_1) \times b_{q_2}(O_2) \times \cdots \times b_{q_T}(O_T) \tag{2.10}$$

[2]multiplication rule

Also, the probability of the hidden states given the system λ is :

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \dots a_{q_{T-1} q_T} \tag{2.11}$$

It is clear why we introduced those equalities. Using the multiplication rule the $P(O|\lambda)$ probability can be written as:

$$P(O|\lambda) = \sum_{allQ} P(O|Q,\lambda) P(Q|\lambda) = \sum_{q_1,q_2 \cdots q_T} \pi_{q_1} a_{q_1 q_2} b_{q_1}(O_1) a_{q_2 q_3} b_{q_2}(O_2) \dots a_{q_{T-1} q_T} \tag{2.12}$$

Indeed we come to a better form of the ecuation starting from the initial but at a deeper look we can see the upside. Knowing that the alphabet of the hidden states is of length N,the length of the observation sequence is T, and by making an iterations over all states we can see that the complexity of the algorithm for finding $P(O|\lambda)$ is of the order $\mathcal{O}(T \cdot N^T)$. That is the drawback. The algorithm is exponential in size and in real world application is unfeasible unleast you use very small values for N, and T. If we want to model bigger systems we must introduce the core of the Hidden Markov Model.

The Forward-Backward Algorithm

Forward-Backward is a dynamic programming algorithm which manages to solve the previous problem in polynomial time. First we have to define the forward variable until time t given the model , the observation series $O_1, O_2, \dots O_n$ and current state q_t as:

$$\alpha_t(i) = P(O_1, O_2, O_3 \dots O_t, q_t|\lambda) \tag{2.13}$$

where t is the current observation The algorithm has 3 steps:

- Initialize the forward variables

$$\alpha_1(i) = \pi_i \cdot b_i(O_1), \text{ where } i = \overline{1, N}$$

- Induction

$$\alpha_{t+1}(i) = \left(\sum_{1}^{n} a_t(i) a_{ij} \right) b_j(O_{t+1}) \text{ , where } t = \overline{1, T-1} \text{ and } j = \overline{1, N}$$

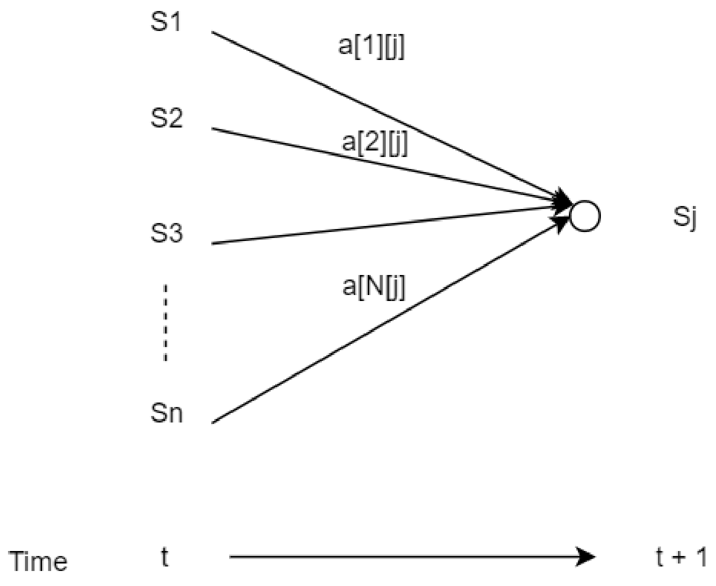

Figure 2.4: Operations sequence for computing $a_t(i)$

- Result

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

We will also introduce the backward algorithm which will be used for the solving of Problem 3. In a similar manner as the above we will introduce the backward variable which describes the probability of the observation sequence from time t + 1 to the end:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, O_{t+3} \dots O_T | q_t, \lambda) \tag{2.14}$$

The algorithm also has 3 steps:

- Initialization:

$$\beta_T(i) = 1, \text{ where } i = \overline{1, N}$$

- Iterate

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j), \text{ where t = T - 1 , T -2} \dots \text{, 1, 1} = \overline{1, N}$$

- Result:

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(O_1) \beta_1(i)$$

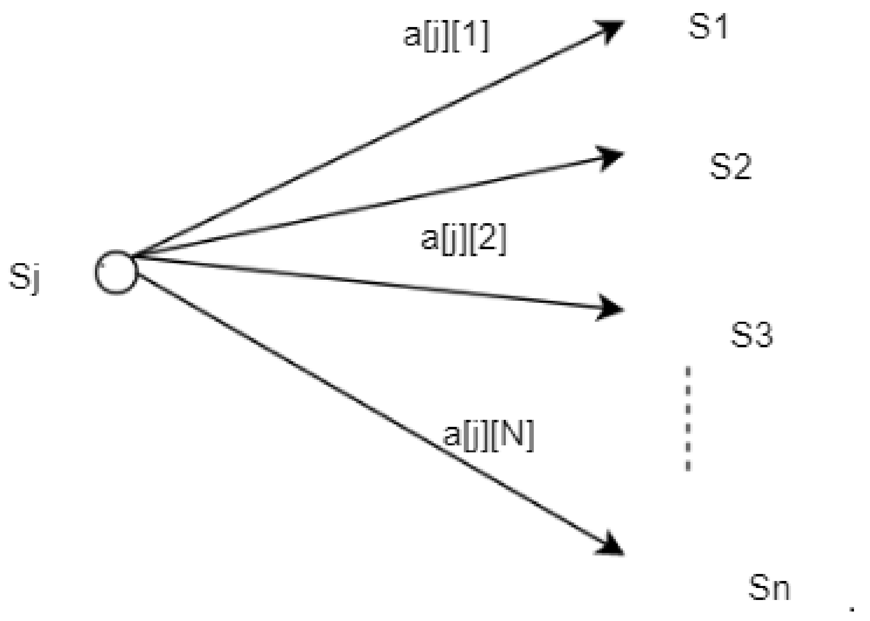

Figure 2.5: Backward

2.4.2 Solution to problem 2 - The Viterbi Algorithm

To find the most likely sequence of hidden states (i.e maximize the quantity $P(Q, O|\lambda), Q = q_1, q_2, \ldots, q_T, O = O_1, O_2, \ldots, O_T$) we will use dinamic programming to solve the maximum path problem and the coresponding sequence of states atached to this path, the Viterbi algorithm.

We will define the quantity :

$$\delta_t(i) = max_{q_1,q_2,\ldots,q_{t-1}} P(q_1, q_2, \ldots, q_{t-1}, O_1, O_2, \ldots, O_{t-1}|\lambda) \tag{2.15}$$

where $\delta_t(i)$ is the maximum score along a path given the first t observations. The algorithm works in the following way; starting from the first observation we recurse backwards retaining at each time the maximum probability from the previous states. The 3 steps of the algorithm:

- Initialization

$$\delta_1(i) = \pi_i \cdot b_i(O_1)$$

$$\phi_1(i) = 0 \text{ where i} = \overline{1, N}$$

- Recursion

$$\delta_t(j) = max_{1 \le i \le N}[\delta_{t-1}(i)a_{ij}]b_j(O_t)$$

$$\phi_t(j) = argmax_{1 \le i \le N}[\delta_{t-1}(i)a_{ij}]$$

where t = $\overline{2, T}$ and j = $\overline{1, N}$

- Result

$$q_T = argmax_{1 \le i \le N}[\delta_T(i)]$$

And the sequence of states will be given by :

$$q_t = \phi_{t+1}(q_{t+1}) \text{ , where t } = \overline{T-1, 1}$$

2.4.3 Solution to problem 3 - The Baum-Welch algorithm

Of all three problems, the third proble surely is the most difficult. It's an interesting problem because given a series of observations $O = O_1, O_2 \ldots O_n$, there is no optimal way to maximize the P(O $|\lambda$), (i.e. adjusting the model parameters (A, B, π) . The reestimation process is decribed by defining the variable:

$$\xi_t(i,j) = P(q_t, q_{t+1}|O, \lambda) = \frac{P(q_t, q_{t+1}, O|\lambda)}{P(O|\lambda)} \tag{2.16}$$

or

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \tag{2.17}$$

From (4.3) we can observe that :

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\xi_t(i,j) = \sum_{t=1}^{N_1} P(q_t, q_{t+1}, O|\lambda) = P(O|\lambda) \tag{2.18}$$

The semnification of this variable is the following: the probability from transitioning from hidden state i at time t to hidden state j given the observation sequence and the model.λ. Lets denote and interpret the sum :

$$\sum_{j=1}^{N}\xi_t(i,j) = \gamma_t(i) \tag{2.19}$$

as the number of times the state q_i is visited over a period of time.

Using the above formulas, we can obtain the parameters of new system $\overline{\lambda}$ as:

$$\overline{\pi_i} = \gamma_1(i) \tag{2.20}$$

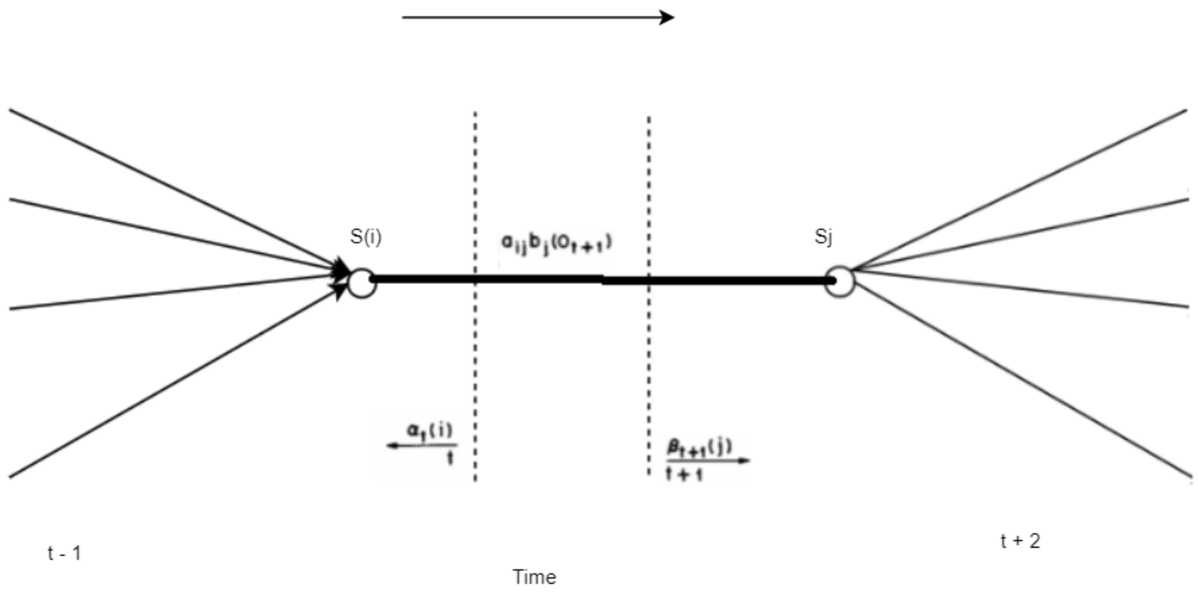

Figure 2.6: Baum-welch

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.21}$$

$$\overline{b_j(k)} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)_{O_t=v_k}}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.22}$$

The formulas from x - y , are used for the reestimation of the new model, thus, using $\overline{\lambda}$ instead of λ, we will better values , in the sense that $P(O|\overline{\lambda}) \geq P(O|\lambda)$.

In fact, the above results can be also obtained using an EM algorithm. In the case of HMMs, the Baum-Welch algorithm steps are equivalent to the EM algorithm steps, where the above reestimation formulas and results can be obtained by maximizing the Baum's auxiliary function obtaining:

$$max_{\overline{\lambda}}[Q(\lambda, \overline{\lambda})] \rightarrow P(O|\overline{\lambda}) \geq P(O|\lambda) \tag{2.23}$$

where:

$$Q(\lambda, \overline{\lambda}) = \sum_{allQ} P(Q|O, \lambda) log[P(Q|O, \overline{\lambda})] \tag{2.24}$$

2.5 N-grams

An n-gram is a continuous sequence of items of size n from a given text. The type of the items can vary between phonemes, letters, words and others according to the application in which they are used. Depending of the size of n we can meet the following names, unigram for 1-gram , bigrams or digrams for 2-ngram, or trigrams (3-gram). In the following paper we going to build our system using the extracted n-grams, where n = $\overline{1,3}$.

2.6 K-means clustering

K-means clustering is a unsupervised machine learning method used in cluster analysis with the aim to partition a set of M observation into k clusters, each belonging to nearest cluster. The result of this algorithm can be visualized as a Voronoi diagram.

The the k-means clustering algorithm is the following: Given a set of observations S = $(x_1, x_2, \ldots, x_n) \in R$, we define the function:

$$J(\mu^{(t)}) = \sum_{x_i \in S} \parallel x_i - \mu^{(t)}(x_i) \parallel^2 \tag{2.25}$$

where $\mu^{(t)}$ denotes the set of centroids at iteration t, $\mu^{(t)}(x_i)$ is the cluster center to which x_i is assigned to, and $\parallel \parallel$ is the euclidian norm. The purpose of this algorithm is that at each iteration of the algorithm, to satisfy the following inequality:

$$J(\mu^{(t)}) \leq J(\mu^{(t-1)}) \tag{2.26}$$

or simply put, given an initial partition of size k, P = $S_1, S_2, \ldots, S_n$ to find the values such that :

$$argmin_S \sum_{i=1}^{k} \frac{1}{2|S_i|} \sum_{x,y \in S_i} \parallel x - y \parallel^2 \tag{2.27}$$

This is done by the 2-step algorithm in it's simples form, in the following way:

- Step 1 - The assignment step

  Each observation will be assigned to a closest cluster according to the euclidian norm.

$$S_i^{(t)} = \{x : \parallel x - \mu_i^{(t)} \parallel^2 \leq \parallel x - \mu_j^{(t)} \parallel^2 \forall j, 1 \leq j \leq k\} \tag{2.28}$$

- Step 2 - The update step

  Calculate the new centroid positions:

$$\mu^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2.29}$$

Chapter 3

Proposed solution

3.1 A HMM aproach for URL classification

We will try to solve the problem of malicious URLs detection using a HMM aproach. The way in which we will going to start is by constructing a n-gram Model over a hidden markov model.

An n-gram model is a probabilistic model used in language processing for predicting the next item in a sequence describeable by a Markov Model of order (n - 1).

3.1.1 Building the model

The first step in constructing the model is to identify how would the system look like in what we are interested, the identification of clean and malware URLs.

As we showed in the theoretical background a HMM can be described as $\lambda = (A, B, \pi)$, where :

- A - N x N matrix, the state transition matrix , where N - numbers of hidden states
- B - N x M matrix, the observation probability matrix, M - is the number of observations
- π - 1 x N row vector, initial state distribution

The first step we have to make is to identify in our case what these parameters will be. First we have to identify the numbers N and M, which need to be fixed for our system to our. For the beginning we will start by defining the number of hidden states as N = 2.

There are a number of reasons why we want to start from here. The number of hidden states , equal to 2 , will let us to clear categorize an URL in a category or another, in our case these states will be called Clean and Malware.

It is clear why we made this choice, both hidden states refer to complex processes, with very many parameters to consider, which we want to model in a simpler way. So for now we will keep N = 2.

Before establishing what M is , we have to respond to the following questiong : In the case of URLs, what will be the observations ? In this paper we will test two kinds of observations from the many that exist on which systems will be built.

Because our input consists of raw URLs, we will first normalize them, then get from this normalized form the n-grams (substrings of length n) - the observations which we will feed to the system. Because of this, the size of M will be equal to the size of the n-gram alphabet. Here we can observe that M is actualy a function :

$$M : N \times N \to N, M(n, m) = C_n^{m-1} + D(n, m),$$

where:

- n - the size of the n-gram
- m - the size of the alphabet
- D - function, $D : N \times N \to N$ that can be written as a liniar combination using the function M, as we will see later

Normalization

The research was conducted on a database of clean and malware URLs, provided by the Bitdefender Cyber Threat Intelligence Lab, which have been further used for the training process. It is obvious that the accuracy of the system will greatly depend both on the quality of the training data, its quantity and its representation.

To build the input for the system, we will proceed as follows. For example, we have the following raw URL : “http://www.aBC.co.uk/a”. First we will split this string using the special character ’/’ (we will call these substrings ”splits”). Each split will represent a part of the url (for example the first split will be the protocol, then the tld, hostname and others).

Because we want to make a general system capable of identifying benign and malware URL using only one lexical feature of the input : the URL itself, we will have to do the following. We will take each "split" and identify all the punctuation marks and divide each split in even smaller substrings by those marks.

After applying this procedure to all the splits we will have a list of substrings from which we are going to exclude some special words which will not count in the final verdict (words like "www" or protocol words - http , htts, ftp). Now we can create the observations which will train the system.

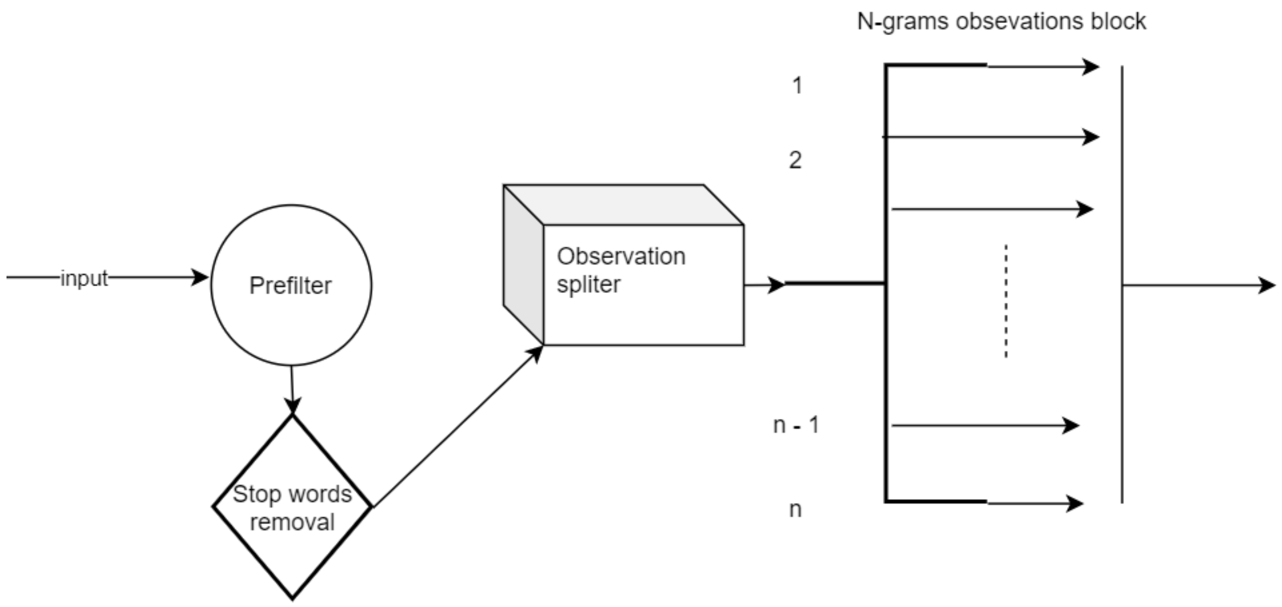

Figure 3.1: Normalized input observation splitter

In this paper we will consider 2 types of observations for the Markov models:

1. The alphabet [a-z0-9\∅]*

2. The alphabet [a-zA-Z0-9\∅]*

We denoted with \∅ - NULL character - the absence of an letter. It will be used when a split has the length smaller the size of the n-gram, so we will append it at the end until we have the size n, the minum to make a word in the n-gram alphabet.

Now we can see what the meaning of the D function is given by the insertion of the special symbol. Because in a word in which is inserted the \∅ symbol ,the length of the word will be between 0 and n - 1. Given this we can see that the function D can be written as:

$$D(n,m) = \binom{m}{n-1} + \binom{m}{n-2} + \cdots + \binom{m}{0} = \sum_{i=0}^{n-1} \binom{m}{i},$$

because 1 or more special characters will be concatenated with all the i-grams, i = $\overline{0, n-1}$

Because of this our final form for M will be:

$$M = \sum_{i=0}^{n} \binom{m}{i}$$

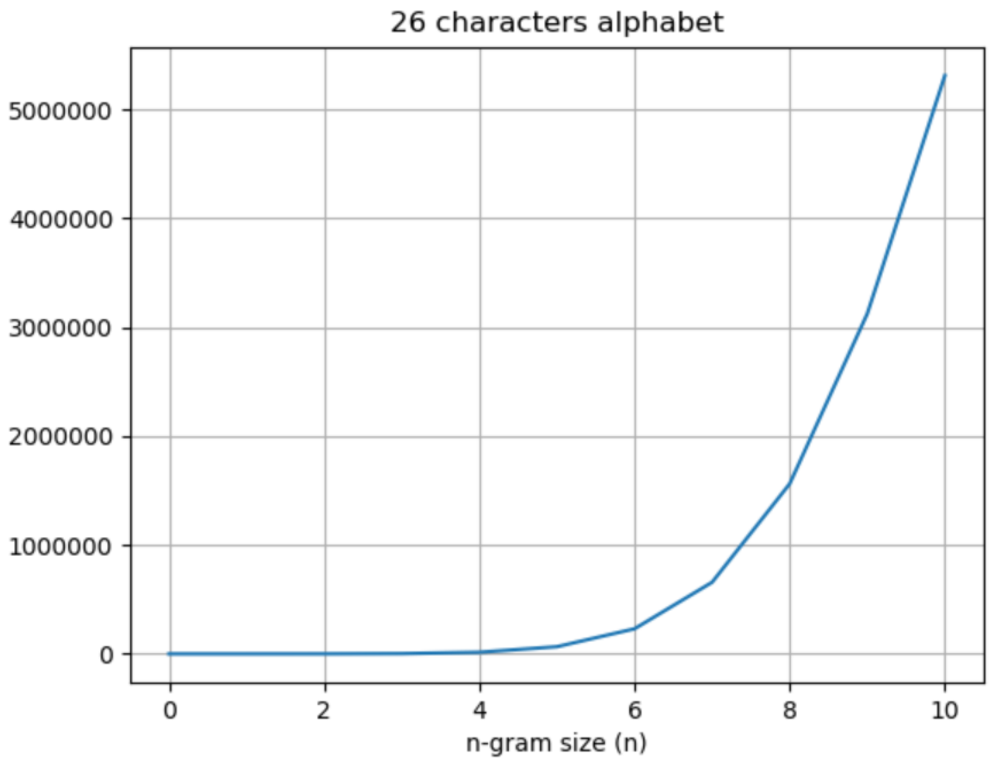

Figure 3.2: N-gram size graphic - alphabet size = 26

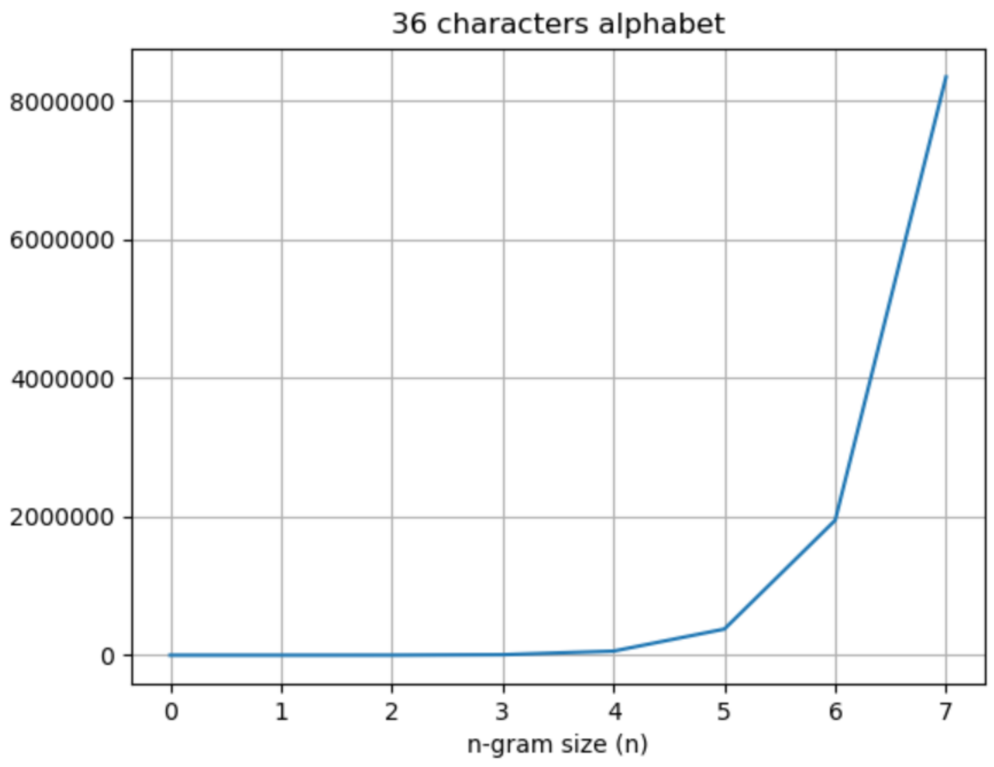

Figure 3.3: N-gram size graphic - alphabet size = 36

We have the url "http://www.aBC.co.uk/a". After splitting we obtain the folowing list : [http , www , aBC , co ,uk, a]. After excluding some special words we will have now [aBC , co ,uk , a] From this subset we will create series of observations according to the 2 alphabet and depending of the size of the n-gram.

For this particular case using the first alphabet we will have following obsevations:

- for 2-grame: [ab , bc , co, uk, a$\backslash\emptyset$]
- for 3-grame: [abc, co$\backslash\emptyset$, uk$\backslash\emptyset$, a$\backslash\emptyset\backslash\emptyset$][1]

And for the second alphabet:

- for 2-grame: [aB , BC , co, uk, a$\backslash\emptyset$]
- for 3-grame: [aBC, co$\backslash\emptyset$, uk$\backslash\emptyset$, a$\backslash\emptyset\backslash\emptyset$]

The basic flow of the whole system can be summed up by the following diagram

By now we have described all the parts up to the last 2:

[1]Observation: all the combinations with $\backslash\emptyset$ dealing with a n-gram will be of the form [a-z0-9]{x}($\backslash\emptyset$){y}, where x ¿ 0 and x + y = n

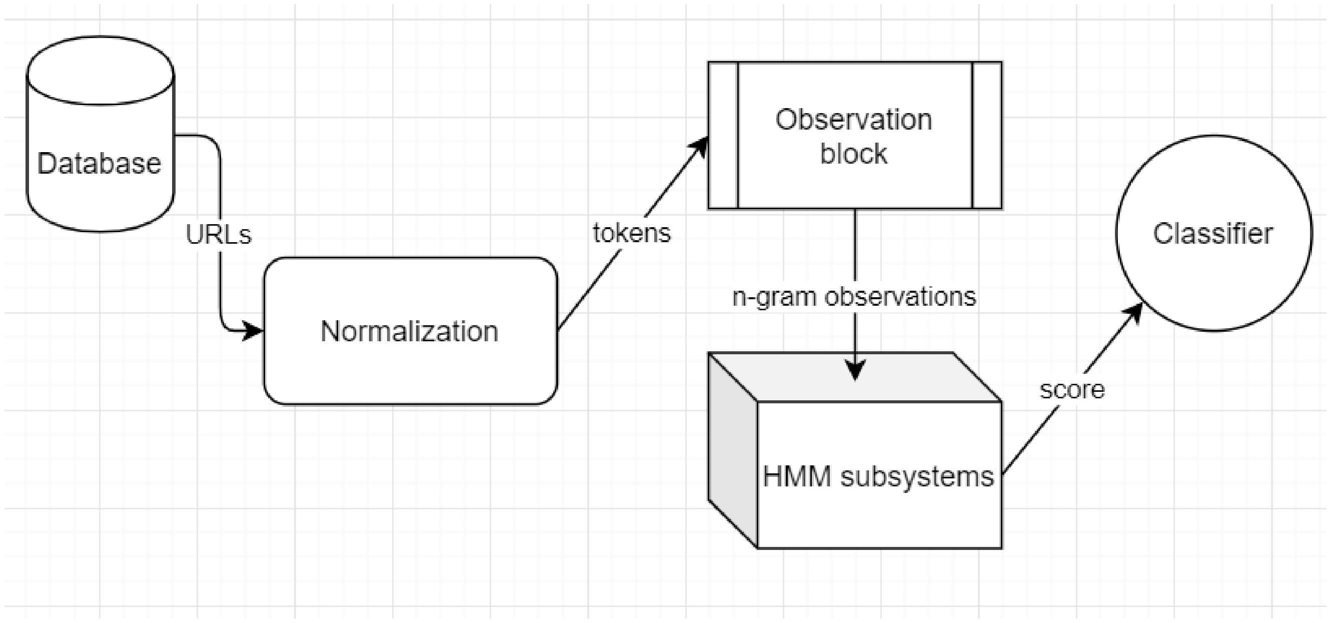

Figure 3.4: System diagram

- The HMM subsystems block - composed of 2 subsystems in which the parameters are trained differently. The output of this block will be given using the Viterbi algorithm from which we will build a Clean-Malware score with which to feed:

- The Classifier - using the output of HMM subsystems block, we will use the K-means algorithm , for k = 2. In this way we obtain the scores (positions) of the 2 centroids for Clean and Malware. In this way we will classify and URL depending to its absolut distance from one of those 2 points.

3.1.2 The first subsystem

The first subsytem will be a custom HMM. This will be used as a reference, a benchmark for testing against the second system.

As we established above, we will have the following system: $\lambda_c = (A_c, B_c, \pi_c)$ where:

1. A_c - the state transition matrix for the custom system. Because we have established that N = 2, we will make an asumption : that there is an equal probability to go to either of the states (neither one or the other is a preferential state). Now we have a constant matrix where every entry is $a_{ij} = 0.5$, i , j = $\overline{1, N}$

2. π_c - the initial probability vector. Because we made the above assumption , we will also keep this vector constant,where every entry $a_i = 0.5$, i = $\overline{1, N}$

3. B_c - the transition matrix of size N x M

As we saw above, M is a function depending on 2 parameters, the size of the n-gram and the size of the alphabet. The HMM is a mathematical model , so we have to find a way convert our observations (which are n-grams) into numbers.

We know that our obsevations are combinations of an alphabet, so when can sort them lexicographically. Because of this we will use a techniques know as feature hashing (or the hashing trick). It is a very simple way of turning features into indices in a vector or matrix, using a hash function. Basically given the observation string, we convert it to a linear index using a one-to-one correspondence :

$$(a1, a2, ..., an) \rightarrow a1 * k^{n-1} + a2 * k^{n-2} + \cdots + an$$

where (a1, a2, ..., an) are refering to the indexes in the symbol tabel of the alphabet used in the current subsystem.

Now that defined the parameters of the system, lets see how we will train this model.

3.1.3 Training

As we said above, the first 2 parameters of the system will be kept constant because we would like to see how the system is evolving by without making any difference between the two states , and the third parameter, the transition matrix will be the only one updating.

This parameter will retain the appeareance frequency for each observation divided by the number of all the observation. The update will be made only on the row which corresponds to the initial verdict of the URL from the training set. Here we should mention also that = k fold cross validation.

The begining of the training procedure can be shown by the following diagram:

The tokens input obtained after the normalization process are feed to the observation block which creates input for a specific HMM subsystem, meant to be trained by a specific n-gram size.

Then, each i-gram traing HMM(i) (i = $\overline{1, n}$ reprents both the i-th HMM and also the size of the i-gram with whom it is trained) takes place the folowing process:

- Step 0:

  A database is created for each HMMs transition matrix. The size of the matrix will be 2 · M (2 rows, for Clean and Malware, which must be initialized at first with

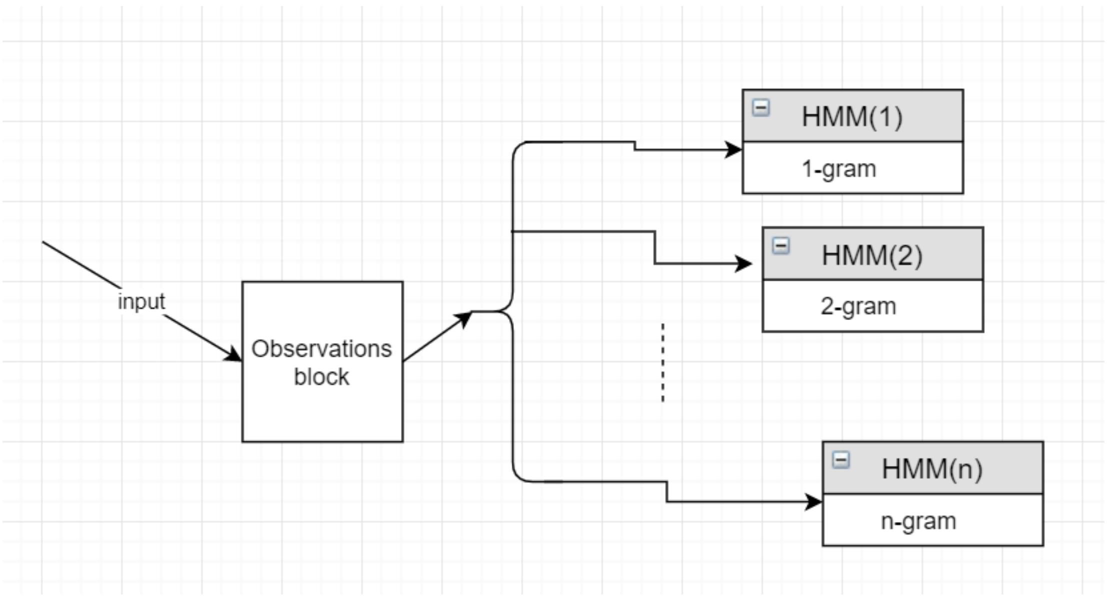

Figure 3.5: HMM subsystems input

any infinitesimal value, so the e row sum shoul always be equal to 1. We will choose to set each field from each row at first with $\frac{1}{M}$. It is necesary that that any field is not equal to 0 because, there will be cases when we will find unknown observations which will need to be modeled by our system.

- Step 1:

  Having the observations and their probabilities we can initialize the forward-backward algorithm which will give up the score P(O $|\lambda$) using forward-backward algorithm variables α, β for the URL.

- Step 2:

  The score obtained will be used by the Viterbi algorithm to give us the sequence of hidden states that best describes URL probability score.

- Step 3 :

  Using the sequence given in Step 2 we will create another secondary score based on the ocurrence frequency of the hidden states. Using this other partial score we will use the k-means algorithm to create centroid points for clean and malware, giving a verdict acording to the absolut distance from one of those 2 points.

- Step 4 :

  Now we have a verdict for our input. Based on this verdict, we have the possibility to update the transition matrix, in the row coresponding to the current verdict.

The flow of this procedure can be summed up to the following diagram:

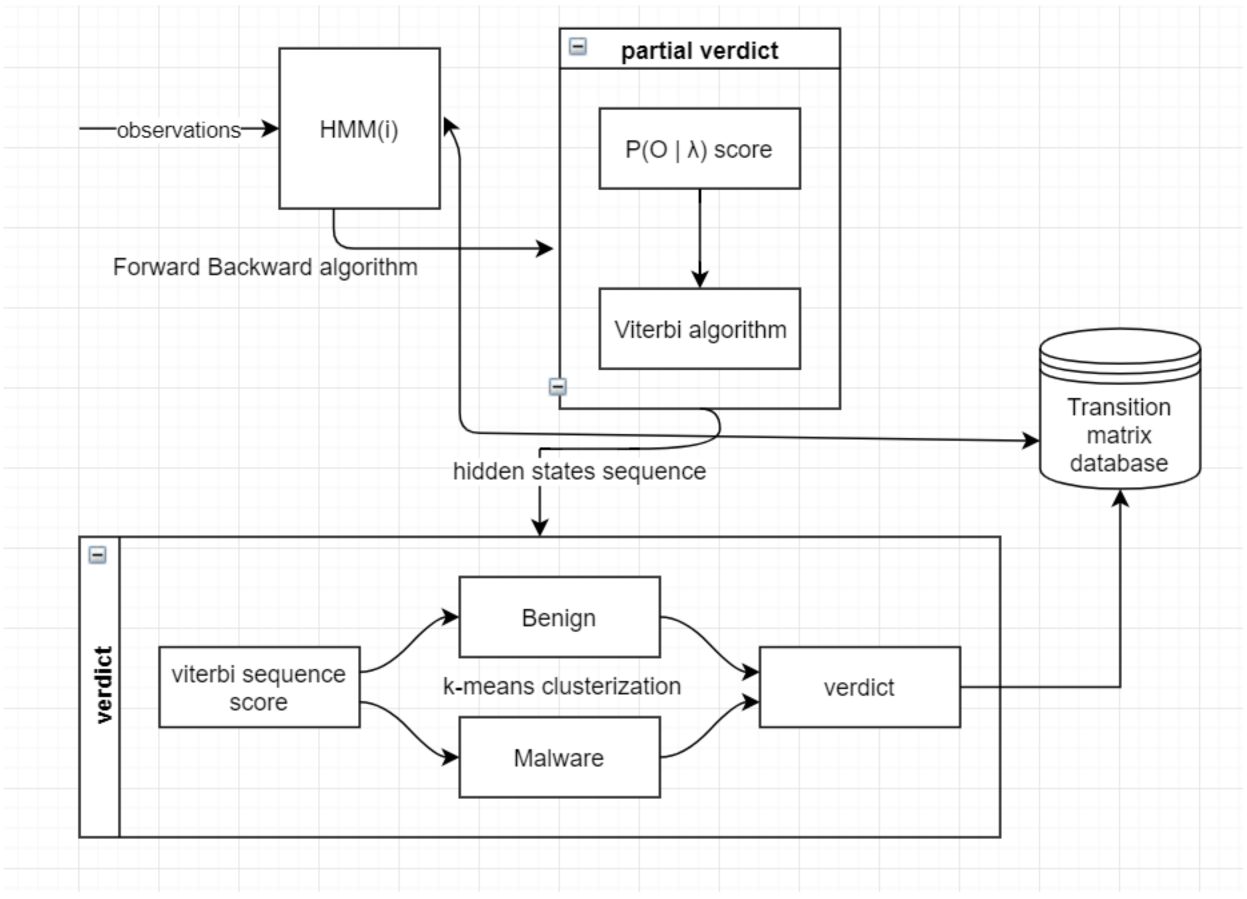

Figure 3.6: HMM subsytem verdict process

3.1.4 The second subsystem

The next part of the system will also be a HMM, this time a regular one, where all the parameters will be updated , using the Baum-Welch algorthm, described in the theoretical background section. We will denote this system as $\lambda_r = (A_r, B_r, \pi_r)$.

The flow of this system is similar to the one present in subsystem 1. The n-gram observations extracted from the raw URLs from the database will go through HMM(n) subsystem trained with this type of observations.

In the beginning all the parameters are initialized as we have seen in the first subsystem. First the A, the state transition matrix, will be initialized with 0.5 for each of the table , because we have N = 2 hidden states, and because of this the elements of the initial probability matrix will be also set to 0.5, and like before the transition matrix will be initialized in all fields with the value $\frac{1}{M}$.

Every input given to the system will go through the next series of steps:

- Step 0 :

  The initialization of the database, and the HMMs subsystems (if it wasn't done before)

- Step 1 :

  Calculate the α, β variables acording to the curent parameters of the HMM and create the variable ξ and using the ecuations presented in the theoretical background.

- Step 2 :

  Set the curent model $\lambda(A, B, \pi) = \overline{\lambda}(\overline{A}, \overline{B}, \overline{\pi})$, where $\overline{\lambda}$ is the new model which maximizes the probability $P(O|\lambda)$ (so now we have $P(O|\lambda) \leq (P(O|\overline{\lambda})$, with parameters reestimated using the Baum-Welch algorithm, that will be further used in dealing with new observations given to the subsystem.

- Step 3

  Using the viterbi algorithm we will obtain a partial score that will be used by the k-means algorithm, and similary to subsystem we will give a verdict for the input.

3.2 Results

Now that we have presented the whole system, we will present the results obtained by each of the 2 subsystems created. In order to clusterize an URL (set a verdict), we built a collection of 300000 URL samples. From this samples, by applying different techniques with the aim of creating a good dataset, by removing duplicates , removing outliers that could produce noise, damaged URL and other.

Afterwards, with a dataset of 120000 URLs, we built the mathematical model. Using a 6-fold cross-validation, we will have a training set of size 100000 composed of both benign and malicious URLs, in approximately the same size proportion, and similary a testing set with the rest of the samples.

After the model was generated we tested the first subsystem , for the 3 kinds of observations (1,2,3 - grams) which provided the following results: The results of the first subsystem are contained in the following tables:

Type [2]	Se[3]	Sp[4]	Tn[5]	Tp[6]	Acc[7]	Fn	Fp
Unigrams	70.15%	69.78%	7015	6978	70.13%	29.15%	30.22%
Bigrams	69.8%	70.70%	6980	7070	70.25%	30.2%	29.3%
Trigrams	71.78%	71.80%	7178	7180	71.79%	28.22%	28.2%

Table 3.1: N-gram observations results - first alphabet

Type	Se	Sp	Tn	Tp	Acc	Fn	Fp
Unigrams	70%	70.87%	7000	7087	70.435%	30%	29.13%
Bigrams	71.9%	70.9%	7190	7090	71.4%	28.1%	29.1%
Trigrams	72.12%	71.21%	7212	7121	71.665%	27.88%	28.79%

Table 3.2: N-gram observations results - second alphabet

The detection procentage is not as good as we wanted to be, because the fp rate is quite high. This happens due to the fact that the system is a static one - the initial probability vector and the π and the state transition matrix A are being kept constant the on the whole duration of the training.

This means that the last parameter has to deal with all the training process. Because of this we will want to see how to system is working when all the parameters are updating. The results for the second subsytem trained with the first alphabet are contained in the following tables:

Type	Se	Sp	Tn	Tp	Acc	Fn	Fp
Unigrams	99.82%	97.72%	9772	9982	98.72%	2.28%	% 0.18
Bigrams	100%	99.4%	10000	9940	99.7%	0 %	0.3%
Trigrams	88.92%	86.91%	8892	8691	89.35%	11.08%	13.09%

Table 3.3: N-gram observations results - first alphabet

Type	Se	Sp	Tn	Tp	Acc	Fn	Fp
Unigrams	99.05%	90.90%	9905	9090	94.975%	0.95%	9.1%
Bigrams	100%	91.14%	10000	9114	95.57%	0%	8.86%
Trigrams	97.22%	93.12%	9722	9312	95.179%	2.78%	6.88%

Table 3.4: N-gram observations results - second alphabet

We can see that the fp rate has dropped drastically in the new subsystem.This subsystem is performing better than the previous one because the Baum-Welch algorithm updates at each new iteration the 3 parameters of the subsystem (A, B, π). It is performing better at estimating the verdict for a given input because it behaves in a dynamic manner in comparison with the previous.

Conclusions

Given that the number of cyber attacks has increased steadily over the last decade, the need for proactive detection methods has emerged.

Old detection methods such as blacklist become helpless in the case of unknown menaces such as zero-day threats. Considering the these aspects, new detections started to rely on probabilistic heuristics and machine learning techniques, that are capable of surpassing the reactiv state of many detections.

One such method is described in the current paper. Our solution based on the mathematical model of the Hidden Markov model is capable to differentiate between benign and malware URLs using only lexical features.

Taking into account the results, we can say that our detection is a solution for the problem of malware URL detection with many ways of increasing the current accuracy in malware detection.

Future research

As a future work our mathematical model can be optimized by increasing the number of HMMs, thus increasing the size of the n-gram and of the observations. This would lead to some problem in implementation because, firstly, the number of observations grows exponentialy.

If memory and time aren't a problem, our system is good enough. But if we want for our system to work fast, and to manage the massive data that the system would hold we will have to make some changes.When the system be initialized, another data structure must be used instead of the usual matrix (for example a sparse matrix) because it will arise memory problems because of the big dataset.

An interesting idea would be to divide the verdict space into smaller and smaller chunks. This would transform our system from a discrete model to a continuous model.

The first step to this idea would be to increase the number of hidden states from 2 to 10, where 0 will be the Clean state and 1 will be the Malware state, and all the states in between (0.1, 0.2 . . . 0.9) will represent different degrees of clean-malware, leading to a increase in the number of centroids given by the k-means algorithm.

After this the increase in size we could go on because the k-means algorithm will be able to clusterize efficiently and give even meaningful verdict for a given URL input .

Bibliography

1. L.R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition", Proc. IEEE, vol. 77, no. 2, pp. 257-286, 1989.

2. Thomas H. Austin Mark Stamp, *Hidden Markov models for malware classification*, 2017

3. May 2014, Journal of Computer Virology and Hacking Techniques 11(2)

4. Mark Stamp. *A Revealing Introduction to Hidden Markov Models*

5. Saradha Ravi , N. Balakrishnan and Bharath Venkatesh . *Behavior-based Malware Analysis using Profile Hidden Markov Models.* In Proceedings of the 10th International Conference on Security and Cryptography - Volume 1: SECRYPT, (ICETE 2013) ISBN 978-989-8565-73-0, pages 195-206. DOI: 10.5220/0004528201950206

6. Lawrence R. Rabiner,*An introduction to hidden Markov models*;IEEE ASSP Magazine , Volume: 3 , Issue: 1 , Jan 1986 , Pages: 4 - 16

7. M. Khonji, Y. Iraqi, and A. Jones. "Phishing detection: a literature survey in IEEE Communications Surveys Tutorials, vol. 15, no. 4, pp. 20912121, 2013.

8. J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. *Beyond blacklists: learning to detect malicious web sites from suspicious urls* in Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD09, pages 12451254, New York, NY, USA, 2009. ACM

9. S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair. *A comparison of machine learning techniques for phishing detection* in Proceedings of the antiphishing working groups 2nd annual eCrime researchers summit. ACM, 2007, pp. 6069

10. N. Spirin and J. Han. *Survey on web spam detection: principles and algorithms*, ACM SIGKDD Explorations Newsletter, vol. 13, no. 2, pp. 5064, 2012.

11. M.-Y. Kan and H. O. N. Thi. *Fast webpage classification using url features* in Proceedings of the 14th ACM international conference on Information and knowledge management, CIKM 05, pages 325326, New York, NY, USA, 2005. ACM.

12. Baum, L. E.; Petrie, T. (1966). *Statistical Inference for Probabilistic Functions of Finite State Markov Chains.* The Annals of Mathematical Statistics. 37 (6): 1554–1563.

13. Rabiner, Lawrence. *First Hand: The Hidden Markov Model.* IEEE Global History Network.

14. Broder, Andrei Z.; Glassman, Steven C.; Manasse, Mark S.; Zweig, Geoffrey (1997). *Syntactic clustering of the web.* Computer Networks and ISDN Systems. 29 (8): 1157–1166.

15. Hans-Peter Kriegel , Erich Schubert1 , Arthur Zimek. *The (black) art of runtime evaluation: Are we comparing algorithms or implementations?*

16. MacKay, David (2003). *Chapter 20. An Example Inference Task: Clustering.* Information Theory, Inference and Learning Algorithms. Cambridge University Press. pp. 284–292.