FACULTY OF SCIENCES

Network intrusion detection by means of sequential Bayesian methods.

Stijn Huysman
Student number: 02013102

Supervisors: Prof. dr. Koen De Turck, Prof. dr. Dieter Fiems

Master's dissertation submitted in order to obtain the academic degree
of Master of Science in Statistical Data Analysis

Academic year 2022-2023

GHENT UNIVERSITY

FACULTY OF SCIENCES

Network intrusion detection by means of sequential Bayesian methods.

Stijn Huysman
Student number: 02013102

Supervisors: Prof. dr. Koen De Turck, Prof. dr. Dieter Fiems

Master's dissertation submitted in order to obtain the academic degree
of Master of Science in Statistical Data Analysis

Academic year 2022-2023

GHENT UNIVERSITY

FACULTY
OF SCIENCES

Stijn Huysman

Prof. Dr. Koen De Turcq
Prof. Dr. Dieter Fiems

June 2023

GHENT
UNIVERSITY

FOREWORD

During my master's dissertation, I unexpectedly delved into the world of cybersecurity, comparing myself to the blue dot on a GPS system. Instead of guiding physical roads, I studied the powerful Sequential Monte Carlo (SMC) algorithm or particle filter to track and detect potential threats in computer networks. It was a fascinating and valuable journey, uncovering the importance of these techniques in safeguarding our digital domains.

Foremost, I would like to express my deepest gratitude to my wife, Marijke. The past three years have presented immense challenges, and achieving success would have been unattainable without your unwavering love and support. Your constant encouragement throughout my statistical journey has been invaluable. Within our 'state space model', you are my indispensable counterpart. Thank you Anna, Felix and Vic, our children. I owe you my full-time dad-dedication again. Although, despite the circumstances imposed by the Covid quarantine, it always brought a smile to my face you unexpectedly appeared in the video background during our online classes. Also, I am so grateful to my parents for their belief in my engineering studies and enthusiasm for this program.

I appreciate my supervisors, Professor De Turcq and Professor Fiems, for their invaluable assistance. Your personalities fueled my motivation to delve into a subject that was initially beyond my scope. I dived into the realm of advanced and intricate mathematics. As real sparring partners, both of you clarified and put my concerns into perspective. Throughout this dissertation, the framework by Prof. Chopin offered a comprehensive approach applying sequential Monte Carlo, including a pretty complete Python package ('particles').

Finally, I would like to acknowledge the numerous individuals who have contributed to my learning process in data analysis through various means. I am grateful to the lecturers who went the extra mile to record their lessons and fellow students with whom we engaged in extensive meetings and discussions to comprehend the material fully. Lastly, I express gratitude to my employer, IW, and manager, Dave, for your belief and support throughout this challenging endeavor.

Let us start the Sequential algorithm!

TABLE OF CONTENTS

ABSTRACT

Network intrusion detection is crucial to guarantee the security of computer networks. This study proposes an advanced approach to network intrusion detection using state space models (SSMs) and the Feynman-Kac formalism. The objective is to improve anomaly detection accuracy by capturing the network behaviour's temporal dependencies and underlying dynamics.

The first approach models the data from the KDD99 dataset (Tavallaee et al., 2009) by a logistic regression model. When new observations occur, the regression parameters are considered the hidden state parameters from the model.

Secondly, state space models are designed to represent the hidden states underlying observed network data. By incorporating the temporal dependencies among network features, one aims to capture its sequential nature. Furthermore, parameters such as correlation and covariance matrices are utilized to tune the SSMs to fit the specific characteristics of the analysed class. Finally, we utilize the Feynman-Kac formalism to incorporate the SSMs into a detection framework (Chopin & Papaspiliopoulos, 2020), which combines the SSM (state space models) instances and the corresponding network data in a bootstrap strategy.

Using Sequential Monte Carlo sampling techniques, the posterior distribution of the hidden states is estimated. The resampling step, known as the "bootstrap," ensures diversity in the particle set and prevents particle degeneracy.

Univariate statistical tests, including paired t-tests and Wilcoxon rank-sum tests, are conducted to compare the states of selected features between the non-harmful and anomaly classes. Significant differences in mean values between the two groups are observed, indicating distinct normal and anomaly network class characteristics.

The performance of the multivariate distribution-based classification model is evaluated using a confusion matrix. Further tuning of the model optimizes for minimum precision and enhanced accuracy. Finally, hypothesis testing is conducted to statistically evaluate the model's performance against random guessing statistically, demonstrating that the state space model significantly outperforms.

Overall, the proposed approach based on state space models and the Feynman-Kac formalism offers a robust and flexible solution for network intrusion detection. Furthermore, improved accuracy in identifying anomalies is achieved by incorporating the temporal dependencies. Although, other machine learning algorithms, like Random Forest, perform better. Therefore, the proposed Sequential Bayesian Monte Carlo algorithm offers a strategy to detect anomalies quickly with a low computational cost.

This research makes a valuable contribution to the progress of network security and provides a foundation for further refinement and optimization in specific network environments.

Keywords: *Network intrusion detection, logistic regression, state space models, Feynman-Kac formalism, anomaly detection, temporal dependencies.*

1 INTRODUCTION

1.1 Introduction

Networks in the early sixties were related to mainframes and telephony services. However, the Advance Research Project Agency (ARPA) funded a network design for the United States Department of Defence. This network (ARPANET) became the blueprint of our modern internet. (*The History of Computer Networks Information Technology Essay*, n.d.). This technology was one of the first networks built on the TCP/IP protocol that, besides computer networking, also exchanges information packet-wise with lots of devices. This extension led to the Internet of Things (IoT) connecting every person and many devices.

However, the exponential growth of this network traffics constitutes a vulnerability to a high occurrence of network attacks, which vary a lot in their setup and behaviour. Ransom software, Denial of Service attacks, computer viruses, and others (further denoted as network intrusion or NI) are considered. Firstly, they are a network risk, but they can also harm every business and public service that use those affected networks. (*What Is a Network Attack?* 2021).

The challenge is identifying all potential network intrusions in time with a correct algorithm. Literature shows a wide palette of Artificial intelligence techniques to detect such anomalies. Support Vector Machines (SVM) and other classification algorithms (Random Forest, Decision trees, Gradient boosting) are among the used techniques. Besides theoretical modelling, many security companies offer such services (Khan et al., 2021).

1.2 Problem statement and research question

For this thesis, we wish to take a sequential (="online") approach, meaning that as new network data becomes available in a sequential order, it is instantly used to update the detection procedure. Here, we assume that a network intrusion will result in an observable change in these network data. Therefore, the problem becomes a special case of change detection.

When changes are detected, three benchmarks will be updated and statistically evaluated. First, the false alarm rate will be monitored (false positives), as well as the misdetection rate (false negatives). As a final challenge, calculating the delay of detection will be covered. Because of the enormous impact that network intrusion could have, it is evidently fundamental that all the above benchmarks are set very stringently. This setting demands an appropriate algorithm.

1.3 Dissertation flowchart

In Chapter 2, the methodology used and its background will be clarified, and the exploration of the dataset used during this dissertation will be discussed. Then, chapter 3 and Chapter 4 will highlight the results and discussion, respectively. Finally, the last chapter will present some practical implementations of this thesis.

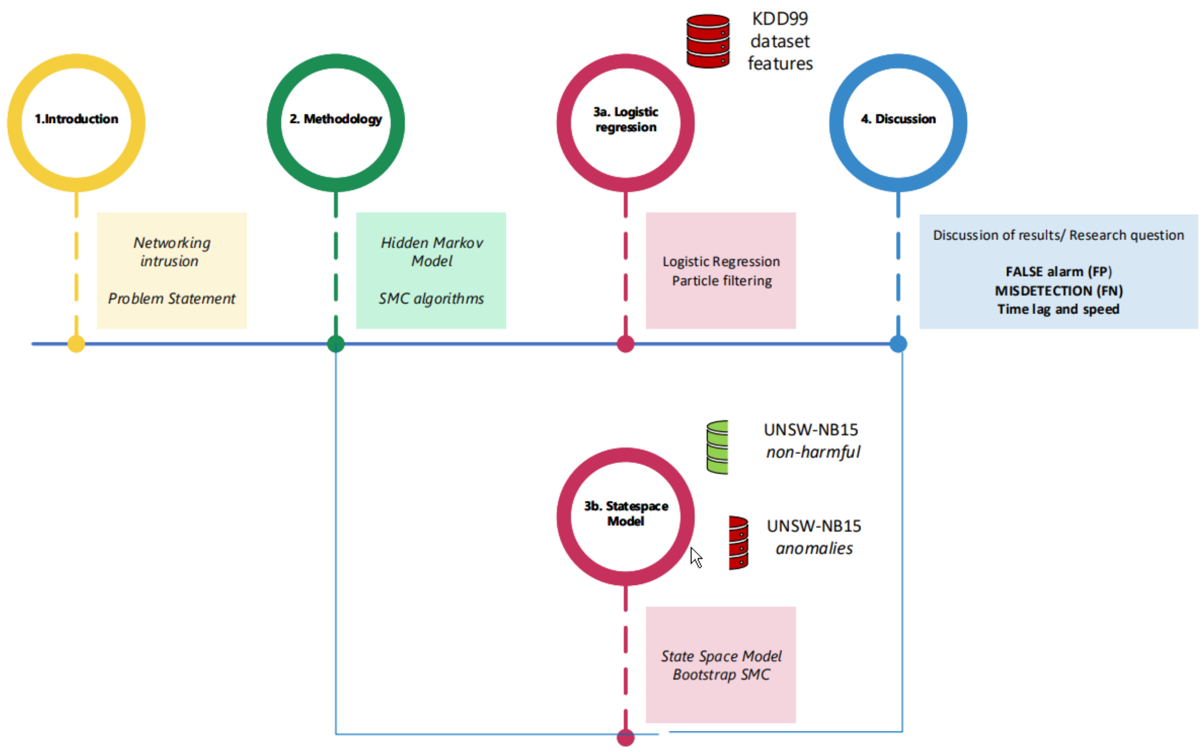


Figure 1: Dissertation flowchart

2 METHODOLOGY

2.1 Sequential Monte Carlo

We will develop this algorithm based on sequential Monte Carlo techniques. Specifically, this concept (particle filtering) is explained further and analysed for its potential use in streaming network data. Particle filter is seen as an ensemble-learning technique. Here, a specified N number of particles are generated from a prior distribution and resampled along, according to their weights. A particle filter algorithm is widely used in robotics, computer vision, and signal processing, where the system's state needs to be estimated from noisy measurements (Del Moral et al., 2006).

We start with theoretical guidance through the concepts of Hidden Markov Models and sequential Bayesian learning. More specifically, the concept of particle filtering will be explained. Next, the aim is to model the metadata of the network (TCP/IP) packets and to distinguish the “good” packets from the “malicious” ones. Ultimately, the SMC algorithm will result in a classification model that is continually updated as new data becomes available. The classification can be achieved by searching for underlying parameters of a classifier, such as logistic regression. Another option is to partition the available data into both categories and model the PDF function of each class.

Secondly, we explore the KDD99 dataset (Tavallaee et al., 2009) that is widely used for research in intrusion detection and machine learning-based classification of network traffic. Here, some sequential Bayesian filter algorithms are investigated on their proper use. Possible shortcomings in the dataset could complicate a sequential filter process.

Additionally, this thesis will cover potential Concept-drift will be discussed separately because of its usefulness during online tracking. Sequential Monte-Carlo algorithm

2.1.1 Hidden Markov Model

A Hidden Markov Model (HMM) is a double stochastic process describing a system's evolution over time when the system is only partially observable. It can be seen as a Markov Chain and a general stochastic process.
This Markov Chain uses the transition probability to describe the movement from the current state to the next state. It does not depend on the other states. Additionally, an HMM requires

time homogeneity, meaning that the probability of transition between the states does not depend on the transition's time.

The general stochastic process describes the relationship between the hidden state and its observations. In other words, the HMM will generate observations ‘related’ to the hidden system. Here, the measurements are contaminated with noise. Estimating the HMM, based on its observations, is called a filtering process (Z. Chen, 2003). Both Kalman and particle filters are powerful Bayesian filter algorithms to estimate the posterior distribution of the hidden state. Both are influenced by noise, each with their strengths and limitations.

While a Kalman filter presumes Gaussian noise and linear state transitions, a particle filter does not assume this. A Kalman filter achieves higher accuracy $wrt\ the\ MSE$ $(\mathrm{E}_\theta[(\hat{\theta}-\theta)^2])$, compared to a particle filter, the latter is more robust when the linearity and Gaussian assumption must be relaxed. This covers most of the real-life examples. In the context of this dissertation, the HMM is a statistical model (regression or classification) where the system being modelled (intrusion or not) assumes to follow a Markov process. This system's states or classification parameters (X) are unobservable and hidden in this scenario. Otherwise, the outcomes (Y) of these unobservable states are influenced by them in a known way. When the X states are not observed directly, one can infer X by its observations of Y.

The Markov condition is a fundamental concept in probability theory and stochastic processes. It states that the future state of a system depends only on its current state and is independent of its past states. This assumption is particularly important when modelling a system. The number of parameters and the computational effort of the statistical model will drastically decrease. The outcome does not depend on any earlier states or observations. Figure 2 illustrates such a multi-layered HMM with the hidden layer (X) and the observation layer (Y)

We can depict a simple Markov process as follows:

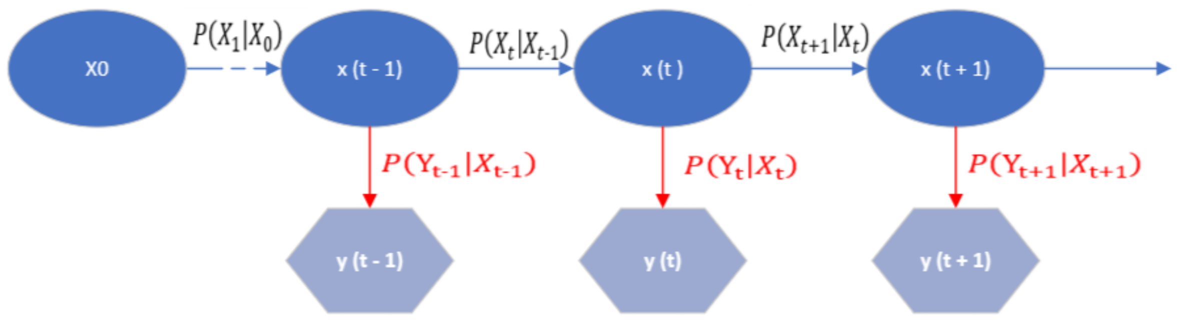


Figure 2: Schematic of a Hidden Markov Model. (Schematics based on: (Bonaccorso, 2020)

$(x_1, x_2, \dots, x_n)$ represents the hidden layers' multidimensional state set (vector) at a given moment or time. In our network intrusion approach, these can be seen as the model parameters of a classification distribution every time an observation (data packet) enters. Therefore, we call it a set of regression parameters (beta) to classify the outcome. This methodology allows for a classification of a hidden state with continuous updates.

$\mathrm{Y} = (y_1, y_2, \dots, y_n)$ represents the multidimensional vector with observations
In our network intrusion approach, this is the set of relevant features from the streaming data.

Due to the nonlinear transition function of network intrusion classifiers and undefined (non-Gaussian) noise, this dissertation will focus on the Particle filter (SMC) and its resampling techniques. First, however, the key ingredients to construct the SMC algorithm will be discussed in the context of the Intrusion Detection System (IDS) and the KDD99 dataset.

2.1.2 Particle Filtering

To estimate the state of a system (HMM), a particle filter algorithm (PFA) can be proposed. Here, one can use it to estimate the posterior distribution of the hidden classification parameters. In the context of this dissertation, this points to the binary classifier parameters to detect potential intrusion. Given a set of metadata features of network traffic, the algorithm will estimate the probability that a data package belongs to one of the other classes at a given moment. Here, both classes are noted as 'normal' and 'malicious' in the case of an intrusion.

The basic idea behind the filter algorithm is that it initializes a set of N "particles". This set represents the initial distribution or prior distribution of the hidden states. As mentioned in

Figure 2, The observation Y_t is determined by its hidden state X_t , with random noise added. The particle filter estimates the posterior distribution at the hidden state time t numerically, given all the observations up to that moment $P(X_t|Y_{0:t})$. Intuitively, the hidden state X_t has no direct relationship with all the observations till then $(Y_{0:t})$. As a solution, Bayes-rule can be used sequentially

The basic PF algorithm follows these principles:

At time t_0, the initial distribution is the probability distribution at that time, given the observations up to that moment, namely $P(X_0|Y_0)$. If there are no observations available at t_0, this distribution is typically chosen based on prior knowledge about the system (P_0). The PF samples a given amount (N) of particles from this prior distribution.

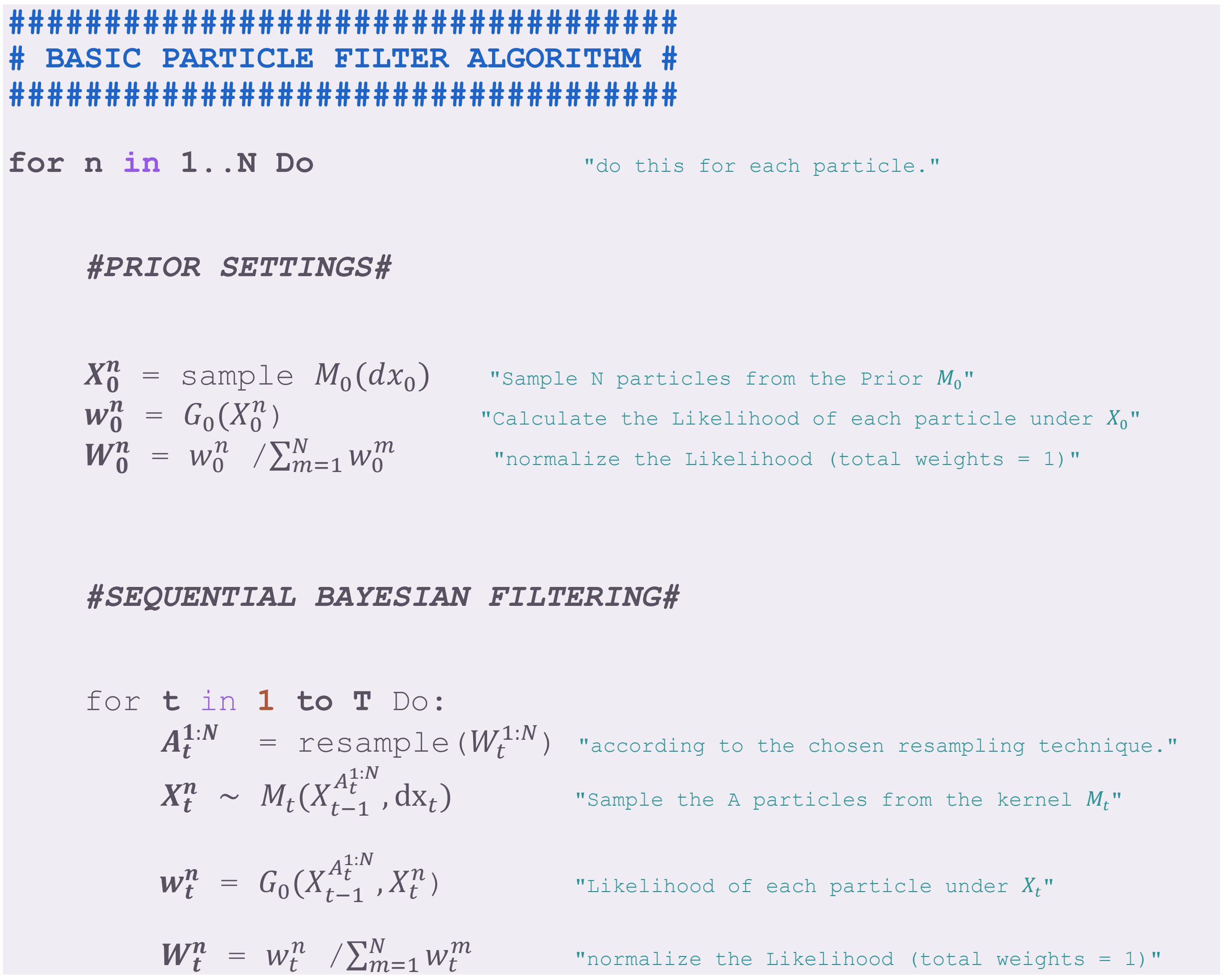
A decent choice of the prior will be further discussed. However, the selection of the prior distribution should be based on a comprehensive understanding of the underlying system, as it can influence the posterior distribution estimates by the particle filter. At each step beyond, the base algorithm proceeds as the following recurring loop:

First, the N particles (X_t^i) are resampled at each time step, using a sampler to obtain the weights w_t^i of observation given a certain particle ($w_t^i = P(Y_t|X_t^i)$. Different samplers exist; a decent choice will affect the filter's accuracy.

Next, the particles are propagated forward in time to explore the posterior distribution of the hidden state $P(X_t|X_{t-1})$. Here, each particle X_{t-1}^i is propagated to X_t^i , forward in time according to its transitional probability, aka. the Markov kernel of the state.

Thirdly, the weight of each of the N particles is 'updated' based on its observation likelihood under the proposed distribution. The posterior distribution is estimated the as the weighted sum of particles $P(X_t|Y_t) \cong \sum_{i=1}^{N} w^i \, . X_t^i$.

The pseudocode in Figure 3 shows the different steps of the generic algorithm. Here, the particle filter algorithm will be modelled and aligned with the code of the Python library **'particles'** as developed by Nicolas Chopin (https://github.com/nchopin/particles).
Along the dissertation, relevant code snippets will be highlighted. The appendix will contain the full scripts.



```
####################################
# BASIC PARTICLE FILTER ALGORITHM #
####################################

for n in 1..N Do                          "do this for each particle."

    #PRIOR SETTINGS#

    $X_0^n$ = sample $M_0(dx_0)$        "Sample N particles from the Prior $M_0$"
    $w_0^n$ = $G_0(X_0^n)$              "Calculate the Likelihood of each particle under $X_0$"
    $W_0^n$ = $w_0^n$ $/\sum_{m=1}^{N} w_0^m$     "normalize the Likelihood (total weights = 1)"


    #SEQUENTIAL BAYESIAN FILTERING#

    for t in 1 to T Do:
        $A_t^{1:N}$ = resample($W_t^{1:N}$)   "according to the chosen resampling technique."
        $X_t^n$ ~ $M_t(X_{t-1}^{A_t^{1:N}}, dx_t)$    "Sample the A particles from the kernel $M_t$"

        $w_t^n$ = $G_0(X_{t-1}^{A_t^{1:N}}, X_t^n)$    "Likelihood of each particle under $X_t$"

        $W_t^n$ = $w_t^n$ $/\sum_{m=1}^{N} w_t^m$     "normalize the Likelihood (total weights = 1)"
```

Figure 3: Generic Sequential Monte Carlo (Particle filter), inspired by Chopin & Papaspiliopoulos (2020)

In this proposed SMC method, a sequence of weighted samples is generated, with each sample representing a possible state of the system at a particular time point (Del Moral et al., 2006). The importance weights are calculated by comparing the likelihood of the current state to that of the previous state, and the particles are resampled according to their weights to generate a new set of particles for the next time point. Importance sampling and resampling are two crucial elements of the SMC algorithm. These will be further explained below, along with a graphical insight into the filtering process (Figure 6).

2.1.3 Importance sampling

Important sampling is the most widely used method to generate samples from a distribution in a particle filter (Figure 4). Here, the main idea is to sample particles from a distribution q(x) that is easy to sample from rather than the distribution under the scope. Afterwards, the generated samples are reweighted based on the **ratio** of the target distribution and the proposal distribution.

$$E[f(x)] = \int_{-\infty}^{-\infty} f(x).p(x).dx \propto \sum_{i=1}^{N} f(x_i).\frac{\boldsymbol{p(x_i)}}{\boldsymbol{q(x_i)}}.q(x_i)$$

$$\propto \sum_{i=1}^{N} f(x_i).\boldsymbol{w_i}.q(x_i)$$

Here, with $p(x)$ the target distribution that needs to be sampled from. Besides that, $q(x)$ distribution is proposed as the proposal distribution, easy to sample from. Very often, a Gaussian distribution with noise can serve the needs. The importance of the sample, also called its weight, as used in the particle filter, can be written as

$$\boldsymbol{w_i} = \frac{p(x_i)}{q(x_i)}$$

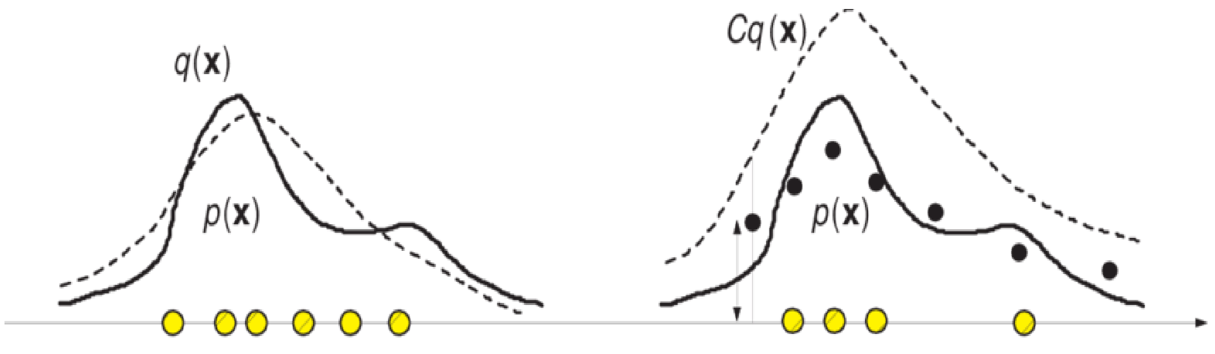


Figure 4: Importance sampling (left) and Acceptance-Rejection sampling, where p(x) is the true PDF and q(x) is the proposal distribution (Z. Chen, 2003).

Initially, a particle set, size N, is generated from the proposal distribution $q(x)$

As described above, the sampling and weighing steps are repeated for all the N particles. This results in N importance weights $w_1, w_2, \dots, w_N$. This set can be used to estimate the expectation of a function f(x) concerning the target distribution P($X_t|Y_t$).

This importance sampling method is useful in the generation process of samples taken from a proposal distribution rather than the target distribution. The latter is too complex to sample from.

Statistically

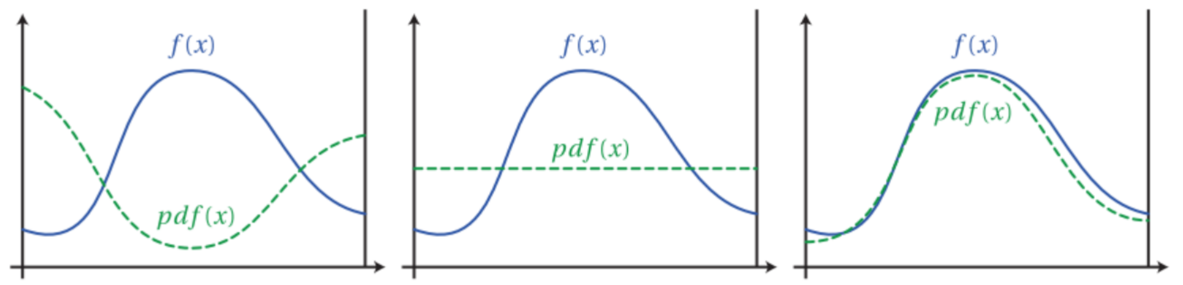
Increasing the number of particles used in the SMC algorithm will certainly increase the accuracy of the sequential predictions. However, this can be computationally expensive, particularly for high-dimensional target distributions. This is the main reason that Importance Sampling is introduced, to reduce the variance drastically. Below, the motivation for this clever reformulation trick is explained (T. Yu et al., 2019).

$\boldsymbol{E}[\boldsymbol{f}(\boldsymbol{X})] = \int_{-\infty}^{+\infty} \boldsymbol{f}(\boldsymbol{x}) p(x) dx \quad => \quad estimated\ as\ \bar{f}(x) = \sum_{i=1}^{N} f(x_i)$ and the variance is given by $\frac{1}{N} Var[f(x)]$.

Reformulating this same estimation towards another distribution in the IS framework, we get $\boldsymbol{E}[\boldsymbol{f}(\boldsymbol{X})] = \int_{-\infty}^{+\infty} \boldsymbol{f}(\boldsymbol{x}) \frac{\boldsymbol{p}(\boldsymbol{x})}{\boldsymbol{q}(\boldsymbol{x})} q(x) dx \quad => \quad estimated\ as\ \bar{f}(x) = \sum_{i=1}^{N} f(x_i) \frac{p(x)}{q(x)}$ and the variance is given by $\frac{1}{N} Var[\frac{p(x)}{q(x)} f(x)]$.

Mathematically, one can bring this IS-variance to zero when choosing the $q(x) = \frac{p(x).f(x)}{E[f(x)]}$. Of course, E[f(x)] is the unknown denominator, but since the density q(x) can be chosen freely, q(x) can be constructed whenever the product $p(x) * f(x)$ is high. Again, as visually explained in Figure 4, the variance is lowest when those distributions are close to each other.

Importance sampling is a fast and efficient method compared to other samplers. This is especially the case when the distribution estimated is highly concentrated. But the choice of the proposed sampling distribution ($q(x)$) could affect the robustness of the sampling method. An unwisely considered choice of the proposed solution will produce Biased results. A good example to illustrate this is estimating a function within a standard normal distribution. If one sample from a Uniform distribution is a proposal and weighs all the generated samples as the ratio between the target and proposed distribution, the results will be biased. This is because of the significant difference between both distributions, causing skewness in the estimates (Figure 5).



Comparison of three probability density functions. The PDF on the right provides variance reduction over the uniform PDF in the center. However, using the PDF on the left would significantly increase variance over simple uniform sampling.

Figure 5: impact of the probability density functions on variance reduction. (Wojciech, n.d.)

2.1.4 Resampling step

At every time step t, a weighted sample is taken from the IS distribution. This sample generates a new set of N particles with higher weights and an effective sample size.
Using this technique sequentially, one could redistribute the particles, focusing on the distribution under the scope. As a result, the most informative particles are (re)generated, and the impact of the less informative ones is reduced.

This loop is repeated until the posterior distribution converges. This means that the particle filter obtained enough accuracy to estimate the posterior of the hidden state. The effective sample size (ESS) represents the number of independent samples as a convergence metric. This is calculated by dividing the sum of the weights by the sum of the squared weights (Kuptametee & Aunsri, 2022). According to Chopin & Papaspiliopoulos (2020), this is explained 'intuitively' as a sample size. Here, given the fact that $ESS(W^{1:N}) \in [1,N]$, and that if k weights equal one, and the other weights (N – k) equal zero, then $ESS(W^{1:N}) = k$.
In Bayesian statistics, a correlated MCMC process causes correlated draws. This suggests that, in most cases, the effective sample size is typically less than the number of draws. As a result, when assessing the convergence of an MCMC model, it is common practice to rely on the effective sample size instead of the actual sample size. $ESS(W^{1:N}) = \frac{[(\sum_{n=1}^{N} w(X^n)]^2}{\sum_{n=1}^{N} [w(X^n)]^2}$

As an example of this intuition, achieving an ESS=10 in an Importance Sampling with N=100 is similar to drawing ten independent samples (*Webinar Centre International de Rencontres Mathématiques, 2018*).

When the ESS is low, the MCMC has not explored the parameter space well enough, and the samples obtained do not represent the posterior distribution. In such cases, resampling can help obtain more representative samples.

As a guideline applied in the Python particles library, the resampling is triggered whenever a minimum ESS ratio is below a threshold. The *Particles.core.smc.py* program sets this default at 0.5 (trigger when $\frac{ESS}{N} < ESSrmin = 0.5$)

Given the above background on importance sampling (IS) and resampling, it is possible to draw the algorithm from Figure 3. Here, particles are sampled from the prior and weighted on their likelihood with an importance sampling. The predictions are updated with the resampling at each step (Figure 6).

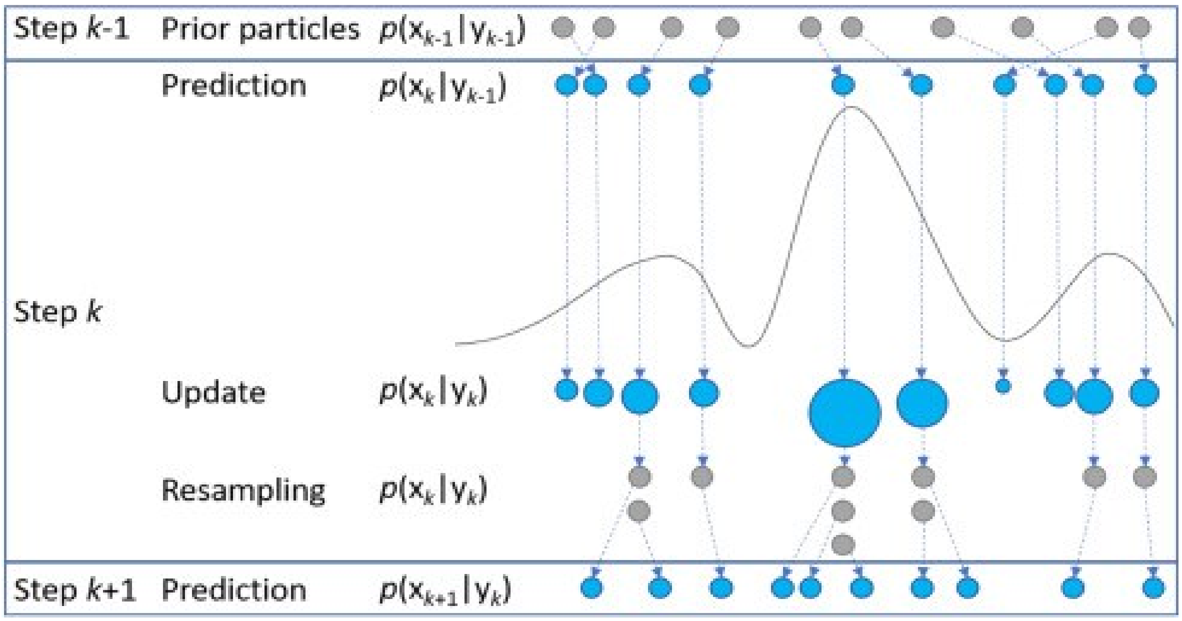


Figure 6: The concept of a particle filtering process. (Kuptametee & Aunsri, 2022)

This loop is repeated until the posterior distribution converges. This means that the particle filter obtained enough accuracy to estimate the posterior of the hidden state. Here, the effective sample size (ESS) values the sampling for good convergence, given by dividing the sum of the weights by the sum of the squared weights.

2.2 Training dataset and context

The KDD99 dataset is a widely used benchmark dataset in academic research for evaluating intrusion detection systems (IDSs). The paper of Tavallaee et al. (2009) highlights the structure and features of this dataset. Initially, it was created by the Defines Advanced Research Projects Agency (DARPA) in 1998 as part of their Intrusion Detection Evaluation Program (IDEA). The KDD99 dataset continues to be utilized for researching IDS systems (Özgür & Erdem, 2016), with over 150 articles studying its characteristics and applications. For this dissertation, a subset of this data will be used to process the PF algorithm.
This dataset is published on Kaggle (*Network Intrusion Detection*.) and published publicly here: https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection.

In a TCP/IP protocol, each data packet is treated as an independent communication unit and is encapsulated with its header containing information about the packet, such as the source and destination IP addresses, the packet sequence number, and the packet length (Ondeng, 2017). This information is used to ensure the reliable delivery of data and to manage the flow of data between the two endpoints of the connection.

Attention!
An especially important remark is that the KDD99 dataset includes features extracted from these packet headers and some other features related to the network connection, but it has no information on the sequence numbers. Therefore, another dataset, UNSW-NB15 (Moustafa & Slay, 2015a), was developed to counter those issues. One approach in the results will use this dataset.

2.3 Modelling choice of the parameters

2.3.1 Sequential modeling of θ

Given the problem statement, where one wants to model the classifiers for malicious network packets vs. the rest, the sequential Markov Chain will be used to model these classifiers of the HMM at a given time. In a generalized Bayesian way, it can be written as $p(\theta_{1:t}|y_{1:t})$ where θ is the set of parameters or classifiers from the hidden model under the scope. They can be inferenced based on the observations $y_{1:t}$ over time. The Markov chain (Figure 2) assumption requires the hidden state to be only dependent on its former state $p(\theta_t|\theta_{1:t-1}) = p(\theta_t|\theta_{t-1})$, where the history is forgotten. Also, the HMM extends this assumption and includes that the observation at a given time step is 'only' defined by the Markov state θ_t at time t. The distribution of that observation is simply depicted, conditional on its current state, or: $p(y_t|\theta_{1:t}) = p(y_t|\theta_t)$.

These Markov assumptions form the base for sequentially generating the model's posterior PDF (probability Density Function). Mathematically, when applying Base Theorem, this can be noted as $p(\theta_t|y_{0:t})$= $p(\theta_t|y_t) = [\prod_{i=1}^{t} p_i(\theta_i|y_i, \theta_{i-1})].p_0(\theta_0|y_0)$

The above formula shows the base calculation for the parameter θ_t given the data, starting with a first prior distribution $p_0(\theta_0|y_0)$. When new data comes in, the parameter is updated with a series of conditional probabilities. This can be seen as a matrix multiplication where the individual states of the θ_t-parameter set is calculated.

Because one cannot sample for this posterior density function, here is where a simple proposal function q(x) comes in, where the samples are easy to generate. Then, based on the ratio of p(x) by q(x), the importance weights at every time step are calculated with a similar recursion:

$$w_{0:t} = w_0 . \prod_{i=1}^{T} \frac{p_i(\theta_i|y_i, \theta_{i-1})}{q_i(\theta_i|\theta_{i-1})}$$

The right factor is the incremental weight of the sample.
Most ingredients for an SMC algorithm are present, a meaningful prior distribution, and state parameter dynamics are generated at every step.

2.3.2 Logistic regression as a classifier

Network intrusion data will be classified binary into malicious and normal. A logistic regression model is a supervised machine learning algorithm with a dichotomous outcome as a result. Here, we are looking to infer hidden states from the observations, meaning that the hidden classifiers will help model the binary outcome, given independent variables. In the field of machine learning, logistic regression is easier to understand and interpret, compared to other algorithms. Moreover, it is computationally efficient, even with large datasets. In the field of machine learning, it is worth mentioning that these, relatively basic operations can be accelerated using GPU parallelization. Python packages such as Tensorflow and PyTorch have these built in.

Mathematically, the regression formula, where a malicious network event happens, is noted as follows:

$$P(class = \ 'malicious'|Y_{network\ observations}) = \frac{1}{(1 + e^{-\beta Y})}$$

The classifiers will be estimated using a linear regression function, employing the ordinary least squares method to estimate the beta parameters. The logarithmic probability of being classified as "malicious" versus "normal," represented by the log-odds, will be utilized in this process.

$$log_e \left[\frac{P(class = \ 'malicious'|Y_{network\ observations})}{P(class = \ 'normal'|Y_{network\ observations})} \right] = \beta_0 + \beta_1 Y_1 + \ \dots + \beta_n Y_n + \varepsilon$$

When formulating a regression equation, the assumptions that hold are evaluated (Zach, 2020). First, the response variable is binary, allowing to use this regression type as a classifier. Second, the features of the network observations need to be pre-processed to avoid such multicollinearity. Here, highly correlated features are removed or combined. In this step, extreme outliers need identification and an extra pre-processing step to deal with.
Third, the observations need to be independent of each other. A residual plot of the logit regression is a technique to reveal potential dependencies between the observations.

Finally, a logistic regression model assumes a linear relationship between the logit of the response variable and each feature used in the model. The Box-Tidwell test can check for this (as an R package). If one variable is found to be nonlinear, it can be resolved by transforming the variables with a higher-order polynomial.

When running this algorithm during the classification test, these tasks will be scheduled as a pre-processing recipe: loading data, setting their data types, removing duplicates, and converting categorical features into binary dummy variables. Then, feature engineering steps will remove low-variance and multicollinearity, focusing on the features with the most important weight.

2.3.2.1 Estimation of the posterior PDF

On purpose, the regression parameters are globalized as a hyperparameter θ, that combines all chosen β_n and their respective variances σ_n^2 ?

As described above, the posterior density function of the θ parameters will be stated by its prior and likelihood. Sequentially, for every set of N particles up to time t-1, the posterior density function is known from the individual ith sample (of N particles) and its importance weight at that moment. $\{\theta^i_{(1:t-1)}, w^i_{(1:t-1)}\}$

1. At time t, the N particles are generated from the proposal distribution q(x), conditioned on the most recent particle distribution PDF(θ^i_{t-1})
2. For each particle, all N incrementing weights are calculated.
   Looking more closely, this incremental weight is nothing more than the likelihood of the particles under the proposed PDF.
   $$\boldsymbol{w}^{\boldsymbol{i}}_{(\boldsymbol{t}|\boldsymbol{t}-\boldsymbol{1})} = \frac{p_t(\theta_t|y_n, \theta_{t-1})}{q_t(\theta_t|\theta_{t-1})}$$
3. Update the particle weights by their increment.

$$w^i_{(1:t)} = w^i_{(1|t-1)} \cdot \boldsymbol{w}^{\boldsymbol{i}}_{(\boldsymbol{t}|\boldsymbol{t}-\boldsymbol{1})}$$

The algorithm above will continuously update as new data comes in.

2.3.2.2 Variance

$$\mathrm{Var}(\theta|y) = \frac{1}{N}\sum_{i=1}^{N} \omega_i\left(\theta_i - \hat{\theta}\right)^2$$

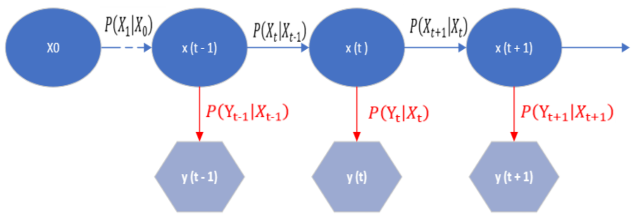
The variance of the parameter $\boldsymbol{\theta}$ is obtained through its weighted mean square error. Therefore, increasing the number of particles (**N**) in the estimation procedure tends to reduce the variance of the posterior distribution. This effect will be evaluated during the model-building phase.

2.3.3 State Space model of classes

Another approach is determining the likelihood of 'malicious' network traffic versus the other. Here, the intention is to use a particle filter for both categories separately on a labelled training dataset. However, the challenge will be to cope with unknown state dynamics.

2.3.3.1 Noise only model

In our problem statement, no information is available on the state dynamics necessary for constructing a filtering process. In the absence of these, we still try to estimate the states based on the previous ones, added with random Normal noise.



$$X_t = X_{t-1} + u_t \ \ with\ u_t \sim N(0, cov_u)$$

$$Y_t = Y_t + v_t \ \ with\ v_t \sim N(0, cov_v)$$

Figure 7: Approach the posterior as a state space model with noise only.

This way, even with limited dynamics, one can filter the posterior likelihood of the states, given the data. Here, a separate Posterior distribution for each category is modelled.

2.3.3.2 Likelihood classification and Goodness of fit

An approach to classification involves evaluating the likelihood of each distribution and comparing the most probable one. For example, to check the likelihood of an observation vector y given a probability distribution of the hidden state x, we can express it mathematically as:

$LL_{safe} = \mathrm{P}(\mathrm{Y}|X_{safe})$ and $LL_{malicious} = \mathrm{P}(\mathrm{Y}|X_{malicious})$

The likelihood ratio of both distributions is stated as $LL_{RATIO} = \frac{LL_{safe}}{LL_{malicious}}$.

As a test statistic $LL_{statistic} = -2\ log(LL_{RATIO})$ ~ χ2(0) follows a χ2 distribution

Here, the H_0 will check if the observation is equally likely to originate from either of the classes.

2.4 Concept drift

Investigating a network data stream demands a statistical inference of the hidden model over time. Particularly, this behaviour must be monitored when training a classification model to classify malicious network traffic. In this context, not adapting for this, the model trained will not accurately reflect the underlying distribution anymore. To address this, it is necessary to update or retrain the model on the current data periodically. Another option to ensure accurate modelling is to use online algorithms that update the model at every additional data entry. (Wang & Abraham, 2015).

As these models the classification boundaries, here focusing on logistic regression coefficients, the particle filter method can be a viable candidate to track changes in these classifiers over time.

The most common way to manage these state changes over time is through ensemble learning techniques. They achieve better accuracy and robustness by combining multiple classifiers into one general prediction. The Weighted Majority, for example, uses a group of M models, where a weighted average estimates one model, the weighing based on the beliefs or performance of each separate classifier.

$$\overline{pred} = \frac{\sum_{i=1}^{M} w_i pred_i}{\sum_{i=1}^{M} w_i}$$

A common method is to start with equal weights for each classifier and halve it when classifying wrong. Other methods are bagging with bootstrapping on the training data.

The model classifiers could change in the network traffic over time and generate concept drift. In detecting malicious packets, it is necessary to treat this effect. Especially there must be a technique to pick up the most recent data in the stream and forget the classifiers trained from old data. Intuitively, Markov Chains would be a proper choice to manage this because it refers to a change of states over time. However, one must remember that a minimal condition is the time-homogeneity of the transition kernel. Chances are that this condition is not fulfilled, and other techniques will be searched for. As mentioned above, the ensemble methods come in handy to oversee this, literally using the most recent data and forgetting the past features from the training data.

Here, in this framework of PF, the Adaptive filter, a variant of the PF algorithm, could be beneficial to handle concept drift. Particularly, the data is filtered sequentially to model the hidden classifier parameters of the hidden model.

Summary measures are considered when following concept drift, as visualized in Figure 8 (Wang & Abraham, 2015):

$p(\theta_0)$: the prior probability of the classifying scheme may change significantly. For example, this can appear when the amount of data in both classes is imbalanced.

$p(\theta_t\,|y_0)$: the posterior probability of the classifiers may change over time, diagnosing a real drift of the classifier.

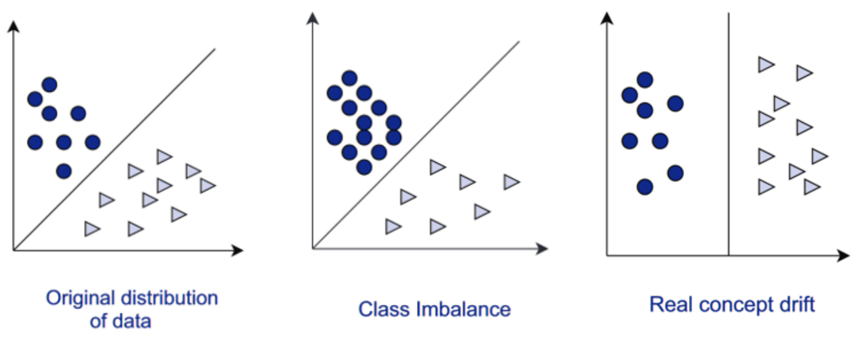


Figure 8: Summary measures show class imbalance and real concept drift. (Wang & Abraham, 2015)

A change in concept can be abrupt or continuous. Also, it can be incremental or recurring. Therefore, the hidden model classifiers generated from the data will predict whether the potential category is malicious. In addition, the potential drift of the model will be detected from the sequential filtering, where the most focus will be on the abrupt drift.

In any case, an ideal algorithm to cope with concept drift needs the following: fast adaptation to drift, robustness to potential noise, and recognizing of recurring concepts (Tsymbal, 2004).

2.5 Classification performance and quality

A confusion matrix will assess the classification performance of a binary outcome dataset, where the categories are defined as "Intrusion" and "OK." The confusion matrix allows for evaluating the classification algorithm's effectiveness in accurately predicting the true labels. Each row of the matrix corresponds to the true labels, while each column represents the predicted labels. Examining the confusion matrix, metrics such as accuracy and precision are calculated to evaluate the algorithm's capability in distinguishing between instances categorized as "Intrusion" and "OK."
After a visual approach, a (Pearson) chi-square test statistic will check the statistical performance of the classification algorithms (Howell, 2011).

2.6 Curse of dimensionality

Although out of the scope of this dissertation, it is worth checking the impact of systems having higher dimensions. SMC algorithms face several challenges in this respect.

The number of particles required to maintain a representative sample of the state space grows exponentially with the dimensionality ($\frac{q(x)}{m(x)} = \prod_{t=0}^{T} \frac{q(x_t)}{m(x_t)}$) $with\ a\ weight\ variance\ ratio^{T+1} - 1$ (Centre International de Rencontres Mathématiques, 2018). This exponential growth increases computational complexity and resource requirements, making SMC algorithms computationally demanding and sometimes infeasible for high-dimensional problems.

In high-dimensional state spaces, where only a few particles retain significant weights, particle degeneracy occurs more frequently. As a result, the effective sample size decreases, reducing the representativeness of the particle approximation and adversely affecting the accuracy of the SMC algorithm.

Numerous techniques have been developed to address these challenges, including advanced resampling strategies, adaptive methods for adjusting the number of particles, and dimensionality reduction techniques. The Particles package (*Nchopin/Particles*, github.com) deals with this, but studying this in detail is out of scope of this thesis.

3 RESULTS

This chapter will discuss two frameworks for filtering network intrusion data and performing classification. First, some guidance is given in the Python libraries used and the Data transformation. Then, the logistic regression model (2.3.2) is constructed, classifying data when new observations are streaming in. Afterwards, the data is modelled into a simple state space model (with a model for each category where the posterior is sequentially updated (2.3.3)).

3.1 Python libraries and data transformation.

During the modelling work, the sequential monte carlo (SMC) framework follows the structure explained in this book: *An Introduction to Sequential Monte Carlo*. (Chopin & Papaspiliopoulos, 2020). Especially the accompanying Python library 'particles' is being used (*https://github.com/nchopin/particles*). This library is well-equipped with the necessary modules for an SMC algorithm. The modules *distributions*, *state space models,* and *core SMC* are consolidated in one library and widely scalable with parallelization.

In addition, arrays are preferred during data transformation with NumPy operations. NumPy is optimized for calculations involving arrays and offers many built-in functions and methods for manipulating and transforming arrays. Also, the particles library fully utilizes NumPy input and calculations during its processing. Pandas operations would drastically drop performance (Byeon, 2020). However, Pandas and Seaborn are utilized for certain transformations, metrics, and visualizations. The OpenAI code generator assists in common data operations and visualizations (M. Chen et al., 2021).

The SMC sampler function of the core module takes a Feynman Kac object (**fk**) as a framework to encompass the model and its data, together with the number of particles (N). Optionally, the resampling technique, processors, and collected summaries can be specified.

Finally, outcome metrics are calculated with some built-in methods of the scikit-learn library.

3.2 Logistic regression classifier

3.2.1 Pre-processing

First, deducting from the KDD99 network data, feature selection is made to create a high-quality dataset with classification on useful features only.

Initially, categorical variables are converted to indicator variables (*pandas.get_dummies()*). Then, only features with high variances are kept. Finally, to keep meaningful results, only features that explain a minimum of 80% of the variance are kept (T. Yu et al., 2019). The concept behind variance thresholding is that features with a low variance are often less valuable than those with a high variance. This method entails computing each attribute's variance and eliminating any attributes that do not meet the designated threshold for variance (*sklearn.feature_selection.VarianceThreshold(threshold = 0.8)*).

Finally, the remaining features' Variance inflation factor (VIF) is calculated (Dupuis & Victoria-Feser, 2013). The VIF is calculated as the ratio of the variance of the estimated coefficients of a model that includes 'all' independent variables to the variance of the estimated coefficients of a model that excludes a particular independent variable. Mathematically $VIF_{X_i} = \frac{1}{(1-R^2_{X_i})}$, is regressing each feature with the other features. A VIF_{X_i} larger than 1 reveals some degree of multicollinearity. A robust technique to regularize later models, features with $VIF_{X_i} > 5$ are removed (Sohil et al., 2022). After feature engineering, the modelling is done on 25192 observations with 7 features.

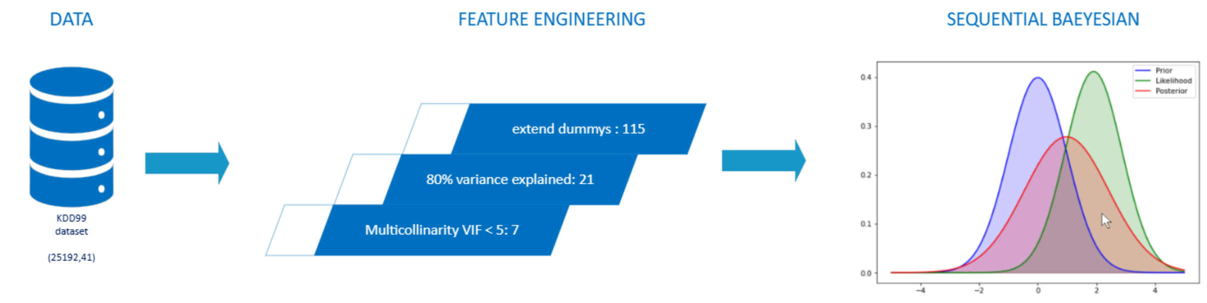


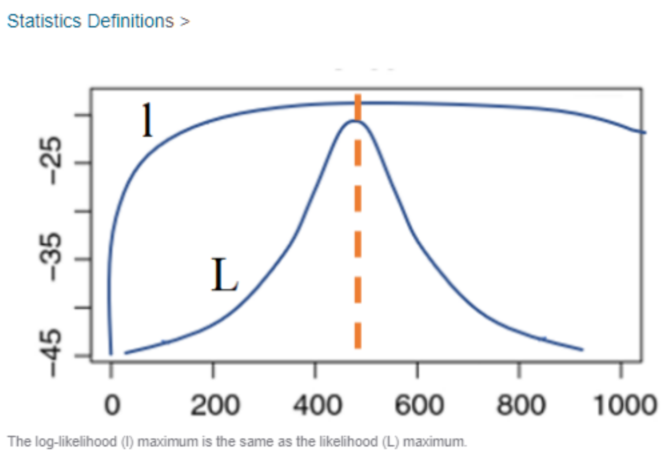
Figure 9: Data processing towards the Bayesian filtering

3.2.2 Feynman-Kac object based on Logistic model

A Bayesian logistic regression model is now constructed, where a prior distribution and a likelihood function are specified to calculate a posterior distribution. The prior distribution is a multivariate normal distribution with a diagonal covariance matrix.

Technically, a state space model class is constructed to track the hidden θ regression parameters ($\theta = \{\beta, \sigma\}$) along with the income observations of the 7 features.
Here, the prior distribution of the β_{i} coefficients are defined from a multivariate normal with a mean of 0 and a deviance with a 0.5 scale. One limiting factor is that larger scales (>.8) will assign larger variance on the prior. This could be computationally unfeasible due to matrix calculation limitations, specifically an impossible Cholesky matrix decomposition when reaching singular matrices.

Given this prior, the Logistic Model Class (LMC) will be defined, calculating the likelihood of the data under the current regression parameters. When calculating the likelihood of a model given the data, many probabilities are multiplied together as a joint probability. This can become very small and difficult to work with when data points are large. Taking the logarithm of the likelihood function allows us to transform the product into a sum, which is easier to work with numerically. Technically, this logarithm trick avoids computational



overflow errors.

Figure 10: Loglikelihood function for optimization of Bayesian models ((Stephanie, 2021)

Below, the Bayesian model, which will be repeated sequentially to filter out the hidden classification parameter, is given. The aim to achieve the posterior parameter estimate will be recursively achieved by $P_t(\theta_t|y_{0:t}) = P_t(y_t|\theta_{t-1}) * P_{t-1}(\theta_{t-1}|y_{0:t-1})$.
By defining a prior and a class object, this framework is defined.

```
# PRIOR

scales = 0.5 * np.ones(p)      # for all column beta
scales[0] = 20                 # intercept has a larger scale

'''Define a prior with multivariate Normal distro'''


prior = dists.StructDist(
                {'beta':dists.MvNormal(scale=scales, cov=np.eye(p))})


''' The prior is MVN distribution with an eye cov matrix. '''


# BAYESIAN MODEL = PRIOR x LIKELIHOOD

class LogisticRegression(ssps.StaticModel):
       def logpyt(self, theta, t):
        # loglikelihood factor t, for given theta
       lin = np.matmul(theta['beta'], data[t, :])
        #Matrix product of two arrays. => dot product of θt * Xt
       return -np.logaddexp(0., -lin)


''' the numpy.logaddexp() function is used to calculate the logarithm of
the sum of exponentiations of the inputs. This function is useful in
statistics where the calculated probabilities of events may be so small as
to exceed the range of normal floating-point numbers. (“numpy.logaddexp2 —
NumPy v1.24 Manual”) ''' (“numpy.logaddexp2 — NumPy v1.24 Manual”)
```

Here, the aim is to estimate the final posterior $P_t(\theta_t)$ in a filtering process. An Iterated Batch Importance Sampling (IBIS) is used. This algorithm tracks a chain of posteriors where importance sampling is applied to sample from a proposal distribution over a given Markov chain length.

The algorithm needs to compute the likelihood factor of two subsequent distributions. (Chopin & Papaspiliopoulos, 2020). Mathematically $P_t^{\theta}(y_t|y_{0:t-1}) = \frac{\gamma_t(\theta)}{\gamma_{t-1}(\theta)}$.

The Feynman-Kac framework can be used to express the probability density function (PDF) of the system state at a future time, given the PDF of the state at the current time and the transition probabilities between states. This can prove beneficial when dealing with scenarios where a set of differential equations governs a system's state, and the objective is to estimate the system's state using noisy observations.

```
# DEFINE FEYNMAN-KAC formalism

from particles import smc_samplers as ssps
logistic_model = LogisticRegression(data=dataset, prior=prior)

''' The Iterated Batch Importance Sampling mechanism is used'''
fk_LR = ssps.IBIS(model=logistic_model, len_chain=10)
```

Computational Attention!

Here, attention is needed to control the computational challenge. A set of N particles represents the posterior distribution of the system state over time. The likelihood ratio is a measure of the fit between the observed data and the current estimate of the system state. This updates the weights of the particles along with income data. Given T time steps, the total particles need N*T updates. In addition, every particle ∈ N gets its likelihood ratio evaluated in the weighing step at every time. The computational complexity grows quadratically $\vartheta(N^2)$. *Example: for 10 particles during 5 time steps, there are 50 particle updates computed, as well as 50 times computing the likelihood.*

3.2.3 Sampler run of the fk model

The Feynman-Kac formalism above defines the likelihood of the data as a Markov Chain that sequences the hidden states of the model. In this approach, the posterior distribution of the model parameters is obtained by sampling intermediate distributions based on the data collected up to that specific point. During the SMC sampling process, particles are resampled and reweighted to obtain this.

```
# RUN THE SMC SAMPLER
'''use the core module of the particles library to sample the fk-formalism
with N particles. Optionally, define the resampling technique'''


import particles result_LR = particles.multiSMC(fk=fk_LR,
                            N=amount_of_particles,
                            resampling = 'systematic')
```

The moments of the estimators are tracked for later analysis and visualization.

3.2.4 Visualization

As mentioned, the computational effort for obtaining the posterior θ distributions depends on the number of particles used in the importance sampling. Iterated Batch Importance Sampling (IBIS) can be used to explore "static models". The chain length determines the number of iterations performed within each batch. During each iteration, the parameters are updated based on importance sampling, which involves generating new samples from an importance distribution and reweighting them based on their likelihood (Chopin, 2002). Figure 11 illustrates the impact of multiple chain lengths and sampled particles on the computational cost.

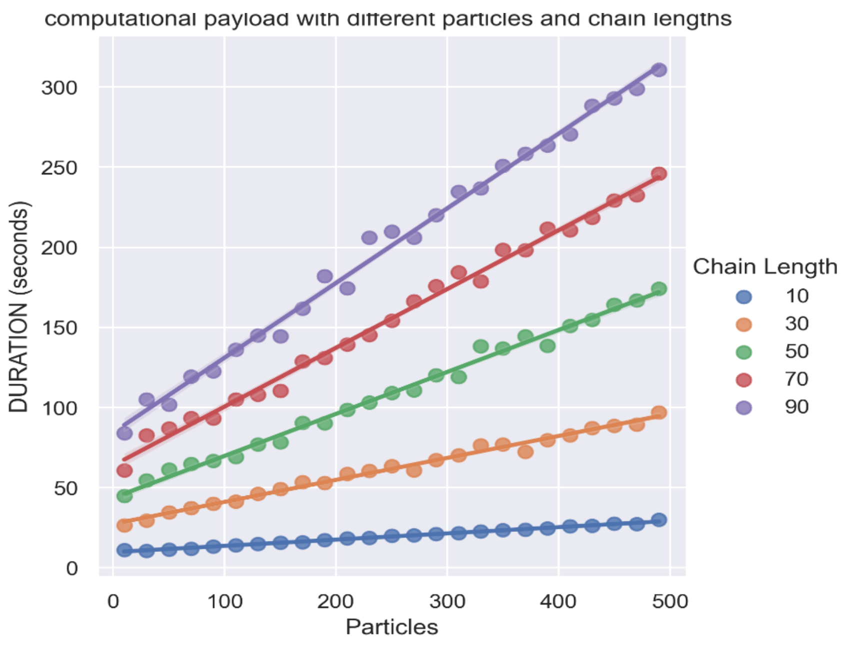


Figure 11: Computational cost of the SMC sampler by the number of particles and chosen chain length.

This "cost" prospect was estimated on the last 5000 observations of the intrusion dataset. With this insight, the tested combinations are chosen as N = [100, 300], each with 2 IBIS models (chain length = [10, 50]). Using more particles in the θ states leads to more stable estimation. In addition, the first part of the chain (ca. 30%) can be seen as the burn-in period for the Iterated Importance Sampling. During the burn-in time, the initial set of particles is sampled from the prior distribution and then sequentially propagated through the model using importance sampling. This burn-in phase allows the particles to explore the state space and reach regions of high likelihood before the iterative updates start (Nguyen et al., 2015).

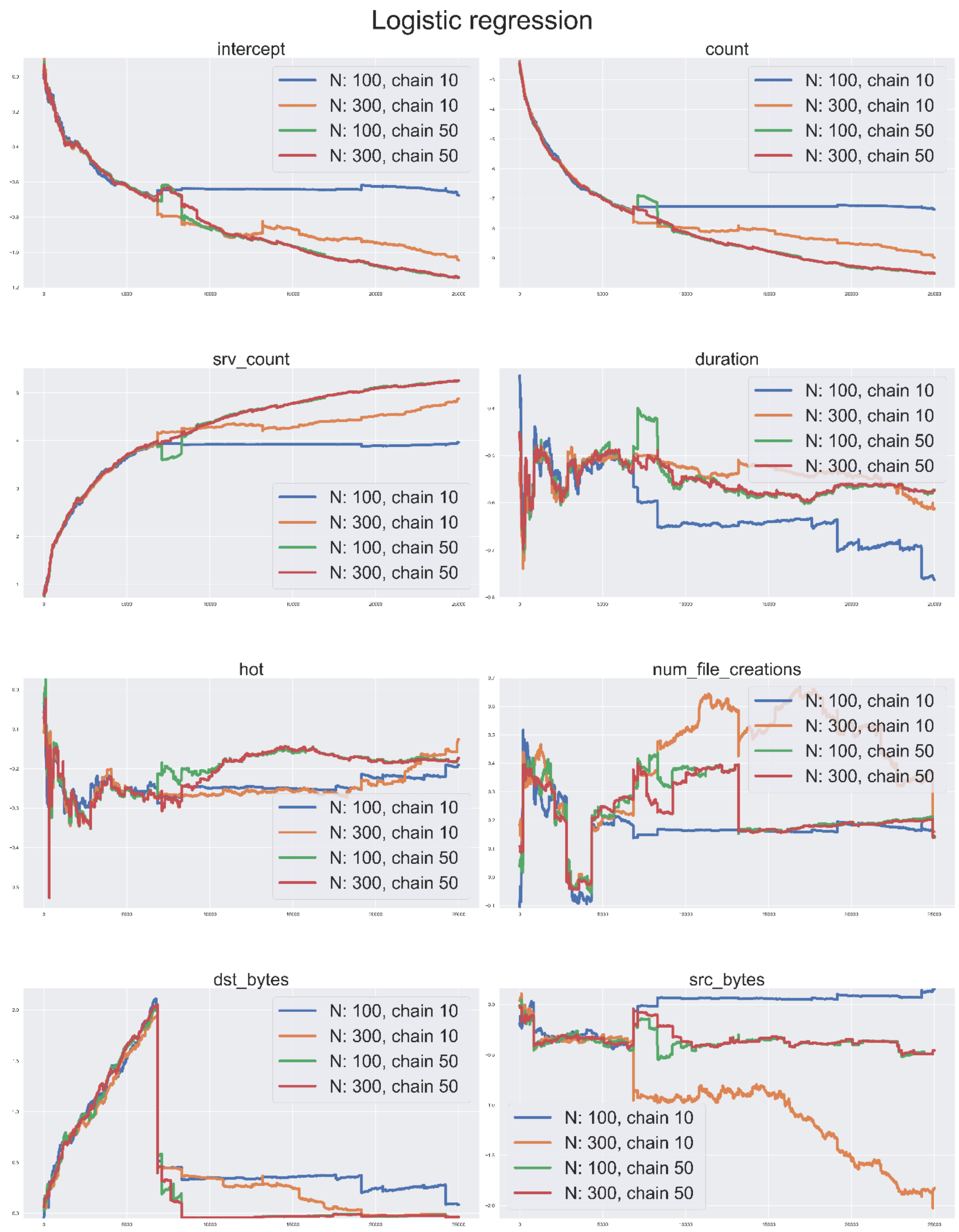


Figure 12: Evolution of the SMC chain of θ, visualized as time updated values of the β-estimators.

Before drawing initial conclusions, it is evident from the data that longer Markov chains exhibit greater stability, particularly when utilizing a higher number of particles in the sampling process. On the other hand, it is observed that the features such as **num_file creations**, **dst_bytes**, and **src_bytes** experience a certain concept drift within the model. This could indicate a discrepancy in data quality or a drift in the entire model, which falls outside the scope of this thesis. Ultimately, the aim is to predict to what category ("non-harmful" or "anomaly" a network packet belongs, given its 7 selected features and an intercept.

Statistically

Remember that interpreting a β-coefficient of a logistic regression model is somewhat tricky. Its value represents the change in the log odds of the outcome variable (log$\frac{no\ intrusion}{intrusion}$). Intuitively, given the chosen model, β_i = a, will result in an impact on the odds $\frac{no\ intrusion}{intrusion}$ exponentially (e^{a}), given all other $\beta -\ values$ constant (Sperandei, 2014). Finally, the subsequent section will evaluate this model using a confusion matrix.

mathematically: P(Y = *no intrusion)* = $1\ -\ \frac{1}{1+e^{-(\beta_0+\ \beta_1 x_1+\ldots+\ \beta_7 x_7)}}$

3.2.5 Confusion matrix of the model

The logistic regression model allows predicting the odds between two classes based on the 7 features selected in the process as observations. Any predicted probability above the threshold is assigned to the positive class, and the predicted probability below the threshold is assigned to the negative class. Commonly, the threshold is set at 0.5. However, the threshold can be tuned based on the specific requirements of this problem or the desired trade-off between precision and recall. Here, we want to balance between false positives and false negatives. In this problem, the semantic wording needs attention. These are defined below:

```
Rows represent the true class labels,
Columns represent the predicted class labels.
The four quadrants of the matrix are defined as follows:

True Positive (TP): Correctly predicted ANOMALY
True Negative (TN): Correctly predicted NON-HARMFUL

False Positive (FP): Incorrectly predicted ANOMALY (Type I error)
False Negative (FN): Incorrectly predicted NON-HARMFUL (Type II error)


threshold = 0.5

for each combination in sampler_results:
    Y_pred = [1 / (1 + np.exp(-rt)) for rt in regressionterm]
    Y_est = [1 if y > threshold else 0 for y in Y_pred]

    TP , FP, FN , TN = confusion_matrix(Y_orig, Y_est).ravel()
    Accuracy = (TP + TN) / (TP + TN + FP + FN)
    Precision = TP / (TP + FP)  #correctly predicted anomalies

100 particles, chain:10 = {'Accuracy': 0.69  , 'Precision':0.35}
300 particles, chain:10 = {'Accuracy': 0.68  , 'Precision':0.34}
100 particles, chain:50 = {'Accuracy': 0.68  , 'Precision':0.35}
300 particles, chain:50 = {'Accuracy': 0.69  , 'Precision':0.35}
```

Firstly, it should be noted that the classification performance is relatively consistent. Therefore, the model with 300 particles and a chain length of 50 here will be used for further comments. As stated, threshold 0.5 results in an accuracy of 69% where the model correctly detects the class. If the classes are balanced, a random classifier will achieve 50% accuracy by guessing the classes randomly. In this case, our classifier outperforms the random chance and could be considered a valuable option. However, the aim is to detect an incoming network observation and classify a network intrusion correctly. Zooming on this precision, the model performs weakly. As a last option, one could 'tune' the threshold. As a result, forcing a minimum of 50% precision degree, a tuning mechanism, and applying a grid search, obtains another balance : Accuracy: 35% and Precision 51%. The threshold for comparing the predicted to the real data was moved to 0.77 to obtain this.

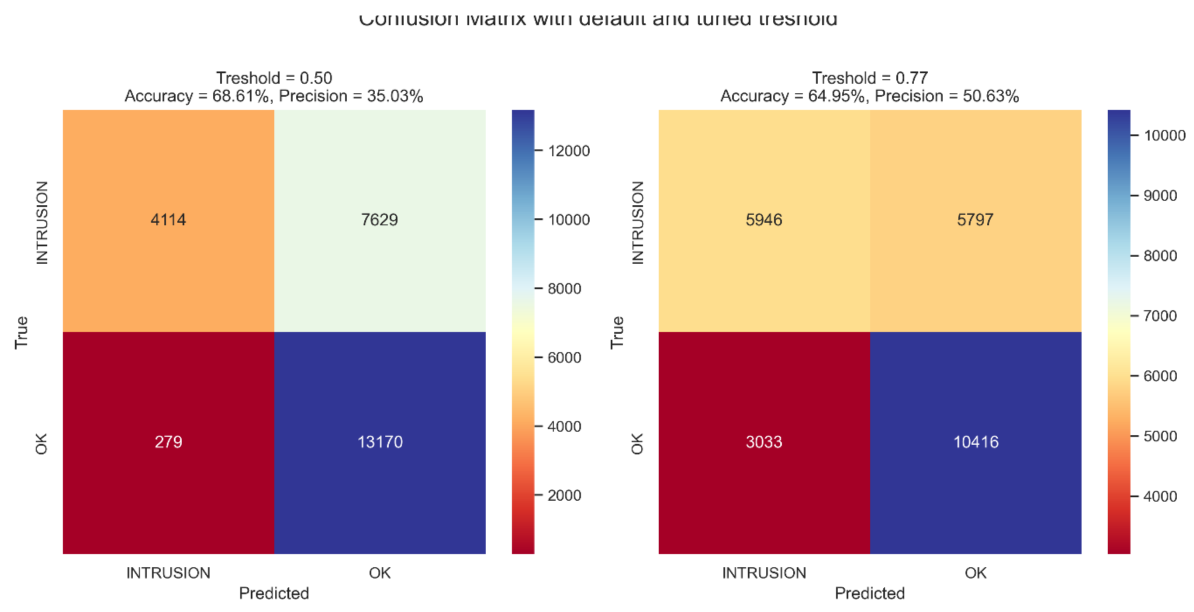


Figure 13: Confusion matrix results with default threshold and tuned threshold

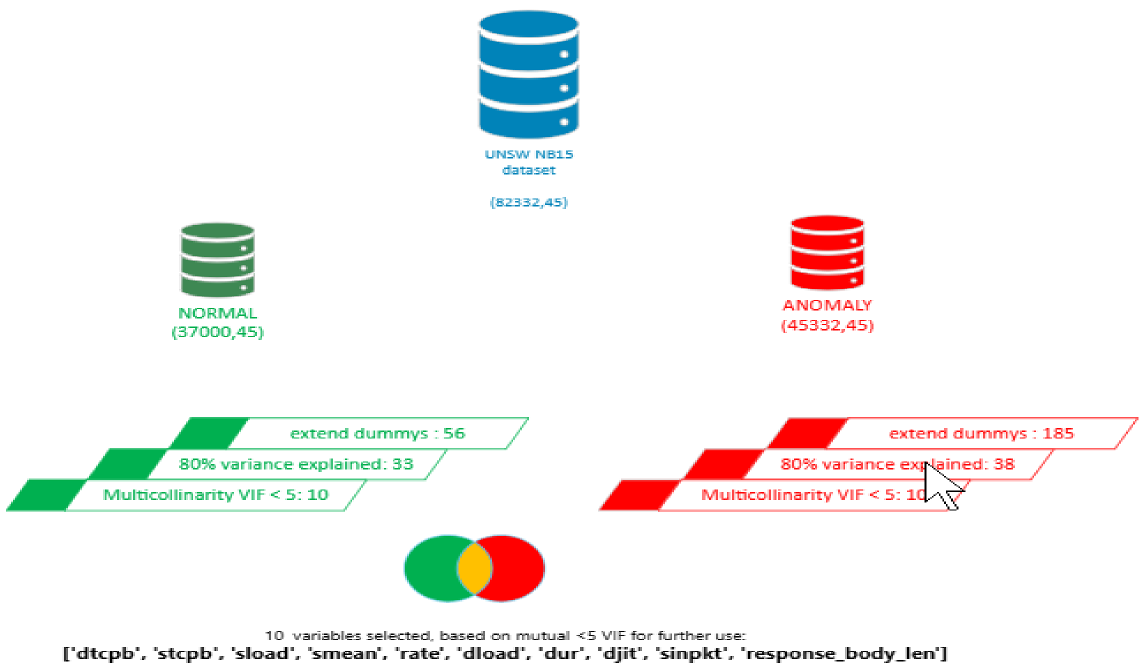
In conclusion, using 7 selected features of the network intrusion dataset can assist in detecting potential intrusion when challenging new incoming data entries. However, the accuracy and precision are low compared to other machine learning techniques.

In literature, where different models are compared to each other with respect to this KDD99 dataset (Ravipati & Abualkibash, 2019), the Random Forest (RF) technique outperforms with 99% accuracy and a 2.4% false alarm rate. However, compared to other techniques, such as the proposed regression, it has some disadvantages. Especially when dealing with a high dimension, the computational cost is large. In addition, that type of model is prone to overfitting. Therefore, applying an RF technique in anomaly detection can fail with unseen data. This dissertation focuses on the power of particle filtering and its potential. Another option is discussed in the next section.

3.3 State Space model

3.3.1 Pre-processing

Another approach to making the sequential Bayesian inference on hidden states is constructing it as a state space model (see section 2.3.3). The UNSW-NB15 (Moustafa & Slay, 2015) dataset is used here. Particularly, this data includes consequent time indices of the observations, which is much needed to design the state space model.


UNSW NB15
dataset
(82332,45)
NORMAL
(37000,45)
ANOMALY
(45332,45)
extend dummys : 56
80% variance explained: 33
Multicollinarity VIF < 5: 10
extend dummys : 185
80% variance explained: 38
Multicollinarity VIF < 5: 10

10 variables selected, based on mutual <5 VIF for further use:
['dtcpb', 'stcpb', 'sload', 'smean', 'rate', 'dload', 'dur', 'djit', 'sinpkt', 'response_body_len']

Figure 14: Data processing on the State Space Model of the UNSW-NB15 dataset.

The dataset is labelled malicious and non-harmful data packets in two classes. A state space model is used for each class to characterize its distribution. A reduced feature set is engineered like the former algorithm (3.2). After creating dummy variables, variance thresholding, and using the non-correlating VIF features, 10 features are mutually selected for a multivariate model. Note that the features occur in both distributions to enable the assessment of the probability of new observations. These features will be tracked in the model and used for later likelihood mapping.

selected features:	dur	djit	response_body_len	sinpkt	sload	rate	dload	dtcpb	stcpb	smean

3.3.2 Feynman-Kac object based on State Space Model

First, a state space model for the data observations is modelled. Typically, this consists of two components: a state equation that describes the evolution of the underlying system or process (aka. the transition kernel). Secondly, an observation equation that relates those hidden states to the observed data is defined.

Here, there are no explicit dynamics, and therefore the usefulness of the state space model could be limited. As an assumption, the model captures the 'moments' of the observation distribution as the hidden state in a time step. The dynamics of the transition function are naively defined as random noise posed on its former state. Awareness is needed to assess when the moments are non-Gaussian. Using the features' moments alone can be inaccurate.

GENERAL STATE SPACE MODEL for ***each*** category:

$\boldsymbol{P}(\boldsymbol{X}_0) = MvNormal(0,1)$ as the **prior** distribution for the hidden state

$\boldsymbol{P}(\boldsymbol{X}_t|\boldsymbol{X}_{t-1}) = MvNormal(\boldsymbol{X}_{t-1},\Sigma_x)$ as the **Markov Kernel** extends from the state $\boldsymbol{X}_{t-1}$ **to** state $\boldsymbol{X}_t$

$\boldsymbol{P}(\boldsymbol{Y}_t|\boldsymbol{X}_t) = \mathrm{MvNormal}(\boldsymbol{X}_t,\Sigma_y)$ as the observations, as the **observation function** of the state

The goal is to estimate the posterior distributions with the SMC algorithms:

$\boldsymbol{P}(\boldsymbol{X}_t|\boldsymbol{Y}_{0:t})$, or the posterior distribution of the states, given all the observations, from $time_0$ to $time_t$ is estimated sequentially. More particularly :

$\boldsymbol{P}_{NON-HARMFUL}(\boldsymbol{X}_t|\boldsymbol{Y}_{NON-HARMFUL_{0:t}})$ and $\boldsymbol{P}_{ANOMALY}(\boldsymbol{X}_t|\boldsymbol{Y}_{ANOMALY_{0:t}})$

```
# STATE SPACE MODEL CLASS

import particles
from particles, import state_space_models as SSM

''' The hidden Markov model defines multivariate states that proceed with
RANDOM noise according to a multivariate Normal Distribution of the
preceding state and noise.

The observations follow their corresponding State and random noise. '''

class MULTIVARIATE(SSM.StateSpaceModel):

    default_params = {'rho': 1, 'nvars': 1,  'start': np. zeros(1)}

    def PX0(self):  # The law of X_0   MVN around start
      return dists.MvNormal(loc=self.start,
                            cov=np.eye(self.nvars)/np.sqrt(1-self.rho**2))

    def PX(self, t, xp):  # The law of X_t conditional on X_{t-1}
      return dists.MvNormal(loc=self.start +self.rho*(xp - self.start),
                            cov=np.eye(self.nvars))

    def PY(self, t, xp, x):  # the law of Y_t given X_t and X_{t-1}
      return dists.MvNormal(loc=x,
                            scale = np.abs(x),
                            cov=np.eye(self.nvars))
```

This state space model class only allows generating the 10 hidden distribution moments based on the selected features by adding normal noise to the state transitions. In addition, this approach defines two Feynman-Kac formalisms, one for each network intrusion class: the non-harmful and anomaly data packets.

```
# DEFINE FEYNMAN-KAC formalism

''' Define SSM with parameters with rho accounting for the dynamics
proportion'''

N_SSM = MULTIVARIATE(rho = 0.9, covY=np.eye(NumberVars), nvars= NumberVars,
start = mean_NON_HARMFUL)

A_SSM = MULTIVARIATE(rho = 0.9, covY= np.eye(NumberVars), nvars=
NumberVars, start = mean_ANOMALY)

''' Define fk elements with parameters'.''

from particles import state_space_models as SSM



#Feynman-Kac framework that encompasses the SSM and the data in a bootstrap
strategy.

N_fk_SSM =ssm.Bootstrap(ssm=N_SSM, data=NON_HARMFUL.tolist())
A_fk_SSM =ssm.Bootstrap(ssm=A_SSM, data=ANOMALY.tolist())
```

The rho parameter represents the correlation or dependence between the variables in the SSM. In this case, a value of 0.9 is assigned for both N_SSM and A_SSM, indicating a strong positive correlation between the variables. Therefore, this parameter in the model can be tuned for. The next step involves using the Feynman-Kac framework to incorporate the SSM instances and the corresponding data into a bootstrap strategy. Then, as mentioned in 0, importance sampling is applied in the sequential algorithm, followed by resampling (2.1.4). The resampling step, often referred to as the "bootstrap" step in the context of SIR, is crucial for maintaining diversity in the particle set and preventing particle degeneracy during the estimation of the posterior (Kuptametee & Aunsri, 2022).

3.3.3 Sampler run of the fk model

```
# RUN THE SMC SAMPLER

''' The SMC sampler is executed for both classes'''

result_SSM = particles.multiSMC(fk=[ N_fk_SSM, N_fk_SSM],
                                N=amount_of_particles,
                                resampling = 'systematic')
```

Like the logistic classification estimators, the estimated moments of the hidden states are traced. The model resamples 1000 particles sequentially with bootstrapping.

3.3.4 Visualization

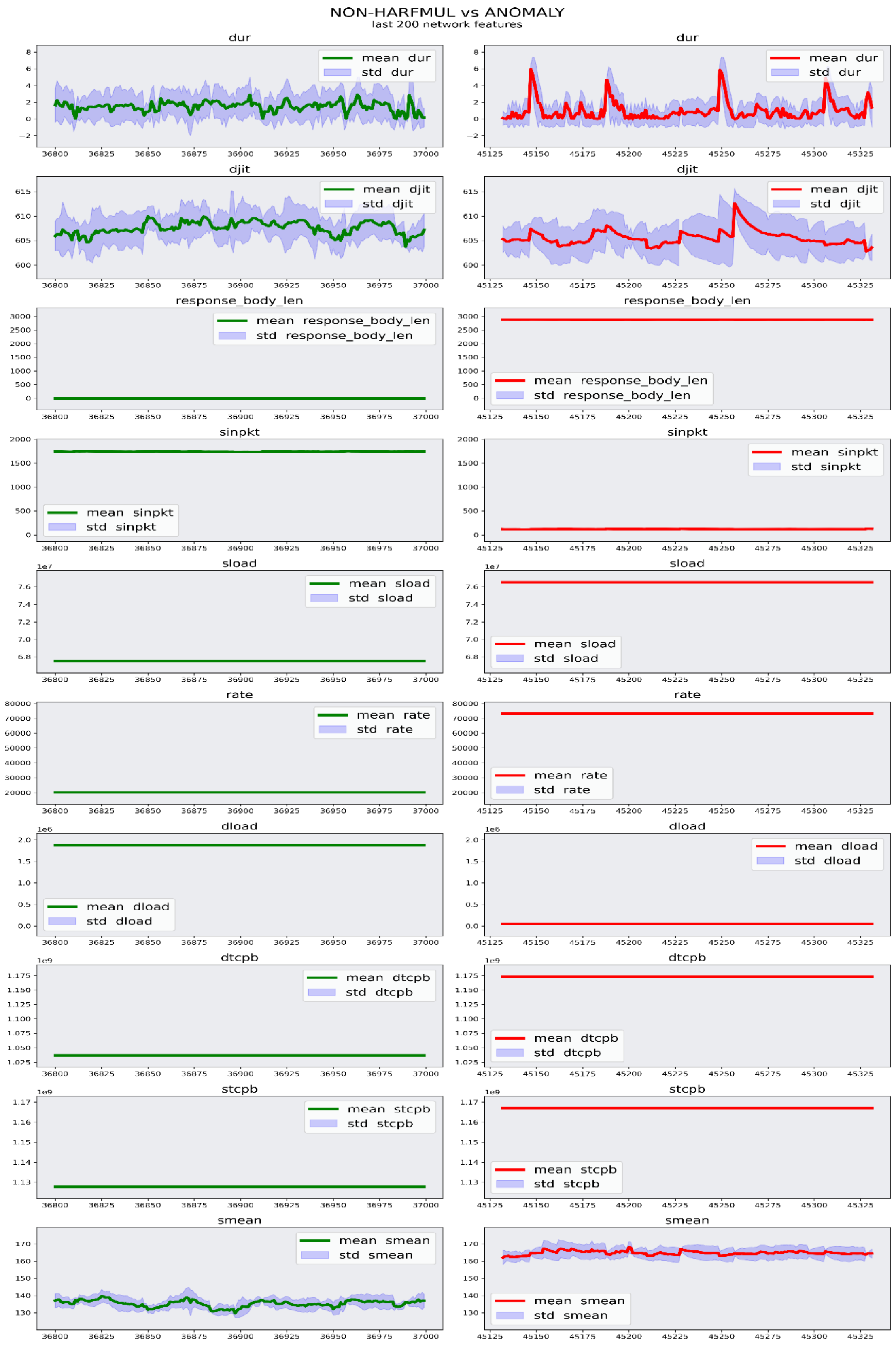


Figure 15: Moments of the 10 features in the UNSW-NB15 State Space Model

During the sequential sampling of the 10 model features, travel along their Markov Chain. The observations tune this path according to their likelihood.

Figure 15 reveals some differences in the states of the selected features. To evaluate whether the feature in one state space model is significantly different from the same feature in the other state space model, a paired t-test can be used. In this sequential model, the means of the feature values between the two models using a paired t-test.

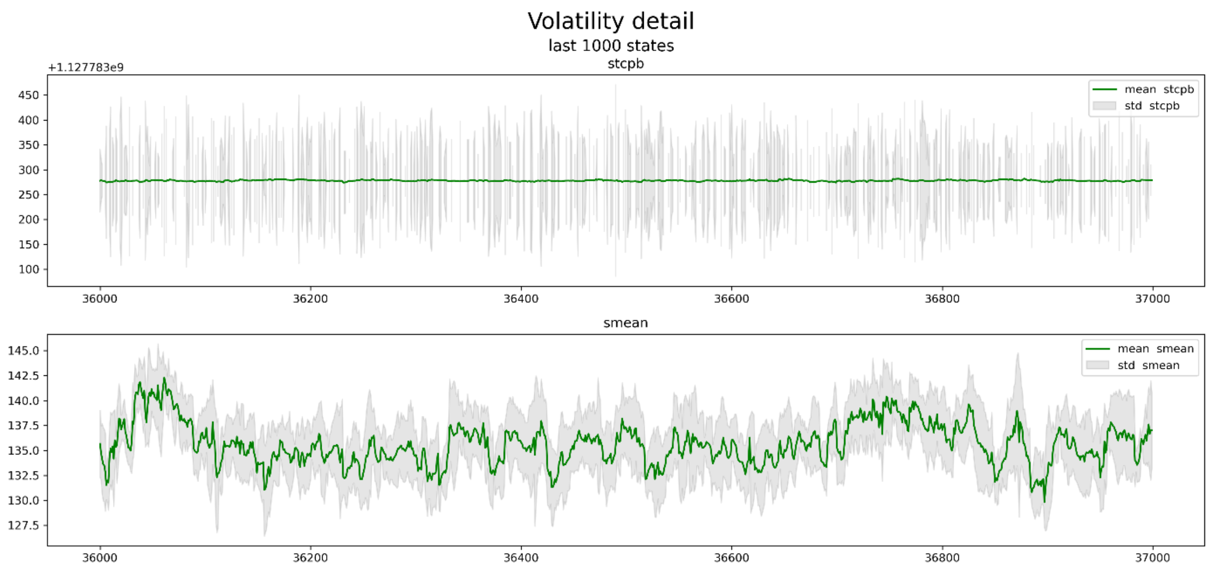


Figure 16: Detail of 2 ‘NON-HARMFUL’ intrusion features, constant and volatility

First, a univariate statistical test checks upon differences between both groups. Here, a paired t-test checks the differences between both. Although, as a more robust technique that relaxes the necessary assumptions for a student-T test, the Wilcoxon rank-sum test is used additionally. The test statistic determines the likelihood of two group samples (here, respectively, the ‘non-harmful’ group and the ‘anomaly’ group (Rosner et al., 2006).

FEATURES	MEAN NON-HARMFUL	MEAN ANOMALY	H_0: P(non − harmful) = P(anomaly) parametric and non − parametric tests: $P < 0.001$	
			PAIRED T-TEST *T-STATISTIC*	WILCOXON RANK-SUM *U-STATISTIC*
DUR	0.44	1.86	-2.56	-8.61
DJIT	606.34	603.52	6.05	-8.61
RESPONSE_BODY_LEN	0.40	2 879.07	-9 557.25	-8.61
SINPKT	1 747.39	123.51	952.64	-8.61
SLOAD	67 539 764.73	76 510 352.50	-26 940 086.11	-8.61
RATE	20 175.21	73 133.23	-177 760.20	-8.61
DLOAD	1 879 055.53	48 783.41	3 227 774.46	-8.61
DTCPB	1 037 201 379.71	1 173 286 245.14	-283 950 514.55	-8.61
STCPB	1 127 783 278.62	1 167 101 849.60	-141 859 541.11	-8.61
SMEAN	137.01	164.15	-134.04	-8.61

Table 1: Univariate distribution test of the selected features.

Table 1 summarizes the mean values of both sampled populations, where both tests reveal a clear difference between both. Based on these univariate results, the further approach is to check the likelihood of the observations, given each distribution. As demonstrated at the top of Figure 16, one can represent both distributions. As a secondary illustration, the distributions of 2 features (‘dur’, and ‘smean’) are compared for both groups (Figure 17). Here, some distributions (e.g., dur) overlap, while others differ clearly (e.g., *smean*).

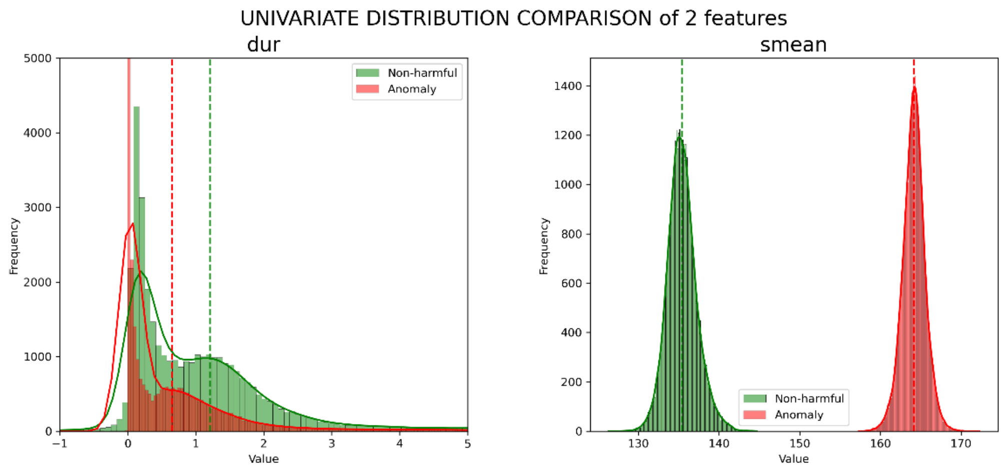


Figure 17: Differences between Non-harmful and Anomaly distributions as a univariate comparison.

3.3.5 Confusion matrix of the model

Similar to logistic regression classification, the utilization of a confusion matrix is prevalent in the context of multivariate distributions. As elucidated earlier, classifying incoming data hinges on assessing the probability of its association with a particular class. To determine the class assignment of observation, it is necessary to approximate the likelihood as a multivariate joint distribution of 10 features. The highest likelihood among these features dictates class membership.

```
''' calculate mean and var vectors, only the last chain is kept (lag)
vectors are referred for both 'non-harmful' as 'anomaly' groups'''

N_MEAN[-lag:]
A_MEAN[-lag:]

N_VAR[-lag:]
A_VAR[-lag:]

''' represent two multivariate normal distributions (MVN) with means and
variance on the diagonal'''

N_mvn = dists.MvNormal(loc=N_MEAN[-lag], cov=np.diag(np.abs(N_VAR[-lag])))
A_mvn = dists.MvNormal(loc=A_MEAN[-lag], cov=np.diag(np.abs(A_VAR[-lag])))

''' By utilizing the natural logarithm (base e) of the probability density
function evaluated at x, predictions are made. Employing logarithms
simplifies the computational process. '''

Prediction = [0 if N_mvn.logpdf(x) > A_mvn.logpdf(x)
                else 1 for _, x in enumerate(df_full[index])]

Ncount, Acount = Prediction.count(0), Prediction.count(1)
```

The algorithm above results in a prediction or classification between one of the two groups. Every observation of network features (x-vector) will belong to one of the Multivariate distributions according to their likelihood.

Also, here, some model tuning can assist towards better accuracy.

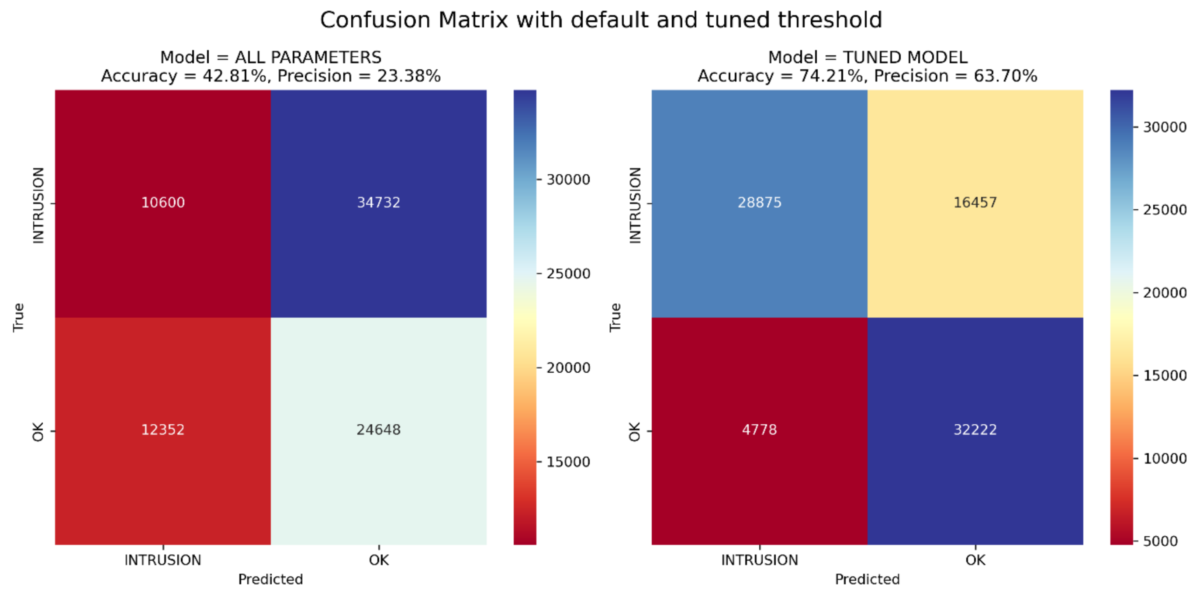


Figure 18: Confusion matrix results with all 10 features model, and tuned for a minimum precision.

A low accuracy is obtained with the full model and a very bad precision (23.38%). Therefore, a tuning loop optimizes for a minimum precision, set arbitrarily at 55%, to obtain the highest possible accuracy. Here, all feature combinations are evaluated for performance on these 2 indicators. Mathematically: $\sum_{r=1}^{10} \frac{10!}{r!(10-r)!} = 1022$ combinations are evaluated as potential candidates. In addition, the time lag is optimized, where tuning is done for the last value only, up to the last 10 values of the state space chain. One could incorporate more information into the evaluation by considering multiple recent values instead of just the last one. This additional information can provide a more comprehensive understanding of the system's behaviour and leads to improved performance. This time lag seems to smooth out the noise and better estimate the underlying system (Zuo et al., 2022). After tuning the state space model, the following features are retained for future testing, as indicated in Table 2:

TUNED FEATURES	ACCURACY	PRECISION
DJIT		
RESPONSE_BODY_LEN		
RATE	74,21 %	63,70 %
TIME LAG = 8		

Table 2: Tuned features of the state space model.

3.3.6 Hypothesis testing of the state space model

H_0: *The state space likelihood model is just a random guess (50/50) classification.*

H_A: *The state space likelihood model is **NOT** a random guess (50/50) classification.*

		PREDICTED	
		INTRUSION	*NON-HARMFUL*
TRUE	***INTRUSION***	$Expected_{H_0}$: *22666*	$Expected_{H_0}$: *22666*
		Observed:28875	**Observed: 16457**
	NON-HARMFUL	$Expected_{H_0}$ *:18500*	$Expected_{H_0}$ *:18500*
		Observed:4778	**Observed:32222**

Table 3: Tuned features of the state space model.

This allows us to check statistically the null hypothesis. Mathematically the Chi-square test statistic will be constructed $\chi^2 = \sum_{i=1}^{4} \frac{(O_i - E_i)^2}{E_i}$. Given a 2x2 table and known totals, given 1 variable will reveal the values of the other, therefore having one degree of freedom (df=1).

$$\chi^2_{df=1} = \frac{(28875 - 22666)^2}{22666} + \frac{(16457 - 22666)^2}{22666} + \frac{(4778 - 18500)^2}{18500} + \frac{(32222 - 18500)^2}{18500} = 23758$$

Given this high test statistic ($P < 10^{-8}$), the state space model outperforms the random guess.

4 DISCUSSION

The tuning process identified specific features, such as 'djit', 'response_body_len', and 'rate', that contributed significantly to the model's performance. These features provide valuable information for accurate classification, enabling the model to distinguish between non-harmful and anomaly instances more effectively. The results of the hypothesis testing provided additional evidence of the state space model's superiority compared to random guessing. The statistically significant chi-square test statistic ($P < 10^{-8}$) indicated that the model's classification performance was not attributable to chance alone but rather to its capability of capturing the underlying patterns in the network intrusion data.

In conclusion, the performance evaluation of the state space model for network intrusion classification revealed promising results. The model demonstrated its ability to capture the data dynamics and achieve reasonable accuracy. However, it is essential to note that the maximum precision attained by the model was 63.7%, implying a 36.3% rate of false-negative predictions for anomalies.

This precision is relatively low compared to some other models that can achieve levels exceeding 85%. An overview involving high-precision intrusion detection systems based on multi-scale convolutional neural networks (MSCNN) illustrates those levels (J. Yu et al., 2022). However, it is crucial to consider the trade-off between precision and computational cost. Higher precision models often require more computationally intensive algorithms, which may not be feasible or practical in specific scenarios.

The state space model presented in this study offers a balance between computational efficiency and reasonably accurate predictions. It incorporates multivariate states and observations, considers temporal dependencies through the time lag parameter, and utilizes relevant features to improve classification performance. These features make it suitable for real-time or resource-constrained environments where computational efficiency is a priority.

It is worth noting that network intrusion detection is a challenging task, and achieving high precision while minimizing false negatives remains a significant research area. Future studies could focus on refining the model further by exploring alternative algorithms, advanced feature selection techniques, or incorporating additional contextual information to improve precision without significantly increasing computational costs.

Overall, while the state space model in this study may not achieve the precision levels of some higher-computational-cost models, it provides a useful approach for network intrusion detection that balances accuracy and computational efficiency. Furthermore, ongoing research and development can enhance its performance to achieve higher precision rates while considering practical implementation constraints.

5 APPENDIX

For reproducibility and further reference, the code and data source used in this dissertation are made publicly available in a GitHub repository :
https://github.com/stijnhuysman/DissertationData/tree/main

More specifically, the KDD99 dataset is available as *Test_data.csv* and *Train_data.csv.* In addition, the set of the UNSW-NB15 dataset is available as *UNSW_NB15_training-set.csv.*
All code is written in Python and presented as 2 Jupyter Notebooks :
SMC LOGISTIC REGRESSION MODEL.ipynb and
SMC STATESPACEMODEL.ipynb

A concise code is given in the sections below

5.1 Necessary Python libraries and modules

```
import pandas as pd
import numpy as  np
from matplotlib import pyplot as plt
from sklearn.feature_selection import VarianceThreshold
from sklearn.metrics import confusion_matrix
import seaborn as sns
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import mannwhitneyu, ttest_ind, multivariate_normal

import particles  # https://github.com/nchopin/particles
from particles import distributions as dists
from particles import resampling as rs
from particles import smc_samplers as ssps
from particles import state_space_models as ssm
from particles. Collectors import Moments

%matplotlib inline
```

5.2 Data preparation

5.2.1 Logistic regression approach

```
train_data_url="https://raw.githubusercontent.com/stijnhuysman/Dissertation
Data/main/Train_data.csv"
dftot = pd.read_csv(train_data_url)
Xtrain = pd.get_dummies(data=dftot.iloc[:,:(-1)] ,
                        drop_first=True,
                        columns=['protocol_type', 'service', 'flag']
                         , dtype=np.uint0)

Xtrain_selected = variance_threshold_selector(Xtrain, 0.8)
VIF_table = calc_vif(Xtrain_selected).sort_values(by=['VIF'],
ascending=False)
VIF_select = VIF_table[(VIF_table['VIF'] <= 5)]
selection = VIF_select.iloc[:,0].values.tolist()
```

5.2.2 State Space Model Approach

```
train_data_url=https://raw.githubusercontent.com/stijnhuysman/DissertationD
ata/main/UNSW_NB15_training-set.csv
dftot = pd.read_csv(train_data_url)
#SPLIT DATA
normal = dftot[dftot.label.eq(0)]
anomaly = dftot[dftot.label.eq(1)]

#CREATE DUMMIES, VARIANCE THRESHOLD, AND VIF SELECT

normal = pd.get_dummies(data=normal.iloc[:,:(-2)] , drop_first=True,
columns=['proto', 'service', 'state'], dtype=np.uint0)
anomaly= pd.get_dummies(data=anomaly.iloc[:,:(-2)] , drop_first=True,
columns=['proto', 'service', 'state'], dtype=np.uint0)

normal_selected = variance_threshold_selector(normal)
anomaly_selected = variance_threshold_selector(anomaly)

VIF_select_normal  = VIF_table_normal[(VIF_table_normal['VIF'] <= 5)]
VIF_select_normal  = VIF_table_anomaly[(VIF_table_anomaly['VIF'] <= 5)]
```

5.3 Logistic regression

```
# PRIOR AND MODEL
scales = 0.8 * np.ones(p)     # for all column beta scales  of 5 = >variance
scales[0] = 20  # intercept has a larger scale   #the beta_0, of course, is
larger, we adjust it to start from

# AMOUNT OF RUNS OF THE SAMPLER
nruns = 1

# PRIOR
'''Define a prior with multivariate Normal distro'''

prior=dists.StructDist({'beta':dists.MvNormal(scale=scales,cov=np.eye(p))})

''' The prior is nothing more than an MVN distribution with an
eye cov matrix'''

#BAYESIAN MODEL: LIKELIHOOD X PRIOR
class LogisticRegression(ssps.StaticModel):  #build  a SMC sampler LR class
    def logpyt(self, theta, t):
        # loglikelihood factor t, for given theta
        lin = np.matmul(theta['beta'], data[t, :])

''' Matrix product of two arrays.  => The log model takes all X-es and
multiplies with the beta matrix (that is a multivariate normal of
dimension'''

        return - np.logaddexp(0., -lin)

''' the numpy.logaddexp() function is used to calculate the logarithm of
the sum of exponentiations of the inputs. This function is useful in
statistics where the calculated probabilities of events may be so small as
to exceed the range of normal floating-point numbers. In such cases, the
logarithm of the calculated probability is stored. This function allows
adding probabilities stored in such a fashion.'''
```

```
fullmodel = LogisticRegression(data=dataset[:], prior=prior)  #the model is
the base for the FK element

fk = [ssps.IBIS(model=fullmodel, len_chain=10), ssps.IBIS(model=fullmodel,
len_chain=50)]

MASTER = particles.multiSMC(nruns=nruns, #A sampling runs
                            nprocs=0,    #MULTIPROCESSING POWER
                            fk=fk,       #B FK objects
                            N=[100,300],  #C N options
                            collect=[Moments],
                            verbose=False,
                            resampling = 'systematic')#D resampling options
```

5.4 State Space Models

```
  ‵′′ The Hidden Markov model defines multivariate states that proceed  with
RANDOM noise according to a multivariate Normal Distribution of the
preceding state and noise

THE MEASUREMENTS ARE SIMULATED from their corresponding State and noise.′′′

class MULTIVARIATE(ssm.StateSpaceModel):

    default_params = { 'rho': 1, 'nvars' : 1,  'start' : np.zeros(1)}

    def PX0(self):  # The law of X_0   MVN around start
        return dists.MvNormal(loc=self.start ,
                                    cov=np.eye(self.nvars)/np.sqrt(1-self.rho**2))

    ‵′′ The Hidden Markov model defines multivariate states that proceed
with RANDOM noise according to a multivariate Normal Distribution of the
preceding state and noise′′′

    def PX(self, t, xp):  # The law of X_t conditional on X_{t-1}
        return dists.MvNormal(loc=self.start +self.rho*(xp - self.start),
                                      cov=np.eye(self.nvars))

    def PY(self, t, xp, x):  # the law of Y_t given X_t and X_{t-1}
        return dists.MvNormal(loc=x,
                                      scale = np.abs(x),
                                      cov=np.eye(self.nvars))
```

```
N_SSM1 = MULTIVARIATE(rho = 0.9, covY=np.eye(NumberVars),
                       nvars= NumberVars, start = mean_NORMAL)

A_SSM1 = MULTIVARIATE(rho = 0.9, covY= np.eye(NumberVars),
                       nvars= NumberVars, start = mean_ANOMALY)

N_fk1 =ssm.Bootstrap(ssm=N_SSM1, data=NORMAL.tolist())  #identity cov 100 %
A_fk1 =ssm.Bootstrap(ssm=A_SSM1, data=ANOMALY.tolist()) #identity cov 100 %

#NETWORK DATA WITH NORMAL CHARACTER
RESULTS= particles.multiSMC(fk=[ N_fk1, A_fk1], nruns=1,
                            N=[1000], nprocs=0,#Sampling 1000 particles
                            store_history = False,ESSrmin=0.5,
                            collect=[Moments],resampling = ['systematic'])
```

5.5 Classification

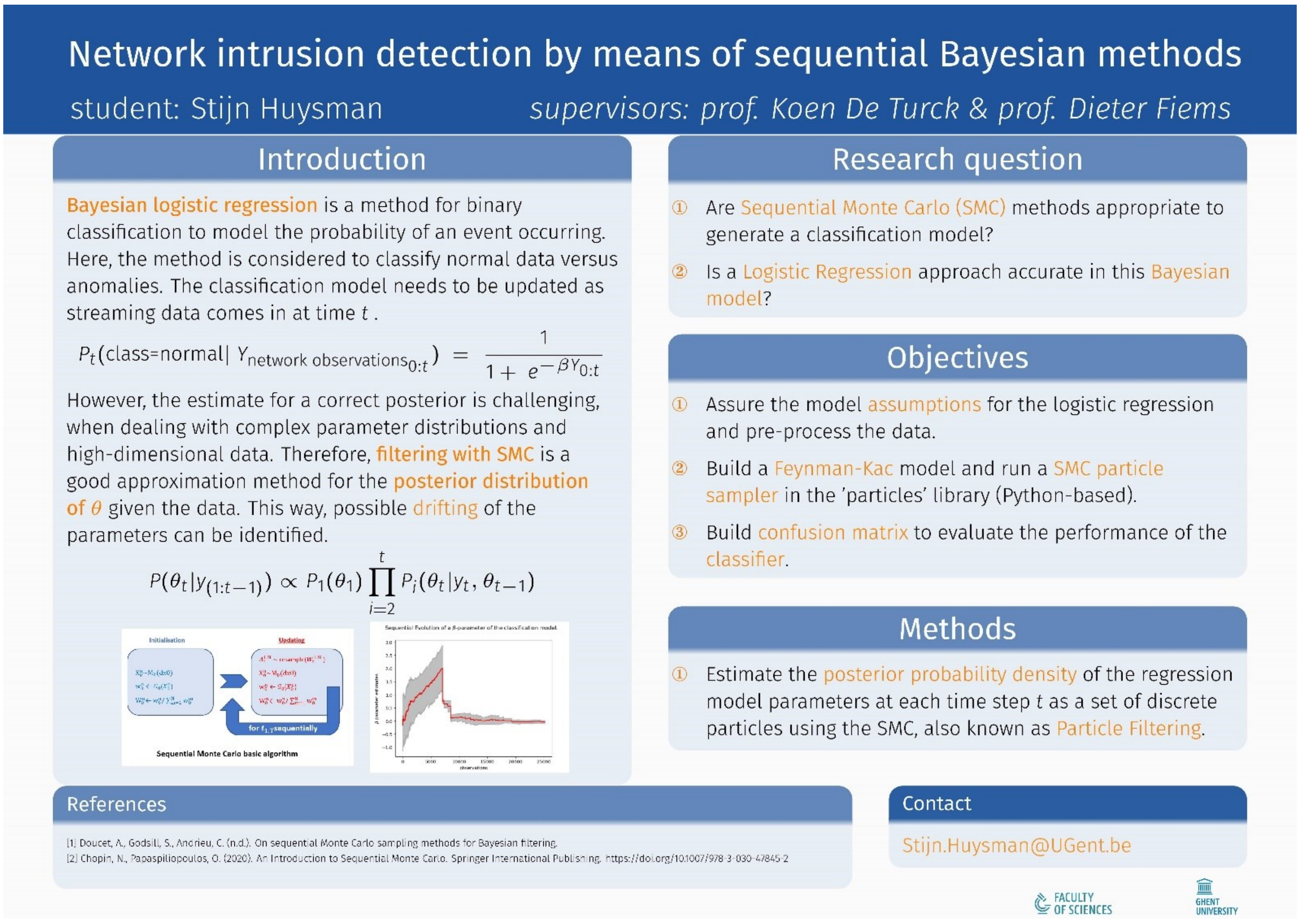
5.5.1 Logistic regression approach

```
betameans_set = SUMMARY[i][2]

# TRUE VALUES
Y_orig = [0 if y == -1 else 1 for y in Y][a:]
labels = ['INTRUSION', 'OK']


fig, axs = plt.subplots(1, 2, figsize=(12, 6))
# PLOT 2 CONFUSION MATRICES. DEFAULT 0.5 AND 0.77 TUNED VALUE
for j, param in enumerate([0.5,  0.77]):
    regressionterm_components = np.array(betameans_set[a:]) * X[a:]
    regressionterm = [np.sum(comp) for comp in regressionterm_components]
    Y_pred = [1 / (1 + np.exp(-rt)) for rt in regressionterm]
    Y_est = [1 if y > param else 0 for y in Y_pred]
    conf_mat = confusion_matrix(Y_orig, Y_est)
    TP, FP, FN, TN = conf_mat.ravel()
    Accuracy = (TP + TN) / (TP + TN + FP + FN)
    Precision = TP / (TP + FP)

    # create a heatmap with labels
    ax = sns.heatmap(conf_mat, annot=True, cmap='RdYlBu', fmt='g',
        xticklabels=labels, yticklabels=labels, ax=axs[j])
    ax.set_xlabel('Predicted')
    ax.set_ylabel('True')
    ax.set_title('Treshold = {:.2f}\nAccuracy = {:.2f}%, Precision = {:.2f}%'.format(param,
                 Accuracy * 100, Precision * 100))
    fig.suptitle('Confusion Matrix with default and tuned Treshold ', fontsize=16, y=1.01)
plt.show()
```

5.5.2 State Space Model Approach

```
def predict(N_moments, A_moments,index, lag, df_full):
    # create a multivariate normal distribution object
    N_mvn = dists.MvNormal(loc=N_moments['means'],cov=np.diag(np.abs(N_moments['var'])))
    A_mvn = dists.MvNormal(loc= A_moments['means'], cov=np.diag(np.abs(A_moments['var'])))


    Ncount, Acount = 0
    Prediction = []
    for i in range(len(df_full)):
        x = df_full[i][index]


        N_likelihood = N_mvn.logpdf(x)
        A_likelihood = A_mvn.logpdf(x)


        if (N_likelihood > A_likelihood):
            Ncount += 1
            Prediction.append(0)
        else:
            Acount += 1
            Prediction.append(1)


    return Prediction
```

5.6 Poster



Network intrusion detection by means of sequential Bayesian methods

student: Stijn Huysman   *supervisors: prof. Koen De Turck & prof. Dieter Fiems*

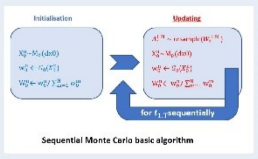
Introduction

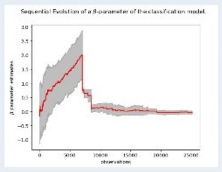
Bayesian logistic regression is a method for binary classification to model the probability of an event occurring. Here, the method is considered to classify normal data versus anomalies. The classification model needs to be updated as streaming data comes in at time t .

$$P_t(\text{class=normal}\mid Y_{\text{network observations}_{0:t}}) = \frac{1}{1+\ e^{-\beta Y_{0:t}}}$$

However, the estimate for a correct posterior is challenging, when dealing with complex parameter distributions and high-dimensional data. Therefore, filtering with SMC is a good approximation method for the posterior distribution of θ given the data. This way, possible drifting of the parameters can be identified.

$$P(\theta_t|y_{(1:t-1)}) \propto P_1(\theta_1)\prod_{i=2}^{t} P_i(\theta_t|y_t, \theta_{t-1})$$


Initialisation
$X_0^n \sim M_0(dx0)$
$w_0^n \leftarrow G_0(X_0^n)$
$W_0^n \leftarrow w_0^n / \sum_{m=1}^{N} w_0^m$
Updating
$A_t^{1:N} \sim \text{resample}(W_t^{1:N})$
$X_0^n \sim M_0(dx0)$
$w_0^n \leftarrow G_0(X_0^n)$
$W_0^n \leftarrow w_0^n / \sum_{m=1}^{N} w_0^m$
for $t_{1:T}$ sequentially
Sequential Monte Carlo basic algorithm



Research question

① Are Sequential Monte Carlo (SMC) methods appropriate to generate a classification model?

② Is a Logistic Regression approach accurate in this Bayesian model?

Objectives

① Assure the model assumptions for the logistic regression and pre-process the data.

② Build a Feynman-Kac model and run a SMC particle sampler in the 'particles' library (Python-based).

③ Build confusion matrix to evaluate the performance of the classifier.

Methods

① Estimate the posterior probability density of the regression model parameters at each time step t as a set of discrete particles using the SMC, also known as Particle Filtering.

References

[1] Doucet, A., Godsill, S., Andrieu, C. (n.d.). On sequential Monte Carlo sampling methods for Bayesian filtering.
[2] Chopin, N., Papaspiliopoulos, O. (2020). An Introduction to Sequential Monte Carlo. Springer International Publishing. https://doi.org/10.1007/978-3-030-47845-2

Contact

Stijn.Huysman@UGent.be

FACULTY OF SCIENCES   GHENT UNIVERSITY

REFERENCES

Bonaccorso, G. (2020). Mastering Machine Learning Algorithms: Expert techniques for implementing popular machine learning algorithms, fine-tuning your models, and understanding how they work, 2nd Edition. Packt Publishing Ltd.

Byeon, E. (2020, December 14). Speed Testing Pandas vs. Numpy. Medium. https://towardsdatascience.com/speed-testing-pandas-vs-numpy-ffbf80070ee7

Centre International de Rencontres Mathématiques (Director). (2018, November 15). Nicolas Chopin: An introduction to particle filters. https://www.youtube.com/watch?v=mE_PJ9ASc8Y

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating Large Language Models Trained on Code (arXiv:2107.03374). arXiv. https://doi.org/10.48550/arXiv.2107.03374

Chen, Z. (2003). Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond. Statistics, 182. https://doi.org/10.1080/02331880309257

Chopin, N. (2002). A Sequential Particle Filter Method for Static Models. Biometrika, 89(3), 539–551.

Chopin, N., & Papaspiliopoulos, O. (2020). An Introduction to Sequential Monte Carlo. Springer International Publishing. https://doi.org/10.1007/978-3-030-47845-2

Del Moral, P., Doucet, A., & Jasra, A. (2006). Sequential Monte Carlo Samplers. Journal of the Royal Statistical Society Series B: Statistical Methodology, 68(3), 411–436. https://doi.org/10.1111/j.1467-9868.2006.00553.x

Dupuis, D., & Victoria-Feser, M.-P. (2013). Robust VIF regression with application to variable selection in large data sets. The Annals of Applied Statistics, 7. https://doi.org/10.1214/12-AOAS584

Howell, D. C. (2011). Chi-Square Test: Analysis of Contingency Tables. In M. Lovric (Ed.), International Encyclopedia of Statistical Science (pp. 250–252). Springer. https://doi.org/10.1007/978-3-642-04898-2_174

Khan, A. A., Beg, O. A., Alamaniotis, M., & Ahmed, S. (2021). Intelligent anomaly identification in cyber-physical inverter-based systems. Electric Power Systems Research, 193, 107024. https://doi.org/10.1016/j.epsr.2021.107024

Kuptametee, C., & Aunsri, N. (2022). A review of resampling techniques in particle filtering framework. Measurement, 193, 110836. https://doi.org/10.1016/j.measurement.2022.110836

Moustafa, N., & Slay, J. (2015a). The Significant Features of the UNSW-NB15 and the KDD99 Data Sets for Network Intrusion Detection Systems. 2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 25–31. https://doi.org/10.1109/BADGERS.2015.014

Moustafa, N., & Slay, J. (2015b). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). 2015 Military Communications and Information Systems Conference (MilCIS), 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

Nchopin/particles. (n.d.). GitHub. Retrieved 11 April 2023, from https://github.com/nchopin/particles

Network Intrusion Detection. (n.d.). Retrieved 11 April 2023, from https://www.kaggle.com/datasets/sampadab17/network-intrusion-detection

Nguyen, T. L. T., Septier, F., Peters, G. W., & Delignon, Y. (2015). Efficient Sequential Monte-Carlo Samplers for Bayesian Inference (arXiv:1504.05753). arXiv. http://arxiv.org/abs/1504.05753

Ondeng, T. (2017). TCP/IP Technology.

Özgür, A., & Erdem, H. (2016). A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015 [Preprint]. PeerJ Preprints. https://doi.org/10.7287/peerj.preprints.1954v1

Ravipati, R. D., & Abualkibash, M. (2019). Intrusion Detection System Classification Using Different Machine Learning Algorithms on KDD-99 and NSL-KDD Datasets—A Review Paper (SSRN Scholarly Paper No. 3428211). https://doi.org/10.2139/ssrn.3428211

Rosner, B., Glynn, R. J., & Lee, M.-L. T. (2006). The Wilcoxon signed rank test for paired comparisons of clustered data. Biometrics, 62(1), 185–192. https://doi.org/10.1111/j.1541-0420.2005.00389.x

Sohil, F., Sohali, M. U., & Shabbir, J. (2022). An introduction to statistical learning with applications in R. Statistical Theory and Related Fields, 6(1), 87–87. https://doi.org/10.1080/24754269.2021.1980261

Sperandei, S. (2014). Understanding logistic regression analysis. Biochemia Medica, 24(1), 12–18. https://doi.org/10.11613/BM.2014.003

Stephanie. (2021, April 20). Log Likelihood Function. Statistics How To. https://www.statisticshowto.com/log-likelihood-function/

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, 1–6. https://doi.org/10.1109/CISDA.2009.5356528

The History of Computer Networks Information Technology Essay. (n.d.). Retrieved 13 December 2022, from https://www.ukessays.com/essays/information-technology/the-history-of-computer-networks-information-technology-essay.php

Tsymbal, A. (n.d.). The problem of concept drift: Definitions and related work.

Wang, H., & Abraham, Z. (2015). Concept Drift Detection for Streaming Data (arXiv:1504.01044; Version 2). arXiv. http://arxiv.org/abs/1504.01044

What is a Network Attack? (2021, February 17). Forcepoint. https://www.forcepoint.com/cyber-edu/network-attack

Wojciech, J. (n.d.). Efficient Monte Carlo Methods for Light Transport in Scattering Media. Wojciech Jarosz. Retrieved 11 April 2023, from https://cs.dartmouth.edu/~wjarosz/publications/dissertation/

Yu, J., Ye, X., & Li, H. (2022). A high-precision intrusion detection system for network security communication based on a multi-scale convolutional neural network. Future Generation Computer Systems, 129, 399–406. https://doi.org/10.1016/j.future.2021.10.018

Yu, T., Lu, L., & Li, J. (2019). A weight-bounded importance sampling method for variance reduction (arXiv:1811.09436). arXiv. https://doi.org/10.48550/arXiv.1811.09436

Zach. (2020, October 13). The 6 Assumptions of Logistic Regression (With Examples). Statology. https://www.statology.org/assumptions-of-logistic-regression/

Zuo, S., Liu, X., Jiao, J., Charles, D., Manavoglu, E., Zhao, T., & Gao, J. (2022). Efficient Long Sequence Modeling via State Space Augmented Transformer (arXiv:2212.08136). arXiv. https://doi.org/10.48550/arXiv.2212.08136