Advanced Wireless Network Activity Inference

Monitoring wireless communication meta data.

Christian Hilligsøe Toft

Mathematical Engineering

Fall 2019 – Spring 2020

Masters Thesis

Mathematical Engineering
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Advanced Wireless Network Activity Inference +

Theme:
Data analysis

Project Period:
Fall 2019 – Spring 2020

Project Group:
MATTEK

Participant(s):
Christian Hilligsøe Toft

Supervisor(s):
Thomas Arildsen
Petar Popovski
Rasmus Waagepetersen

Copies:

Page Numbers: 90

Date of Completion:
September 17, 2020

Abstract:

This thesis investigate if applications can be identified from publicly avialable meta data, from a Wi-Fi network.

The classification was done through clustering by a Gaussian mixture model – on data with restricted labelling. The data was captured at Aalborg University by passively monitoring Wi-Fi traffic. By utilising three different preprocessing methods – Simple standardisation, transformation through Factor analysis of mixed data and a trained Gated recurrent unit - autoencoder, encoding – features of the data is transformed for the clustering.

Even with the used preprocessing methods, the clustering were found no better than random guessing for identifying applications for the captured data labelling combination.

Contents

Preface **vii**

1 Introduction **1**
1.1 Problem Analysis – Data leakage by use of Wi-Fi 1

2 Prileminary **5**
2.1 Preliminary results and data 7

3 Data collection and labelling **17**

4 Preprocessing and simple clustering **23**
4.1 Data of interest 23
4.2 Data point considerations and limitation 30

5 Autoencoder and neural networks. **33**
5.1 Recurrent neural networks and Gated Recurrent Units 34
5.2 Autoencoder 37

6 Training of autoencoder and backpropagation through time **43**
6.1 Backpropagation through time 43
6.2 Training and the gradient step 47

7 Preliminary data processing results **49**
7.1 FAMD – Inertia analysis 49
7.2 Model validation and Training results 52

8 Results **57**
8.1 Clustering results 59

9 Discussion **67**
9.1 Discussion – results 67
9.2 General Discussion 68

10 Conclusion **71**

11 Future studies **73**

Bibliography **75**

A Abbreviations and notation **79**

B Backpropagation through time – First time step. **83**

C Visualised confusion matrix – models 0 through 5 **85**

D Code Overview **89**

Preface

This masters thesis is written by Christian Hilligsøe Toft for the education of Mathematical Engineering at Aalborg university, spring 2020. The main theme of this project is data analysis of Wi-Fi signals and what can be inferred from passively monitoring Wi-Fi signals/a Wi-Fi connection.

Throughout this thesis when a term is introduced it will be written in italic, and some of the terms will be abbreviated. If this is the case the abbreviation will be written in parenthesis following the term e.i. *Medium Access Control* (MAC). Furthermore, appendix A contains all abbreviated terms used, their abbreviation along with a short description.

The citation follows the IEEE-style, with the bibliography ordered after their first appearance and citations written as [citation number, page number]. Multiple citations without specific page numbers is shortened if possible with a dash, i.e. [5],[6],[7],[8] becomes [5]-[8]. When referring to sections, figures and table the associated number for the current chapter is used e.g. Section [number]. For equations only the equation number is written in a parenthesis.

Aalborg University, September 17, 2020

1 | Introduction

Data analysis can be a powerful tool. The increase in computational powers in the previous decades have enabled analysis of large and larger data sets. Due to this development, tools such as machine learning and especially neural networks have been of interest. This increase in interest have enables analysis to be performed in new ways, with new data, gaining valuable information from data or improving quality of life for users. This rapid development also raises questions to the use and availability of personal data, as well as the steps needed to ensure privacy.
This chapter will introduce the reasoning behind this project, as well as a literature review for a better understanding of the problem at hand. Furthermore, the scope of the project will be outlined by a problem statement. The chapter will conclude with the delimitation for this projects.

1.1 Problem Analysis – Data leakage by use of Wi-Fi

Wireless devices is for many an everyday commodity. In Denmark it is estimated that 83% of the population[1] accesses the internet daily using mobile-/smartphones [1].
Smart devices such as watches, tv and home operations, general *Internet of things* (IoT)[2], is an increasing trend with the number of IoT devices forecasted to increase [3], [4]. Wireless connections enables all of these different devices to access the internet. This leads to a large amount of wireless communication occurring at all times.

The technological advancement has let to wireless networks being ubiquitous. Research have shown that multiple elements can be inferred by passively monitoring Wi-Fi connection. Some mobile stations can perform a probe request in order to automatically connect to known Wi-Fi connections, and to detect when a network is available. By periodically broadcasting a probe message on all available channels, a connection can be established between a *station* (STA) and an *access point* (AP). [5]–[8]
Especially can probe request from mobile station be used for tracking the location of these mobile stations. This can as an example be used for gaining information of public transport by tracking the number of user and their movement, though it could more nefariously be used for tracking a specific target giving enough information. [7], [8]

[1] Between the age of 16-74, survey conducted for Danmarks Statistik 2017

In this masters thesis it will be investigated what other information can be gained by monitoring a Wi-Fi Connection. To answer this, meta data from a Wi-Fi connection will be investigated and machine learning methods for data analysis will be utilised.

A Wi-Fi connection is restricted on the *Physical layer* (PHY), e.g. the frequencies for which a transmission can occur and the physical transmission of the signal from attenuation and interference. Furthermore, multiple stations might want to transmit on the same wireless medium, further restricting wireless communication. This restriction of wireless communication is handle by a Medium Access Control layer. The *Medium Access Control* (MAC) will attempt to fairly distribute the access to the shared wireless medium. In order to distinguish the different stations transmitting and control the transmission environment MAC address are utilized, uniquely distinguishing STAs and APs. [5, p. 246], [9]
On the MAC layer data are packed into frames[2] each split into three basic components. Per IEEE-802.11 these subsets are a MAC header comprising of control information, a frame body of variable length containing the payload and a frame check sequence, by a cyclic redundancy code validation. [5, p. 636].

The payload frame may be encrypted, as such no information is expected to be gained by monitoring the frame body. Instead all the information is to be gained from the un-encrypted MAC header, in addition to the information that can be gained from receiving the radio transmissions. As the information from a STA is broadcasted by radio waves, placing a receiver in close proximity to the AP will enable the observation of communication, without interfering with the connection. From the network point of view, nothing observable is going on in this type of setup and the "intruder" can freely monitor the connection. As the AP is the gateway to a distribution service, monitoring information to and from the AP, possible within a close proximity, will give a full view of the Wi-Fi connections to the distribution service, through that one access point. The PHY aspects and transmissions architecture will be further elaborated in chapter 3.

To ensure quality and user experience some application have requirements that needs to be fulfilled. This is especially the case when applications are required to run in real time, e.g. as with *Voice over IP* (VoIP) or other live streaming services. VoIP is restricted, as delays and high latency reduce the user experience, with it inherently being a two way communication.
The following is a theoretical example of how some of the unencrypted information could possibly be used for application identification. Some simple information that can be gained from the meta data is when a device receives and transmits information. Another information that can be gained is how large a packet received is. By

[2] MAC frames

using these two elements in conjunction, a very rough estimate of the required data rate for all running applications on a STA can be estimated.

The idea is simple, different application will have different behaviours. If this behaviours is possible to exploit, then a register of application could be created and used in order to identify the behaviour of users. Furthermore, if the only information used to create this register is "publicly" available then anyone could monitor or be monitored.

1.1.1 Problem statement

As explained in the problem analysis, the MAC header and other meta data such as transmission and size, is readily available to anyone in close proximity to the connection. This thesis will investigate what can be inferred by this meta data or more concisely:

- **Can applications be inferred by monitoring a wireless connection?**

In order to answer the problem statement the following is addressed in this thesis.

- The content of a MAC header is investigation through the IEEE Wi-Fi 802.11 protocol.
- Data collection for access points at Aalborg university campus.
- Data analysis is performed – Taking the information from the MAC header and meta data, un-supervised machine learning methods is investigated and utilised.

These elements will then give an insight into the information gained from the meta data.

1.1.2 Limitations on the inferred information

The following limitations restrict the outcome of this project.

All the information will be gained through Wi-Fi connection, as such any device without Wi-Fi or with Wi-Fi disabled will not be recognized and aid in information acquiring. The main goal of the project is to investigate what information can be achieved by passively monitoring a Wi-Fi connection, without physical interference and previous knowledge.

The monitoring will occur at access points at Aalborg University, as such the data will represent a mixture of students and personal from the university. This will restrict the expected data but also the hours in which data can/will be observed, and

under which conditions the data is required (personal vs professional use). A private network is expected to yield greater information about a specific user though a less specific network can be seen as a proof of concept and investigate what information can be gained from a multitude of users. The number of smart devices is expected to be reduced in such an environment, compared to a home network, as such this analysis will lack any information that can be gained by such devices[10].

No information from the frame body will be taken into consideration, as it can be expected to be encrypted and as such unavailable.

This thesis is finalised during the 2020 covid19 pandemic and consequently shutdown of both university and library utilities in Denmark, limiting the possibilities for data collection. Due to the covid19 pandemic a large part of the data, including all the labelled data, was captured during lockdown severely limiting the different devices and users. With the limited access to the university the only devices active, during these captures, are devices for which their full behaviour is known and labelled. This contradicts with the original scope of the thesis to passively investigate traffic in an office environment – even though the devices aren't any different, besides being less varied, the networks conditions are. Furthermore, the applications running and the exact behaviour of the active application are for the labelled data, is not as natural as would be expected for a real scenario. This is due to the fact that all the activity for the labelled data, is manually controlled and labelled, as well as having it be representative of the different expected scenarios for the captured data – The reasoning behind the labelling is . As no other devices are active on the network, regular conditions such as multiple active users, with their individual tendencies and behaviour, isn't a factor in this captured data.

In other words: Due to the covid19 pandemic some of the data is limited to only a few devices in a controlled environment and can possibly lack the "scope/real life scenario" which capturing in an uncontrolled office space would produce. Furthermore, the capturing device is limited to a single device.

2 | Prileminary

In this chapter related works, and preliminary analysis is introduced.
If for a device, a specific interpacket/size cluster, that indicate a e.g. Skype call, can be found, then it is possible to check if a similar cluster exists with the STA as the source (*Source address* (SA)/ *Transmitter address* (TA)). If a STA is only on the receiving end of a live communication feed, it isn't required to transmit the same amount of information, as would be required for the two way communication. As such VoIP applications enables a second possible clustering for interpacket arrival time/size. This can enable a differentiation between a passively receiving source, from a live feed vs an active participant. If something is live streamed (live event), then up to a few minutes delay might be tolerable for the receiver which could possibly further distinguish the different application.
High audio quality and a low latency are two wanted properties when it comes to VoIP, for this a certain amount of information needs to be transmitted within a certain time frame. If a there is a combination (cluster) of packet length and time that correlates to e.g. Skype, then anyone can know if a STA are using Skype or not, just by being in its proximity.

In both [11] and [12], classification of Skype activity is performed based upon the information gained from packet length and the interarrival time, defined as the time between to consecutive packets. This identification of Skype is from a network management point of view, as even with full access to network information can Skype be hard to identify.

A lot of work have been put into identifying Skype, from a network management point of view [11]–[21]. A common method for this identification, is the utilisation of flows and flow analysis as in [11], [16]–[21]. Research have shown that both supervised and unsupervised machine learning methods can be used to identify Skype and other applications using features from the flow. In [21], Azab et. al describes a flow as a collection of packets all sharing certain properties, such as the source and destination IP, transport layer protocol and ports used. This information requires access/decryption of the packets and are, as aforementioned, not publicly available. Flow analysis is therefore not an option.

Even though [11] also use flow based analysis, Do et al. utilise machine learning methods to classify traffic into Skype, VoIP and Others using a Decision Tree classification. In this paper the authors find that the features of the flow with the highest

effect of for classification is the packet length and the packet inter arrival time with a 99% true positive rate. As both packet lengths and packet inter arrival time is available, this thesis investigate whether classification can occur, without knowing if packets belong to a certain flow.

Methods

Two different machine learning methods can be used to investigate if e.g. a Skype conversation is active at a given point. Supervised and Unsupervised machine learning. Both method have a draw back, that in order to investigate performance, and in the case of the supervised method use them, data points needs to be classified/labelled .

For unsupervised learning clustering methods like K-means and Gaussian Mixture models can be utilised. [22, p. 425] These methods will try to exploit the behaviour/features of the data, in the case of k-means the distance between data points, in order to make clusters of data points. The clusters would then correspond to (possibly unknown) properties of the data. If a cluster exist that specifically corresponds to e.g. Skype, then new data points that also corresponds to Skype should fall within the same cluster. The task would then be to identify the cluster that corresponds to the specific elements. A down side of using unsupervised methods is that if your looking for a specific "cluster" the method might not find it, instead finding other prominent features of the data.

For supervised methods the machine is taught what features belong to what class by labelled data points. The methods will then try to make a decision rule based on the features/labels such that new unknown data points can be classified. Features can also be handcrafted or self taught. If the features are hand crafted then a investigation of the data for the features or combination of features can be performed, this could as an example be performed using a Decision Trees [22, p. 663]. In such methods a combination of smaller decisions or models are used in concession for the final classification. Another option is to have the machine learning method find the features of interest by it self, based upon the labelled data. Neural networks is a popular group of such methods, utilising repeated training on the data, along with hyper parameters, to control and shape the network to achieve specific properties.

In both the supervised and unsupervised method, any new data point can be allocated to the "classes". Depending on the methods, a decision boundary will be created and the algorithm will decide which class any data point belongs to. For some algorithms this is a binary decision, where for other methods a data point is said to belong to a class with a certain probability.

A problem can especially arise for labelling the data. Two possible methods is to be considered.

Firstly label all the data within a time range as e.g. Skype or Other, this can be achieved by logging when a Skype call is active on a computer and then cross referencing this information with the collected data. The downside of using this method is, if the STA in question is utilising the network for other application as well as Skype, the other applications might obscure the Skype information. It should be taken into consideration, that in real world scenarios ideal conditions often aren't meet. If the classification method can handle multiple applications at the same time and still generalise well to new a data sets, it can then handle a broader spectrum of scenarios.

The other methods could be to classify Skype packets/flows and other applications, by using other classification methods with full access, as in [11], [16]–[21], and then only use the features from these packets to construct the classifier. This is expected to be similar results as in the first case if all the STAs are limited to one applications at a time, but with a higher precision . A downside with using this methods is that all the data needs to be captured twice, once with full access and once without. Furthermore, a classifier per each of the data captures is also needed.

2.1 Preliminary results and data

Some information about STAs, such as their coming and going, can easily be gained on a network. If the STA isn't turned of probe request and general activity of the STA will show up in the captured data. As previously mentioned the MAC addresses of the devices is available in the MAC header (Receiver/transmitter – source/destination). A Wi-Fi sniffers can even give a signal strength indication, up two 3 axis x,y,z strength. By using this information spatial information can be gained, in addition to knowing if the STA is present, somewhere in the vicinity of the sniffing device.

The MAC addresses are also a point of interest as they typically aren't randomly generated and assigned to a device. For ease of use, a need for control of the MAC addresses arises. A MAC address is a combination of 6 pairs of hexadecimal numbers and will typically be assigned by the device/hardware manufactures. It is as such not only possible to see if a device enters or leave and gain a bit of spatial information relative to the receiver/Wi-Fi sniffer, the MAC address give information about the type of devices. [5, p. 636]

The MAC address can contain an *organisationally unique identifier* (OUI) [5, p. 169], which, as the name implies, uniquely identifies a organisation (Vendor/manu-

factures). These OUI can then be resolved and the organisation can be identified, i.e. you can know if the STA is an Apple, Samsung or Cisco product etc. For some devices it might not be as evident for which the OUI refers. This is especially the case where components from different manufactures are used to create a single device, e.g. the "same" device can have two different network cards as a possible hardware specification and as such two different OUI.

But is knowing what devices are active a "problem"? Devices will more often than not be tied to a specific person, with enough data if multiple devices belong to the same person, their coming and going can be closer followed.
In conjunction knowing what the different devices are doing, the possibility to track them gives insight into the behaviour of the users.

Captured Data examples

The captured data contains information that can be split into different categories, MAC and Radio transmission. The MAC information is any information that is directly read from the packet data, the MAC header, and the Radio transmission information is any information related to the physical transmission.

Depending upon the specific type of transmission source and destination address aren't always present in a MAC header. Furthermore, the source and destination address pair aren't necessarily identical to the transmitter and receiver address pair[1]. A combination of source and destination information is therefore used to gain the full insight into the behaviour of the STAs

As the information is transmitted via a wireless medium a transmission can fail or the packet can be malformed on arrival. If this is the case the information from the packets isn't reliable. This is even true when just monitoring a wireless network, as information such as sender and receiver can be corrupted. The IEEE-802.11 protocol takes this into account by utilising a frame check sequence as the last part of the transmission [5, p. 636]. If this sequence doesn't corresponds to the received packet, the information from the packet should be considered an outlier. It should be noted that just because a Wi-Fi sniffer receives a malformed packet, the packet could be successfully transmitted between the source and destination STAs.

As mentioned the classification of application can possibly be performed based upon the length of the different packets and their arrival in time. Starting out by simply plotting packets lengths as a function of time, gives insight into when a STA is more

[1] E.g. STA (a) communicating to STA (c) through AP (b) could yield a source address and transmitter address of (a), but a destination and receiver address of (c) and (b) respectively.

or less active.

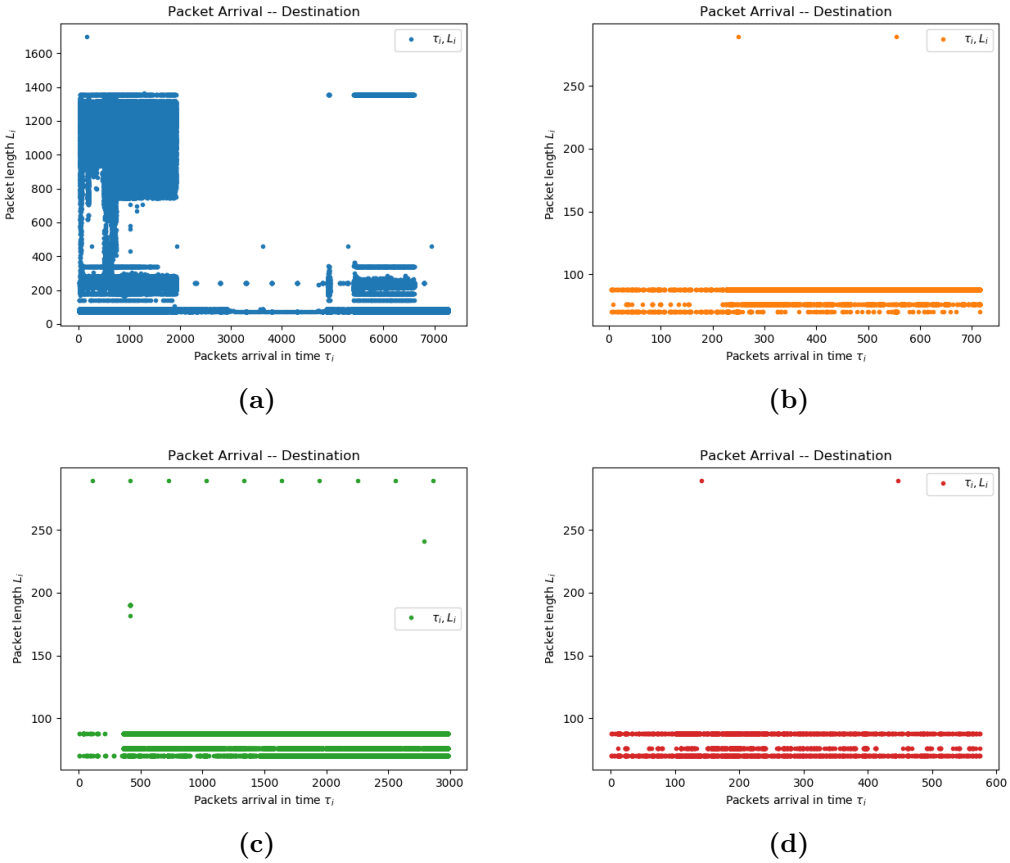


Figure 2.1: Packet length as a function of time for the same STA under different condition.

Figure 2.1 illustrates captured data examples for the same STA under different conditions [2]. For 2.1 the Wi-Fi sniffer, were placed in close proximity to the STA in order to decrease the chance of transmissions errors. Furthermore, the Wi-Fi sniffer were set to only monitor the Wi-Fi network which the STA were using, in order to not jump between channels potentially missing information. After the data were captured it was filtered to only include the information from the specific STA. For figure 2.1, the information is further limited to only the cases for which the STA is the destination (*Destination Address* (DA)/*Receiver address* (RA)) of a packet.

Subfigure 2.1a clearly have activity in the start and the end of the capture. Subfigure 2.1b doesn't have the same activity as 2.1a, with only a minor change in behaviour occurring after around [200] seconds.

[2]Note that the axis aren't normalised and espcially 2.1a has significantly larger values compared to the other subfigures

For subfigure 2.1c the activity corresponds similarly to that for 2.1b with a change in behaviour after around [400] seconds. Subfigure 2.1d shows a more sporadic pattern, for which it could be argued that no clear indication of a change or atleast multiple changes are present.
As these captures were performed under a somewhat controlled environment, the activity of the STA is known for each capture. Starting out by taking a look at the two edge cases. For 2.1a the STA had a Skype conversation active in the begging and the end of the capture, with regular internet use (Web searches, streaming services etc.) through out the capture. For 2.1d the STA was left idle during the entire capture.

That leaves the two somewhat similar looking captures from 2.1b and 2.1c. In both these cases the STA was idle in the start of the capture and then an application was started. Though the two captures look somewhat similar, 2.1c is a live streaming service and 2.1b is actually another Skype call. The applications where started a few minutes after the capture, lining up with the visual change in behaviour. Comparing 2.1a and 2.1b a large difference can be observed. The reasoning behind this difference could possibly be found in the other activity the STA where performing, but a comparison with a second STA for which the first where connected via a Skype call reveals the same behaviour. There are some other differences between the two captures. With the major ones being that for 2.1a both sender and receiver is connected to the same AP and the two captures where performed on for two different APs.

Instead of plotting the different packets arrival in time, packet lengths can instead be plotted as a function of the inter arrival time of the packets. As this is still a two dimensional plot visual inspection can be utilised to gain insight into the behaviour of this domain.

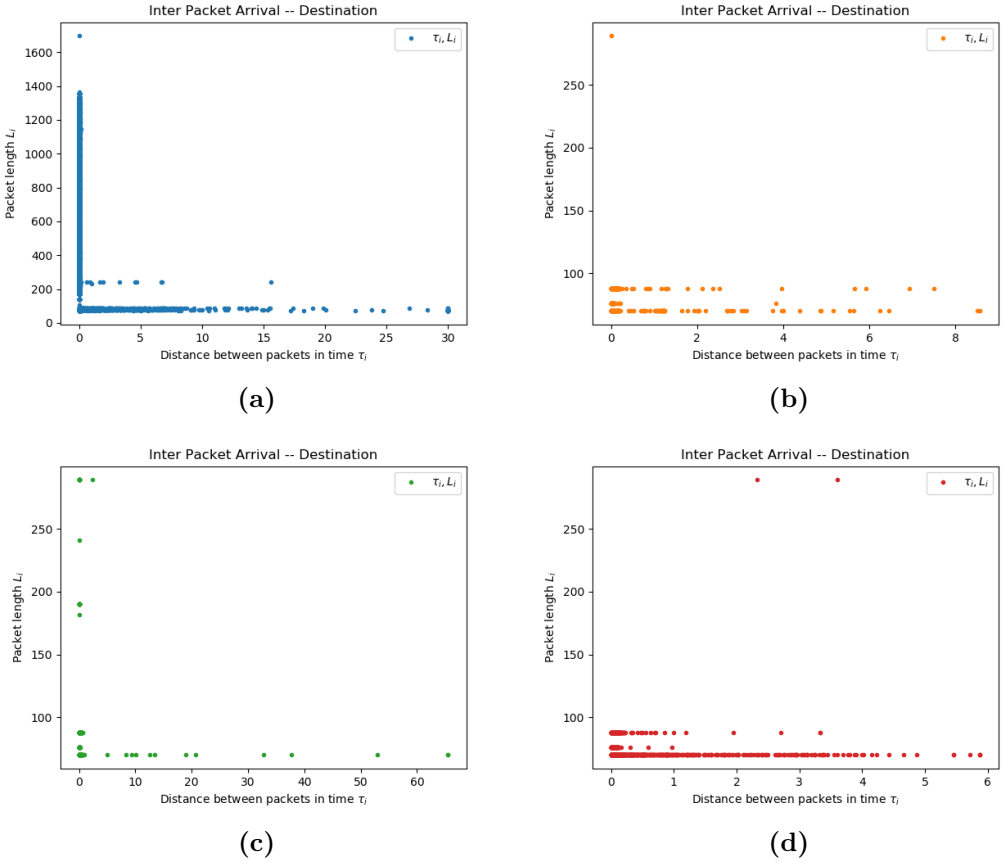


Figure 2.2: Packet length as a function of time since the last packet (Inter arrival time) for the same STA under different condition.

From figure 2.2 clusters can be visually identified. In all of the different captures from figure 2.2 multiple miner clusters can visually be seen. This is especially easy for 2.2b and 2.2c and less so for 2.2a and 2.2d, though as this is a visual assessment it is left to the reader to asses from it what they please[3]. For 2.2b and 2.2c the number of clusters appear to greatly exceed the number the number of applications active, which where either one or none depending on the given time. As such multiple cluster or maybe a combination of clusters. indicate the same application and fewer cluster could be looked for in the data. Another property that needs to be pointed out is the proximity of the cluster to each other. If a distinctions between nearby clusters are needed in order to correctly identify an application, then an increase in unique clusters that should be identified is also needed.

[3]Note again the differences in axis as in the case with 2.1

K-means

The clustering performed on the date example form figure 2.2 is performed by use of K-means clustering. K-means clustering or K-means algorithm is a non-probabilistic technique for finding clusters. The algorithm aim to partition the data set into K clusters, for which distance for points within the cluster is small, compared to distances outside the cluster, i.e. points in close proximity are grouped together. The algorithm can be seen as an optimisation problem. To perform this optimisation the appropriate notation will firstly be introduced. The notation follows that of M. Bishop in [22, p. 424].

Let $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a data set consisting of N observation of a random D-dimensional euclidean variable x. For each cluster introduce a D-dimensional vector $\boldsymbol{\mu}_k$ with $k = 0, \ldots, K-1$, each representing the centres of the K clusters. To distinguish which of the clusters a data point x_n is allocated a set of binary indicator variables, $r_{nk} \in \{0, 1\}$, is introduced. For $r_{nk} \in \{0, 1\}$, $k = 0, \ldots, K-1$ denote the different possible clusters, such that if $\mathbf{x}_n$ is allocated to cluster k then $r_{nk} = 1$, and $r_{nj} = 0$ for $j \neq k$. The goal is then to find the set of cluster centres, $\{\boldsymbol{\mu}_k\}$, and cluster assignments, for each data point $\{r_{nk}\}$, such that the sum of squares of the distances for each data point, x_n, to its associated cluster center is minimized. This can be expressed in the objective function, J,

$$J = \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} r_{nk} ||x_n - \mu_k||^2.$$

The optimisation can be performed through an iterative two step process in which, per iteration, r_{nk} and $\boldsymbol{\mu}_k$ is optimised successively keeping one objective fixed while minimising over the other. [22, p. 424 - 425]

The values for which r_{nk} and $\boldsymbol{\mu}_k$ is optimised can be found to be

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j ||\mathbf{x}_n - \boldsymbol{\mu}_j||^2 \\ 0 & \text{otherwise} \end{cases}$$

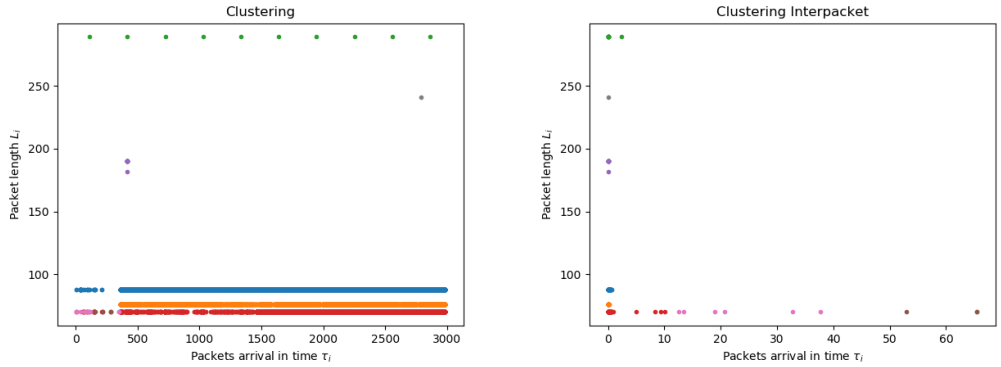
and

$$\boldsymbol{\mu_k} = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}.$$

This corresponds to each data point being associated to the nearest cluster center, in distance, and setting the cluster centres as the mean of values associated to the cluster.

The only parameter that can be tuned when using this method is the number of

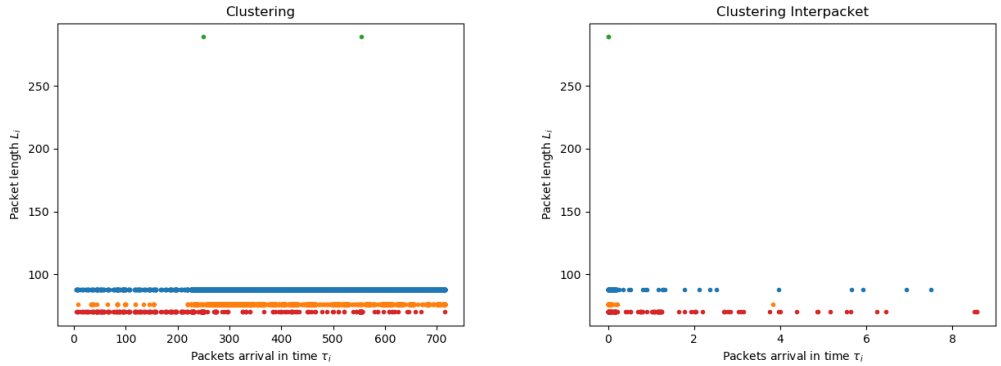
clusters, K. An example of clustering on the 2.2c, is shown on figure 2.3. For this the K-means algorithm is used with an arbitrary choice of $K = 8$. From subfigure 2.3, it can be seen that a cluster, the orange color, correlate to the change in behaviour, and it can as such be identified.



(a) **(b)**

Figure 2.3: Packet as a function of time **(a)** and inter arrival time **(b)**, colorised after clusters $K = 8$. Capture from 2.1c

Sadly this identified cluster wont necessarily uniquely identify the application used, but rather the specific change in behaviour. A way to support this hypothesis would be to use the same cluster centers on similar looking data, which is using another application.



(a) **(b)**

Figure 2.4: Packet as a function of time **(a)** and inter arrival time **(b)**, colorised after clusters $K = 8$. Data displayed corresponds to 2.1b though cluster center are those found from 2.3

In figure 2.4, the cluster centers found from 2.3 is used to find clusters for this case. Here the orange cluster doesn't necessarily indicate the same change in behaviour,

but notice how the different cluster indicate different packet lengths, also evident in 2.3. The found clusters are likely only classifying based upon the packet lengths, with a need for more clusters to uniquely identify different application based upon packet length/inter arrival time if possible.
By using the data as is, the large discrepancy between packet length and inter packet arrival comes to fruition, indicated by mostly finding clusters from packet lengths.

Gaussian mixture model

Besides the use of the simple clustering by k-means, relying on the distance between data points in order to perform the clustering, *Guassian mixture model* (GM) are instead used as a method.

Instead of the distance between observations, GM assumes that the variables are derived from different sources producing Gaussian distributed variables. The goal is thus to estimate the parameters from the different Gaussian distributions. When the distributions are established, the clustering is then performed by investigating the probability of the different variables, x_i, belonging to a specific distribution/cluster, k.

It is possible to find the parameters, θ, and perform the clustering through the iterative *expectation-maximisation* (EM) algorithm. Given some initial parameters, in this thesis found through the k-means algorithm, find the variables that are expected to belong to the different clusters – The expectation step.

In order to calculate the probability of a variable belonging to a cluster, a K-dimensional binary random variable, $\mathbf{z}$ is needed, with K being the total number of clusters. The random variable is limited to only a single non-zero entrance, e.g. the k'th element is equal to 1, with all other elements equal to 0, resulting in K possible different states for such a variable, z_k. By use of such a random variable the conditional probability, $\mathbf{z}$ given data $\mathbf{x}$, is

$$\begin{aligned} p(z_k = 1|\mathbf{x}) &= \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\mathbf{x}|z_j = 1)} \\ &= \frac{\rho_k \mathcal{N}(\mathbf{x_n}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \rho_j \mathcal{N}(\mathbf{x_n}|\mu_j, \Sigma_j)}, \end{aligned}$$

using Bayes theorem, where ρ_k is the prior probability of a variable belong to a cluster and μ/Σ the parameters of a multivariate Gaussian distribution. [22, ch. 9]

With every variable belonging to a clusters, the next step is to update the parameters, $\theta = \{\mu, \Sigma, \rho\}$, maximising the likelihood function, $\mathcal{L}(\mathbf{x}|\theta)$ for all the different

clusters, K, through:

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} p(z_k = 1|\mathbf{x})\mathbf{x_n}$$
$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} p(z_k = 1|\mathbf{x})(\mathbf{x_n} - \mu_k)(\mathbf{x_n} - \mu_k)^T$$
$$\rho_k = \frac{\sum_{n=1}^{N} p(z_k = 1|\mathbf{x})}{N}$$

The log likelihood function can then be evaluated,

$$\ln p(\mathbf{x}|\theta) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \rho_k \mathcal{N}(\mathbf{x_n}|\mu_k, \Sigma_k) \right\}$$

and check it against convergence criterion – Then reiterate. [22, ch. 9]

In other words, the EM algorithm first assign variables to a normal distribution, then updates the distribution such that they fit the variables better. With the updated parameters, the process can then be reiterated, allowing the parameters to gradually update. The update occurs since with new parameters, the variables can fit better with a different, compared to the previous, distribution.

The advantage for utilising GM compared to k-means, is that variance for the variables are taken into consideration, allowing variables to be assigned a cluster though the variable is closer to another clusters center. Though as with k-means, GM require knowledge of the number of clusters/sources producing the data.

Since these, relative simple clustering algorithm are not the main focus of this thesis, further elaboration is omitted. For further detail see [22, ch. 9].

3 | Data collection and labelling

In this chapter the theory relevant to data collection and the prepossessing/data management is introduced. This is done in order to give the necessary insight into the restrictions the data collections have and the considerations that needs to be taken, in order to perform the analysis/inference. The chapter includes relevant theory of the *physical layer* (PHY) of the IEEE 802.11 protocol as well as practical aspects of the data collection, in order to gain a better understanding of how and why the captures are possible and what can be expected from them.

Before the monitoring is performed some setup is required. Passive monitoring of a network with a focus on specific APs is used in this project. The Wi-Fi sniffers are focused on specific APs by limiting their monitoring to channels utilised by the APs. On one hand this is going to limit the possible STAs that can be observed, as any STA not communication through the given channels is outright ignored, on the other hand no information is missed due to channel searching.

The IEEE-802.11 standard introduces multiple *wireless local area network* (wlan) architectures, with the *basic service set* (BSS) as a building block for them all. An infrastructure BSS is a specific extension of a BSS with limited structuring and the main interest for this project. It is possible for infrastructure BSSs to be connected to another BSS/network. A *distribution system* (DS) is the IEEE-802.11 designated term for the architectural component used to interconnect infrastructure BSSs. As an AP is any entity with STA properties and distribution system access function, an AP is needed for STAs to establish a connection to the DS. The infrastructure BSS, with an AP connected to a DS, will have all STAs communicating with the AP, which in turn will relay information to STAs from other STAs and the information from the DS. As such communication on an infrastructure BSS is controlled with the AP as an intermediate node. [9] [5, ch. 4.3]

The Wi-Fis from which the data is collected is from an infrastructure BSS of the extension, *Extended Service Set* (ESS). An ESS consist of multiple infrastructure BSS, where STA can fluently switch from BSS to BSS, by changing which AP they are associated with.
Information transmitted via Wi-Fi uses two main frequency bands, 2.4 GHz and 5 GHz. These frequency bands are then in turn divided into radio channels. APs are free to choose which channel to utilise, though any interference from nearby AP, utilising the same or overlapping channels, can restrict the effectiveness of cer-

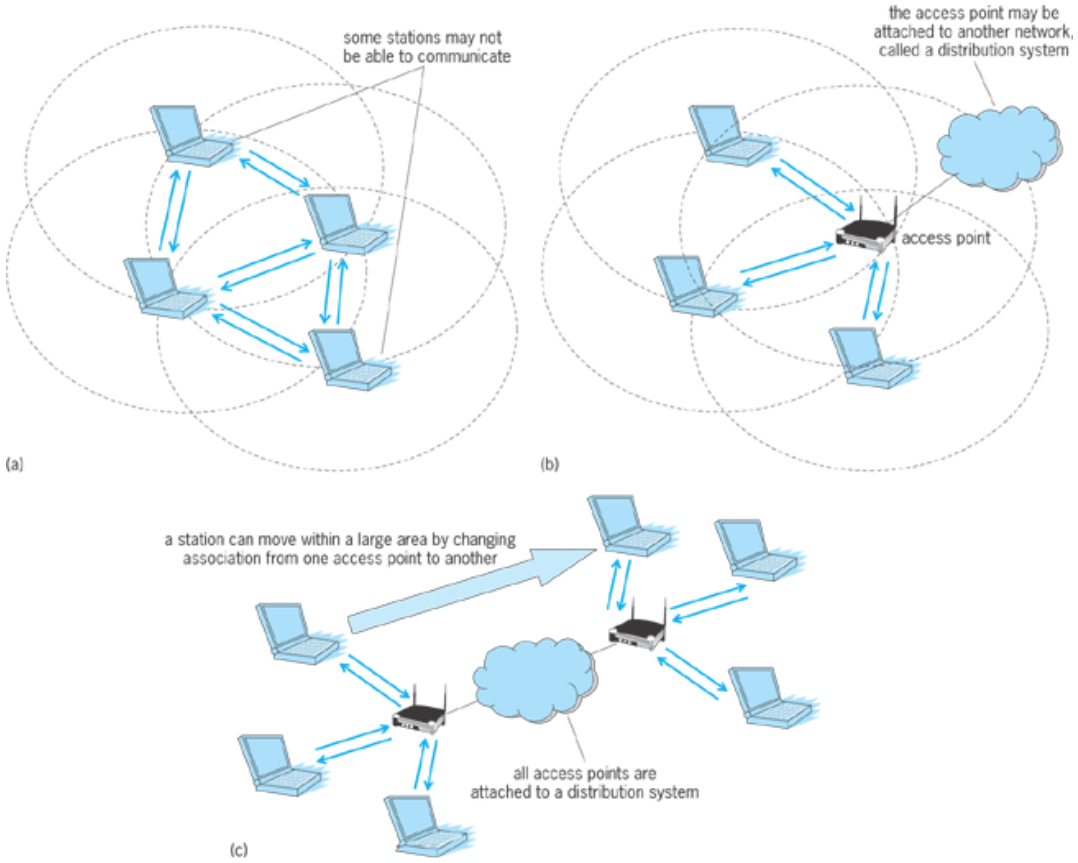


Figure 3.1: Illustration of independent BSS (a), infrastructure BSS (b) and ESS (c) from [9]

tain channels and as such limit possible channels. The available frequencies and the bandwidth of the channels, restricts the number of channels and number of non-overlapping channels – e.g. for 802.11b in the United States 11 channels are available, with only three channels being non-overlapping [9]. APs will try limit communicate on the same channels as other APs and will actively switch channels to avoid interference. [5, Ch. 19, p. 1747]

Multiple channels can be a potential problem for monitoring Wi-Fi Connections. If the device used for the monitoring is restricted to only listening in on a single channel, then transmissions on other channels will be missed. In order to diminish this effect the Wi-Fi sniffer could spend a short amount of time on each channel, capturing information on all available channels.
This channel sweep can be useful when monitoring for activity, but is drastically limiting if a coherent capture is needed. As the discussed methods will be relying on the interpacket arrival time, potentially missing multiple packet from an STA could prevent the detection of the wanted application. To counteract this, a possible solution could be to increase the time the Wi-Fi sniffer stays on each channel, based upon knowledge from classification, at the cost of an increase in time between consecutively visiting the same channel. Unless channels are randomly selected, or some other visiting scheme is used, the time between each revisit is linearly dependent on the number of channels and the time used on each channel

A classifier needs to be created, in order to identify applications. As before mentioned classifiers can be supported by labelled data, if it isn't out right required.
By having access to some/most of the STAs on the network a python script can be utilised to record exact times for when e.g. Skype call is initiated, as the Skype API isn't available. The utilised python script monitors the Skype process and sub-process for changes to their connectivity behaviour. If their aren't any network restrictions, Skype will establish connections between the users through an UDP connection. Checking and logging when this connection is established can give the time for a Skype call. The only found offset by using this methods is that Skype periodically creates a few short UDP connections when idle. These UDP connection would then show up as a very short Skype call and could possibly be classified as outliers, though would still indicate that a Skype call is active.

As the network where the data is recorded isn't restricted in anyway, Skype will follow its default behaviour, how this differs from a network where UDP connections is restricted isn't investigated in this project, and simply limits itself to the default case. Having the python script run on startup on the STAs, while logging Skype activity and the STAs MAC address, enables cross referencing afterwords to the collected data, and hence labelling of it. The behaviour for the Skype is based upon

empirical observations, as the ins and outs of Skype isn't public information.

Using only passive labelling through UDP packets has its limitations. With Skype allowing for both audio and video call, indistinguishable from each other by the aforementioned simple labelling method, it is of interest to utilise another methods. In this thesis any labelling of data is done so manually, in other words, the time and change in application is manually investigated. The manually labelled data can, for the Skype part, be cross-referenced with the UDP activity. With the Covid-19 pandemics shutdown of Aalborg University, during the data capturing for this thesis, the manual labelling is an option as the number of active devices is reduced. This form for time labelling has the added benefit, compared to the simple Skype monitoring, that it keeps track of multiple different application and whether they are active in the same time frame.

The time labelling is performed as follows: The exact time for the start of a an application is noted and every received packets within the given time frame is now belonging to this label. Multiple application within the same time frame will also be investigated with (Streaming and Skype in conjunction etc), though "gaming" is left out as a possible category for labelling.

If a specific network is of interest, listening in on the channels associated with the BSS is useful. By placing a Wi-Fi monitor close to an AP and only listening in on that specific AP, the amount of information correctly received would correspond roughly to that of the AP. It could then give insight into how the specific AP is used, and the behaviour of the associated STAs. As the devices and capturing of data is limited for this project, due to covid19, instead of the more realistic scenario with a Wi-Fi sniffer located near an AP it is kept in close proximity to the active devices. When restricting to specific APs, where STAs have known behaviour and labelling, then a classifiers can be constructed. If a classifier already is available, then a restriction to a specific AP isn't as necessary allowing for channel sweeping. Channel sweeping can be useful for a broader search for STAs using the specific application or whom are actively transmitting.

In conclusion by monitoring an AP, any information, though possibly encrypted, that is transmitted between STAs on that specific BSS or coming from the DS, whom the AP is connected to can be monitored. APs can utilise multiple different channels which needs to be considered when capturing the data.

The collected data is stored in a `.PCAPNG` data fill, a standard often used for network monitoring applications such as Wireshark.

All monitoring devices are set to monitor- and promiscuous mode. These two different modes relays information to the network (802.11) adapter, in the monitoring devices, on how they should handle incoming communication.
Promiscuous mode will tell the network (802.11) adapter to ignore any *MAC address filters* it has, which often will be enabled as standard. A MAC address filter will limit the network (802.11) adapter to ignore/filter out all packets not intended for the MAC address associated with the network interface [23]. For regular use, ignoring the information, not intended for the receiver device, at the network interface is useful, as the system aren't required to deal with this unwanted information more than necessary.

With monitor mode active the network adapter is told to capture and hand over all packets on the selected channels, regardless of *service set identifier* (SSID) [23]. Even if only information from a single SSID is required, a combination of monitor mode and promiscuous mode is be needed, in order to successfully capture all packets on a network. Due to monitor mode capturing of packets at the radio level, not enforced by promiscuous mode, it isn't enough with just one of these settings.

Not all network interfaces card/802.11 adapters in STAs allows for these changes and as such special equipment for Wi-Fi monitoring can be useful. In this project a standard PC that allows for these changes is used. The PC is an Asus A551LN laptop, with a Intel Wireless 7260 network interface card running, Linux – Ubuntu version 18.04. The data collected for this project consist of 26 unlabelled captures, lasting anywhere from 10 min to multiple hours and 14 labelled captures of one hour each.

The capturing was performed with a combination of Wireshark and Aircrack-ng. Though Wireshark should be sufficient, it was found that Aircrack-ng had to be used for this specific capturing setup. Aircrack-ng is utilised in order to set the network interface card into monitor mode as well as directing the capturing to a specific channel. [23]

4 | Preprocessing and simple clustering

As the preliminary analysis showed, using simple clustering on data consisting on inter arrival time for packets and packets size, could result in the loss of the temporal aspect with clusters mostly corresponding different packet sizes. The initial clustering showed little diversity between clusters, in the initial case only showing packet size, loosing much of the temporal information an increase in possible significant features is of interest. In the following section the features of interest is expanded upon including, reasoning for the interest in the features – as an extension to the previous analysis.

4.1 Data of interest

Skype and other application will actively try and adapt their behaviour to accommodate for specific transmission environment, in order to achieve the best possible quality given the circumstances.
If an AP is especially busy, or a transmitting STA is obscured/far away from the AP, the throughput for which the STA can achieve is reduced. The specific modulation, compared to the send packet size, can be changed in relation to the quality of the connection. The transmitting of packets are as such not just a one thing catch all, with aspects such as how different network chip manufacturers handle different situation drastically changing how packets are transmitted. For the 802.11ac and 802.11n amendment a *modulation and coding scheme* (MCS) is utilised. The MCS determines the number of spatial channels, coding, and modulation hence specifying the conditions a transmission occurs, and the amount of information transmitted compared the transmitted packet size. The IEEE-802.11 standard specify index values for the MCS from 0 to 76, concatenating useful network information into a single values. [5, p. 2353]

The access to the two different frequency bands, 2.4 GHz and 5 GHz, is governed by specific IEEE protocols amendment, with the older amendments of 802.11b and 802.11a and the newer 802.11g/n/ac amendments. The amendments directly specify how the network adapters handle throughput and network conditions.
The more information that can be gained from the conditions of the network the better, as these features influence how the applications behave, forming the foundation for the decision making. By monitoring the airtime/total load on the network,

i.e. every broadcast and not just limited to a specific STA, the conditions of the possible throughput could be further gleamed. With only one transmission at a time, restricted by the MAC layer providing fair access to the wireless medium, gaining a sense of the network load through packets in a time period, prior to the packet of interest, is to be monitored.

Another indicator for the condition of the transmission environment, is the transfer rate for packets, the higher the transfer rate the better a connections can be suspected to be, between the STA and AP.

It isn't of interest to know the exact second an application is started, but a rough time period[1] for which the activity occurs is sufficient. This further enables the possibility of averaging information and using this information in the decision making, e.g. by accumulating information from a time periods and then the making the decision based upon this information.
Even though a flow can't be established information such as the average throughput within a certain time period, T, can be used. Though without knowing the flow, an average throughput for a time period could originate from a combination of application and not just a single application.

A problem could occur for using the roughly estimated throughput, based upon the packet size and interarrival time. The estimate is based upon a STA receiving packets such that an application has the required information, for the time period, until the STA next receives a packet. If multiple applications are active, then the following packets aren't necessarily meant for the same application, as the previous received packet. When this is the case any estimate of the required information for applications is overestimated. Even though such an overestimation can occur it can give insight into the behaviour of a STA and active applications.

When having to calculate the average required throughput estimate, for a STA, the edge cases and how the average is to be calculated needs to be considered. Running, trailing or centred moving average takes values on either side of the current values. To circumvent edge cases problems the classifier can be allowed an initialization period, essentially ignore packets at the start of capture in order to calculate the necessary averages.
Calculating the average based upon the last X samples or on the last T seconds will not correspond to each other, as the arrival of packets isn't equidistant. If the time period, for which the average is performed, is to narrow, then an over estimate of the required throughput can occur, this will especially be the case when a STA seldom receives packets. As the throughput, based upon packets size, since the last packet is

[1] e.g. could be within a few minutes

Data Type	Association	Type
Packet size	STA	Integer
Packet inter arrival time	STA	Float
Average throughput estimate for a time period T	STA	Float
Transfer rate	STA	Float
Airtime (total load)	AP	Float
MCS	STA	Categorical
Type of Wi-Fi (amendment)	AP	Categorical
IEEE Wi-Fi amendment	AP	Categorical

Table 4.1: The different data types of interest and the device for which the data is associated.

already calculated this could be utilised instead of calculating the information anew.

An option for the classification model could be to force causality and not wait for future packets before the decision making. For a network load assessment of a packet, it makes sense to only take the previous packets (and possibly the following few packets) into consideration, as these will be the conditions for which the STA is attempting to transmit the packet. It should be noted that even though a network have been busy previous to the transmission of a packet, the STA could still have been allowed to transmit its packet without delay. If this is the case the previous activity on the network wouldn't influencing the current packets interarrival time or MSC value. Even though two STAs are operating under the same network conditions, both connected to the same network etc., their behaviour when running the same application might be different if the STAs are vastly different. This difference in behaviour could stem from elements such as software and hardware differences, i.e. differences in versions for the *operating system* (OS), applications and a multitude of manufacturers producing different network chips.

If an STA has a valid OUI, i.e. it is present and isn't spoofed, the OUI is information about the manufacture of the network chip, when resolved. Knowing the network chip manufacturer gives insight into what conditions the STA is transmitting under. A problematic aspect of using the OUI for application is that the same manufactures can produce a multitude of hardware, as well as new and old hardware from the same manufactures existing. Though the OUI can be telling about the STA, a relation between OUI and OS aren't likely as the OS is software limited and can be changed by the user. As such the OUI isn't deemed a data type of interest for this project.

Table 4.1 contains a list of all the mentioned data and their associated device (STA or AP) as well as a data type (Integer, float or categorical labels) for each.

4.1.1 Data standardisation

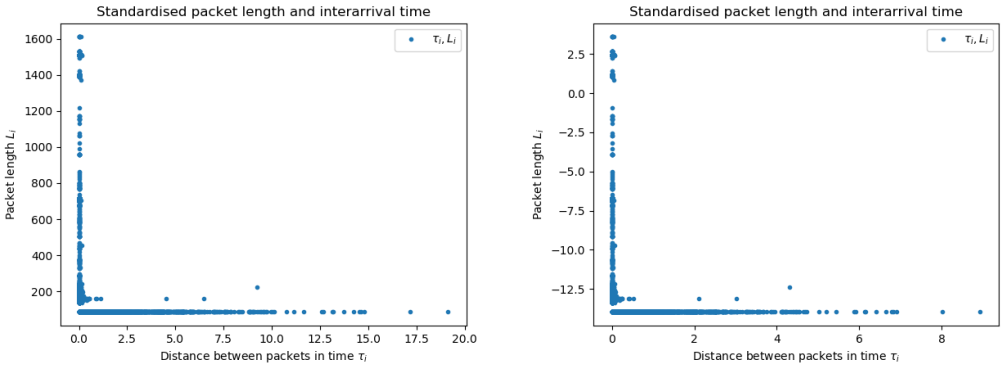
In this thesis a regular normalisation/standardisation is performed by subtracting the sample mean, $\bar{\mu}(X)$, and dividing by the sample standard deviation, $\bar{\sigma}(X)$,

$$\text{Standardise}(X) = \frac{X - \bar{\mu}(X)}{\bar{\sigma}(X)}.$$

The standardisation is performed in order to better compare the multitude of different from 4.1. Furthermore, standardisation have also shown great results if the data is to be used when training a neural network.

If the variables of X follows, or is expected to follow, a normal distribution the standardisation will express the variables in a distance, number of standard deviations, from the mean.

When standardisation is used, it will align the different data under the same premise, subsequently mitigating potentially dominating data, allowing for a better comparison between them, e.g. a variable with a range form 0 to 300 outweighing a variable with range from 0 to 1, as evident in the simple clustering from chapter 2. Figure 4.1 is an example the difference of standardisation makes for data similar to which is shown in chapter 2.



(a) Original data before standardisation **(b)** Data after standardisation

Figure 4.1: Regular and standardised data

4.1.2 Time series and word clustering

When the data is captured packets are inherently received as a time series. The captured data will contain packets from all devices transmitting, under the conditions

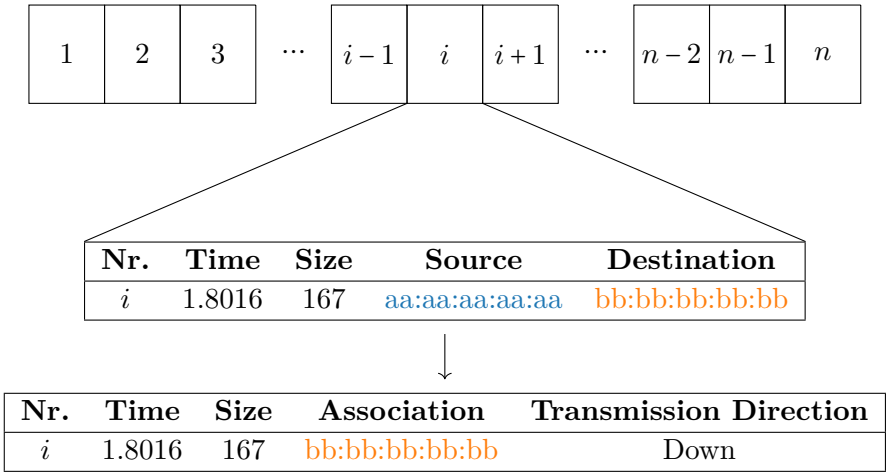


Figure 4.2: Example of a time series with a packet send from an AP to a unique STA implying association with the STA and a downlink.

for which capture is limited, e.g. Network type/frequencies and channel/BSSID etc. From a single captured file multiple time series can be constructed.

Under the assumption that the network is an infrastructure BSS network, every communication is passed through the AP and as such an association to an STA can be conducted as illustrated in figure 4.2. Furthermore, a direction of transmission can also be decided depending on if the associated STA is the transmitter or receiver a given packet. With an association STA, the original time series can be split up into unique time series, only containing packets with association to specific STAs. Figure 4.3 is an visualisation of this split up.

Applications such as Skype will typically, e.g. when a Skype call is active or inactive, either be transmitting multiple packets within a time frame, or not transmitting anything[2]. If sufficient time passes, without a STA transmitting/receiving packets, it can be expected that the STA doesn't have an active application running. These gaps of silence can then be used to split up the created time series into shorter time series, for where applications are active.

These shorter time series can be of interest, as they possibly remove discontinuity between packets in the same time series. The packets in the smaller time series are expected to have more in common, than if a large delay occur between the packets. The reasoning for this, is that a longer pause can indicate inactivity for an STA. Any packet transmitted before and after the inactivity, depending upon the length

[2]Possibly transmitting/receiving a single packets – e.g. fetching new info such as your current Skype Status etc.

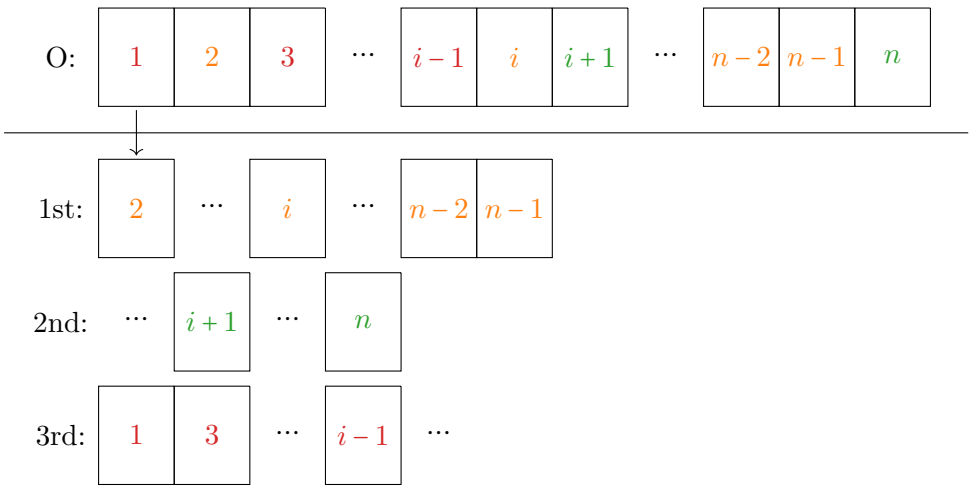


Figure 4.3: Example of splitting up a time series, O, into three distinct time series, (1st, 2nd and 3rd), depending upon packet association, with each color {Orange, Green and Red} indicating a unique STA association.

of inactivity, would be uncorrelated. In this thesis the time series are split if a gap larger than 200 ms occur, based upon arguments from [24].

If it isn't wanted to do these short splits, the initial handshakes between STA and AP can instead be used to split up the time series. Though splitting by the initial handshakes would require that the handshakes are captured.

By working with the data as time series, instead of simple packets, the labelling needs to be taken into consideration again. Instead of giving a simple categorical label, such as an integer number, an array with length equal to the number of categories is used in stead, with each entrance corresponding to a categorical label. In order to prevent a single differently labelled packet to equally influence an entire time series, each entrance in the aforementioned array is equal to the percentile appearance of packets with a given application/application-type in the transformed time series. This association score, s_i, for the i'th application/application-type is simply calculated by taking the number of packets received within the different labelled time frames, $\#P_i$, and dividing by the total number of packets, $\#P_T$, in the entire time frame,

$$s_i = \frac{\#P_i}{\#P_T}. \tag{4.1}$$

As time frames can be overlapping, packets in the original time series can be assumed belonging to multiple different application. In order to take this into consideration

packets, just as the entire time series, are allowed to have more than one label. By allowing multiple labels for a single packet, a problem would then occur when trying to calculate the association score, as the total number of labels is greater than the total number of packets, e.g. let s_i be an association score then $\sum_{i=1}^{K} s_i > 1$, with K different classes. If the goal is to have an association scores $s_i \in [0, 1]$, which represent the percentile appearance of a label in a packet, such that the sum of all these score equate to 1, the mismatch between the total number of labels and the total number of packets is to be prevented. Furthermore, if the association score s_i is calculated by using the sum of the total number of labels, $\#P_i$,

$$s_i = \frac{\#P_i}{\sum_{k=0}^{K} \#P_k}, \text{ with K number of classes.} \tag{4.2}$$

Instead of the total number of packets, the associations score would slightly favour any overlapping time frames/labels, as these packets are disproportionally represented compared to labelled without overlap.

Another idear, and what is used in this thesis, would then be to replicate the principal of the association score, having each label represent a percentile association with active applications. In this case prior knowledge, such an expected number of packets transmitted in a time frame, of the different application/application-types could be usefull. The use of such a prior would be to distribute the percentile association with the applications, e.g. an application being twice as likely to be transmitting as another in an overlapping time frame would create a split value of $\frac{2}{3}$ and $\frac{1}{3}$ for these two labels. With no known prior a common solution/method is to use a uniform distribution – assuming equal preference for transmitting for all applications. By use of such a prior a given transmitted packet is labelled, with an even split, depending upon the number of overlapping time frames.

Instead of calculating the association score, s_i, as in equation 4.1, the total number of packets with a specific label, $\#P_i$ is substituted with a sum of all labelled values for the i'th class.

$$s_i = \frac{\sum_L l_i}{\#P_T} \text{ for } l_i \in L_i \tag{4.3}$$

where P_i is the set of all received packets belonging to the i'th class and L the set of all labels.

Compared to both 4.1 and 4.2 calculating the association score as in 4.3 – not only produces values $s_i \in [0, 1]$ with the sum of all association score equating to 1 – but unlike 4.2 only ever take a packets into consideration once, and still allow for overlapping time frames. For labelling the single packet, a uniform prior is used, with the sum being limited to that single packet witch would just result in the prior.

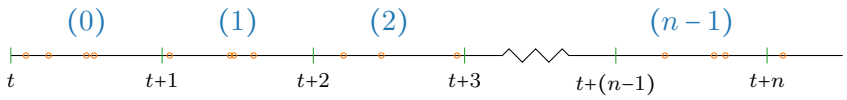


Figure 4.4: Visual example of quantitative to qualitative convergence – All data points within the same range, marked by vertical lines, are allocated to the same class

4.2 Data point considerations and limitation

The behaviour of the data can have consequences for clustering and the classification. Especially should the collected data be representative for a large verity of possible, conditions, as clusters will be skewed if it is misrepresentative.

Another problem for clustering using data with categorical labels is that e.g. the measure for dissimilarity with k-means is the squared euclidean distance. For these reason k-means clustering isn't an appropriate methods if the data contains variables representing categorical labels. [22, p. 427]

As such if simple clustering is to be used, then either the categorical labels should be ignored, or a transformation is needed. An option could be to transform the quantitative data points into qualitative data by breaking down the data into ranges with each range corresponding to a class.
Figure 4.4 illustrates how, as an example, some form of temporal data could be converted into qualitative data. Discretising the data is also a loss of information and doesn't change the fact that clustering with k-means still isn't an option, but could possibly enable other methods requiring a consistent data type. Instead of discretising the continuous quantitative data, another option would be to transform the data in such a way that the discrepancy between qualitative and quantitative data is eliminated.

4.2.1 Factor analysis of Mixed Data.

Factor analysis of Mixed Data (FAMD) is a statistical analysis method that allows for comparison between quantitative and qualitative data – a mixed data set.

The method has it root in a combination of *principal components analysis* (PCA) and *multiple correspondence analysis* (MCA), methods that deals with quantitative and qualitative data respectively. The goal of FAMD is to bring these two, relatively similar methods together in a way that allows for a combination of the two types of data.

Though FAMD, MCA and PCA can be used in gaining statistical information about the data, such as correlation coefficients, another property is dimensionality reduc-

tion and transformation. It is possible to transform the data into a lower dimensional subspace by use of FAMD, similarly to PCA, by only keeping the basis vectors, in a transformation matrix, corresponding to the most information/variance between data points. As with PCA, this information can be calculated based upon eigenvalues. By use of FAMD the issue of balancing out the qualitative and quantitative variables is handled.

The process is a follows:

- Find the basis vector with the highest inertia – corresponds to eigenvector with the largest eigenvalue.
- Find the vector with highest inertia which is orthogonal any other found basis vector and iterate until the collection of vectors span the entirety of original space – Eigenvectors sorted after eigenvalues.
- Keep the number of basis vector based upon their inertia.

The inertia indicate how well the data is represented through that basis vector. When utilising less basis vectors, the data is transformed into a lower dimensional subspace. The inertia allow for control of the quality for this dimensionality reduction. The total kept inertia is given by the sum of the inertia for the different basis vector due to each vector being independent, of any other vector, through their orthogonality. As the basis vectors are constructed based upon maximising inertia, using the first few vectors, possibly corresponding to a specific percentile of the total inertia [3], allows for this control. [22, ch. 12]

The previous might as well be a description for dimensionality reduction through PCA, but FAMD, compared to PCA, works by segregating the data into quantitative and qualitative data, preprocessing these separately before bring the data back together. After the data is preprocessed, the procedure can follow that of a regular PCA – by finding eigenvalues and their corresponding eigenvectors. [22, ch. 12][25, ch. 3]

For the continues qualitative data the preprocessing is the standardisation, described in subsection 4.1.1, but a likewise procedure cant be performed on the categorical data. If the categorical data is labelled with regard for some hierarchical order, some methods for data processing can handle this type of data. If the methods can handle this, the categorical data is seen as a quantitative variable. FAMD requires the quantitative data to behave as for MCA (disjunct table with normalisation indicator), rewriting the qualitative data as a binary indicators and expanding the number of dimensions, letting each dimensions corresponding to a qualitative category. [25, ch. 3]

[3] Total inertia – sum of the inertia for all basis vectors.

In [25, ch. 3] it is argued that in order for the inertia, for the qualitative data, to have the same inertia properties of MCA, the binary indicators should be divided by the proportion of individuals whom posses each of the specific categories.
When this is the case and the data is setup for a regular PCA to be performed on the data. [25, ch. 3]

5 | Autoencoder and neural networks.

In this chapter the relevant theory for neural network, and how it will be used in this thesis, is introduced. One of the common uses for neural network is classification, where the neural network attempts to learn how to classify through labelled data. This supervised learning focus on feature learning and extraction of the data, in order to perform classification, with a specific combination of features equating to a specific class. As labels are a limiting factor for this project a different approach to neural networks will instead be investigated.

In chapter 4 FAMD was introduced as a way to handle categorical data and possibly perform dimensionality reduction. Another method for handling this type of problems is Autoencoders. An autoencoder is a machine learning method that is trained to find a representation of the data by extracting information from the data, producing a "code word". The "code word" can then be utilised to reproduce the original data by use of a decoder, as such the code word contains the essential information needed to reconstruct the original data. [26, ch. 14]

A problem with FAMD is that it doesn't gleam the entire time series as a single entity. It can only take information from the packets and decide what observed value has the largest variance, compared to all other values, and doesn't take statistical properties between packets into consideration.

The goal of the transformations is to achieve a feature space, for which classification can be performed. If this feature space allows for clustering methods to classify the applications, the problem with labelling of data can be circumvented or atleast reduced.

The unsupervised clustering, as previously mentioned, doesn't require labels to find the different clusters only to associates the clusters, if possible, to the associated applications. As the encoder can train on unlabelled data, any collected data can be used for this training. Once the encoder is trained it can then be used to transform a subset of the data, containing labels, on which clustering can be performed and labels associated.

Depending on the quality of encoders, in conjunction with the possible dimension re-

duction, loss of information might occurs in the procedure of encoding and decoding the information.

5.1 Recurrent neural networks and Gated Recurrent Units

By looking at the captured data as time series some nice properties can be exploited. Instead of having each packet be an individual value, a data point might, when seen in the context of the entire time series, behave predictively. Any STA can of course change behaviour from one time instance to the next, depending on the whim of the user. Though unless only a single packet is transmitted for an application then some association between packets is expected. Another advantages of using a recurrent model is that it allows for calculations of properties, without having to specify it to the network – i.e. calculating a rough throughput, with the problem it entails, can be ignored letting the model itself find and calculate information.

Neural networks are self taught based upon the data and choice of hyper parameters and are expected to approximate some function f, such that $f(\mathbf{x})$ for any data $\mathbf{x}$ corresponds to the labelled true output y. Even though a manual approximation or derivation of f might be cumbersome a neural network will try and approximate it. [26, ch. 6]

A blind approach with neural networks, though possible, might not yield noteworthy results, as such exploiting prior knowledge of the data and behaviour leads to a better approximation of f. By repeatedly training the neural network on the data set, it is expected to become better at identifying the features of the data, for which the different labels corresponds. The function, f, is expected to find the right labels no matter the data, . This is often one of the elements investigated by having dedicated training and test data with no overlap between the two set.[26, ch. 6]

In order to successfully train a neural network, the required data needs to be representative of the given scenario. If a scenario is wrongfully under or over represented in a data set it can skew the classifier. This is often one of the advocates for more data being better.

A *Recurrent neural network* (RNN), is a type of neural network that allows information from previous calculations to linger and influence calculations later on. The input for such a network is sequential data, such as a time series $\mathbf{x} = \{x^{\langle 0 \rangle}, x^{\langle 1 \rangle}, \ldots, x^{\langle T \rangle}\}$. Figure 5.1, is a visualisation of a simple RNN with input $\mathbf{x}$, output O and hidden state H. The specific way the RNN stores and passes along information is dependent upon the specific type of RNN, with e.g. some network types being bidirectional passing information both forwards and backwards in time. [26, ch. 10]

5.1.1 Gated recurrent unit

A *gated recurrent unit* (GRU) is a type of RNN network that use a gated architecture in order to prevent gradient problems and manage information flow [27]. The performance for a GRU is comparable to that of *Long-short term memory* (LSTM) [28], though have the advantages of a simpler gating mechanism and the need for fewer calculations.

Neural network "Historically" works by having neurons as active and inactive, with the specific combination of activate and inactive neurons yielding the output of a layer. This methodology has later subsided, especially with activation functions such as the Rectified Linear unit and variation here of, outputting any positive real value. Though the methodology has changed the naming have stuck around, as such the activation for a GRUs hidden unit is given by the following set of equations

$$r^{\langle t \rangle} = \varphi_s \left(\left[W_r x^{\langle t \rangle} + b_r \right] + \left[U_r h^{\langle t-1 \rangle} + u_r \right] \right) \tag{5.1}$$

$$z^{\langle t \rangle} = \varphi_s \left(\left[W_z x^{\langle t \rangle} + b_z \right] + \left[U_z h^{\langle t-1 \rangle} + u_z \right] \right) \tag{5.2}$$

$$\hat{h}^{\langle t \rangle} = \varphi_t \left(\left[W_{\hat{h}} x^{\langle t \rangle} + b_{\hat{h}} \right] + \left[U_{\hat{h}} (r^{\langle t \rangle} \odot h^{\langle t-1 \rangle}) + u_{\hat{h}} \right] \right) \tag{5.3}$$

$$h^{\langle t \rangle} = (1 - z^{\langle t \rangle}) \hat{h}^{\langle t \rangle} + z^{\langle t \rangle} h^{\langle t-1 \rangle} \tag{5.4}$$

where φ is an activation function, either sigmoid φ_s or tanh, φ_t and (5.4) is the activation. [29] Throughout the set of equations, $\{(5.1)(5.2)(5.3)(5.4)\}$, W_* is the learned weights associated to the input values and U_* the learned weights associated with the previous hidden state. b_* and u_* are the learned biases for the network. [1] Though the bias in these equations is represented with two different values, one corresponding to the current input and one for the previous hidden state, the bias is train and calculated as a single entity. Figure 5.2 depicts the flow of information of a GRU cell.

Information flow for any hidden unit is regulated by the gated architecture of the GRU. The activation value for a specific hidden unit for time step t, e.g. $h^{\langle t \rangle}$, is, as per equation 5.4, a weighted combination of the activation value for the previous time step, $t-1$, and the new value $\hat{h}$. The exact weight of these values is regulated by what is known as an update gate, z [29]. The update gate value is always between 0 and 1, as per the definition of the Sigmoid function,

$$\varphi_s(x) = \frac{1}{1 + e^{-x}},$$

and as such the weight of z and $1 - z$ can be seen as a choice of the old or the new hidden state. Though the Sigmoid function predominantly output values close

[1] The $*$ is used as a substitute for the multiple appearances of weights and biases.

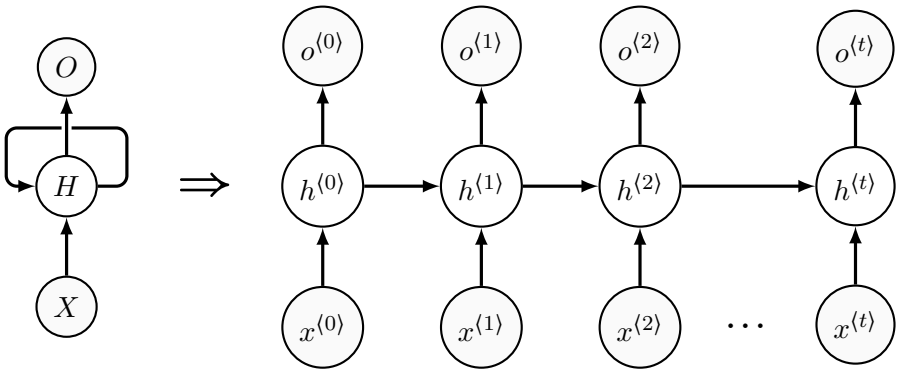


Figure 5.1: Unfolded Recurrent neural network.

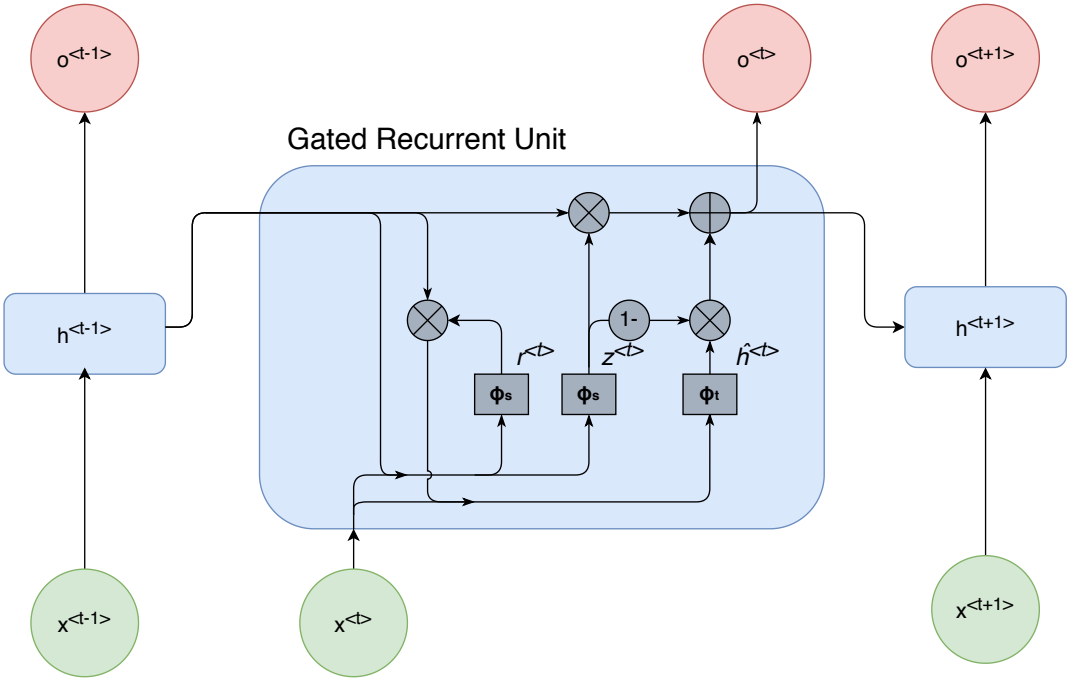


Figure 5.2: Gated recurrent unit cell – expanded.

to 0 and 1, it maps any real value into the range $\{0,1\}$, $\varphi_s : x \in \mathbb{R} \to \{0,1\}$ implying that the activation value can be upto a 50/50 mix of the old and new hidden value.

The value, $\hat{h}$, contains the reset gate, r, given in (5.1). The reset gate enables the network to drop the previous hidden state and only update the new hidden state through the current input values, as per (5.3). As with the update gate, the reset gate is a value passed through a Sigmoid function but unlike the update gate, the reset gate is used through the Hadamard product[2] between the reset gate and the previous hidden state in (5.3). If both the update and reset gate is 0 or close to it, then the new hidden state for time t, is only depending upon the input values dropping any information from the previous state.[29]

5.2 Autoencoder

Autoencoders can consist of any type of neural network architecture the user wants. If a neural network is used for classification, the data $\mathbf{x}$ and the labels Y is used to train the neural network, but as the best representations of $\mathbf{x}$ is unknown this can't be used as a label for training the encoder. Instead the neural network is trained in an unsupervised manner exploiting the properties of neural network architecture. [26, ch. 14][29],[24]

The objective of the encoder is to map the input, $\mathbf{x}$, to a code word, h, with a compact representation. Compared to a regular neural network where the goal is to find some function $f(\mathbf{x}) = Y$ for all inputs $\mathbf{x}$ and associated labels Y. Autoencoders simply "search" for or "construct" a function $\mathcal{E}(x) = h$, by also training the inverse function $\mathcal{D}(h) = \bar{x}$, such that

$$\mathcal{D}(\mathcal{E}(x)) = \bar{x}.$$

The idea is then to training a neural network to reconstruct x or in other words have $\bar{x} = x$. The loss function used in training a neural network with that type of premise can be a measure of dissimilarity between the input and the output.
Once the entire network is trained to reconstruct x, the encoder part of the neural network architecture can then be used independently. As such for autoencoder it is still of interest to find a function, $\mathcal{E}(x) = h$ where h is the compact representation, but the entire setup can be trained in an unsupervised manner with x as both the input and target. [26], [29]

[2] Element-wise/entry-wise multiplication between two matrices.

5.2.1 Autoencoder model

The following section contains information about the used Autoencoder model architecture, as well as the data and data management needed in order to utilise the network. This includes input/output dimension and what exactly is going on beside what a single GRU cell is.

A neural network architecture is defined by its hyperparameters, these are any parameters not directly optimised through the training of the network.
For a single GRU cell the only hyperparameter is the "depth" or hidden size of the cell replacing the conventional notations of neurons in a network. This hidden size specify the dimensions of the output for a GRU cell, and as such the the number of weights that is to be trained. Multiple GRU cells can be used in conjunction by stacking the cell on top of each other creating multiple layers in the RNN network. This allows for the model to achieve greater complexity through a hierarchical feature representation. [28]
If the stacking of GRUs is utilised the input to the GRU cell in a later layer to a specific time step, is the output value for the previous layer at the same time step.

As previous mentioned, the overall input to a GRU cell is a time series, with a batch consisting of multiple time series. Each time series consisting of individual data points, in this thesis corresponding transmitted packets. The recurrent nature of the network allows the values for older time steps, to influence the output for the current time step. Whence the entire time series have been run through the network, for each of the time steps, a hidden state is calculated by the GRU cell. By this, two elements of interest occurs. The first element of interest is the features extracted, the hidden state, for each of the time steps. And the second element of interest is the hidden state of the last time step n. The hidden state for the last time step has had to take all information, from any of the previous time steps, into consideration, where as the features/hidden state for each of the time steps is dependent on the evolution of the time series. Both the last hidden value and the collection of values is used when an autoencoder is constructed from GRU cells.

The autoencoder model consist of three independent models, encoder, center and decoder model, with figure 5.3 depicting the architecture of a simple multilayered autoencoder.

The encoder and decoder models are in architecture, two identical RNNs consisting of stacked GRU cells following the flow of picture 5.2. The center model is a *Fully connected neural network network* (FNN) – A linear combination inputs and weights, $Wx + b$, passed through an activation function – with as many neurons as the depths of the GRU cells. As an intermediate value, the center model acts as both

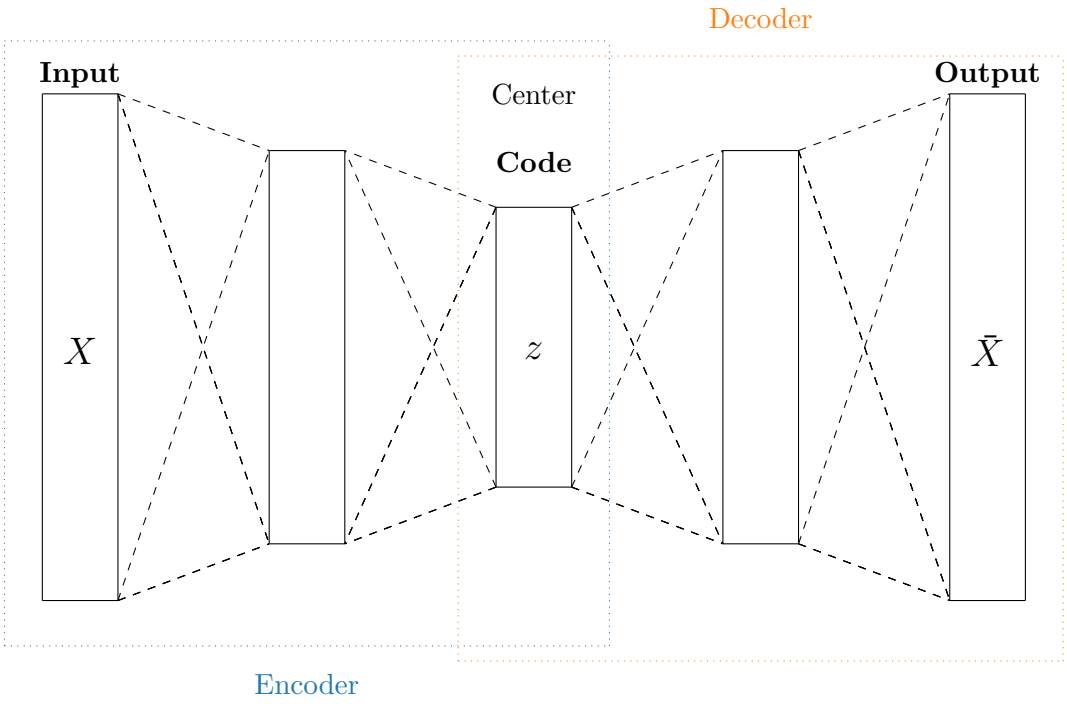


Figure 5.3: Theoretical three layers autoencoder architecture.

the output layer of the encoder, as well as the input layer for the decoder. For neural networks the number of layers are conventionally only used to refer to the number of hidden layers. With the autoencoder, in theory consisting of multiple models, the number of hidden layers, for the overall model, is one greater than the sum of the encoders and decoders hidden layer – Since the layer between the encoders isn't hidden in regular terminology [26, ch. 6]. In order to encompass the entire autoencoder in independent network components the center model is thus used. In figure 5.3 the center model (code word), h, is depicted as belonging to both the en- and decoder.[24], [29]

As input the autoencoder takes in a variable length time series calculating hidden values for each of the time steps in the time series. The last hidden state calculated by the encoder have taken the entirety of the time series into consideration. This hidden state is the end product of the encoder model and the value passed on to the center model.

The decoder is expecting a time series as its input, just as the encoder, but is only passed the hidden state from the encoder through the center model. This hidden state or code word, is passed to the decoder model as its initial hidden state, h_{-1}, along with a start of sequence value, e.g. a null value. From the start of sequence

value and the hidden state, the decoder can perform a calculation for a single time step, $\bar{x}_0$. This value, the feature output for the decoder model, is the first output value of the encoder-decoder model. In the optimal situation the first output of the decoder should be identical to the first value of the input time series, x_0. The decoder output for its first time step, $\bar{x}_0$, is then used as the input for the decoder at the second time step, with the process iterating for the entire time series. [24], [29]

Multiple different sets of weights is trained per GRU cell. Compared to the conventional FNN where a set of weight is trained for each layer with the dimensions of the trained weights, W, fitting to the input value and the number of neurons in the layer, a GRU cell have a set of weights for each of its gates and its new hidden value. The dimensionality of each of these weights, w_*, is the same as that of a single FNN layer.

Even if a single GRU cell requires more computation, compared to a single layer FNN (of related dimensions), and further requires a larger number of weights, it has the distinct advantages of handling variable length inputs, while producing a fixed length codeword. With the codeword produced by the encoder being the last hidden state of a GRU cell, the dimension of the code word is always $M = L \times D$, where L is the number of layers and D the depth of the GRU, no matter the length of the input time series.[24], [29]

By transforming any length time series to a fixed length one, any two time series will become comparable. The end goal is thus to investigate if time series fixed length representation contain the necessary information to determine the application, for which the time series is associated. This is all under the assumption that the time series are fully defined by the active application, in conjunction with transmissions conditions of the network.

5.2.2 Teacher Forcing

Teacher forcing is a method for training sequence autoencoders. This methods can be used to speed up both the training and the convergence of the autoencoder network. Instead of using the predicted features from the previous time step as the input feature to the decoder in the subsequent time step, the teacher forcing method rely on a supervising teacher correcting the errors of the model before the output value is used as the following input value.

Under normal circumstances the input values for the decoder when training an autoencoder or when purely using the decoder, are values for which most are calculated by the decoder itself. Starting out with the a start of sequence character, in this thesis

a null character, a sequence would be given as:

$$null,\ \bar{x}_0,\ \bar{x}_1, \ldots,\ \bar{x}_{n-2},\ \bar{x}_{n-1}$$

in such a sequence only the initial null value is given, with the rest calculated by the decoder.[30] In practice teacher forcing is performed by substituting the calculated values, $\bar{x}$, with values from the original time series, x, such that the decoder input series become:

$$null,\ x_0,\ x_1, \ldots,\ x_{n-2},\ x_{n-1}.$$

With the use of teacher forcing the training of the autoencoder is expected to converge faster as the input for the decoder, is now a constant value for every epoch and doesn't change when weights are updated – reducing the number of necessary training epochs. A constant input value allows for a more direct updating of the weights, as the error isn't calculated based upon "wrong inputs", with errors possibly accumulating with the passage of time. In other words, the system doesn't first have to learn how to correctly encode the first value, and then the second etc. but the weight can be updated for a correct input of an entire time series. [24], [30]

Further advantages for the training time can be achieved, by using known input values for every time step. Besides possibly achieving a faster convergence for the weights, it is possible to perform an increased number of calculation in parallel – reducing the calculation time, per training epoch. Instead of having to wait for the results of the previous time steps, the calculations for any given time step can be completed – The calculation of $\bar{x}_i$ can be done simultaneously with $\bar{x}_{i-1}$, since $\bar{x}_{i-1}$ isn't used in the calculation of $\bar{x}_i$, when using teacher forcing.

Though teacher forcing can speed up the process of training a neural network, it can be memory intensive depending on the implementation. The naive solution, is to generate a time series from the first $n-1$ values of the original data and the prepend a null values at the start. This new time series, in conjunction with the original data, can then be used with one-to-one values, as the new series corresponds to the correct decoder input for any given data point, with the same index. If pointers to the original data isn't utilised, and instead a copy of the old data is created, the total amount of data is doublet compared training without teacher forcing.

Often, as in the case for this thesis, a GPU is used for training neural networks. Though a GPU is good performing calculation, transferring data to and from the GPU is a relative slow process. With limited computational storage, especially for the easily accessed RAM memory duplicating the data bad memory management can be a potential consideration. [31, ch. 1]

6 | Training of autoencoder and backpropagation through time

In this chapter the theoretical and practical aspects of training autoencoders and GRU cells is introduced. This includes derivation for the backpropagation of the error through time, for the first few time steps, and how this can be used in updating/training the autoencoder.

6.1 Backpropagation through time

Neural networks is an optimisation problem with the goal of minimising the objective function, often some measure of difference between a target and the output of the network. The optimisation is performed using a gradient based uptimisation algorithm, in this thesis the adam algorithm [22]. Gradient based optimisation work by calculating the gradient of the objective function, with respect to the current weights of the network, $\frac{\partial E}{\partial w_{ji}}$. The following is the back propagation of the gradient for a GRU cell for the first time steps.

For the autoencoder, the last GRU produces output values for the time series. As previously mentioned this value can be compared to the input values and a measure of dissimilarity can be calculated. If the error function is differentiable then the gradient, with respect to the weights, W_*, can be calculated and back propagated through the network.

An example for such an error function is the ℓ_2-norm, such that

$$E_t = \frac{1}{2}(h_t - y_t)^2,$$

where h_t is the hidden state from the GRU at time step t, with y_t the corresponding target.

The total error, E, for all the time step, is the summation of the errors for the different time steps,

$$E = \sum_t E_t.$$

This type of error backpropagation, is sometimes referred to as backpropagation through time, as the change of error is investigated for the different time steps.

There is a total of six different weights that needs to be taken into consideration for a given hidden state, as per the set of equations, {(5.1)(5.2)(5.3)(5.4)}. As differentiation is linear, an option would be to start with the first time step and iterate through all value possible time steps. For a conceptual understanding of the Gradients behaviour, the gradient for the first few times steps is derived for one of the six weights, in this case the weights of the update gate, z.

In order to derive the gradient for GRU cell it should be noted that the reset gate, (5.1), update gate, (5.2), and the new suggested hidden state, (5.3), are composite functions of weights, as such the chain rule is to be used. In order to distinguish between the outer and inner functions for these equations a subscript is introduced, such that:

$$\begin{aligned} z^{\langle t \rangle} &= \varphi_s\left(\left[W_z x + b_z\right] + \left[U_z h^{\langle t-1 \rangle} + u_z\right]\right) \\ \frac{\partial z^{\langle t \rangle}}{W_z} &= \varphi'\left(\left[W_z x + b_z\right] + \left[U_z h^{\langle t-1 \rangle} + u_z\right]\right)\ (x^{\langle t \rangle}) \\ &= \frac{\partial z_o^{\langle t \rangle}}{\partial z_i^{\langle t \rangle}} \frac{\partial z_i^{\langle t \rangle}}{\partial W_z}, \end{aligned}$$

where o is the out function and i the inner function. The above equation only holds for time steps given that previous hidden states $h^{\langle t-1 \rangle}$ isn't dependent on W_z.

For the first time step, $t = 1$, the initial hidden state $h^{\langle 0 \rangle}$, can be seen as a constant within the scope of the specific GRU, even though it might not be in the context of the entire neural network. With $h^{\langle 0 \rangle}$ a constant value, E_1 only depends on W_z through the update gates themself, and as such the partial derivatives through the use of the chain rule is

$$\frac{\partial E_1}{\partial W_z} = \frac{\partial E_1}{\partial h^{\langle 1 \rangle}} \frac{\partial h^{\langle 1 \rangle}}{\partial z_o^{\langle 1 \rangle}} \frac{\partial z_o^{\langle 1 \rangle}}{\partial z_i^{\langle 1 \rangle}} \frac{\partial z_i^{\langle 1 \rangle}}{\partial W_z}, \tag{6.1}$$

or with the equations inserted and derived and using the ℓ_2-norm for the error:

$$\frac{\partial E_1}{\partial W_z} = \left(h^{\langle 1 \rangle} - y\right)\left(h^{\langle 0 \rangle} - \hat{h}^{\langle 1 \rangle}\right)\left(\varphi_s(W_z x^{\langle 1 \rangle} + U_z h^{\langle 0 \rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 1 \rangle} + U_z h^{\langle 0 \rangle} + b_z))\right)(x^{\langle 1 \rangle}).$$

Like wise the above can be done for any of the other weights, e.g. W_r,

$$\frac{\partial E_1}{\partial W_r} = \frac{\partial E_1}{\partial h^{\langle 1 \rangle}} \frac{\partial h^{\langle 1 \rangle}}{\partial \hat{h}_o^{\langle 1 \rangle}} \frac{\partial \hat{h}_o^{\langle 1 \rangle}}{\partial \hat{h}_i^{\langle 1 \rangle}} \frac{\partial \hat{h}_i^{\langle 1 \rangle}}{\partial r_o} \frac{\partial r_o}{\partial r_i} \frac{\partial r_i}{\partial W_r}.$$

The number of calculations that needs to be performed in order to calculate the gradient for the second time step increases, due to the models recurrent nature. The partial derivative of E_2, with respect to W_z, is as with E_1 only depending through the current hidden value, $h^{\langle 2 \rangle}$,

$$\frac{\partial E_2}{\partial W_z} = \frac{\partial E_2}{\partial h^{\langle 2 \rangle}} \frac{\partial h^{\langle 2 \rangle}}{\partial W_z},$$

though, where the first hidden state only dependent upon W_z through the update gate for the second time step, it also depends on W_z through the previous hidden state. The previous hidden state is used in calculating both the current reset and update gate, as well as the new suggested hidden state, and is, in itself, also used in the calculation of the new hidden state. The following is the partial derivative of the second time steps hidden state with regards to W_z. In order to keep the information comprehensible, and to explain the different steps, the derivative is split up for the gates and hidden state through a line break.

$$\begin{aligned}
\frac{\partial h^{\langle 2 \rangle}}{\partial W_z} =& \frac{\partial h^{\langle 2 \rangle}}{\partial h^{\langle 1 \rangle}} \frac{\partial h^{\langle 1 \rangle}}{\partial z_o^{\langle 1 \rangle}} \frac{\partial z_o^{\langle 1 \rangle}}{\partial z_i^{\langle 1 \rangle}} \frac{\partial z_i^{\langle 1 \rangle}}{\partial W_z} && (6.2) \\
&+ \frac{\partial h^{\langle 2 \rangle}}{\partial z_o^{\langle 2 \rangle}} \frac{\partial z_o^{\langle 2 \rangle}}{\partial z_i^{\langle 2 \rangle}} \left(\frac{\partial z_i^{\langle 2 \rangle}}{\partial h^{\langle 1 \rangle}} \frac{\partial h^{\langle 1 \rangle}}{\partial z_o^{\langle 1 \rangle}} \frac{\partial z_o^{\langle 1 \rangle}}{\partial z_i^{\langle 1 \rangle}} \frac{\partial z_i^{\langle 1 \rangle}}{\partial W_z} + x^{\langle 2 \rangle} \right) && (6.3) \\
&+ \frac{\partial h^{\langle 2 \rangle}}{\partial \hat{h}_o^{\langle 2 \rangle}} \frac{\partial \hat{h}_o^{\langle 2 \rangle}}{\partial \hat{h}_i^{\langle 2 \rangle}} \left(\frac{\partial \hat{h}_i^{\langle 2 \rangle}}{\partial h^{\langle 1 \rangle}} \frac{\partial h^{\langle 1 \rangle}}{\partial z_o^{\langle 1 \rangle}} \frac{\partial z_o^{\langle 1 \rangle}}{\partial z_i^{\langle 1 \rangle}} \frac{\partial z_i^{\langle 1 \rangle}}{\partial W_z} \right) && (6.4) \\
&+ \frac{\partial h^{\langle 2 \rangle}}{\partial \hat{h}_o^{\langle 2 \rangle}} \frac{\partial \hat{h}_o^{\langle 2 \rangle}}{\partial \hat{h}_i^{\langle 2 \rangle}} \left(\frac{\partial \hat{h}_i^{\langle 2 \rangle}}{\partial r_o^{\langle 2 \rangle}} \frac{\partial r_o^{\langle 2 \rangle}}{\partial r_i^{\langle 2 \rangle}} \frac{\partial r_i^{\langle 2 \rangle}}{\partial h^{\langle 1 \rangle}} \frac{\partial h^{\langle 1 \rangle}}{\partial z_o^{\langle 1 \rangle}} \frac{\partial z_o^{\langle 1 \rangle}}{\partial z_i^{\langle 1 \rangle}} \frac{\partial z_i^{\langle 1 \rangle}}{\partial W_z} \right) && (6.5)
\end{aligned}$$

The first partial derivative that is taken into consideration is with respect to the first hidden state, h_1, in equation (6.2). This partial derivate is similarly to that of (6.1), though with the addition of the partial derivative of the second hidden state, with respect to the first.

For (6.3) the change of error through the current update gate, while changing W_z, is derived. As with the first time step W_z is used in calculating the current value of the update gate, but as with (6.1), $h^{\langle 1 \rangle}$ is also used when this calculation is performed, as in:

$$z_i^{\langle 2 \rangle} = \left(W_z x^{\langle 2 \rangle} + U_z h^{\langle 1 \rangle} + b_z \right)$$

The partial derivative of $z_i^{\langle 2 \rangle}$, with respect to W_z, is as such the partial derivative of $h^{\langle 1 \rangle}$, through further use of the chain rule, and the derivative of $W_z x^{\langle 2 \rangle}$, for W_z, i.e.

the last $x^{\langle 2 \rangle}$ in (6.3).

For equation (6.4) and (6.5), $\hat{h}^{\langle 2 \rangle}$ is the focus. W_z is used in this calculation through the previous hidden value and the previous hidden value in the reset gate. These two values is multiplied using the Hadamard product as described in chapter 5, and as such the product rule is used creating the split into the two different functions, with the differences inside the parenthesis.

This general notation used for the partial derivative, not only gives insight into the chain of partial derivatives that is to be performed in order to calculate the gradient, but also how computational tricks can be used to speed up the process. Note how a lot of the computations from (6.1) is repeated through the partial derivative for the second time step. When $\frac{\partial h^{\langle 1 \rangle}}{\partial z_o^{\langle 1 \rangle}} \frac{\partial z_o^{\langle 1 \rangle}}{\partial z_i^{\langle 1 \rangle}} \frac{\partial z_i^{\langle 1 \rangle}}{\partial W_z}$ is calculated once, the value can then be reused for later calculations.

Future time steps are all reliant on the previous hidden values, due to the recurrent nature of the model, and as such values from previous time steps are repeated in the in future calculations. Calculations can be saved and hence the training can be speed up compared to a direct calculation, by starting in a chronological order. With the calculations being dependent on the length of a time series, and the finite memory of computers, a limitations for calculating the exact gradient can occur if a time series is of a certain length – with the length dependent on the computers limitation. In order to, atleast, get an approximated gradient the calculation can be initiated as normal but then be truncated / stopped before the computer runs out of memory.

As previously mentioned exploiting and vanishing gradient can be a problem for RNN networks. With the current value directly dependent on the previous time step, e.g. as in (6.2), a chain of partial derivative, for later time steps, would all be dependent upon every previous values. This will lead to

$$\frac{\partial h^{\langle t \rangle}}{\partial W} = \frac{\partial h^{\langle t \rangle}}{\partial h^{\langle t-1 \rangle}} \frac{\partial h^{\langle t-1 \rangle}}{\partial h^{\langle t-2 \rangle}} \cdots \frac{\partial h^{\langle 2 \rangle}}{\partial h^{\langle 1 \rangle}} \frac{\partial h^{\langle 1 \rangle}}{\partial W},$$

where W is the weight for which the error gradient needs calculation. If the gradient for these value where all less than 1, and the only values used in the calculation (As typical with a RNN network [27]), the problem of vanishing gradient arises, as

$$\prod_{i=0}^{n} g_i \to 0$$

with $0 < g_i < 1$ for any i and $n \to \infty$. With the finite precision of a computer, n only have to reach a large enough size for the value to become zero. If this is the case the

gradient is said to vanish and the information for how the weights should be updated is lost.

6.2 Training and the gradient step

When the gradient is calculated the weights can be updated through the gradient step,

$$W_{\text{new}} = W_{\text{old}} - \lambda \frac{\partial E}{\partial W},$$

where $\lambda \in \mathbb{R}_+$ is the, not necessarily fixed sized, learning rate and $W \in \mathbb{R}$ the weights that needs updating. The learning rate is the step length of the gradient step and should be chosen sufficiently small to not overstep the minimum. This is the basis, for gradient optimisation algorithms widely used in neural network optimisation. The error function have the largest rate of decrease by updating the weights along the negative gradient the error function [22, p. 240]. Though overstepping the minimum isn't in itself a problem for the training, it can possibly lead to oscillation around a minimum.

Under certain constrains for the objective functions[1], a global minimum can be reached using this type of methods, especially can the learning rate for the gradient step be calculated. This isn't expected to be the case for neural network, among other due to its use of non-linear layers. The learning rate needs to be small enough in order to reach the global/local minimum, but the training time for the model should also be taken into consideration – A smaller learning rate leads to smaller updates to the weights. With the weights being iteratively updated through the gradient decent an increase in the amount of iterations required to reach the minimum is an increase in time and amount of required computations. Though the training time doesn't change the usefulness of a model it can be a constrain if a problem is to be solved. [22, ch. 5]

When a model is constructed via a neural network it is entirely based upon the data, for which the network have trained. To circumvent this the goal of the neural network is generalisation, instead of reaching a null error. When the goal is generalisation to new unknown data the optimal solution for the given data set might not correspond to the best solution overall. If an adequate loss is reached then the optimisation can be seen as good enough for the task at hand. This is also the reason why a local minima can be acceptable for training the neural network, compared to a global minimum.

[1] i.e. convexity – It can be proven that if the functions is convex all local minima are global minima

When a neural network is repeatedly trained, possibly reaching a null error, the generalisation might falter. When this is the case the neural network has been overfitted and will be evident from unstable validation data. [26, ch. 7].

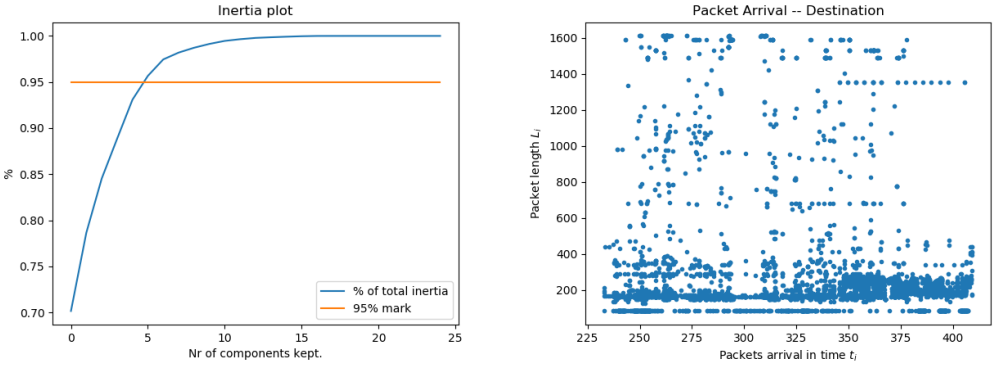
7 | Preliminary data processing results

Throughout this thesis three different data processing methods, standardisation, FAMD and autoencoding, have been introduced. This chapter introduces the preliminary results for these methods, such as the possibility of data reduction through inertia analysis with FAMD and the training results for the autoencoder.

7.1 FAMD – Inertia analysis

As mentioned FAMD allows for dimensionality reduction, through the analysis of inertia, similarly to the more known method of PCA. Furthermore, FAMD will initially expand the dimensions of the categorical data, allowing a new dimension per categorical label.
As FAMD allows for handling the categorical data, it allows for direct use of clustering after the transformation. By keeping a large amount of the information from the data and reducing dimensionality, clustering can be made easier.

By taking the data, portrayed as in chapter 2, and performing dimensionality reduction through FAMD, it allows for a direct comparison between this new and old data. Firstly the inertia can be calculated as explained in chapter 4 and plotted in order to investigate the possible reduction for the dimensions.



(a) Inertia for the data cumulative with 95% visualised.

(b) Original data before FAMD displayed as in chapter 2

Figure 7.1: Cumulative inertia and data reduced to 2 dimensions.

Figure 7.1a is the cumulative inertia plotted for the same data as what shown in figure 7.1b, utilising the expanded data formate shown in 4.1, but visualised as from chapter 2. Unlike the data already shown in chapter 2, this captured data has a more sporadic activity through out its entirety. The expanded dimensions, for this data, increase the dimensions to 25 in total, resulting in the same number of possible dimensions to keep and thus the same number of different inertia values. As is evident from figure 7.1a, most of the inertia is in just a few dimensions. In fact 70% of the total inertia comes from just a single dimensions, with the two largest corresponding to 78% of the total inertia in combination.

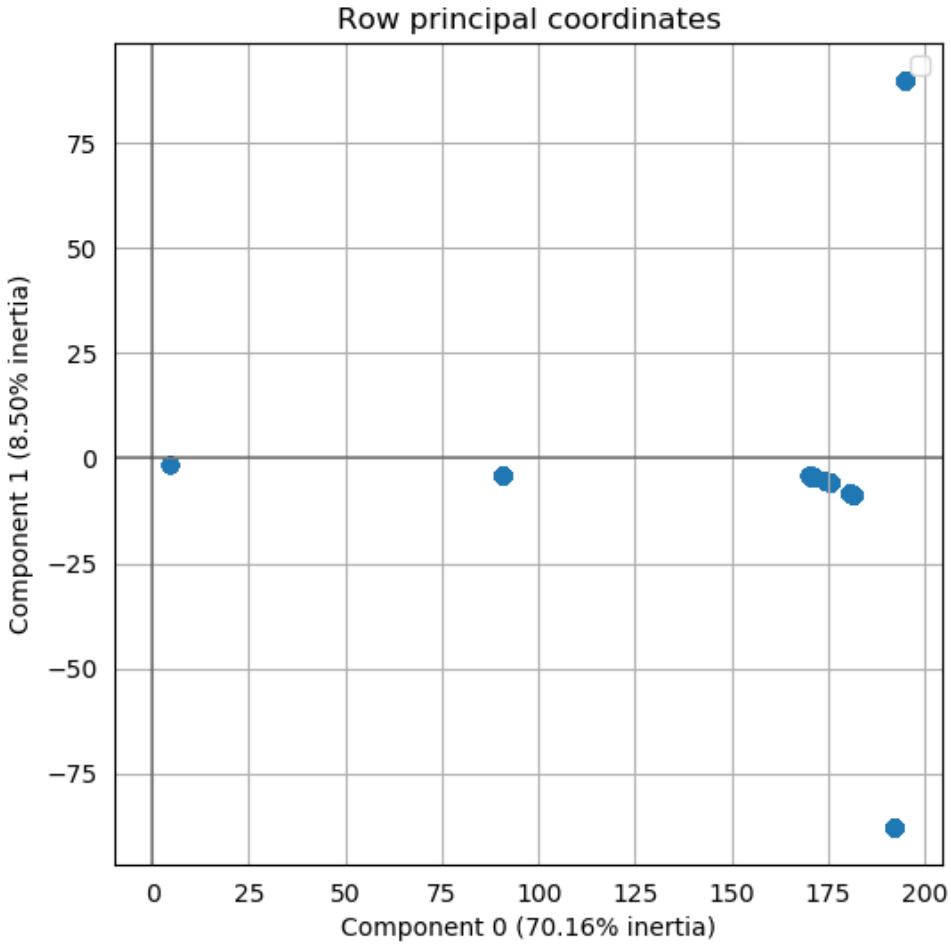
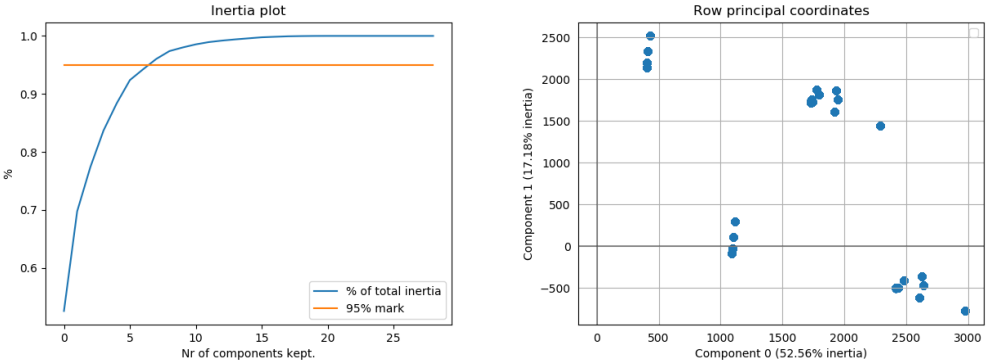


Figure 7.2: Data transformed via FAMD down to 2 dimensions.

With +95% of the total inertia from just 6 components, and the later components corresponding to 0.7% or less of the total inertia, reducing the dimensions to 6 is an option. By keeping 6 components most of the information is kept while adding further components, adds little to no additional information. Figure 7.2 is the same data as in 7.1a and 7.1b, but with two kept components.



(a) Inertia for the data cumulative with 95% visualised.

(b) Data transformed via FAMD down to 2 dimensions.

Figure 7.3: Cumulative inertia and data reduced to 2 dimensions.

In order to make sure that every data point is handled identically, it isn't an option to just reduce every captured data, or new captured data, to the same dimensions. Instead a basis vector is created through FAMD, by looking at every data point. This is an option since every data point have a value, in and of itself, and isn't forced to be in the context of its entire time series.
Figure 7.3a is again the cumulative inertia plotted in descending order as previously seen, with figure 7.1a. The behaviour for the entire dataset is similar to that of the single capture from figure 7.1a. With the similar behaviour the same reasoning can be used, reaching the same conclusion. A change in dimensionality to between 6 or 9 results to between 92% and 97% of the total inertia. With the added data creating a more diverse dataset, the total number of possible components to keep is 29.

The found basis vectors can then be used for transforming the data and clustering can be performed.

As is apparent from figure 7.3b, some natural clusters occur in the data when it is transformed into a 2D space, as such this is also utilised. As with the autoencoder, the best representation might not be the best for application identification. With just two components 70% of inertia is kept, indication that much of the information is still available.

7.2 Model validation and Training results

This section introduces the training results of the autoencoders training and the results on the validation data for specific epochs. This section further elaborates on the behaviour observed during the training and discuss probable cause and effect as

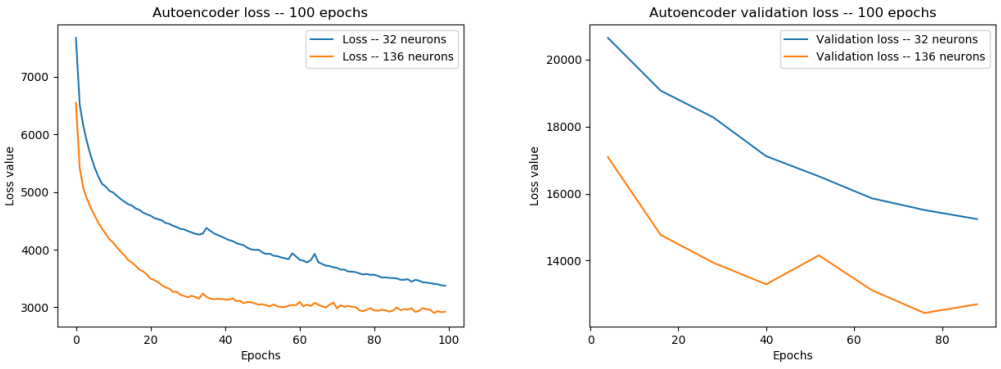
well as explain the observed behaviour.

When an autoencoder is trained, the best possible error for a given set of hyper-parameters is not necessarily 0. As mentioned with the presented GRU setup, time series are reduced in size till a $M = N \times L$ dimensional vector. In theory, if all time series are of length less than m, the autoencoder can learn the indicator function, with the encoded value being identical to the input. When the indicator value is learned, the autoencoder will inherently just pass value through the different layers, leading to no reduction in dimensionality, but also no features extracted.

Compared to FAMD, where the dimensionality reduction id based upon the calculated inertia, the quality of the autoencoder isn't that henceforth. The quality of the encoding, from an autoencoder, is dependent upon the specific weights trained or in another word the specific minimum reached.

7.2.1 Loss and validation-loss

Given a set of hyperparameters, the training the data the autoencoder is observed to converge.
When training the neural network a few things is of interest. First of, its of interest to see if the neural network with a given setup converges, observing a reduction for the loss function as the neural network is exposed to more and more data. When this is the case the neural networks is shown to be able to learn from the setup which, in this thesis, corresponds with learning the encoding scheme.



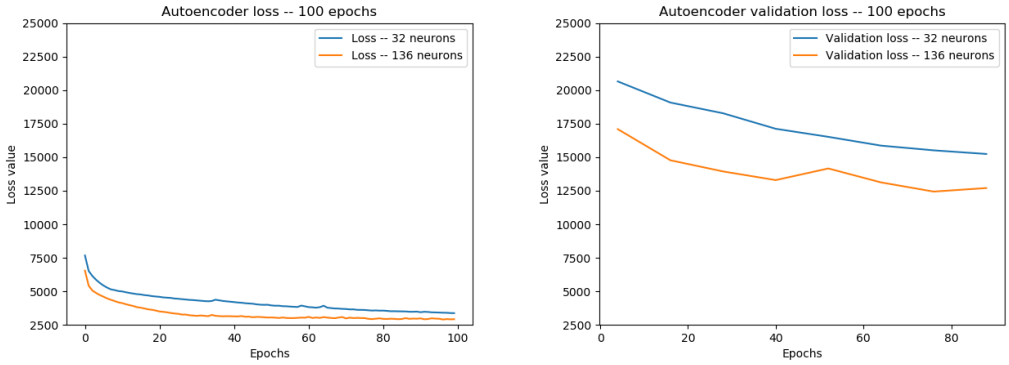
(a) Autoencoder loss for 32 and 136 neurons **(b)** Autoencoder validaion loss 32 and 136 neurons

Figure 7.4: Loss and validation loss.

Figure 7.4a shows the losses as a function of epochs for the autoencoders, with 2 layers as well as 32 and 136 neurons as the hyperparemeters. As can be seen from figure 7.4a the autoencoder is able to learn an encoding scheme, reducing the loss as the number of epochs increases. For the hyperparemeters, an increase in neurons is as mentioned also a decrease in the dimensionality reduction. This corresponds well with the results shown on figure 7.4a, with 136 neurons achieving a slightly better loss at the cost of not reducing the data as much.

Having the neural network iteratively validated, gives insight into the networks behaviour for unknown data. Figure 7.4b is the corresponding validation loss from figure 7.4a. The number of calculated validation losses is significantly fewer, than for the regular loss, in order to save time doing training. When calculating the validation loss it isn't an option to utilise teacher forcing as described in chapter 5 – as this methods can only be used during training – resulting in an increased calculation time. As the general behaviour can still be observed, the obvious choice is to simple not calculate as many validation losses.

Though the validation losses isn't tracked for every single epoch, it is still evident, from figure 7.4b, that when epochs increases validation loss decreases. In other words, the behaviour of the loss for the validation data is similar to what is observed for the training data. That the validation data achieves better results as the number of epochs increases indicate that, not only does the autoencoder learn the training data, it generalises to the validation data as well.



(a) Autoencoder loss for 32 and 136 neurons – scaled axis.

(b) Autoencoder validaion loss 32 and 136 neurons – scaled axis.

Figure 7.5: Loss and validation loss with scaled axis.

It is worth noting the large difference in loss values between the training and vali-

dation data. Figure 7.5, is the same images from 7.4, but with axis scaled such that the loss axis is identical. Though a difference between the loss for the training and validation data is expected, the loss of the validation data for the last checked epochs is around 12500, with the corresponding loss for the training data at around 3000.

A major difference between the two values is that when evaluating the validation data, as previously mentioned, teacher forcing can't be utilised. The reason why this is worth noting, is that every consecutive value for the validation is from the previous predicted value, compared to the previous correct value, as explained in chapter 5. If the neural network isn't great at decoding when an error occurs, then any error might accumulate through the system.

Another thing to take into consideration is that a good autoencoder wont necessarily correspond to a good application identification. To reiterate – the reason for using an autoencoder is to let the encoding identify/extract information useful for performing the identification, as the unencoded information doesn't directly give this.

8 | Results

In this chapter the results of the clustering achieved, with the different preprocessing methods, is introduced with the goal of investigating how useful the found methods are, along with clustering, to identify application through the Wi-Fi traffics meta data.

In order to actually infer anything, a need for investigating how the clusters correspond to the different label occur. Every method used in this project are unsupervised learning – letting the data speak for itself. In order to evaluate the methods and cluster, the rough labelling is used in conjunction with these different methods.

The clusters doesn't inherently belong to a specific label and the described clustering methods will always find exactly the number of clusters asked for. In order to investigate the performance a confusion matrix is used – counting the number of correctly classified variables.

If the problem is reduced to the binary case of either Skype activation (VoIP) or no Skype activation (Non-VoIP) the problem is reduced to four different cases – with two clusters created. If it is assumed that the two produced clusters, either belong to the Skype or non-Skype group, it is possible to investigate which cluster a variable belong too. If a wast majority of variables belong to one cluster, and not the other, all with the same label, the assumption is then that this specific cluster correspond to the given label.
By setting this up in a 2×2 matrix, with the rows and columns corresponding to either labels or a cluster, ordered such that the sum/product of values in the main diagonal is larger than the sum/product of the anti-diagonal, a confusion/matching matrix is constructed, with the pairing in the diagonal equating to the label cluster pairing. The reason for investigate the product or the sum between the values will become evident shortly.

For this simplified binary case with either Skype or non-Skype grouping, the setup corresponds to a True false question. The variables then fall into either a True positive/negative or a False positive/negative for whether a variable is classified correctly. The four different entrances in the binary confusion matrix corresponds with one of these true/false classification – Typically the first entrance in the matrix is made to correspond with the true positive.

From the true/false positives and negatives the performance for this type of classification can be evaluated. A possible metric is to simply calculate how good the system is to correctly identify positives and negatives compared to the total number classification,

$$ACC = \frac{TP + TN}{P + N}. \tag{8.1}$$

For (8.1) TP and TN refer to True positive and negatives, where as P and N refers to the total number of positive and negative predictions respectively, e.g. $P = TP + FP$ where FP is the number of false positives. If the sum of the main diagonal $(TP + TN)$ is greater than that of the anti-diagonal $(FP + FN)$ then the accuracy is as a minimum above 0.5.

The problem with using this type of accuracy would be, if the data is unbalanced with one class dominating another. If so, a high accuracy can be achieved if a system categorise everything identically even though it is unable to distinguish between classes. Another metric that take this fault into consideration is the Youden's J statistic, a value often used in conjunction with receiver operations curve analysis. Youden's J statistic is given by:

$$J = \frac{TP}{TP + FN} + \frac{TN}{TN + FP} - 1 = \frac{TPTN - FNFP}{(TP + FN)(FP + TN)}. \tag{8.2}$$

[32]
Compared to (8.1), if everything is classified identically then $J = 0$, and the method is useless. Furthermore, from equation (8.2) it is worth noting, that the product between the true positive and negative is directly compared to the product between the false positive and negative. This further differs Youden's J statistic from the accuracy calculation. The two methods can then be used depending on what is of interest interested in – How many correct classification performed or should wrongfully classification be taken into consideration. By utilising Youden's J statistic it is possible to achieve a negative value, if the the combination if the product of the true positive/negative is less than the product between the false positive/negative.

In Youden's paper, [32], he states that $J \in [0, 1]$, though as evident from (8.2) it can take on negative values. It is worth noting that Youden's J statistic originally isn't designed for this matching of values, instead only for cases with a specific positive/negative outcomes and fully labelled data.

A problem arises for evaluating in the non-binary case. When dealing with the non-binary case, the need for calculating the number of correctly classified variables arises. The matching between the different clusters and labels become cumbersome,

as the number of possible combinations increases with factorial speed. Furthermore, the notion of True/False positives/negatives doesn't work for an increased number labels, as such the use of Youden's J statistic isn't an option for the non-binary case. When evaluating the non-binary case the accuracy from (8.1) is used in stead, in conjunction with the confusion matrix.

By utilising multiple labels a direct comparison between the predicted labels, from the clusters, and given labels, from the dataset, isn't an option. The fact that the labelling results in values $s_i \in [0, 1]$ with $\sum_i s_i = 1$ for every data point, time series or packet, is then used. After clustering each data point have received a label, corresponding to a cluster. Instead of simply counting the number of different labelled instances, corresponding to a cluster, the label vectors for each data point in a cluster is added together. The cluster will each then have a vector associated to them, with entrances in the vector corresponding to the different labels. The values in these different vectors are then used instead of the counting the instances. By doing it this way, the total number of classification still equal to the total number of data points, even though multiple labels where allowed.

8.1 Clustering results

With some of the data being unlabelled, only the data with an associated label can be used for the actual comparison and matching between clusters and labels. As every data point should be taken into consideration, the clusters are created on the basis of the entire dataset. The following few steps are performed for the different methods in order to achieve the results.

- Transform all the data with the specific method – Autoencoder, FAMD and Standardisation.
- Perform clustering on the transformed data.
- Utilise the transformed data with associated labels and see how they fit in clusters.
- Using one of the described methods for analysis investigate how well the clusters can be utilised for application identification.

The clustering is investigated against both a binary labelling, and the non-binary labelling described in chapter 4, and GM is used for the clusters.

8.1.1 Results – Standardisation

Starting out with the binary labelling for the standardised data. The data corresponds to a total of 141508 labelled packets. With the multiple labels, the clustering

22814.5	49771.5
21223.5	47698.5

Table 8.1: Confusion matrix – standardised binary case.

18910.5	3719	1157.5	15693	20539
0	0	0	0	0
590	120	133	624	1035
11055	3199	1169.5	9461.5	15291
10156	2153	1279	9058.5	16154.5

Table 8.2: Confusion matrix – standardised 5 labels.

achieves an accuracy of 0.5017 or in other words 70995 correctly labelled packet versus 70513 incorrectly for a total of:

$$\frac{70995}{70995 + 70513} = 0.5017$$

The confusion matrix for this results in table 8.1. As this is the binary case Youden's J statistic can be calculated

$$J = \frac{22814.5}{22814.5 + 21223.5} + \frac{47698.5}{47698.5 + 49771.5} - 1 = 0.0074.$$

For the non-binary case, the case with 5 labels, the confusions matrix is given by table 8.2.
This table is further normalised along the rows, then visualised by a heat map in figure 8.1.

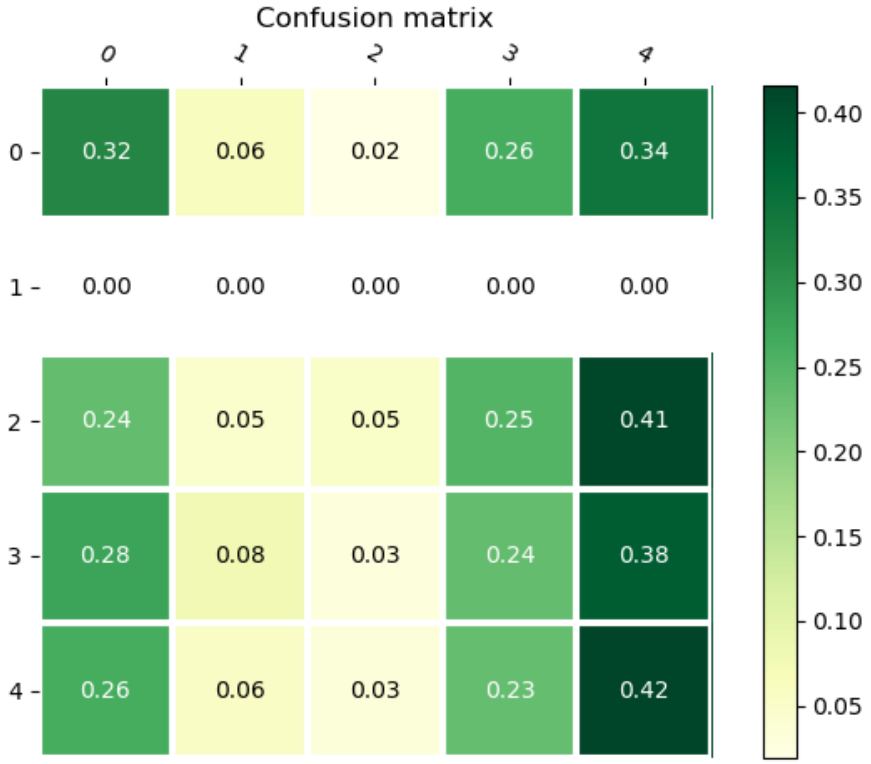


Figure 8.1: Confusion matrix for standardised data – 5 labels.

As Youden's J statistic can't be calculated the accuracy has to suffice. With a total of 44659.5 corresponding to a correct label class combo, and 96848.5 with an incorrect combination, the accuracy for this application identification using standardised data is 0.3155

8.1.2 Results – FAMD

For the FAMD the results are calculated for the different cases, from chapter 7, being a reduction to, 2, 6 and 9 dimensions. As with the standardised data, results are also calculated for both the binary and non-binary labelling.

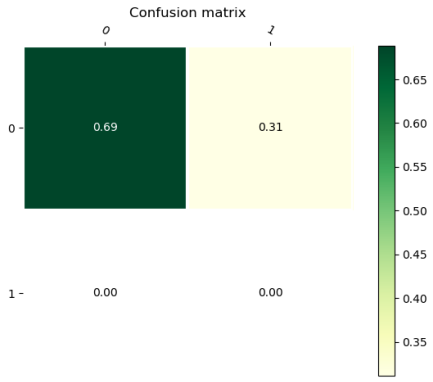
The accuracy results for the three different dimensionality reductions is shown in table 8.3 with Youden's J statistic for the binary case given in 8.4.
As for the standardised data, a normalised confusion matrix, visualised with a heat map is also constructed for the FAMD data.

Dimensionality reduction.	Binary labelling	Non-binary labelling
2	0.6879	0.3747
6	0.6887	0.3747
9	0.6878	0.3747

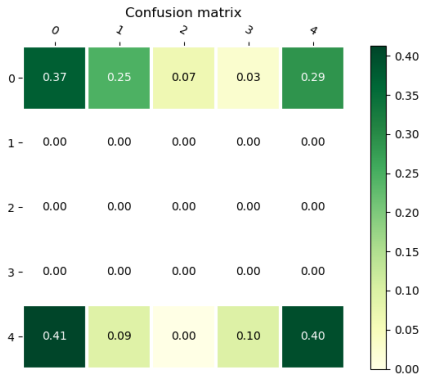
Table 8.3: Accuracy for FAMD transformed data – binary and non-binary labelling

Dimensionality reduction.	J
2	-0.0011
6	-0.3112
9	-0.2185

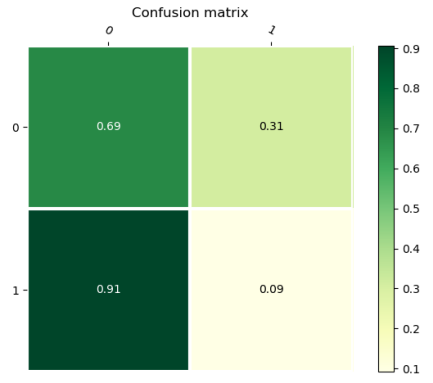
Table 8.4: Youden J statistic for FAMD transformed data.



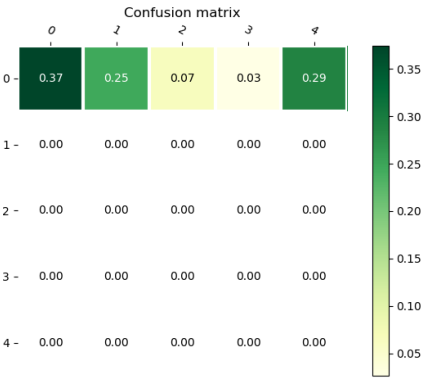
(a) Confusion matrix – 2 dimensions – 2 labels and clusters.



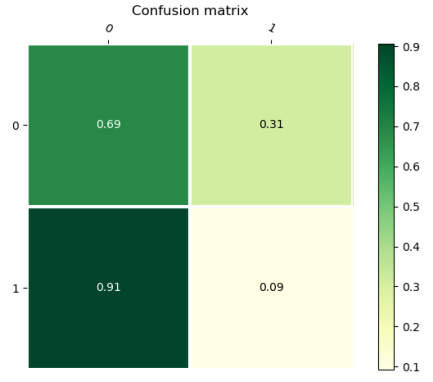
(b) Confusion matrix – 2 dimensions – 5 labels and clusters.



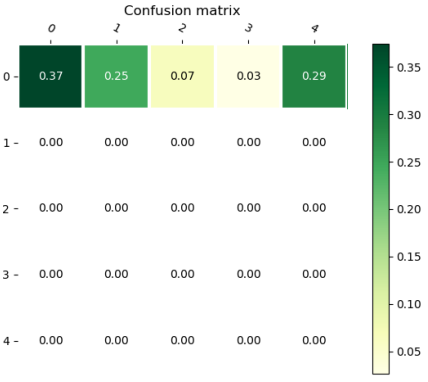
(c) Confusion matrix – 6 dimensions – 2 labels and clusters.



(d) Confusion matrix – 6 dimensions – 5 labels and clusters.

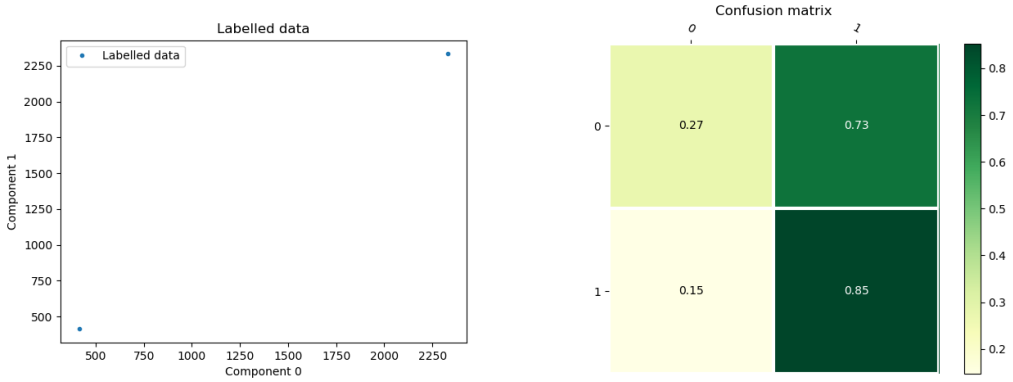


(e) Confusion matrix – 9 dimensions – 2 labels and clusters.



(f) Confusion matrix – 9 dimensions – 5 labels and clusters.

Figure 8.2: Confusion matrix heat map for FAMD transformed data.



(a) All the labelled data transformed via FAMD with 2 kept components.

(b) Confusion matrix with 2 labels and clusters – clusters made from labelled data.

Figure 8.3: Every labelled data transformed by FAMD and the confusion matrix produced by clustering exactly this data.

Figure 8.2 is the collection of every confusion matrix produced by the FAMD data, even the 2×2 from the binary case.

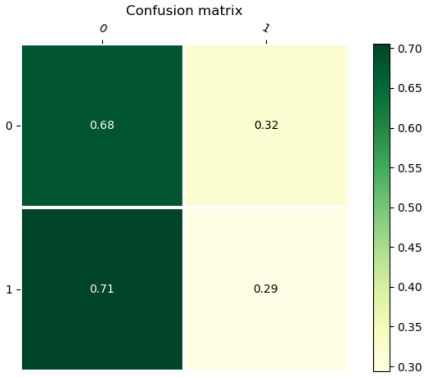
With FAMD being able to transform the data point into a 2 dimensional space, a visual aspect of the behaviour can be gained. Figure 8.3a is the FAMD representation of all labelled packets transformed for visual inspection. Furthermore, clustering created only on the basis of this data subset is created resulting in an accuracy of 0.6719, with as 8.3b the resulting confusion matrix.

8.1.3 Autoencoder results

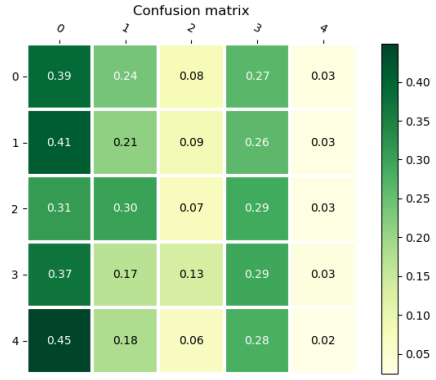
The results in this section for the autoencoder correspond to 9 individual models. The different models is assigned a number, 0 through 8, for which they will be referred in this section. Table 8.5 contains the model specification, that makes the given models unique compare to each other – that is the model number, number of neurons and how many epochs it has been trained. As with the results for the FAMD the results for the autoencoder is both given in a tables, table 8.6 and 8.7, and visualised through its confusion matrix. Figure 8.4 is the visualised confusion matrix for models 6, 7 and 8 [1].

For the autoencoder, the best accuracy is achieved from model nr. 0, for the binary case, and model nr. 5, for the non-binary case. The of the accuracies is greater than 0.67

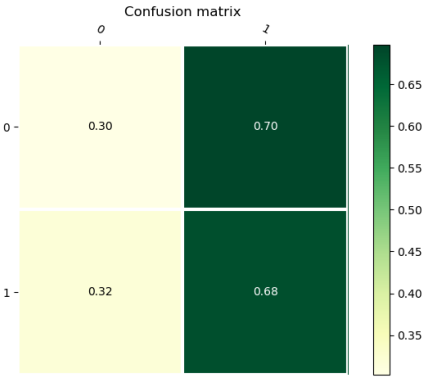
[1] Appendix C contains the confusion matrices for the other 6 models.



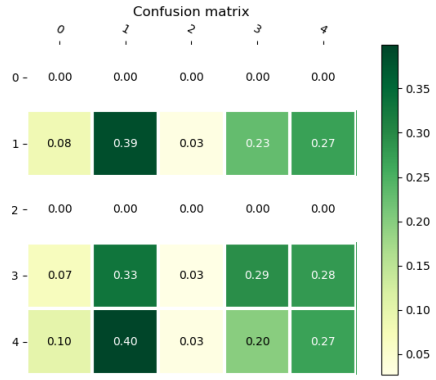
(a) Confusion matrix – model 6 – 2 labels and clusters.



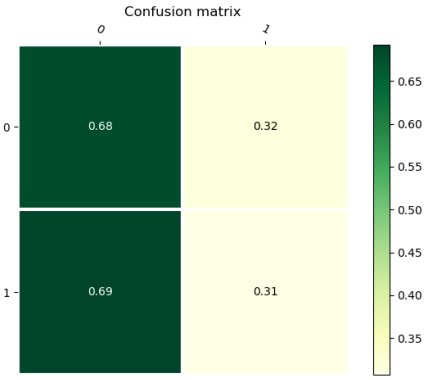
(b) Confusion matrix – model 6 – 5 labels and clusters.



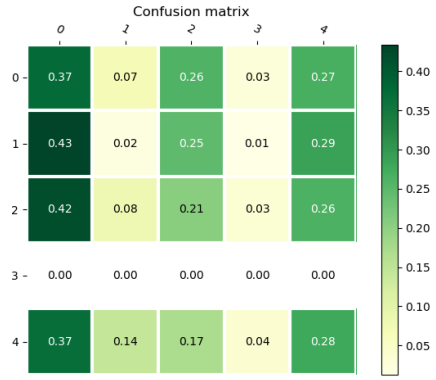
(c) Confusion matrix – model 7 – 2 labels and clusters.



(d) Confusion matrix – model 7 – 5 labels and clusters.



(e) Confusion matrix – model 8 – 2 labels and clusters.



(f) Confusion matrix – model 8 – 5 labels and clusters.

Figure 8.4: Confusion matrix heat map for Autoencoder transformed data.

Results for 8 different models:

Model nr.	Neurons	Epochs
0	128	100
1	128	20
2	136	100
3	140	100
4	140	120
5	140	150
6	32	100
7	64	100
8	64	20

Table 8.5: Model hyperparameters and training epochs

Model nr.	Binary labelling	Non-binary labelling
0	0.6707	0.3483
1	0.6653	0.3231
2	0.5837	0.3135
3	0.5376	0.3248
4	0.6003	0.3186
5	0.6676	0.3650
6	0.6051	0.3356
7	0.5818	0.3628
8	0.5659	0.2986

Table 8.6: Accuracy for autoencoded data – binary and non-binary labelling

Model nr.	J
0	-0.0804
1	0.0401
2	0.0039
3	-0.0146
4	-0.0234
5	0.0448
6	-0.0257
7	-0.0164
8	-0.0108

Table 8.7: Youden's J statistic for autoencoded data.

9 | Discussion

This chapter contains the discussion of the results from chapter 8, as well as general discussion of the project, its limitations and the problems that have occurred along with general observations.

9.1 Discussion – results

As quite evident from the results shown in chapter 8, non of the methods yields results that can be used for inferring application activity. The best accuracy achieved is for the binary FAMD with a reduction to, 6 dimensions. But as this is binary case, Youden's J statistic indicate a bad performance with a value close to 0. The high performance is also originating from the methods classifying everything identically, resulting in the accuracy being identical to the largest percentile of labels instead of giving information about applications. In the ideal situation the confusion matrices produced and visualised in chapter 8, would have the largest concentration of labels along the main diagonal. Instead the results indicate that non of the methods are better than random guessing.

By using any of these methods, along with the captured data and choice of preprocessing/transformation, it isn't possible to correctly identify any applications. An interesting observation is that a lot of the methods classify all the labelled data identically, or as especially evident for the standardised and FAMD transformed data, in a subset of total possible clusters.

Since some clusters aren't used at all, the labelled data can be seen as not being representative for the entirety of the captured data. A reason for this might be that only two different devices has labelled captured data, with the entire dataset possibly containing captures from devices passing by – especially for data capture prior to the covid19 pandemic.
With only access to two different devices when capturing the labelled data, their behaviour might be sufficiently different creating this scenario. With the problem being less severe for the encoder models, it might be an indication that viewing the entirety of a capture, as a single entity, can give relevant information of the different applications. Though from the results achieved for the autoencoder, something is needed in addition for the setup used in this thesis. The performance of the clustering after using the autoencoder indicate that the different autoencoder behave similarly. Especially from table 8.6, it is quite evident that the different trained encoders, though

having different parameters and trained for different epochs, all produce results similar to that achieved by random guessing – similar to both FAMD and standardisation. But unlike FAMD, the encoded data behave atypical when inspecting the confusion matrices, as unlike the other method the results for the encoded data is often distributed in different clusters, e.g. compared to the aggregation of FAMD transformed data.

Ordering a confusion matrix after size is an optimisation exercise in itself, as taking the row column combination with the largest value then eliminating that row and column might not give the best results in the end – and situations where this is the case can easily be constructed. But more importantly it directly decide the calculated accuracy. With the clustering resulting in multiple different labels associated to the same cluster, deciding upon a cluster/label combination isn't as straight forward. The ideal situation would be that the labelled data fall into a unique cluster for each of the labels. Even when increasing the number of cluster the data is still observed to fall within the same clusters.

Transforming the data into 2-dimensions via FAMD, allows for a simple visual comparison. By first transforming the data, and then only plotting the labelled data, as in figure 8.3a the behaviour of the data can be investigated. With the labelled data falling into two unique clusters, and being unable to recognise/distinguish the data for clusters created by the entire data set, clusters are instead constructed from the labelled data only. As the accuracy for even this scenario, the 0.6719, isn't much better than random guessing and with the setup still predominately classifying the data identically, this all indicate that the labelled data fail to convey the sought information.

9.2 General Discussion

Results comparison between the encoded and the non-encoded methods is a tricky one, since a possible error in the labelling has a larger impact for the non-encoder methods. A major problem for this thesis has been the capture of the Wi-Fi data – especially with the covid19 pandemic.

With the shutdown due to covid19 resulting in limited access to the university, the data was captured with regularly available equipment without the use of equipment specific for Wi-Fi monitoring. The captured data might behave differently if such equipment was used, but this can only be speculated. Especially since the available information for the behaviour of a specific network interface cards can be sporadic at best, equipment for the specific use of Wi-Fi monitoring might not have this limitation.

Another option under ideal circumstances would be to capture the data using multiple different devices. The reason for this would be to validate the captured data, making sure that the capturing is performed correctly, with the network interface card behaving probably for both monitor and promiscuous mode. During capturing this wasn't an option since a device can't both capture data and function regularly. Monitor and promiscuous mode will essentially remove the device from the network, with it now tasked with capture everything and do nothing about the received information. Furthermore, having two devices capturing will allow to cross reference the captured packets for any error/discrepancy. This could possibly prevent packet from being labelled wrongly due to a faulty STA association or incorrect information from the packet in general. Since some of the used information originating from calculation performed by the computer (e.g. the transfer rate) and not the MAC header directly, incorrect information is a possibility.

It is worth noting that the labelling is hard labelled – that is if a packet, relevant to a label, is received before or after the labelling cut-off, it will be missed. When a packet labelling is missed in this way it will result in a miss labelled packet influencing the overall indicated performance. This is especially problematic for the non-autoencoder methods, as the cluster for those methods are performed for every individual packet, but with autoencoder performed on the entire time series, a few miss labelled packets might not influence the overall labelling, for the entire time series.

It could be argued that manual labelling of packets through time frames increases the risk of error occurring during this part. Though it can be an error prone process, labelling packets other than through time frames would require full access to the network. If labelling should be performed without full access to a network, it might be of interest to investigate adding a slack parameter for the time frames. Even though a user might start or stop application at a specific time, creating a time frame, the packages transmitted on a Wi-Fi might arrive outside this time frame, due to package aggregation and transmissions delay. Though the best way of identifying how such a slack parameter should behave, would be from full access.

The best way to achieve labelling of the different packets would be with full access to the network, in conjunction with creating an application classifier, possibly through flow analysis. By capturing the data twice, once only capturing the public available meta data, and once capturing everything with full access, the public available information can then be labelled by the results of the full access classifier. The second option would be to create dummy data with full control of every single packet sent – knowing the labels but loosing the real data element which the thesis

is based upon.
Having a ground-truth label for every captured packet would also allow for the utilisation of supervised machine learning methods for classification.

Another element that can be taken into consideration, is whether adding additional variables is just added noise, with fewer variables per packet possibly increasing accuracy. But with both the autoencoder and FAMD reducing the dimensions, the methods should handle redundant information, extracting the features of interest.

An autoencoder could have been used in order to reduce the dimensions of the packets, like FAMD. But since FAMD use statistical optimisation/likelihood optimisation – closed form and not a representation or estimate – a better result is expected compared to what could be achieved through a neural network.

It might be to much to ask of the data, to have it fall within such specific grouping but the premise of this thesis is exactly that: Does a cluster occur for a specific application/application type.

Even if it true that any time series behaviour is fully defined by both the network conditions and the active applications – the basis for this thesis – a problem might arises if its not uniquely defined. If different applications or even application type behave to similar on the meta level, they might be indistinguishable from each other.

10 | Conclusion

The idea behind the project was, that with different application possibly having unique properties when transmitting, the applications could be identified from public available information. It would then be expected that this meta data would create unique clusters for each specific group, when utilising transmissions relevant meta information – like the information presented in chapter 4.

Through captured meta data, whether these clusters exist was investigated. Three different preprocessing methods, simple standardisation, transformation through FAMD and encoding through a trained GRU-autoencoder, was utilised on captured meta data. The goal of using the preprocessing methods was to capture the essence of each transmission, when used in conjunction with clustering.

FAMD transformation and the GRU-autoencoder allow the data to be used as either individual packets or as a coherent time series. With labelled data the clusters produced was evaluated through their accuracy for correctly classifying application types.

The results indicate the achieved clusters, through the transformed data, doesn't contain the necessary information needed, in order to correctly identify the different application. It can't be ruled out, that is possible to achieve application identification, in general or through the investigated methods, even though it wasn't the case for this thesis.

If applications are to be identified through the meta data, the method used in this thesis might be applicable, in combination with data captured and labelled through a different setup. But for this specific case it can only be concluded that it wasn't possible – with results no better than random guessing.

11 | Future studies

In this section option for future studies within the scope of this thesis, besides investigating elements from the discussion, is hypothesised.

Autoencoders are often used in *Natural language processing*, e.g. with a goal of translating between languages. A problem for Natural language processing could be the translation of text between two different languages. The reason this can be a problem that the same word, can have multiple meanings, such as the word "bank". In order to handle this, the method used is required to capture the essence of the sentence, similar to how the autoencoder in this project was expected to capture the essence of a time series.

In 2017, Vaswani et. al. published a paper called: "Attention is all you need." [33]. In this paper Vaswani et. al. compares an Attention neural network – corresponding to the center of an autoencoder – with that of autoencoders constructed through RNN and Convolutional neural networks, concluding that Attention neural network outperform regular autoencoder. If future research into the subject of this thesis is to be conducted, an option might be to investigate how Attention neural network or other Natural language processing methods handle the problem.

Bibliography

[1] statistik, D., *Danmarks statistik - it-anvendelse i befolkningen*, Visited 10.09.2019, 2017. [Online]. Available: https://www.dst.dk/Site/Dst/Udgivelser/nyt/GetPdf.aspx?cid=24235.

[2] Gupta, P., Stanescu, A., Mata-Toledo, R. A., and Accessscience, *Internet of things*, Visited 10.09.2019, 2017. DOI: 10.1036/1097-8542.349850. [Online]. Available: https://www.accessscience.com/content/internet-of-things/349850.

[3] Statista, *Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions)*, Visited 10.09.2019, 2017. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[4] statistik, D., *Danmarks statistik - elektronik i hjemmet*, Visited 10.09.2019, 2019. [Online]. Available: https://www.dst.dk/da/Statistik/emner/priser-og-forbrug/forbrug/elektronik-i-hjemmet.

[5] "Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications", *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, Dec. 2016. DOI: 10.1109/IEEESTD.2016.7786995.

[6] Handte, M., Iqbal, M., Wagner, S., Apolinarski, W., Marrón, P., Muñoz Navarro, E., Martinez, S., Barthelemy, S., and Fernández, M., "Crowd density estimation for public transport vehicles", *CEUR Workshop Proceedings*, vol. 1133, pp. 315–322, Jan. 2014.

[7] Song, B., Poonawala, H., Wynter, L., and Blandin, S., "Robust commuter movement inference from connected mobile devices", in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov. 2018, pp. 640–647. DOI: 10.1109/ICDMW.2018.00099.

[8] Musa, A. and Eriksson, J., "Tracking unmodified smartphones using wi-fi monitors", Nov. 2012, pp. 281–294, ISBN: 9781450311694. DOI: 10.1145/2426656.2426685.

[9] Luo, H. and Accessscience, *Wi-fi*, Visited 11.09.2019, 2017. DOI: 10.1036/1097-8542.802040. [Online]. Available: https://www.accessscience.com/content/wi-fi/802040#802040s002.

[10] Junges, P.-M., Francois, J., and Festor, O., "Passive inference of user actions through iot gateway encrypted traffic analysis", in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Apr. 2019, pp. 7–12.

[11] Do, L. H. and Branch, P., "Real time voip traffic classification", Swinburne University of Technology, Melbourne, AU, Tech. Rep., 2009.

[12] Buonerba, A., "Skype traffic detection and characterization", Master's thesis, Helsinki University of Technology, Helsinki, FL, Sep. 2007.

[13] Ptáček, L., "Analysis and detection of skype network traffic", Master's thesis, Masaryk University, 2011.

[14] Azab, A., Layton, R., Alazab, M., and Watters, P., "Skype traffic classification using cost sensitive algorithms", in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, Nov. 2013, pp. 14–21. DOI: 10.1109/CTC.2013.11.

[15] Di Mauro, M. and Longo, M., "Skype traffic detection: A decision theory based tool", in *2014 International Carnahan Conference on Security Technology (ICCST)*, Oct. 2014, pp. 1–6. DOI: 10.1109/CCST.2014.6986975.

[16] Perenyi, M., Gefferth, A., Dang, T. D., and Molnar, S., "Skype traffic identification", in *IEEE GLOBECOM 2007 - IEEE Global Telecommunications Conference*, Nov. 2007, pp. 399–404. DOI: 10.1109/GLOCOM.2007.81.

[17] Alshammari, R. and Zincir-Heywood, A. N., "Machine learning based encrypted traffic classification: Identifying ssh and skype", in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Jul. 2009, pp. 1–8. DOI: 10.1109/CISDA.2009.5356534.

[18] Bonfiglio, D., Mellia, M., Meo, M., Ritacca, N., and Rossi, D., "Tracking down skype traffic", in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, Apr. 2008, pp. 261–265. DOI: 10.1109/INFOCOM.2008.61.

[19] Zhang, D., Zheng, C., Zhang, H., and Yu, H., "Identification and analysis of skype peer-to-peer traffic", in *2010 Fifth International Conference on Internet and Web Applications and Services*, May 2010, pp. 200–206. DOI: 10.1109/ICIW.2010.36.

[20] Angevine, D. and Zincir-Heywood, N., "A preliminary investigation of skype traffic classification using a minimalist feature set", in *2008 Third International Conference on Availability, Reliability and Security*, Mar. 2008, pp. 1075–1079. DOI: 10.1109/ARES.2008.158.

[21] Azab, A., Layton, R., Alazab, M., and Watters, P., "Skype traffic classification using cost sensitive algorithms", in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, Nov. 2013, pp. 14–21. DOI: 10.1109/CTC.2013.11.

[22] Bishop, C. M., *Pattern Recognition and Machine Learning*, 1st ed. Springer, 2006, ISBN: 0-387-31073-8.

[23] Wireshark.org, *Wireshark - wlan (ieee 802.11) capture setup*, Visited 21.02.2020, 2019. [Online]. Available: https://wiki.wireshark.org/CaptureSetup/WLAN.

[24] Meneghello, F., Rossi, M., and Bui, N., "Smartphone identification via passive traffic fingerprinting: A sequence-to-sequence learning approach", *IEEE Network*, vol. 34, no. 2, pp. 112–120, 2020.

[25] Pagès, J., *Multiple Factor Analysis by Example Using R*, 1st ed. Taylor & Francis Group, 2015, ISBN: 978-1-4822-0547-3.

[26] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[27] Bengio, Y., Simard, P., and Frasconi, P., "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[28] Dey, R. and Salem, F., "Gate-variants of gated recurrent unit (gru) neural networks", Jan. 2017.

[29] Cho, K., Merriënboer, B. van, Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y., "Learning phrase representations using rnn encoder-decoder for statistical machine translation", Jun. 2014. DOI: 10.3115/v1/D14-1179.

[30] Dey, R. and Salem, F., "Gate-variants of gated recurrent unit (gru) neural networks", Jan. 2017.

[31] Larsen, T., Arildsen, T., and Jensen, T. L., *Behavioral Simulation and Computing for Signal Processing Systems*, 1st ed. John Wiley & Sons Inc, 2018, ISBN: 978-0-4707-1134-7.

[32] Youden, W. J., "Index for rating diagnostic tests", *Cancer*, vol. 3, no. 1, pp. 32–35, 1950. DOI: 10.1002/1097-0142(1950)3:1<32::AID-CNCR2820030106>3.0.CO;2-3.

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I., "Attention is all you need", in *Advances in Neural Information Processing Systems 30*, Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., Eds., Curran Associates, Inc., 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

A | Abbreviations and notation

The notation and naming used in this thesis follows that of the IEEE 802.11-2016 protocol, [5]. For ease of reading the definitions used from IEEE 802.11-2016 protocol [5], is here reiterated

- Access point (AP): An entity that contains one station (STA) and provides acces to the distribution services, via the wireless medium (WM) for associated STAs [5, p. 128].
- Basic service set (BSS): Basic architecture component for an IEEE 802.11 wireless network [5, p. 184-185].
- Destination Address (DA): Medium access control (MAC) address for which the information transmitted is intended.
- Distribution system (DS): A system used to interconnect a set of basic service sets and integrated local area networks to create and extended service set [5, p. 132, p. 186].
- Extended service set (ESS): The extended service set is a union of the infrastructure basic service sets (BSS) with the same service set ID (SSID) connected by a Distribution system (DS) [5, p. 186].
- Frame: A unit of data exchanged between peer protocol entities [5, p. 133].
- Internet of Things (IoT): Broad term for multiple devices communication through the internet – without require human interaction.
- Medium access control (MAC): Protocol for controlling access to the medium of communication.
- Medium access control (MAC) address: Unique identification address for each station
- Medium access control (MAC) frame: The unit of data exchanged between MAC entities [5, p. 135].
- Modulation and coding scheme (MCS): The specific modulation and coding scheme used – used for 802.11ac and 802.11n amendment. [5, p. 2353]
- Physical layer (PHY): The Physical layer of a transmissions

- Physical layer frame: The unit of data exchange between physical entities [5, p. 138].
- Receiver Address (RA): Medium access control (MAC) address belonging to the intended receiver of the transmission (Physical).
- Source Address (SA): Medium access control (MAC) address for which the information originates.
- Service Set Identifier (SSID): Identifier (name) for a service set.
- Station (STA): A logical entity that is a singly addressable instance of a medium access control (MAC) and physical layer interface to the wireless medium (WM) [5, p. 141].
  Per the 2007 iteration of the standard: Any device that contains an IEEE 802.11-conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium (WM).
- Transmitter Address (TA): Medium access control (MAC) address belonging to the transmitting device (Physical).
- Operating system (OS): System software for managing software, hardware and resources.
- Wireless medium (WM): Wireless Transmission Medium for which the transmission occurs.
- Voice over IO (VoIP): Voice communication over Internet Protocol.

The following abbreviation represent different methods:

- Expectation Maximisation (EM): An optimisation algorithm that can be used when finding clusters through a GM.
- Factor Analysis of mixed data (FAMD): Factor analysis method for mixed data.
- Fully connected neural network (FNN): Neural network with fully connected properties.
- Gaussian Mixture Model (GM): A clustering methods through a mixture of Gaussian distributions.
- Gated Recurrent Unit (GRU): Specific RNN with gated structure.
- Multiple correspondence analysis (MCA): Data analysis technique for categorical/qualitative data.

- Principal components analysis (PCA): Data analysis method utilising the principal components – for quantitative data.
- Recurrent Neural network (RNN): Neural network with recurrent connection properties.

Appendix A. Abbreviations and notation

B | Backpropagation through time – First time step.

This appendix contains all the equations and calculations for the Backpropagation through time for all the different weights, for the first time step. Every equation can be derived by repeated use of the chain rule as the first time step is assumed independent on the initial hidden state.

The chance of error through $W_{\hat{h}}$:

$$\begin{aligned}\frac{\partial E_1}{\partial W_{\hat{h}}} &= \frac{\partial E_1}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial \hat{h}_o^{\langle 1\rangle}}\frac{\partial \hat{h}_o^{\langle 1\rangle}}{\partial \hat{h}_i^{\langle 1\rangle}}\frac{\partial \hat{h}_i^{\langle 1\rangle}}{\partial W_{\hat{h}}} \\ &= \left(h^{\langle 1\rangle} - y^{\langle 1\rangle}\right)\left(1 - z^{\langle 1\rangle}\right)\left(1 - \tanh^2(W_{\hat{h}}x^{\langle 1\rangle} + U_{\hat{h}}(r^{\langle 1\rangle} \odot h^{\langle 0\rangle}) + b_{\hat{h}})\right)\left(x^{\langle 1\rangle}\right)\end{aligned}$$

The chance of error through U_z:

$$\begin{aligned}\frac{\partial E_1}{\partial U_z} &= \frac{\partial E_1}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial z_o^{\langle 1\rangle}}\frac{\partial z_o^{\langle 1\rangle}}{\partial z_i^{\langle 1\rangle}}\frac{\partial z_i^{\langle 1\rangle}}{\partial U_z} \\ &= \left(h^{\langle 0\rangle} - \hat{h}^{\langle 1\rangle}\right)\left(\varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z))\right)\left(h^{\langle 0\rangle}\right)\end{aligned}$$

The chance of error through $U_{\hat{h}}$:

$$\begin{aligned}\frac{\partial E_1}{\partial W_{\hat{h}}} &= \frac{\partial E_1}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial \hat{h}_o^{\langle 1\rangle}}\frac{\partial \hat{h}_o^{\langle 1\rangle}}{\partial \hat{h}_i^{\langle 1\rangle}}\frac{\partial \hat{h}_i^{\langle 1\rangle}}{\partial W_{\hat{h}}} \\ &= \left(h^{\langle 1\rangle} - y^{\langle 1\rangle}\right)\left(1 - z^{\langle 1\rangle}\right)\left(1 - \tanh^2(W_{\hat{h}}x^{\langle 1\rangle} + U_{\hat{h}}(r^{\langle 1\rangle} \odot h^{\langle 0\rangle}) + b_{\hat{h}})\right)\left(r^{\langle 1\rangle}h^{\langle 0\rangle}\right)\end{aligned}$$

The chance of error through U_r:

$$\begin{aligned}\frac{\partial E_1}{\partial W_{\hat{h}}} =& \frac{\partial E_1}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial \hat{h}_o^{\langle 1\rangle}}\frac{\partial \hat{h}_o^{\langle 1\rangle}}{\partial \hat{h}_i^{\langle 1\rangle}}\frac{\partial \hat{h}_i^{\langle 1\rangle}}{\partial r_o^{\langle 1\rangle}}\frac{\partial r_o^{\langle 1\rangle}}{\partial [r_i^{\langle 1\rangle}}\frac{\partial r_i^{\langle 1\rangle}}{\partial U_r} \\ &= \left(h^{\langle 1\rangle} - y^{\langle 1\rangle}\right)\left(1 - z^{\langle 1\rangle}\right)\left(1 - \tanh^2(W_{\hat{h}}x^{\langle 1\rangle} + U_{\hat{h}}(r^{\langle 1\rangle} \odot h^{\langle 0\rangle}) + b_{\hat{h}})\right)\left(U_{\hat{h}}h^{\langle 0\rangle}\right) \\ &\quad \left(\varphi_s(W_z x + U_z h^{\langle 0\rangle} + b_z)(1 - \varphi_s(W_z x + U_z h^{\langle 0\rangle} + b_z))\right)\left(h^{\langle 0\rangle}\right)\end{aligned}$$

These equations in combination with the equations in chapter 6, is the error back-propagation through time for the first time step of a GRU cell.

Error backpropagation second time step

The following is the full equation of (5.1), (5.2), (5.3) and (5.4) with the partial derivatives inserted.

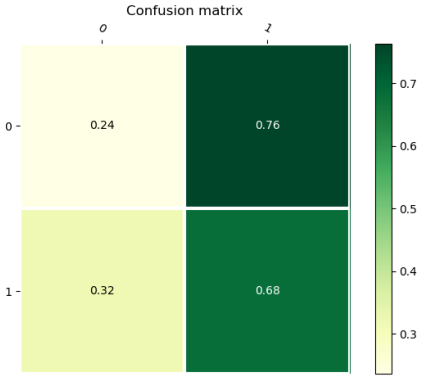
$$
\begin{aligned}
\frac{\partial E_1}{\partial h^{\langle 2\rangle}}\frac{\partial h^{\langle 2\rangle}}{\partial W_z} =& \frac{\partial E_1}{\partial h^{\langle 2\rangle}}\frac{\partial h^{\langle 2\rangle}}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial z_o^{\langle 1\rangle}}\frac{\partial z_o^{\langle 1\rangle}}{\partial z_i^{\langle 1\rangle}}\frac{\partial z_i^{\langle 1\rangle}}{\partial W_z} \\
&+ \frac{\partial E_1}{\partial h^{\langle 2\rangle}}\frac{\partial h^{\langle 2\rangle}}{\partial z_o^{\langle 2\rangle}}\frac{\partial z_o^{\langle 2\rangle}}{\partial z_i^{\langle 2\rangle}}\left(\frac{\partial z_i^{\langle 2\rangle}}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial z_o^{\langle 1\rangle}}\frac{\partial z_o^{\langle 1\rangle}}{\partial z_i^{\langle 1\rangle}}\frac{\partial z_i^{\langle 1\rangle}}{\partial W_z} + x^{\langle 2\rangle}\right) \\
&+ \frac{\partial E_1}{\partial h^{\langle 2\rangle}}\frac{\partial h^{\langle 2\rangle}}{\partial \hat{h}_o^{\langle 2\rangle}}\frac{\partial \hat{h}_o^{\langle 2\rangle}}{\partial \hat{h}_i^{\langle 2\rangle}}\left(\frac{\partial \hat{h}_i^{\langle 2\rangle}}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial z_o^{\langle 1\rangle}}\frac{\partial z_o^{\langle 1\rangle}}{\partial z_i^{\langle 1\rangle}}\frac{\partial z_i^{\langle 1\rangle}}{\partial W_z}\right) \\
&+ \frac{\partial E_1}{\partial h^{\langle 2\rangle}}\frac{\partial h^{\langle 2\rangle}}{\partial \hat{h}_o^{\langle 2\rangle}}\frac{\partial \hat{h}_o^{\langle 2\rangle}}{\partial \hat{h}_i^{\langle 2\rangle}}\left(\frac{\partial \hat{h}_i^{\langle 2\rangle}}{\partial r_o^{\langle 2\rangle}}\frac{\partial r_o^{\langle 2\rangle}}{\partial r_i^{\langle 2\rangle}}\frac{\partial r_i^{\langle 2\rangle}}{\partial h^{\langle 1\rangle}}\frac{\partial h^{\langle 1\rangle}}{\partial z_o^{\langle 1\rangle}}\frac{\partial z_o^{\langle 1\rangle}}{\partial z_i^{\langle 1\rangle}}\frac{\partial z_i^{\langle 1\rangle}}{\partial W_z}\right) \\
=& \left(h^{\langle 2\rangle} - y^{\langle 2\rangle}\right)\left(z^{\langle 2\rangle}\right)\left(h^{\langle 0\rangle} - \hat{h}^{\langle 1\rangle}\right) \\
&\left(\varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z))\right)(x^{\langle 1\rangle}) \\
&+ \left(h^{\langle 2\rangle} - y^{\langle 2\rangle}\right)\left(h^{\langle 1\rangle} - \hat{h}^{\langle 2\rangle}\right)\left(\varphi_s(W_z x^{\langle 2\rangle} + U_z h^{\langle 1\rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 2\rangle} + U_z h^{\langle 1\rangle} + b_z))\right) \\
&\left((U_z)\left(h^{\langle 0\rangle} - \hat{h}^{\langle 1\rangle}\right)\left(\varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z))\right)(x^{\langle 1\rangle})\right. \\
&\left.+(x^{\langle 1\rangle})\right) \\
&+ \left(h^{\langle 2\rangle} - y^{\langle 2\rangle}\right)\left(1 - z^{\langle 2\rangle}\right)\left(1 - \tanh^2(W_{\hat{h}} x^{\langle 2\rangle} + U_{\hat{h}}(r^{\langle 2\rangle} \odot h^{\langle 1\rangle}) + b_{\hat{h}})\right)\left(r^{\langle 2\rangle}\right)(U_z) \\
&\left(h^{\langle 0\rangle} - \hat{h}^{\langle 1\rangle}\right)\left(\varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z))\right)(x^{\langle 1\rangle}) \\
&+ \left(h^{\langle 2\rangle} - y^{\langle 2\rangle}\right)\left(1 - z^{\langle 2\rangle}\right)\left(U_z h z^{\langle 1\rangle}\right) \\
&\left(\varphi_s(W_z x^{\langle 2\rangle} + U_z h^{\langle 1\rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 2\rangle} + U_z h^{\langle 1\rangle} + b_z))\right)(U_r)\left(h^{\langle 0\rangle} - \hat{h}^{\langle 1\rangle}\right) \\
&\left(\varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z)(1 - \varphi_s(W_z x^{\langle 1\rangle} + U_z h^{\langle 0\rangle} + b_z))\right)(x^{\langle 1\rangle})
\end{aligned}
$$

These equation can be quite messy when written out without gaining further inside in the dependencies and the behaviour of the error, as such this is ill-advised compared to the short hand notation.

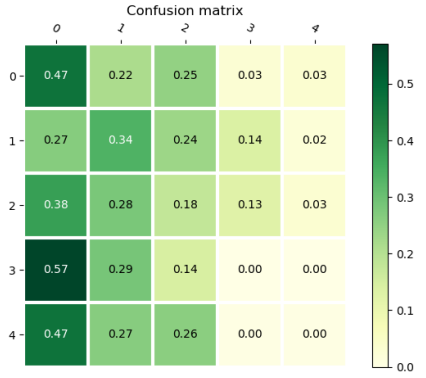
As mentioned in chapter 6, the same procedure as performed for W_z can be done for the other weights at the second time step. This can further be iterated for any future time steps but the amount of equations will increase as seen from the first to the second time step.

C | Visualised confusion matrix – models 0 through 5

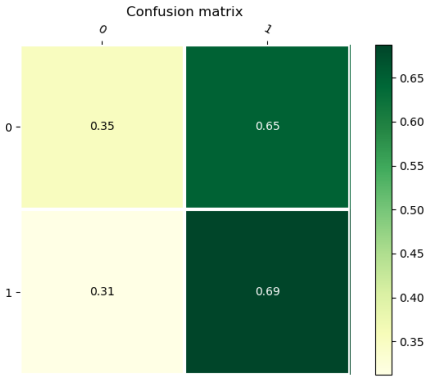
This appendix contains the visualised confusions matrices for the models 0 - 6, not shown in chapter 8.



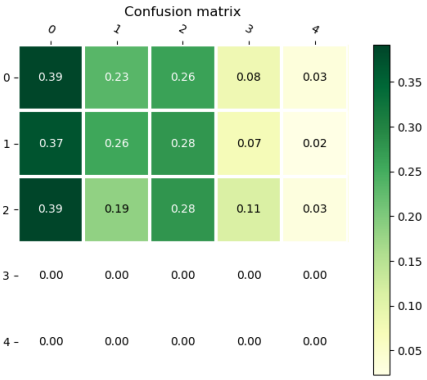
(a) Confusion matrix – model 0 – 2 labels and clusters.



(b) Confusion matrix – model 0 – 5 labels and clusters.

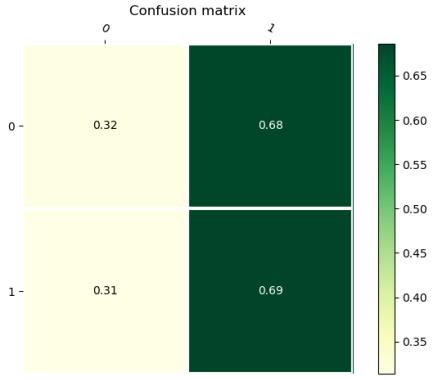


(c) Confusion matrix – model 1 – 2 labels and clusters.

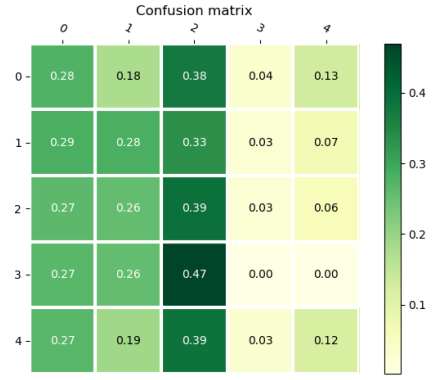


(d) Confusion matrix – model 1 – 5 labels and clusters.

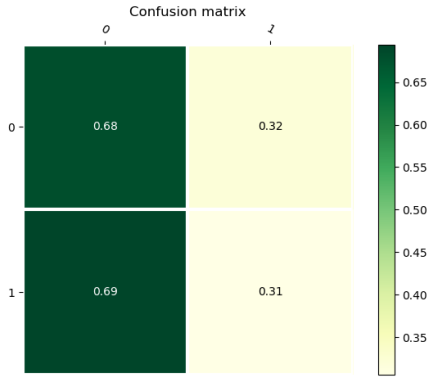
Figure C.1: Confusion matrix heat map for Autoencoder transformed data.



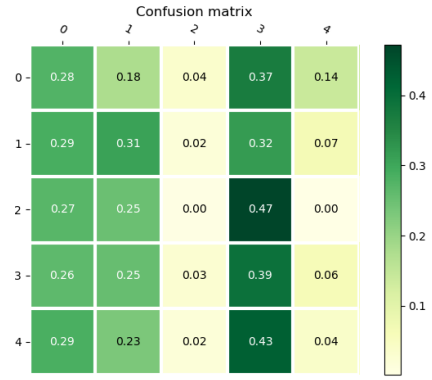
(a) Confusion matrix – model 2 – 2 labels and clusters.



(b) Confusion matrix – model 2 – 5 labels and clusters.

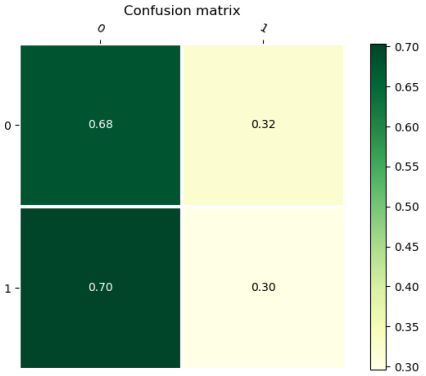


(c) Confusion matrix – model 3 – 2 labels and clusters.

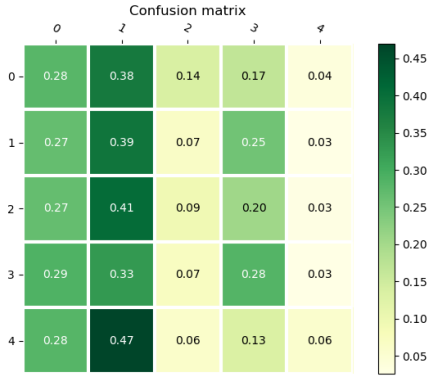


(d) Confusion matrix – model 3 – 5 labels and clusters.

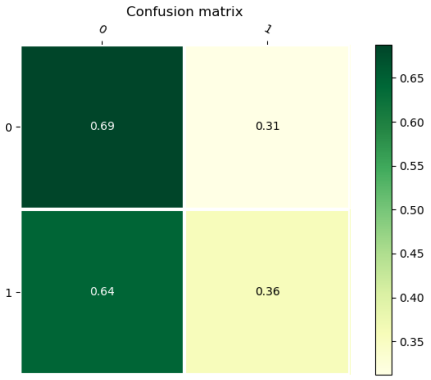
Figure C.2: Confusion matrix heat map for Autoencoder transformed data.



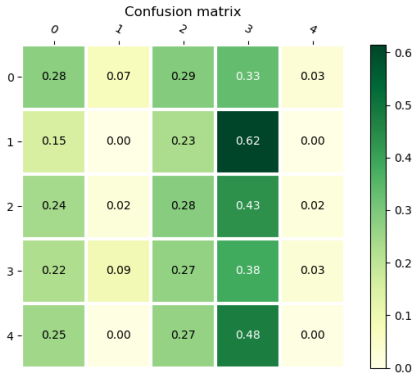
(a) Confusion matrix – model 4 – 2 labels and clusters.



(b) Confusion matrix – model 4 – 5 labels and clusters.



(c) Confusion matrix – model 5 – 2 labels and clusters.



(d) Confusion matrix – model 5 – 5 labels and clusters.

Figure C.3: Confusion matrix heat map for Autoencoder transformed data.

Appendix C. Visualised confusion matrix – models 0 through 5

D | Code Overview

The code produced for this project is available on GitHub @ `https://github.com/Hilligsoe/MattekP10-2020-AAU-enclosure`. This appendix is meant to give an overview of the different scripts and modules, as well as how to use them to gain the different results achieved in chapter 8. As mentioned the scripts are implemented in Python (Python 3.6) and are dependent on `os, numpy, sklearn, tqdm, pytorch, itertools, pathlib, pandas, pickle, prince, scipy` and `matplotlib` libraries. Furthermore, the github link also contain 2 pickled data files, a subset of the total captured data with a labelled and a non-labelled capture, that can be used as a dummy for testing the scripts.

The scripts can roughly be divided in 5 different categories, initial analysis, preprocessing, labels, autoencoder and results.

The initial processing script, the script used in creating the results from chapter 2, is `interpacket.py`. This script takes the original data and visualise the packet lengths as a function of time and inter arrival time.

The preprocessing scripts are:

- `preprocessing-dataloading.py`: Script that loads in the data and performs preprocessing such as ordering and removing errors as well as a conversion between data types.
- `word-grouping-dict.py`: Script that splits and sorts the original captured data in the shorter time series described in chapter 4
- `normalisation.py`: A module containing a simple standardisation function that takes categorical data into consideration.

The label scripts, are simple scripts containing a dictionary with the information needed in order to load in and label packets, with either binary or multi class labels. These scripts are:

- `label_dict.py`
- `bin_label_dict.py`

Furthermore, the script `skype_monitoring.py`, is a passive logging script for Skype, utilising change in UDP connections – Tested for Ubuntu 16.04 but should work for windows – Note the script starts a background thread for the logging.

In order to train the autoencoder 2 scripts are needed.

- `pytorch_modules.py`: Module containing the costume pytorch functions and classes used for the training.
- `training.py`: script that trains the autoencoder using the classes and functions from `pytorch_modules.py`

Finally are the results from chapter 8 from the follow 3 scripts.

- `labelled_encoding.py`: Script that use a saved pytorch model, created by training `training.py`, in order to encode data and perform clustering.
- `FAMD.py`: scipts that transform the data through FAMD before performing clustering.
- `interpacket_standardisation.py`: Script that performs clustering on data behaving as from `interpacket.py`, but with standardisation.

The scripts are setup to running with little to no interaction. If a change is sought, for e.g. a variables such as the binary vs non-binary labelling, it can be done directly in the script.