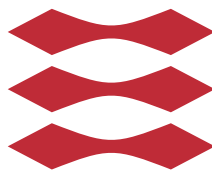


Inferring Pairwise Co-location from Noisy Bluetooth Signals

Constantin Teodor Gherghescu

DTU



Kongens Lyngby 2014
IMM-M.Sc.-2014-????

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk IMM-M.Sc.-2014-????

Summary (English)

The goal of the thesis is to provide a method which infers the existence of physical co-location based on Bluetooth RSSI measurements.

Summary (Danish)

Målet for denne afhandling er at ...

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfilment of the requirements for acquiring an M.Sc. in Computer Science and Engineering.

Lyngby, 20-June-2014

Not Real

Constantin Teodor Gherghescu

Acknowledgements

I would like to thank my supervisors Sune Lehmann Jørgensen, Jakob Eg Larsen, and Vedran Sekara for their guidance and feed-back. Additionally, I would like to thank Arek Stopczynski for his help with the database, and Piotr Sapieżyński and Ole Winther for their help with inferring methods.

Contents

Summary (English)	i
Summary (Danish)	iii
Preface	v
Acknowledgements	vii
1 Introduction	1
1.1 Motivation	1
1.2 Objective	2
1.3 Scope and limitations	3
1.4 Thesis Outline	3
2 Data acquisition	5
2.1 Bluetooth	6
2.2 FriendFinder app	6
2.2.1 App overview and implementation	6
2.2.2 Data	8
2.3 SensibleDTU data	10
2.3.1 Project overview	11
2.3.2 Data	11
2.4 Data merger	12
3 Methods for inferring pairwise co-location	15
3.1 Testing methodology	16
3.2 Neural Networks	17
3.2.1 Theoretical overview	17
3.2.2 Results	20

3.3	Logistic regression	25
3.3.1	Theoretical overview	26
3.3.2	Results	27
3.4	Naive Bayes	29
3.4.1	Theoretical overview	29
3.4.2	Results	31
3.5	Comparison between the algorithms	31
4	Conclusions	33
A	Stuff	35
	Bibliography	37

Introduction

1.1 Motivation

Epidemiology [1], personal health issues [2], group discovery [3], human mobility [4, 5], efficient team creation [6, ?], the analysis of academic success [7], network theory [8], also Fig. 1.1, and psychological research [9]; all of the above have begun making use of the same notion, one that is difficult to quantify [10, 11]: social connections and social interactions between individuals. And although co-location does not necessarily mean physical social interaction, it is a requirement for it. [12].

There are a number of approaches to determining co-location: self-reported data, a more traditional approach, which is prone to cognitive bias, social desirability bias and halo error [14, 15]. A more recent approach involves the use of data provided by modern means of communication, namely mobile phones. This data can come from both the actual phone, in the form of GPS locations or bluetooth readings [13], as well as from the phone company itself in the form of anonymous cell tower records [16, 17]. This has the advantage of being applicable almost anywhere, because of the high percentage of mobile phone penetration (95.5% estimated by the International Telecommunication Union in May 2014). However, there are approaches that yield better results, but come with financial, environmental or other types of additional cost: RFID tags [18],

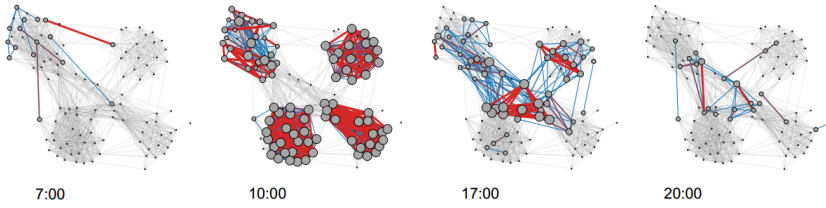


Figure 1.1: Face-to-face interactions over a day for college students. Blue means a low, and red a high frequency of interactions. Image from [13]

audio-video recordings [19], on-body sensors [20] and wifi signals [21], just to name a few.

1.2 Objective

Given a bluetooth RSSI between two phones, the purpose of this paper is to indicate a method which reliably and accurately determines the existence of pairwise co-location between the people carrying the two phones.

Reliability refers to the fact that multiple tries with the same input, and constant settings (same machine learning algorithm, same parameters and same training data), will always yield exactly the same result. While accuracy refers to the precision of the algorithm during testing, or how close to the expected result it is.

This inference is achieved by applying and analysing multiple machine learning algorithms on a data set consisting of two main parts:

- Data automatically recorded by the SensibleDTU data collector app [13]
- Ground truth data obtained by the test subjects by interacting with the FriendFinder app

For each machine learning algorithm the parameter configuration which yields the best results will be chosen, followed by a comparison between the best configurations for each algorithm.

1.3 Scope and limitations

The paper analyses the data obtained from three test subjects. Each subject has been given the same phone model, Samsung Galaxy Nexus. The app built for the phones, FriendFinder, has a purely functional purpose. As such, little to no consideration has been given to the style, theme or design of the app. Fig 2.1 shows the app, and while fit for purpose, it is not very aesthetic.

While considerable testing has been done with regards to the algorithm parameters, the machine learning algorithms list is by no means exhaustive. There are three main algorithms: Naive Bayes, Neural Networks, and Recursive learning, and an explanation on why a fourth, Hidden Markov Model, is unsuitable for this type of data.

When analysing data, we only look at the bluetooth RSSI value, and data derived directly while measuring it. For example, given that measurements are taken every five minutes, we at some point look at the length of an uninterrupted string of measurements, or at the measurements taken before and after (if possible) a specific measurement. GPS traces, phone records, infrared sensors, or facebook/email information are not taken into consideration.

1.4 Thesis Outline

The thesis begins with this introduction, which gives an overview of the general theme of the project. The objectives, scope and limitations and thesis outline are all self-defining.

It continues with a section which describes the data acquisition process and the methodology used to obtain the data from the SensibleDTU database. The section also describes the FriendFinder app, as well as its implementation.

Next, the machine learning algorithms are presented. For each algorithm a theoretical overview is given. The implementation details are discussed, followed by the results obtained by applying it to the data. At the end of this chapter, we do a comparative analysis between the algorithms, followed by the last section, the conclusions, where we will give the final results and recommendations.

CHAPTER 2

Data acquisition

The data used in the paper has been gathered over the course of three months, starting in early March and ending in late May. The data has been obtained with the help of three student volunteers. Each volunteer carried a phone, provided by the SensibleDTU project, the same project this thesis is a part of [22, 13]. The phones are Samsung Galaxy Nexus [23] and are running the Android operating system [24].

Each phone came with two apps. The first recorded regularly a multitude of information, such as Bluetooth RSSI, GPS traces, battery usage and cell tower information, and is a part of the SensibleDTU project [13], while the second allowed the volunteers to manually name the person they are in physical proximity with, and was developed during this thesis. The second app provided the ground-truth information that confirmed the existence of physical proximity.

This gave rise to two types of data. First, a continuous stream of regular measurements from the first app, and punctual messages from the second app. Below we will go into more detail about how exactly the two types of data look, how are they gathered and finally, how the two are combined to create a unitary set which serves as a basis for the machine learning algorithms.

One last thing to mention is that the volunteers were not chosen completely random, but in such a way that guaranteed the existence of both types of data.

On the one hand, the volunteers are all studying for a computer science major at DTU. This ensured that the students have found themselves in each others vicinity at some point during the data gathering period. Also, all three volunteers live in different places. This is important because house-mates or couples living together will introduce a very large amount of data that is very specific, while we are looking to generalize as much as possible. On the other hand, we needed to make sure that eventually, some volunteers are in each other's physical proximity. This was resolved by the fact that the volunteers had a course together. This ensured that at least once per week, usually two because of group work, the volunteers met.

2.1 Bluetooth

2.2 FriendFinder app

2.2.1 App overview and implementation

As the app, FriendFinder, is made for the Android operating system, it is implemented using Java for Android [25]. It has a Google App Engine backend [26]. Fig 2.1 shows the main screen of the app.

FriendFinder is implemented using a client server architecture, where the clients are the apps installed on the phone, and the server is the Google App Engine. Fig 2.2 shows an overview of this particular architecture. In this case, the FriendFinder app (in the role of the client) makes a *save data* request to the Google App Engine (the server). The server in turn responds with the result of the operation, either a *success* or *failure* message.

The app contains a single screen (the one showed in Fig 2.1), which corresponds to the main activity. Activities are the main building blocks of an Android app, and they are used both to interact with the user, as well as provide additional functionalities [27]. On the main screen there are buttons for each volunteer. Once one of the volunteers is in physical proximity with another volunteer (as perceived by either one), they both press the button corresponding to each other. A confirmation message is displayed, as can be seen in Fig. 2.3.

For the data to reach the database on the Google App Engine, an internet connection is required. However, the app can also function off-line, by saving all the data locally, and sending it to the on-line database as soon as an internet

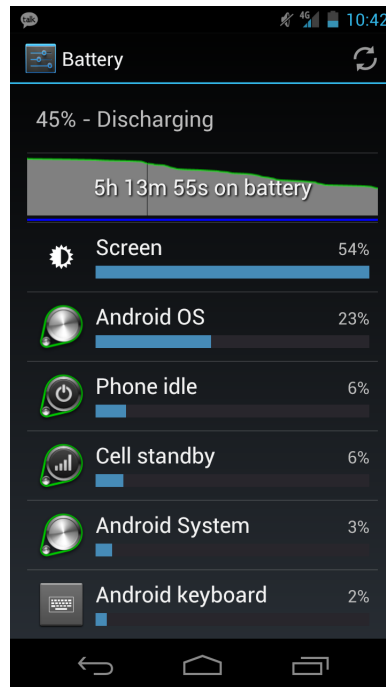


Figure 2.1: FriendFinder app

connection is established. It does this by using an `IntentService` [28], which has two main functions:

- A first function is to store the data locally, until an internet connection is established.
- the second, and most important, is to check periodically for an internet connections. Once an internet connection has been established, all the data is sent to the Google App Engine. The app checks every one minute for internet access. This allows for a relatively fast updating of the database, while at the same time keeping the resource use at a level that does not impede the normal functioning of the phone.

The data is stored locally in RAM of the phone (the volatile memory). This is made possible by the relatively small size of the data being saved (the names of the two people involved, and an ID object that also serves as a timestamp) and

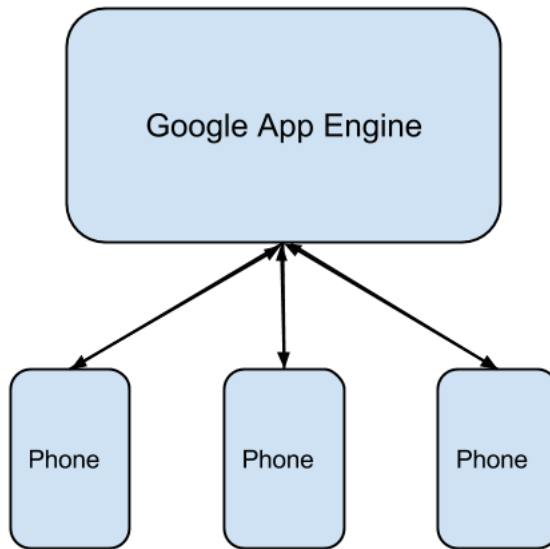


Figure 2.2: Client Server Architecture used by the FriendFinder app

the memory capacity of the phone (approximately 700 MB of RAM). However, this has the disadvantage of being vulnerable to the phone turning off due to lack of battery. The volunteers have been made aware of this fact.

Once the internet connection is established, the actual sending of the data is a simple matter, done with the help of the Google App Engine API. One thing to mention here is that each data entry is sent individually. If at any point, a *save data* request is met by a *failure* answer, the request is repeated until the *success* message is received. In case of *failure*, subsequent attempts have a one second delay between them. This is done to ensure that the app does not impede the overall functionality of the phone.

2.2.2 Data

Once a button with the name of a person is pressed, the data that is saved on the phone, and eventually sent and saved on the online database has the following format:

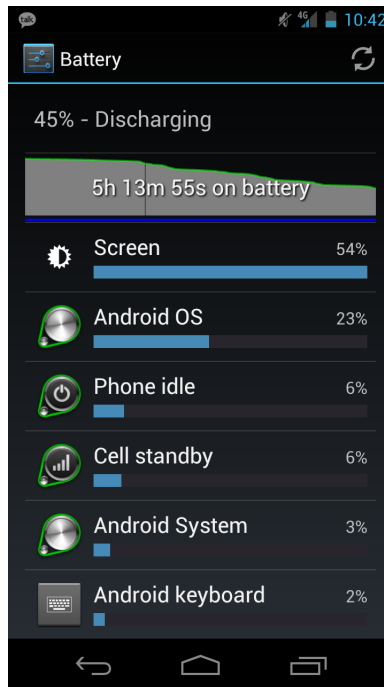


Figure 2.3: Confirmation message on the FriendFinder app

`{ID, owner, target}`

ID It has a double role. First, the ID is a unique identifier, used for differentiating between multiple entries, as well as for various database operations. This field is required by the Google App Engine. Secondly, the ID also plays the role of the timestamp, used in future computations. While there are valid concerns that the ID might not be unique, due to two people pressing the button at the same time, the fact that the timestamp also includes the millisecond, and the fact that there are only three participants in the project results in an extremely low probability of identical timestamps. Obviously, the timestamp corresponds to the moment the button was pressed, and not the moment the data was sent to the database.

owner This field refers to the owner of the phone, which indicates the person that has made the observation.

target This field names the person that has been observed by the owner.

Fig. 2.4 shows an example of how the data looks, with the caveat that the *friend* field in the image corresponds to the *target* field in the description above. The saved data was accessed and retrieved by API calls to the Google App Engine.

⏪ Prev 20 21-40 Next 20 ⏩		
<input type="checkbox"/> ID/Name	friend	owner
<input type="checkbox"/> name=2014-04-02 11:24:37.318	rafa	Magda
<input type="checkbox"/> name=2014-04-02 11:24:58.803	ferdinando	Rafa
<input type="checkbox"/> name=2014-04-02 12:52:05.905	ferdinando	Rafa
<input type="checkbox"/> name=2014-04-02 12:52:08.734	rafa	Ferdinando
<input type="checkbox"/> name=2014-04-06 10:35:34.509	magda	Rafa
<input type="checkbox"/> name=2014-04-06 10:37:02.683	magda	Ferdinando
<input type="checkbox"/> name=2014-04-06 10:37:04.752	rafa	Ferdinando
<input type="checkbox"/> name=2014-04-06 10:37:37.694	ferdinando	Rafa
<input type="checkbox"/> name=2014-04-06 10:44:09.723	bartosz	Rafa
<input type="checkbox"/> name=2014-04-06 13:01:40.166	ferdinando	Magda
<input type="checkbox"/> name=2014-04-06 13:01:42.319	rafa	Magda
<input type="checkbox"/> name=2014-04-07 20:50:01.430	rafa	Magda
<input type="checkbox"/> name=2014-04-07 20:50:06.456	magda	Rafa
<input type="checkbox"/> name=2014-04-09 09:56:46.845	bartosz	Rafa

Figure 2.4: Saved entries for the FriendFinder app

2.3 SensibleDTU data

The second type of data comes from the SensibleDTU data collector app. Based on [13, 22], we will give a short description of the project, with an emphasis on data collection. We will then focus on the actual data, its form, and how it was collected.

2.3.1 Project overview

SensibleDTU is the implementation of a large scale study aimed at observing the various interchanges that take place between humans from a social and communications point of view. It aims to collect data on a large group of individuals, most of them students at DTU (Danish Technical University). The collected data consists of, among others, social relations and the networks that form as a result, geographical locations, and, of special interest to this thesis, face-to-face interactions. It does this by using a wide variety of sources, such as questionnaires, the wireless internet on campus, and mobile sensing through smartphones that were handed out to participants. A part of the data gathered through this last method is used in this thesis.

The smartphone data gathering is done by using a data collection app, based on the Funf framework [29]. This data is saved locally on the phone, and it is sent to a secure database server when the app detects the presence of an internet connection. Due to its sensitive nature, great care has been taken to ensure the privacy, security and anonymity of the saved data. Having said that, the three volunteers that participated in the data gathering for this thesis have agreed to the de-anonymization of their Bluetooth information, as it was required in order to merge the two sets of data gathered.

2.3.2 Data

The Bluetooth data is obtained periodically by the data collection app using the Bluetooth probe. Each phone performs a Bluetooth scan every five minutes. The scan lasts 30 seconds. While the 30 second scan lasts, the app saves all Bluetooth devices (that are discoverable) in its proximity, the time at which the scan has been made, as well as the RSSI [30]. The data is stored on the phone from where it eventually reaches the secure database server, from where it can be downloaded.

In order for the data to be used, it first needs to be *cleaned*, as it contains additional meta-data (fields used by the database for internal accounting), and devices that are not part of the study. The end result has the following format:

```
{timestamp, owner, target,signal_strength}
```

The above tuple has almost the same format as the one resulted from the FriendFinder app, with the addition of a new field, signal strength. At a

first glance, one might think that the two sets of data can be easily combined. However, due to differences in the time when data gathering takes place, desynchronization and the human factor, obtaining a single data set from the two is not a trivial matter. The following section goes into more detail regarding this issue.

2.4 Data merger

With the data merger we aim to obtain a single data set that consists of ordered entries of the following form, for each participant:

```
{timestamp, signal_strength, tag}
```

Where *signal_strength* refers to the RSSI, *timestamp* refers to the timestamp provided by the SensibleDTU data collector app, and *tag* is either *yes* or *no*. The *tag* has the following meaning:

- yes** This tag means that the data has been confirmed through the use of the FriendFinder app. This means that we have co-location.
- no** This tag mean that the data has not been confirmed through the use of the FriendFinder app. While the Bluetooth probe has discovered the device, and a RSSI exists, there is no co-location. The following are examples of situations where this might happen: same room but a relatively large distance between the phones (cafeteria, large course rooms), transitional contacts (volunteers passing by each other), adjacent rooms with signal weakened by the walls.

A first step is separating the SensibleDTU data on an user by user basis. For each volunteer, we obtain two timelines, which detail his interaction with the other two volunteers. Although measurements are taken every 5 minutes, the phones eventually become desynchronized. As such, we split the measurements into time windows, each window having a length of five minutes. This value was chosen because it matches with the period at which measurements are taken by the Bluetooth probe and gives a natural separation of the values obtained. Fig. 2.5 shows a small part of a timeline with the splitting already done.

The above notion of tagging each timestamp now can be resolved by tagging each time window. The volunteers have been instructed to confirm the co-location once at the beginning of the meeting. In case of prolonged meetings,

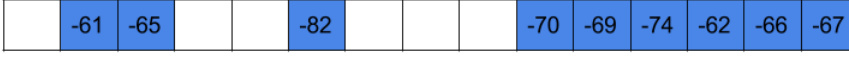


Figure 2.5: Example of timeline for a single volunteer. The time windows are easily visible, as well as the RSSI value measured for each one. The values are expressed as *dBm*.

with separations and reunions, the volunteers were instructed to confirm again after each separation/reunion.

In tagging the windows, we have used the notion of measurement chain. We define it as a continuous, ordered series of one or more time windows, where we have a RSSI value measured for every window in the chain. For example, in Fig. 2.5 we have three distinct measurement chains: the first one with two windows, the second one with only one window, and the last one with six windows.

We tag a window w with *yes* if a FriendFinder app confirmation has been done in any window belonging to the chain that window w also belongs, or the window immediately before, even if that window has no measurement. This last part is done to make sure that confirmations done in between Bluetooth probe measurements are not lost. Fig. 2.6 exemplifies how windows are tagged with *yes*.

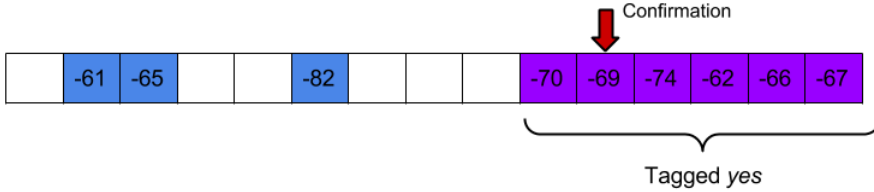


Figure 2.6: Example of how tagging takes place as a result of a confirmation by FriendFinder app data.

After all the confirmations are taken into account and all the relevant chains are tagged with *yes*, everything that remains is tagged with *no*. In the end we are left with a total number of 465 entries, Fig. 2.7 showing how they are divided:

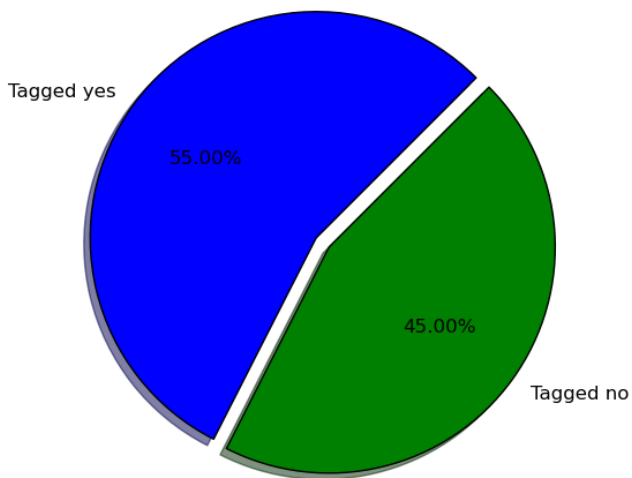


Figure 2.7: The partitioning of the entries by *yes* and *no* tags.

CHAPTER 3

Methods for inferring pairwise co-location

With the data extracted and formatted in suitable manner, as described in the previous chapter, we move on to the main part of the thesis. Given a set of consecutive Bluetooth RSSI measurements, *can we infer co-location?*; and if yes, what is the best way of doing it?

One way of answering these questions is to simply set a threshold, and split the dataset into two parts: the ones that are above the threshold indicate co-location, while the other do not. That is exactly what *Sekara,Lehmann* did in [30], with convincing results. They set the threshold to *-80 dBm*.

On a similar note, we average our two types of data, the ones tagged *yes* and *no*, and we obtain an average of *-64 dBm* for the ones tagged *yes* and an average of *-82 dBm* for the ones tagged *no*. From this result, one can easily see the similarity between the threshold in *Sekara,Lehmann*'s work and the values obtained through this thesis's data.

The above method can successfully be used under any type of measurements. However, it does not take advantage of the additional information that our method of data gathering has provided, such as temporal information, or the significance of continuous measurements. Below we propose three methods of

inferring pairwise co-location in the form of three machine learning algorithms. For each one we will provide a theoretical overview and implementation details, followed by a discussion of the results. The chapter will end with a comparison between the three algorithms where results, ease of use and performance will be taken into consideration.

In Section 2.4 we split our data into time windows. At any one point during the execution of the algorithms, we will be referring to a single time window, the one for which we want to infer co-location. Of course, additional parameters and features will be used, but even if not specified, every run of any algorithm described below will include, even if not specifically noted, at least the RSSI of a single time window, the one mentioned at the beginning of this paragraph.

While the parameters and the inputs used differ from one algorithm to another, one thing that remains constant across all of them is the testing methodology, described in the following section.

3.1 Testing methodology

In order to test the accuracy of the algorithms, we use cross-validation, a well recognized method for computing accuracy estimations [31]. The technique is used here for obtaining an accuracy score for each algorithm and its variations, so that a comparison can be made at the end. Broadly speaking, cross-validation refers to the techniques used to partition the main dataset into two sets: a *training* set, used in the initial phase for training the algorithms, and a *testing* set, used for validating the algorithm with *unseen before* data, after the training has finished. Two methods were used: k-fold validation, with $k = 10$ [31], and repeated random sub-sampling validation, with a 80 – 20 proportion [32].

- K-fold validation is done by partitioning the data in k sets. One set is used for testing, while the other $k - 1$ are used for training. The process is repeated k times, once for each partition. The end result is the average over all k results obtained.
- For random sub-sampling validation, the dataset is partitioned into two sets. The entries are randomly assigned to one of the sets based on the proportion established at the beginning. For this type of validation, the test has been repeated 100 times, and the results averaged.

During the execution of the tests, both k-fold validation and repeated random sub-sampling validation output extremely similar results, most of the time the

results having less than 0.5% difference between them. For this reason, when presenting an algorithm score, that score will be the average of the two methods.

3.2 Neural Networks

A first approach aims to use artificial neural networks (ANN) in order to infer co-location. Specifically, this section will deal with feed-forward ANNs, both from a theoretical and a practical point of view.

3.2.1 Theoretical overview

The introduction of backpropagation by Rumelhart et al. [33] paved the way for the development of the modern artificial neural network. In their current form, ANNs can be described as a set of processing units, called *neurons*, or *artificial neurons* that communicate between each other.

The general layout of a feed-forward ANN can be seen in Fig. 3.1. It consists of an input layer, one or more hidden layers and an output layer. The number of *neurons* on the input layer is equal with the number of features that represent the object we need to classify. The number of *neurons* in the output layer usually equals the number of classes the objects need to be classified in. The number of hidden layers, and the number of *neurons* in those layers varies greatly from problem to problem, and will be discussed in the next section, which tackles the implementation details. In addition to the processing units, the other major components of a feed-forward network are the links between the *neurons*. The type of ANN described here is feed-forward, which means the links are unidirectional. Information only moves from the input *neurons*, to the *neurons* in the hidden layers, to the *neurons* in the output layers. For example, in Fig. 3.1, information only moves from left to right [34].

An artificial neural network gets its name from the similarity it has with the biological network of neurons one finds in the human or animal brain. The same can be said about the way it works. Each *artificial neuron* receives an input from all the other *neurons* on the previous layer (or they receive outside input if they are on the input layer). Once all the inputs are received, the *neuron* in question processes them and outputs the result to *neurons* on the next layer [35]. Fig. 3.2 shows the details of an *artificial neuron*.

Each link between *neurons* is characterized by a weight, w_i . A *neuron* applies

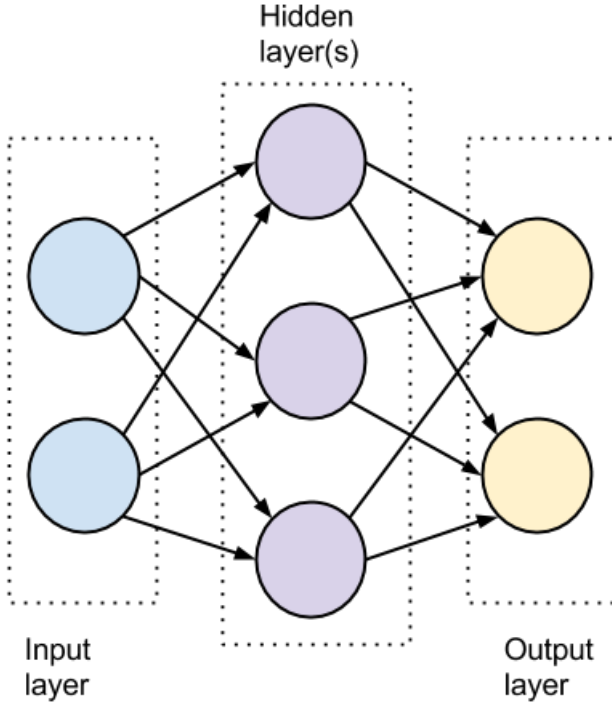


Figure 3.1: General structure of an ANN.

a so-called *activation function* on the weighted sum of the inputs, and outputs the result. Sometimes, a bias is used: θ . Thus, the output of single *neuron* can be expressed as:

$$y = F\left(\sum_i w_i y_i + \theta\right)$$

y The output of the current *neuron*.

F The activation function. This is usually either the *sigmoid* function, $y = F(x) = \frac{1}{1+e^{-x}}$, or the *tanh* function [35].

w_i The weight of an input link.

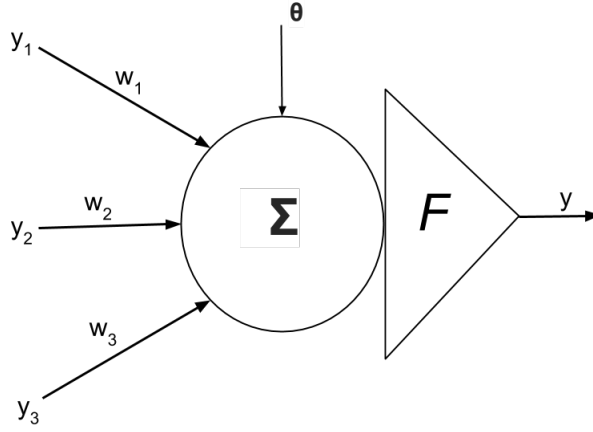


Figure 3.2: Structure of an artificial neuron.

y_i The input, either from a previous layer of *neurons*, or a from outside the network.

θ Bias

In order for the network to output the expected result when presented with an input, it has to be set up. That translates into assigning appropriate values to the weights corresponding to the links between *neurons*. One way of doing this is to manually assign the values. However, this implies a priori knowledge, which is not possible in most cases. The other choice is to *train* the network.

The training method used here is backpropagation, first introduced in [33]. The intuition behind is as follows. We randomly assign values to the weights w_i in the ANN. We apply an input to the network, whose result we already know, and compare the outputted result with the correct one. The aim is to minimize the difference.

While the mathematical procedure through which researchers have reached backpropagation is outside the scope of this thesis, based on [35] we will present the general equations used in the implementation of the algorithm:

We modify each weight with:

$$\Delta w_k = \alpha \delta_k y_k$$

Where α is the learning rate, y_k is the input value on link k , and δ_k is computed as follows:

$$\delta_k = F'(\sum_i w_i y_i + \theta) \sum \delta_l w_{kl}$$

Where δ_l is used in the next layer, and w_{kl} represents a weight between the current layer and the next one. This process continues recursively, until we reach the output layer. In order to finish the computations, we need the δ_o which corresponds to the output layer:

$$\delta_o = (d - y) F'(\sum_i w_i y_i + \theta)$$

Where d represents the expected output, and y the desired output. As one can easily see, we begin from the output layer with computing δ_o , and updating the corresponding weights. We then move further one layer, and compute the corresponding δ values. We continue to go back until we reach the links that connect to the input layer. From this procedure the *back* in *backpropagation* comes from.

3.2.2 Results

For implementation purposes we used a Python neural network, available at [36]. The network used has, besides the mandatory input and output layers, a single hidden layer (which is enough [37, 38, 39]).

The number of *neurons* in the hidden layer differs greatly from problem to problem, with some authors suggesting any number between the number of *neurons* in the input layer, to up to two times that value [40]. Due to the relatively small number of features extracted from the phone data, we have run tests varying numbers of hidden units in the above mentioned range.

During the testing, different values for the α parameter have been tried, in the end, the values that yielded the best results being between 0.15 and 0.4, with little variation. As such, all the tests that are done below are done with an α parameter of 0.2.

We begin our testing with the most basic case. We use a single feature, the measured Bluetooth RSSI. The plot in Fig. 3.3 details the results obtained. While the accuracy is acceptable, we can easily see that varying the number of hidden nodes in the hidden layer has no impact on the result.

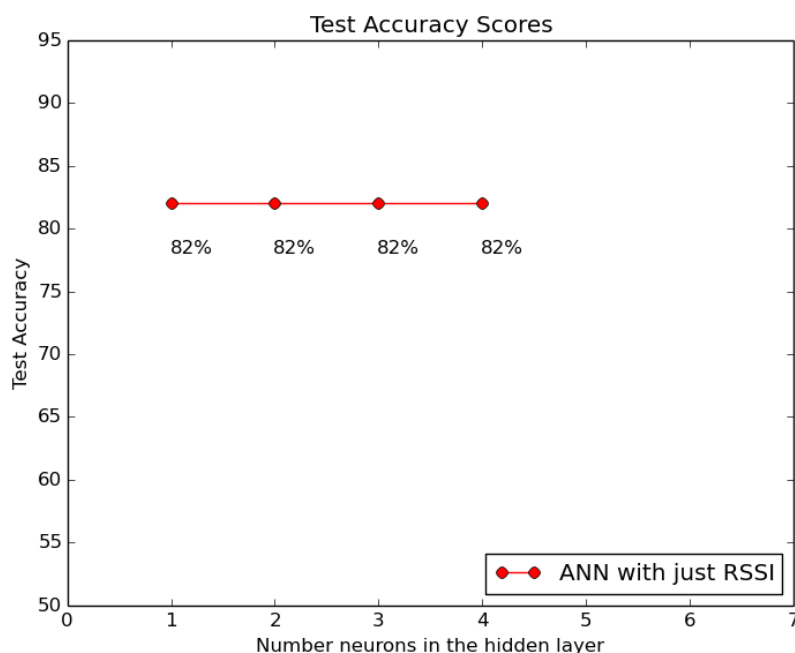


Figure 3.3: Test accuracy for ANN when just the RSSI is used as a feature.

Next we try to make use of the notion of chains, as it was defined in Section 2.4. When trying to infer co-location for a specific time window, we also look at the RSSI of the windows that are in the same chain, but before it. We gradually increased the number of *neurons* in the hidden layer, as well as the number of windows we take into consideration.

As we increased the number of windows, the length of the chains becomes a problem. Different chains have different lengths, but the number of features in the ANN remains constant. For example, when using four previous windows, and a chain has only a length of three, even in the best case scenario, we still have two missing features.

One way to deal with this problem is to simply discard the cases that are missing values. However, this introduces significant bias into the data, as observation of the dataset has shown that shorter chains are more likely to belong to the category tagged with *no*. By discarding them, we greatly reduce the number of samples tagged with *no*, thus unbalancing the overall proportion of training

and testing data. As such, we used imputation to replace the missing values. Two approaches were used.

A first approach used mean imputation. We replaced the missing data with the average of the values in the chain. Fig. 3.5 shows the results obtained. Secondly, we used last observation carry forward [41], as it involved items ordered by time value. The results can be seen in Fig. 3.4. Consideration has been given to multiple imputation [42], however, its cost, and the next result made it unsuitable.

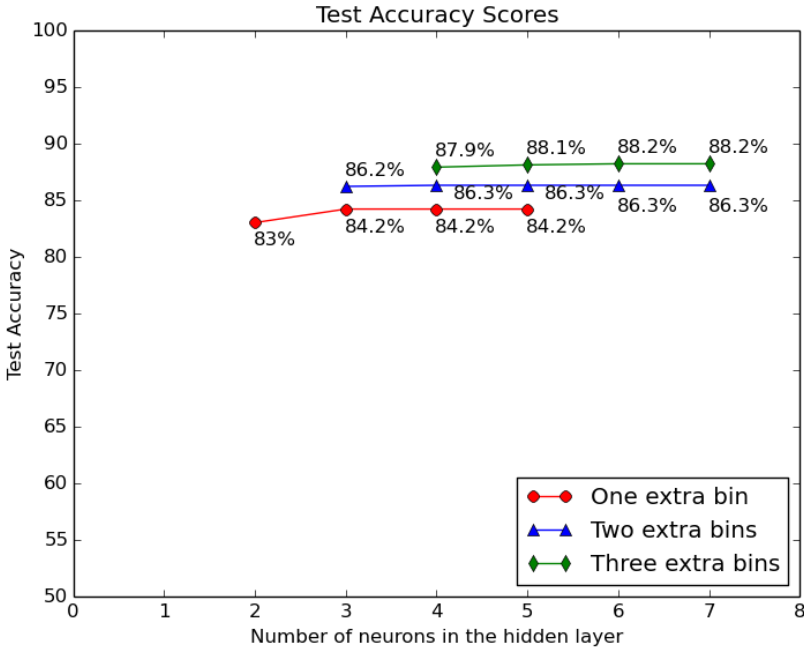


Figure 3.4: Test accuracy for ANN when the RSSI for the current and previous time windows are used as features, with mean imputation.

As we increase the number of time windows, for mean imputation, the accuracy increases. However, this is due to the uniformization of data, as every value that is missing is replaced with the average. As the average is almost the same between the testing set and the learning set, we end up with entries who sometimes look extremely alike. When entries are full of averages, the difference is made by the real data. As the number of time windows with real data decreases, this choice of parameters comes closer and closer to the basic one. However, even

with one extra bin, this choice of parameters is an improvement over it, as every extra time windows has the chance of providing new data.

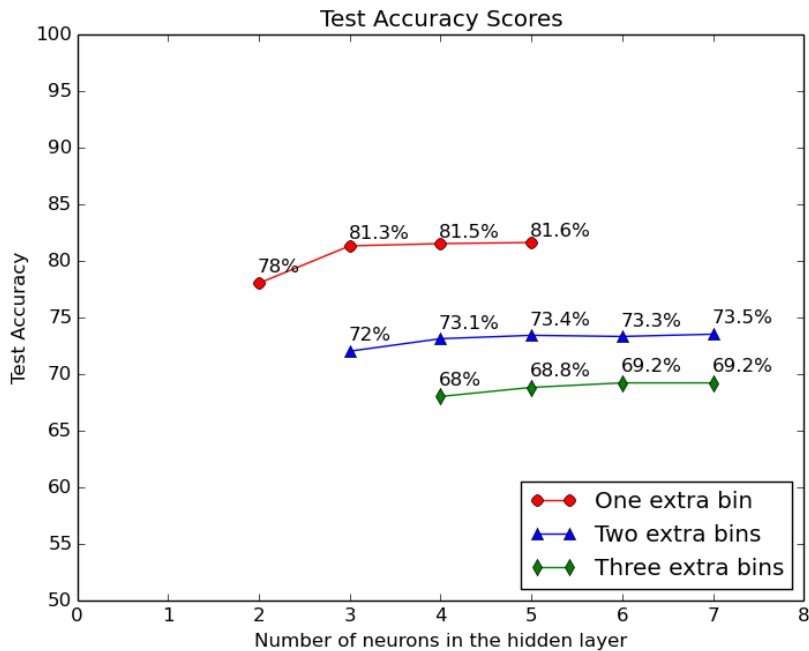


Figure 3.5: Test accuracy for ANN when the RSSI for the current and previous time windows are used as features, with last observation carry forward imputation.

Running the algorithm with a last observation carry forward strategy for replacing missing values produces negative results. Even taking into consideration the best result of this set, obtained with only one extra time window, we are still below or close to the score of the basic set. As we increase the number of time windows used, the accuracy drops markedly, with an almost 7% jump downwards between one and two extra time windows, and a 4% drop between two and three extra time windows. This choice of parameters is unsuitable for this algorithm.

Taking into account the additional information provided by the chain that a window belongs to can improve the result. However, the previous approach had a number of issues, related to missing values, and the choice of how many extra time bins should be taken into consideration. For the next set of parameters,

we only use two features: the RSSI of the current window, and the length of the chain the window is a part of. This has the strength of avoiding any bias introduced by imputation, while at the same time making almost full use of the additional information provided by the chain. Fig. 3.6 shows the test accuracy for this choice of features.

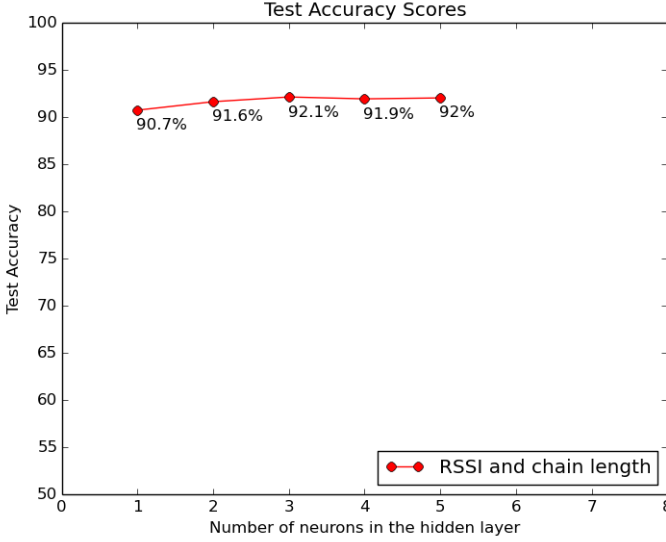


Figure 3.6: Test accuracy for ANN when the RSSI and the chain length are used as features.

Finally, Fig. 3.7 shows a plot with a comparison between the best results for each choice of parameters. Clearly, using just the current window RSSI and the chain length is the best choice. While providing just a minor increase in computation cost by adding just one extra new feature, it is better than the basic case by more than 10%. The parameter set using mean imputation comes close, but still produces inferior results. Furthermore, the use of imputation raises questions regarding the validity of the obtained result, as well as over-fitting.

Across the entire suite of tests, the variation in the number of *neurons* had little to no impact on the overall accuracy of the tests. Even with a minimum number of *neurons*, the results were extremely close to cases where four or five were used. The extra *neurons*, however, did have an impact on the overall performance. The difference between using three *neurons* instead of one translated into a doubling of the training and running time.

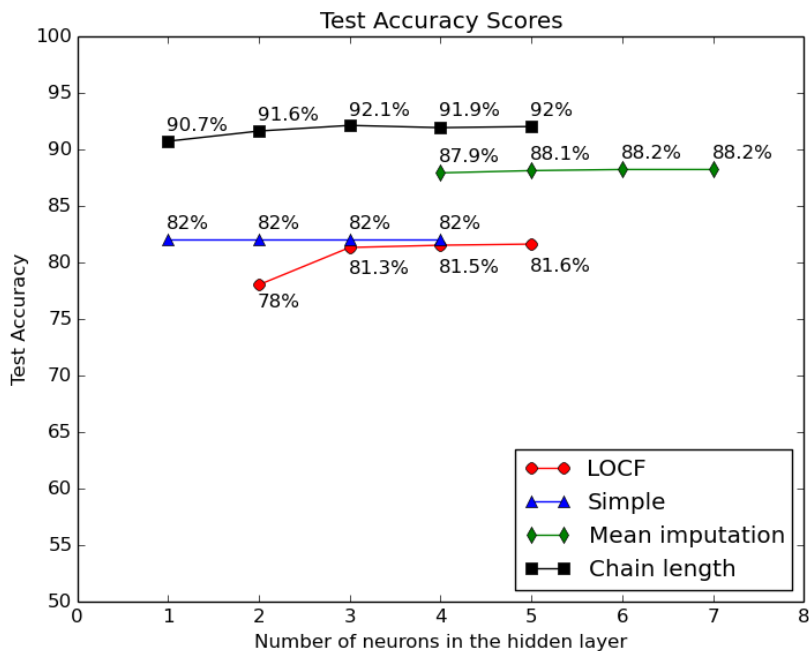


Figure 3.7: Comparison between test accuracy for the best cases for each set of parameters.

3.3 Logistic regression

In order to meet the objective stated at the beginning of this thesis, we look at the measured Bluetooth RSSI, and some adjacent data, and we try to decide if we have co-location. This can be expressed as the output(response) variable is depending, in some form, on the input(explanatory) variables ???. This immediately leads to the idea of regression. The most basic example of this is the linear regression. However, linear regression has an output variable that is continuous, which is unsuitable for the problem raised by this thesis.

Indeed, here we have a dichotomous outcome. Given a set of measurements, there either is or there isn't co-location. For these types of situation, a different type of regression is required, namely logistic regression.

3.3.1 Theoretical overview

Although usually regressions are used to either predict or estimate a certain value, logistic regression is actually a classifier, thus suitable for our problem.

The foundation of the logistic regression consists of the logistic function:

$$f(y) = \frac{1}{1 + e^{-y}}$$

Fig. 3.8 shows the plot of the logistic function. It has several attractive qualities. Based on ??:

- Its output values are between 0 and 1. Thus it can easily map probabilities, which is the result of the logistic model.
- Looking at Fig. 3.8, we can discern the presence two points after which the function increases(decreases) significantly, which can be considered activation thresholds, when looking from the point of view of a classifier.

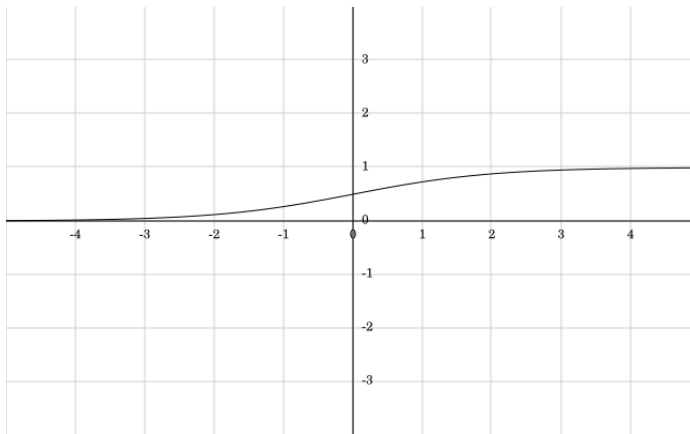


Figure 3.8: Plot of the logisitic function.

Based on [43, 44, 45]: for a set of measurements, the logistic model computes the probability that they belong to either one of the categories. We wish to model the dependence between X , which consists of a set of variables $X = \{X_1, X_2, X_3, \dots\}$, which express the features, and $p(X) = Pr(Y|X)$, where Y

represents the classifier labels. Using the above logistic function, we can write:

$$p(X) = \frac{1}{1 + e^{-X}}$$

But X can be written as a linear combination of the input variables: $X = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \dots$. Replacing in the above equation we obtain:

$$p(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 \dots)}}$$

We also need to compute the *logit*, the inverse of the above function, which looks very much like a regular linear regression:

$$\ln \frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \dots$$

All that is left now is to estimate the unknown quantities $\beta_0, \beta_1, \beta_2 \dots$, through the process of training. This is usually done with maximum likelihood estimation [46]. Intuitively, the method looks for estimates of $\beta_0, \beta_1, \beta_2 \dots$, so that the results obtained using those estimates come very close (or as close as they can) to the expected result. The maximum likelihood estimation makes use of the likelihood function:

$$l(\beta) = \prod_i p(X_i)[1 - p(X_i)]$$

Where the first term corresponds to cases that belong to one category, and the second to cases that belong to the other category. The estimations for $\beta_0, \beta_1, \beta_2 \dots$ are chosen so that the above function is maximized. Solving the above equation entails the use of computationally intensive iterative methods [44], and is outside the scope of this paper. Once discovered, the estimations are plugged back in the logit, and the model can be used for classification.

3.3.2 Results

We used the Scikit-learn [47] implementation for the logistic regressor. For testing purposes we used the same sets of parameters as before. The results can be seen in Fig. 3.9.

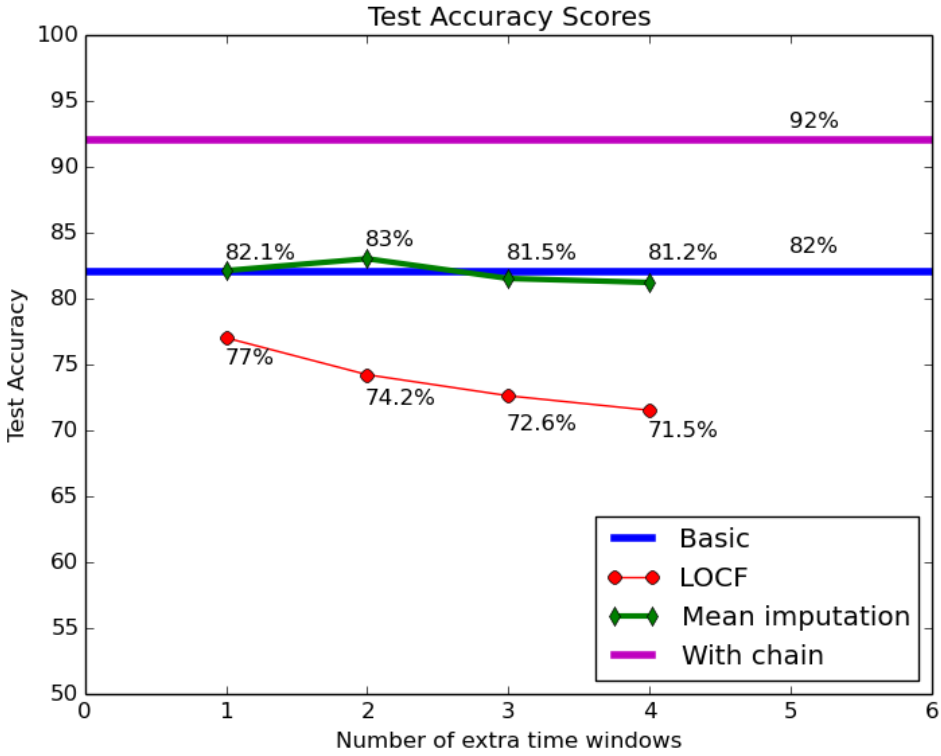


Figure 3.9: Testing results for logistic regression.

For the basic case, where only the Bluetooth RSSI measurement is used, the accuracy reaches 82%. We then continue the tests with the use of multiple time windows. Again, in case of missing windows, we use imputation, both mean imputation and last observation carry forward.

Increasing the number of time windows used has little effect on the mean imputation parameter set, it peaking at 83%, when two extra time windows were used. This value comes very close to the score obtained with basic set, and also introduces a high amount of computational overhead, from the algorithm (as we add more features, more values need to be estimated), as well as the database structure (as the chain structures need to be traversed). Overall, this choice of parameters brings little against the basic set.

The same cannot be said by the variation that uses last observation carry forward. Increasing the number of time windows taken into consideration has a significant effect on the overall accuracy of the algorithm. Unfortunately, it is not a positive one. As we increase that number, the accuracy decreases dramatically. Even with one extra time window, the score we obtain is lower than even the basic case. When we move to two extra windows, the score drops to 74.2%, to 72.6% for three windows, and 71.5% for four windows, and the trend continues. Clearly, this choice of parameters is not suitable for this particular algorithm.

The last choice of parameters is the Bluetooth RSSI of the current time window and the length of the chain it belongs to. This set of parameters provides a significant improvement over all the other cases, with an accuracy of almost 92%. In addition to the improved test score, this choice requires only one new feature to be added to the algorithm, in comparison to the base case.

Although the results obtained using mean imputation are usable, the fact that they are almost identical to the base case gives us no reason to chose it. Overall, the chain length has the most impact on accuracy, while at the same time avoiding the pitfalls of imputation, making it the best choice for logistic regression.

3.4 Naive Bayes

Given that the two algorithms discussed above outputted similar results in their best cases, we considered using a more simple algorithm to check if we obtain the same results. Naive Bayes was the choice.

3.4.1 Theoretical overview

Naive Bayes is a probabilistic classifier [48]. At its core, the Naive Bayes classifier uses Bayes theorem:

$$p(Y|X) = \frac{p(Y)p(X|Y)}{p(X)}$$

In case of the classifier, Y is a variable over the classes the data needs to be classified in, where X is a variable representing the features. X can also

be replaced by multiple variables expressing the presence of multiple features: $X_1, X_2, X_3, \dots, X_n$. Based on [48], replacing this in the theorem we obtain:

$$p(Y|X_1, X_2, X_3, \dots, X_n) = \frac{p(Y)p(X_1, X_2, X_3, \dots, X_n|Y)}{p(X_1, X_2, X_3, \dots, X_n)}$$

One thing to mention here is that when used for classification by comparing values, for the above model, the item located at the denominator is constant (that is, it does not depend on Y), and does not need to be computed.

And here, the *naive* part of the classifier comes in. We assume that the feature variables are independent (the assumption is obviously wrong, however, it has been shown to have good results in practice [49]). Thus we can write:

$$p(X_1, X_2, X_3, \dots, X_n|Y) = \prod_{i=1}^n p(X_i|Y)$$

Replacing in the equation above, we obtain:

$$p(Y|X_1, X_2, X_3, \dots, X_n) = \frac{p(Y) \prod_{i=1}^n p(X_i|Y)}{p(X_1, X_2, X_3, \dots, X_n)}$$

This gives us the model. It computes the probabilities that a certain feature vector, representing an object, belongs to either category. While the probabilities calculated are not correct, the relation between them is valid. That is, if an object to be classified has a higher probability of belonging to class A compared to the probability of belonging to class B , than that object probably belongs to class A .

The parameters are estimated through training, using maximum likelihood [50]. However, depending on the type of data we wish to classify, we need to assume a specific feature distribution, usually called *event model* [51]. For continuous data, the Gaussian distribution is usually used, while for discrete features Bernoulli and multinomial distributions are used [51].

3.4.2 Results

3.5 Comparison between the algorithms

CHAPTER 4

Conclusions

APPENDIX A

Stuff

This appendix is full of stuff ...

Bibliography

- [1] L. A. N. A. H. E. S. Y. b. Fredrik Liljeros, Christofer R. Edling, “The web of human sexual contacts,” 2001.
- [2] A. Madan, S. T. Moturu, D. Lazer, and A. S. Pentland, “Social sensing: Obesity, unhealthy eating and exercise in face-to-face networks,” in *Wireless Health 2010*, WH '10, (New York, NY, USA), pp. 104–110, ACM, 2010.
- [3] S. Mardenfeld, D. Boston, S. Pan, Q. Jones, A. Iamntichi, and C. Borcea, “Gdc: Group discovery using co-location traces,” in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pp. 641–648, Aug 2010.
- [4] J. Sun, J. Yuan, Y. Wang, H. Si, and X. Shan, “Exploring space–time structure of human mobility in urban space,” *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 5, pp. 929 – 942, 2011.
- [5] A. Sevtsuk and C. Ratti, “Does urban mobility have a daily routine? learning from the aggregate data of mobile networks,” *Journal of Urban Technology*, vol. 17, no. 1, pp. 41–60, 2010.
- [6] O. Bandiera, I. Barankay, and I. Rasul, “Social connections and incentives in the workplace: Evidence from personnel data,” *Econometrica*, vol. 77, no. 4, pp. 1047–1094, 2009.
- [7] D. Blansky, C. Kavanaugh, C. Boothroyd, B. Benson, J. Gallagher, J. En-dress, and H. Sayama, “Spread of academic success in a high school social network,” *PLoS ONE*, vol. 8, p. e55944, 02 2013.

- [8] Y. ong-Y. eol Ahn, J. ames P. . Bagrow, and S. une Lehmann, "Link communities reveal multiscale complexity in networks," vol. - 466, no. - 7307, pp. - - 764, - 2010/08/05/print.
- [9] K. K. Rachuri, M. Musolesi, C. Mascolo, P. J. Rentfrow, C. Longworth, and A. Aucinas, "Emotionsense: A mobile phones based adaptive platform for experimental social psychology research," in *Proceedings of the 12th ACM International Conference on Ubiquitous Computing, Ubicomp '10*, (New York, NY, USA), pp. 281–290, ACM, 2010.
- [10] G. ergely Palla, A. lbert-L. aszlo Barabasi, and T. amas Vicsek, "- Quantifying social group evolution," vol. - 446, no. - 7136, pp. - - 667, - 2007/04/05/print.
- [11] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys '09*, (New York, NY, USA), pp. 205–218, ACM, 2009.
- [12] N. Eagle, A. S. Pentland, and D. Lazer, "Inferring friendship network structure by using mobile phone data," *Proceedings of the National Academy of Sciences*, vol. 106, no. 36, pp. 15274–15278, 2009.
- [13] A. Stopczynski, V. Sekara, P. Sapiezynski, A. Cuttone, J. E. Larsen, and S. Lehmann, "Measuring large-scale social networks with high resolution. working paper," *CoRR*, vol. abs/1401.7233, 2014.
- [14] S. Wuchty, "What is a social tie?," *Proceedings of the National Academy of Sciences*, vol. 106, no. 36, pp. 15099–15100, 2009.
- [15] R. M. Gonyea, "Self-reported data in institutional research: Review and recommendations," *New Directions for Institutional Research*, vol. 2005, no. 127, pp. 73–89, 2005.
- [16] J.-P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.-L. Barabási, "Structure and tie strengths in mobile communication networks," *Proceedings of the National Academy of Sciences*, vol. 104, no. 18, pp. 7332–7336, 2007.
- [17] P. Hoevel and A.-L. Barabasi, "Temporal and spatial regularity of mobile-phone data," in *APS Meeting Abstracts*, p. 54005, Feb. 2012.
- [18] A. Barrat and C. Cattuto, "Temporal networks of face-to-face human interactions," in *Temporal Networks* (P. Holme and J. Saramäki, eds.), Understanding Complex Systems, pp. 191–216, Springer Berlin Heidelberg, 2013.

- [19] D. Wyatt, T. Choudhury, and H. Kautz, "Capturing spontaneous conversation and social dynamics: A privacy-sensitive data collection effort," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV-213–IV-216, April 2007.
- [20] D. Olguin, B. Waber, T. Kim, A. Mohan, K. Ara, and A. Pentland, "Sensible organizations: Technology and methodology for automatically measuring organizational behavior," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 39, pp. 43–55, Feb 2009.
- [21] M. B. Kjærgaard and P. Nurmi, "Challenges for social sensing using wifi signals," in *Proceedings of the 1st ACM Workshop on Mobile Systems for Computational Social Science*, MCSS '12, (New York, NY, USA), pp. 17–21, ACM, 2012.
- [22] "Sensibledtu." <https://www.sensible.dtu.dk/>, June 2014.
- [23] "Samsung galaxy nexus." <http://www.samsung.com/us/mobile/cell-phones/GT-I9250TSGGEN>, June 2014.
- [24] "Android." <http://www.android.com/>, June 2014.
- [25] "developer.android.com." <http://developer.android.com/>, June 2014.
- [26] "Google app engine." <https://developers.google.com/appengine/>, June 2014.
- [27] "Android activity." <http://developer.android.com/reference/android/app/Activity.html>, June 2014.
- [28] "IntentService." <http://developer.android.com/reference/android/app/IntentService.html>, June 2014.
- [29] "funf, the open sensing framework." <http://www.funf.org/>, June 2014.
- [30] V. Sekara and S. Lehmann, "The Strength of Friendship Ties in Proximity Sensor Data," *ArXiv e-prints*, Jan. 2014.
- [31] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, vol. 14, pp. 1137–1145, 1995.
- [32] T. Segaran, *Programming Collective Intelligence Building Smart Web 2.0 Applications*. O'Reilly Media, 2007.
- [33] D. avid E. . Rumelhart, G. eoffrey E. . Hinton, and R. onald J. . Williams, " - Learning representations by back-propagating errors," vol. - 323, no. - 6088, pp. - - 536, - 1986/10/09/print.

- [34] S. Cross, R. Harrison, and R. Kennedy, "Introduction to neural networks," *The Lancet*, vol. 346, no. 8982, pp. 1075 – 1079, 1995.
- [35] B. Kröse and P. van der Smagt, "An introduction to neural networks," 1993.
- [36] "Neural network toolbox." <https://github.com/IssamLaradji/NeuralNetworks>, June 2014.
- [37] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.
- [38] E. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with gaussian hidden units as universal approximations," *Neural Comput.*, vol. 2, pp. 210–215, Apr. 1990.
- [39] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [40] D. Stathakis, "How many hidden layers and nodes?," *International Journal of Remote Sensing*, vol. 30, no. 8, pp. 2133–2147, 2009.
- [41] J. Shao and B. Zhong, "Last observation carry-forward and last observation analysis," *Statistics in Medicine*, vol. 22, no. 15, pp. 2429–2441, 2003.
- [42] D. B. Rubin, *Multiple imputation for nonresponse in surveys*, vol. 307. John Wiley and Sons, 2009.
- [43] D. G. Kleinbaum and M. Klein, *Logistic regression: a self-learning text*. Springer, 2010.
- [44] D. W. Hosmer, Jr., S. Lemeshow, and R. X. Sturdivant, *Introduction to the Logistic Regression Model*, pp. 1–33. John Wiley & Sons, Inc., 2013.
- [45] G. J. D. W. T. H. R. Tibshirani, *An Introduction to Statistical Learning*. Springer, 2013.
- [46] S. Menard, *Logistic regression: From introductory to advanced concepts and applications*. Sage, 2009.
- [47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

- [48] D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *Machine Learning: ECML-98*, vol. 1398 of *Lecture Notes in Computer Science*, pp. 4–15, Springer Berlin Heidelberg, 1998.
- [49] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, pp. 41–46, 2001.
- [50] H. Zhang, "The optimality of naive bayes," *AA*, vol. 1, no. 2, p. 3, 2004.
- [51] A. McCallum, K. Nigam, *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48, Citeseer, 1998.