UNIVERSIDAD EAFIT

Engineering School

Imputation Method Based on Recurrent Neural Networks for the Internet of Things

Graduation manuscript presented as partial requirement to obtain the

Master of Science in Engineering

AUTHOR:

Ing. Sebastián Rodríguez Colina

SUPERVISOR:

Ricardo Mejía-Gutiérrez, PhD.

February 2018

Abstract

The Internet of Things (IoT) refers to the new technological paradigm in which sensors and common objects, like household appliances, connect to and interact through the Internet. This new paradigm, and the use of Artificial Intelligence (AI) and modern data analysis techniques, powers the development of smart products and services; which promise to revolutionize the industry and humans way of living. Nonetheless, there are plenty of issues that need to be solved in order to have reliable products and services based on the IoT . Among others, the problem of missing data posses great threats to the applicability of AI and data analysis to IoT applications. This manuscript shows an analysis of the missing data problem in the context of the IoT, as well as the current imputation methods proposed to solve the problem. This analysis leads to the conclusion that current solutions are very limited when considering how broad the context of IoT applications may be. Additionally, this manuscript exposes that there is not a common experimental set up in which the authors have tested their proposed imputation methods; moreover, the experiments found in the literature, lack reproducibility and do not carefully consider how the missing data problem may present in the IoT. Consequently, the reader will find two proposals in this manuscript: i) an experimental set up to properly test imputation methods in the context of the IoT; and ii) an imputation method that is general enough as to be applied to several IoT scenarios. The latter is based on Recurrent Neural Networks, a family of supervised learning methods which have excel at exploiting patterns in sequential data and intrinsic association between the variables of data.

Keywords: Internet of Things, Incomplete Data, Missing Data, Imputation Methods, Recurrent Neural Networks, Machine Learning.

Resumen

El Internet the las Cosas (IoT) es un nuevo paradigma tecnológico, en el cual sensores y objectos comunes, como electrodomésticos, se conectan e interactúan a través de la Internet. Este nuevo paradigma, de la mano con técnicas de Inteligencia Artificial (AI) y técnicas modernas para el análisis de datos, hace posible el desarrollo de productos y servicios inteligentes; lo que promete revolucionar la industria y la forma de vida de los humanos. Sin embargo, existen muchos problemas que deben ser solucionados para poder contar con productos y servicios confiables basados en el IoT. Dentro de estos problemas, el problema de los datos faltantes impide la correcta aplicación de modernas técnicas the AI y análisis de datos en aplicaciones basadas en el IoT. Este escrito presenta un análisis del problema de los datos faltantes en el contexto del IoT, así como métodos de imputatción actuales propuestos a solucionar este problema. Del análisis se concluye que las soluciones actuales tienen grandes limitaciones si se considera lo amplio del contexto de las applicaciones basadas en IoT. El análisis también expone que no hay un marco experimental en común que pueda ser usado por los diferentes autores, y que los experimentos encontrados carecen de reproducibilidad y no consideran adecuadamente como el problema de los datos faltantes se presenta en el contexto en particular del IoT. De acuerdo con lo anterior, este escrito presenta dos propuestas principales: i) un marco experimental que permite evaluar adecuadamente los métodos de imputación que se pretendan evaluar en este contexto; y ii) un método de imputación que es lo suficientemente general como para ser aplicado en los diferentes scenarios del IoT. El método de imputación se basa en el uso de Redes Neuronales Recurrentes, una familia de métodos de aprendizaje supervisado que ha mostrado un buen desempeño explotando patrones de datos sequenciales y relaciones intrínsecas entre variables.

Palabras Clave: Internet the las Cosas, Datos Incompletos, Datos Perdidos, Métodos de Imputación, Redes Neuronales Recurrentes, Aprendizaje de Máquina.

State of Publications

During the development of this research project, I contributed to the following publications:

- Ruiz-Arenas, S., Rodríguez-Colina, S., Rusak, Z., Mejía-Gutiérrez, R., & Horváth, I. (2016). Design of a Low-End Cyber-Physical System TestBed For Testing a Failure Diagnosis Algorithm. In Proceedings of TMCE 2016 (pp. 1–9).
- Rodríguez-Colina, S., Zapata, C., Osorio-Gómez, G. & Mejía-Gutiérrez, R. (2017). An Experimental Analysis for Fault Detection and Diagnosis in the Internet of Things: A Home Refrigerator Case Study. Accepted for oral presentation at IMECE, 2017. (*)
- Rodríguez-Colina, S., Mejía-Gutiérrez, R. (201X). Recurrent Neural Networks as Imputation Method for the Internet of Things. Submitted to *IEEE Internet of Things Journal* (+).

(*) The presentation at this conference was accepted on July 2nd, 2017. However, due to financial matters, it was not possible to present at the conference and the article was hence withdrawn on August 10th, 2017. The article will be upgraded and submitted to an indexed journal.

(+) This article includes the most relevant content presented in this manuscript.

Nomenclature

Acronyms

AI	Artificial Intelligence.
ANN(s)	Artificial Neural Network(s).
AUEC	Area Under the Error Curve.
CPS(s)	Cyber-Physical System(s).
GRU	Gated Recurrent Units.
IoT	Internet of Things.
LSTM	Long-Short Term Memory.
MAE	Mean Absolute Error.
RNN(s)	Recurrent Neural Network(s).
WSN(s)	Wireless Sensor Networks(s).

Variables

$\hat{f}$	Estimated function relating inputs to outputs.
$\hat{X}$	A reconstructed matrix of data (without missing values).
$\hat{x}$	The predicted value of any variable in any time step.
$\mathbf{x}$	Vector-value input or observation in a data set.
$\mathcal{D}$	A data set.
$\mathcal{T}$	A data set used for testing supervised learning models.
ε	Inherent error between a particular prediction and the corresponding true value.
d	Number of variables in a data set.
f	True function relating inputs to outputs.
i	Index over the observations in a data set.
j	Index over the variables in a data set.
k	The probability that any variable (equal for all variables) is missing in a Bernoulli pattern of missing data.
l	Index over the missing values in a data set.
$l_{\not{O}}$	Range of possible lengths of consecutive missing observations in a row wise missing data pattern.
l_O	Range of possible lengths of consecutive non missing observations in a row wise missing data pattern.
m	Number of missing values in a data set.
n	Number of observations in a data set.
nan	Indicator of a missing value.
P	A vector of all P_j.

P_h The higher missing rate for the column wise missing data pattern.

P_j The probability that a particular variable is missing in a Bernoulli pattern of missing data.

P_l The lower missing rate for the column wise missing data pattern.

S_{miss} The set of indices of the variables that will be missing in the row wise missing data pattern.

X A matrix of data without missing values.

X^{nan} A matrix of data with missing values.

x_j Single-valued input or observation of variable.

y Single-valued output or prediction of a supervised learning model.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Computers are now much faster and smaller than they were twenty years ago: "current mobile devices have more computation capabilities than the Apollo Guidance Computer" (Kaku, 2011, pg. 10). In 1965 Gordon Moore (Roser and Ritchie, 2018) predicted that by reducing transistors size, computation would be exponentially cheaper and faster (see figure 1.1). Although it is widely argued that Moore's law is coming to an end of its validity (Simonite, 2016)—due to the upcoming of new computation paradigms such as quantum computing—, it is good for showing out the impressive evolution of computing for more than 40 years.

Wireless communication is also making progress. Several protocols and technologies have been implemented widely on sensors, actuators and machinery. Every electronic device can currently be connected to several others by means of different communication protocols (e.g. Wi-Fi, ZigBee, Bluetooth, GSM; see figure 1.2) (Shuang-Hua, 2014).

As a consequence of such evolution of computation and communication technologies, a wide variety of devices, including sensor-enabled smart devices and all types of wearables, connect to the Internet and power newly connected applications and solutions (Gil et al., 2016). Such phenomena is what is now known as the Internet of Things (IoT).

1.1.1 What is the IoT all about?

In a broad sense, the term IoT has been used to refer to a world where physical objects and beings, as well as virtual data and environments, all interact with each other in the same space and time (CERP-IoT, 2010). Such connectivity has the potential to power *smartness* across very different application areas such as mobility, assisted living, environmental protection, agriculture, etc. And has also gain

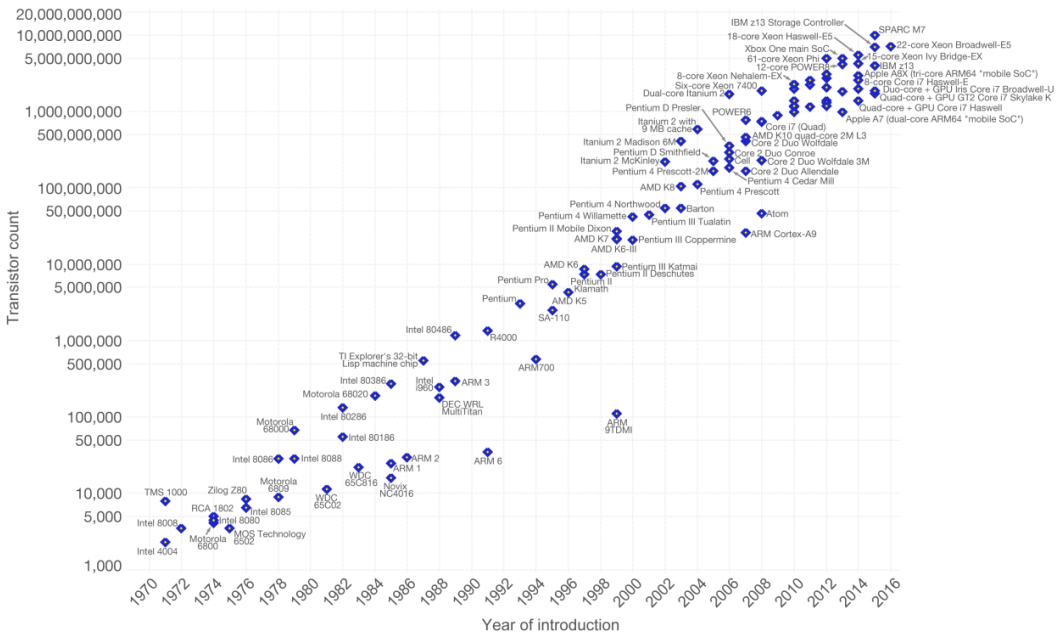


Figure 1.1: *The number of transistors in integrated cirquits chips (1971-2011) (Roser and Ritchie, 2018)*

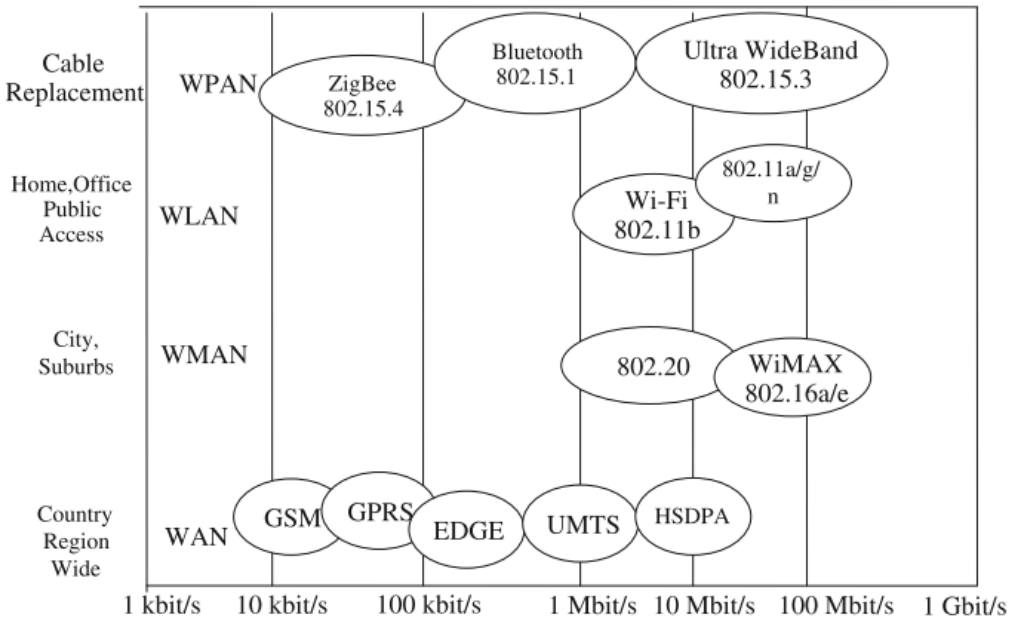


Figure 1.2: *Wireless communication standards, their speed and usage (Shuang-Hua, 2014, pg. 2).*

special interest from several governments (Friess, 2013).

The IoT is not tangible, in the sense of Vermesan et al. (2013):

> "The IoT is a concept and a paradigm that considers pervasive presence in the environment of a variety of things/objects that through wireless and wired connections and unique addressing schemes are able to interact with each other and cooperate with other things/objects to create new applications/services and reach common goals. (pg. 7)"

It is usually said that the view expressed before is not quite accomplished and that there is still plenty of research to be done. Up to now there is no unique solution for the development of the IoT in terms of security, management and data analytics (Alaba et al., 2017; Earley, 2015; Tsai et al., 2014). The data analytics side of the IoT is the responsible for bringing smartness (a.k.a cognitive capabilities) into IoT products (F. et al., 2015). Such smartness has been consistently considered the main objective of the IoT (Vermesan et al., 2013; Wu et al., 2014). This project is thus focused on the data analytics aspect of the IoT, since the development of this aspect is crucial for the true development of the IoT (Arsénio et al., 2014; Alam et al., 2016).

1.1.1.1 Applications

The way in which IoT applications can impact current products and services is by adding smartness to them. As grouped by E. Porter and E. Heppelmann (2017) smartness in products comes in different levels: monitor, control, *analysis*[1] and autonomous reaction to the changes in environment. The distinction of these levels can be seen hierarchically in figure 1.3, and they are described as follows:

Monitoring products are the first level of IoT development, this stage also considers management of alerts and notifications. For example consider the K'Track Glucose. It monitors glucose by a sensor in a wearable watch; it can provide time series plot of the glucose levels; alerts of high levels and reminders of times for medication. Monitoring applications may make use of signal processing to correct errors from sensors.

Controlling products can change the environment. For example Philips Lighting hue light bulbs can change lighting conditions based on user instructions via smartphone. Users can also provide timers and set alarms. However, industrial or more complex applications (see the Roomba robot) require more control than it is provided by most platforms; it is therefore, typical to use cascade controls where more complicated and real-time algorithms are applied locally, whereas parametrisation of such algorithms takes place remotely.

Analysis makes use of Artificial Intelligence (AI) and data analysis tools to build a model from the *constrained* world of the applications and improve the way the product gets things done. It can

[1] The original word used by the author was optimisation, but the word analyse seems to be more suitable for this project.

also help maintenance by providing product diagnostics based on data. Applications at this stage are simply a help for people and there is no autonomy exhibited.

Autonomy is the uppermost level of smartness. It builds on top of the other three levels and it is what products exhibit when they adapt without human intervention to different scenarios or user preferences.

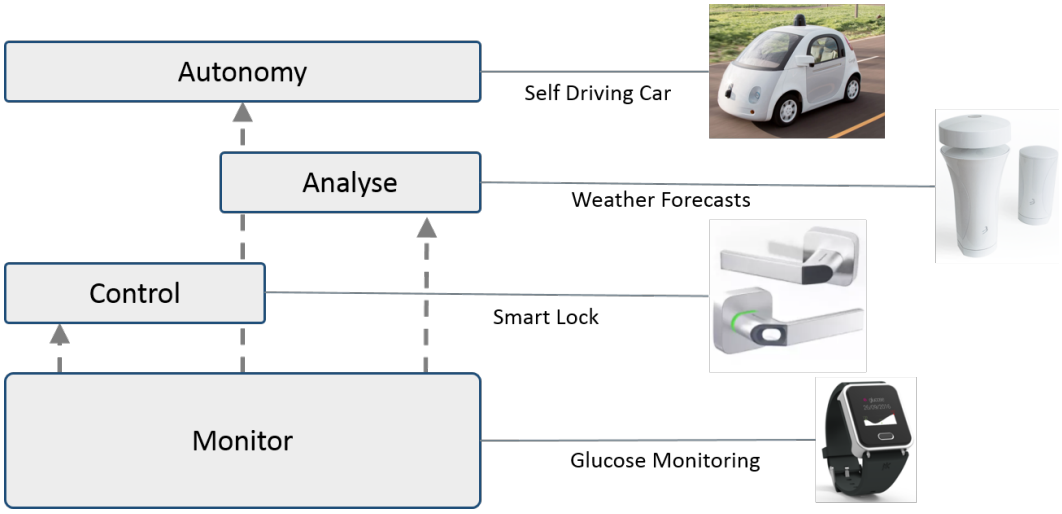


Figure 1.3: *Smartness hierarchy of IoT products*

There are several platforms that allow applications with any level of smartness to be developed more easily. Every IoT platform allows the monitoring stage of development (see for example Ubidots). However not all IoT platforms support the later three levels (ThingSpeak, ThingWorx, AWS IoT, IBM Watson IoT and Google IoT Solutions are platforms that do)[2]. It is worth mentioning that none of these platforms provides support in the form of plug and play solutions —besides monitoring and very simple control— they rather provide access to the tools for development of the other levels of smartness in the form of software libraries.

1.1.1.2 Prototypical architecture

Figure 1.4 shows a typical representation of the IoT. It shows some application areas and how they integrate with the Internet. Some domains such as smart homes (Alaa et al., 2017) would need a gateway [3] to connect to the Internet, whereas some objects or *Things* in other domains may connect

[2] `https://thingspeak.com/apps`, `https://www.thingworx.com/`, `https://aws.amazon.com/iot-platform/`, `https://www.ibm.com/internet-of-things/` and `https://cloud.google.com/solutions/iot/`

[3] A device that connects two systems that use different protocols (IEEE Standard Glossary of Computer Networking Terminology).

directly.

The gateway allows wired and wireless connections between objects inside a small network such as the one in a smart home environment; however, connections are usually wireless, which allows a dynamic distribution of objects. The last description is what is known as Wireless Sensor Networks (WSN) (see Shuang-Hua, 2014, for details). Gateways allow independent environments to have access to a bigger network to expand their capabilities for computation and perform analytics in a larger scale, in which developers can include data from all those independent environments. In contrast, real-time (but limited) decisions require some processing to be made locally (Arsénio et al., 2014).

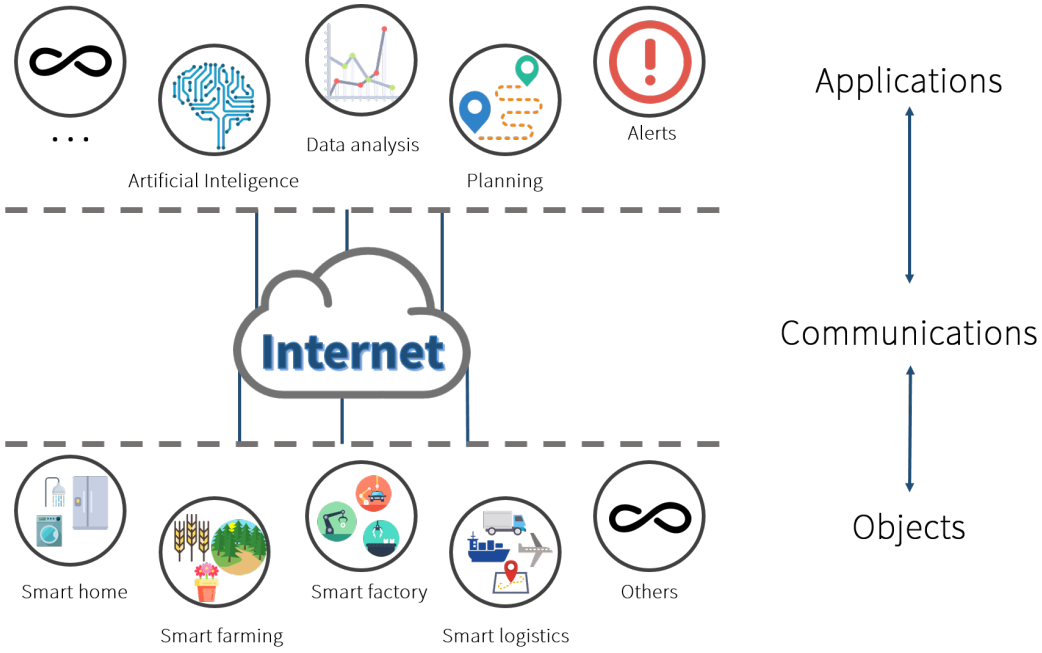


Figure 1.4: *Broad scale representation of the IoT.*

In the cases of Smart Factory and Smart Logistics shown in figure 1.4, other terms have also emerge: Cyber-Physical Systems (CPS) and Industry 4.0. All these terms share a lot of or make use of the IoT, but their subtle differences are not relevant to understand this manuscript; please refer to Henning et al. (2013); Henning, Kagermann. Wolfgang, Wahlster. Johannes (2013) for a deep description about these.

As the amount of objects connected to the Internet grows, complicated applications start to emerge and some previously unrelated objects start to make sense when used together (e.g. factories can adjust production in connection to real-time logistics by considering supply and market predictions; see Stolpe, 2016). Such applications rely on data analysis being performed remotely; furthermore, local computer power is usually very limited and applications may share different sources of data (Tsai et al., 2014). Remote analytics can take advantage of all data and greater computational power

to achieve a desired application (e.g. predictive maintenance, real-time route planing, etc.).

1.1.2 The potential of IoT applications

It is clear that the era of the IoT facilitates data extraction across several domains. Thus, there is a huge opportunity to extract knowledge from IoT applications. Moreover, the IoT relies on data analysis to exploit its true potential; leading to innovative new business models, products and services (Stolpe, 2016). It is not just about connectivity, but about how such connectivity can be used to enhance products' behaviour and learn from their context (Stolpe, 2016; Tsai et al., 2014).

In the following paragraphs, I present the hypothetical evolution of a bike rental system. It should be noted that at any stage, other enhancements could take place. However, the presented ones are specially chosen to emphasize the impact of using data analysis to exploit the true potential in IoT products.

IoT for remote monitoring. This is the first level of smartness in a product. It would be interesting to add a GPS to every bike in the rental system. The location of each bike can then be traced by users or administrators via a map interface on a web page. Knowing the location of bikes will allow users to decide which station to go for a bike; will help administrators to define new stations; and might prevent a bike from being stolen.

IoT for interaction with the system. It is easy to build a rule-based system which can alert administrators of the rental system of bikes leaving a predefined area. It could also respond to user queries about the availability of bikes near his current location. At this stage, people would not have to see a map for themselves, they can just ask queries or subscribe to notifications.

IoT to extract knowledge from the system. A truly reliable bike rental system should guarantee that, at any time, there is a certain amount of bikes available. This can be done with data analysis techniques tailored to predict the demand of bikes based on the time of the day and the day of the week. Predictive maintenance and fault detection of every bike can also be developed by the use of AI and data analysis.

Please refer to Stolpe (2016) for an elaborated description on how manufacturing; transportation and distribution; energy and utilities; public sector; and health care and pharmaceutical; can benefit from applications of AI and data analysis to the IoT.

1.2 Research Problem

As said before, the focus of this work is on the analytics side of the IoT . Analysis of the data from the IoT can only be carried out by what is known as machine learning or data mining[4] (Tsai et al., 2014; Alam et al., 2016).

Machine learning, a form of AI,is a very active area of research which deals with finding patterns in data, where data can be anything from pictures or words to sensor readings. These patterns can be used for very different tasks, for example, they are the driving force of autonomous driving systems (Aeberhard et al., 2015; Ziegler et al., 2014) and translation systems (Singh et al., 2017). Systems built using machine learning show a great level of *adaptation* to the environment presented to them in terms of data.

Machine learning can be broadly classified into three types of algorithms: *supervised*, *unsupervised* and *reinforced*. Here I introduce some fundamental concepts of supervised learning. The following provides only enough material to understand the research problem of this project, but its scope is very limited in terms of all the possibilities provided by machine learning. Interested readers are encouraged to review and expand the subject in any or some of the following: Izenman (2008); Hastie et al. (2009); P. Murphy (2012); Bishop (2006). I also take this opportunity to introduce some useful notation.

1.2.1 Supervised machine learning basics

The idea of supervised learning is to give computers examples of the *task* a human want them to perform. Such procedure is called *training*. The computer is then left to perform the task by itself, which may be referred as the *operation time* or *operation stage*.

The task is always to predict the value of a variable y based on a vector $\mathbf{x}$. In general, $\mathbf{x}$ and y can be any type of data, but they are always defined in terms of categorical variables (i.e. variables with a finite set of values like people's gender and whether a household appliance such as a refrigerator is on) or continuous variables[5] (e.g. people's height and the temperature of water). The task mentioned above is called *classification* if y represents a categorical variable, whereas it is called *regression* if y corresponds to a continuous variable.

Usually, training means to feed a *learning algorithm* with a number of examples from a *training data set* $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where n is the number of training examples; every $\mathbf{x}_i$ is a d-dimensional input vector called an *instance* or an *observation* representing sensor readings; and finally, y_i represents a particular example of what we want the computer to output from the input vector.

[4] In this work, the term data mining will not be used. Please refer to Izenman (2008, chap. 1) for clarification on the difference (if any) between the two terms.

[5] It is not relevant whether a variable is strictly continuous or practically continuous.

The scope of IoT applications is so vast that there could be applications requiring $\mathbf{x}$ and y to be a mixture of categorical and continuous variables; nonetheless, it is usually required that every pair $(\mathbf{x}_i, y_i) \in \mathcal{D}$ has the same structured format. Every dimension along the d-dimensional input vector is called a *feature* or a *variable*; thus, every $\mathbf{x}_i$ (boldface and indexed by i) has the same set of variables. Each of the variables are denoted x_j (regular type face and indexed by j) for $j \in \{1, 2, ..., d\}$; unless I refer to the value of the variable in a particular observation, in which case I write $x_{i,j}$.

For all practical purposes, a *learning algorithm* is a computer procedure which aims to find a function f called the *model*. The model refers to a correspondence between every y_i and every $\mathbf{x}_i$. In other words, supervised learning is an attempt to find f such that $y_i = f(\mathbf{x}_i) + \varepsilon_i \ \forall \ i$, where ε_i is the error between a particular prediction by the model and the real observed value. I omit the index i and refer to such correspondence as $y = f(\mathbf{x})$, when the interest is not in a particular pair $(\mathbf{x}_i, y_i)$.

In practice, after training, there is no way of knowing whether the algorithm has found the exact f; therefore, the algorithm is said to find an approximate model $\hat{f}$ such that $y \approx \hat{f}(\mathbf{x})$. This $\hat{f}$ is then evaluated using a *test set*. The test set $\mathcal{T}$ can be said to be every available pair $(\mathbf{x}_i, y_i) \notin \mathcal{D}$. The performance of the learnt model $\hat{f}$ is estimated by comparing y_i with $\hat{f}(\mathbf{x}_i)$ for every $\mathbf{x}_i$ and y_i available in $\mathcal{T}$. Once a *good* performance measure has been achieved, the model is set on operation: it will be provided with new $\mathbf{x}$ values and humans or other computational systems will rely on the model's estimated value $\hat{f}(\mathbf{x})$.

Most of the techniques for supervised learning impose little or no assumptions about $\mathbf{x}$ and y to be able to make a good estimate of f. Moreover, regardless of violations to the assumptions imposed in the mathematical derivation of the learning algorithms, the resulting models are judged based on their performance on the test set. On the other hand, it is required that all possible $\mathbf{x}_i$ in $\mathcal{D}$ or $\mathcal{T}$ or in operation time have the same set of features (format); the value that every variable takes for a particular $\mathbf{x}$ can obviously be different.

1.2.2 Problem Statement

As said by Stolpe (2016), IoT-enabled devices will become part of a highly dynamic network, in which devices can come and go from time to time. For illustration, consider the bike rental system described in section 1.1.2. If a sensor inside a particular bicycle is out of battery, its value would not be recorded for a certain period of time. If a supervised learning-based application relies on the value of that sensor (and others), it would not work until the sensor has enough power to transmit its value. Mathematically, this means that some of the x_j may be missing from some of the $\mathbf{x}$ (observations), making the model $\hat{f}$ useless (Yan et al., 2014). Figure 1.5 shows more reasons that may cause missing one or many variables from any observation.

In other words, the dynamic nature of IoT networks implies that every new observation $\mathbf{x}$ may

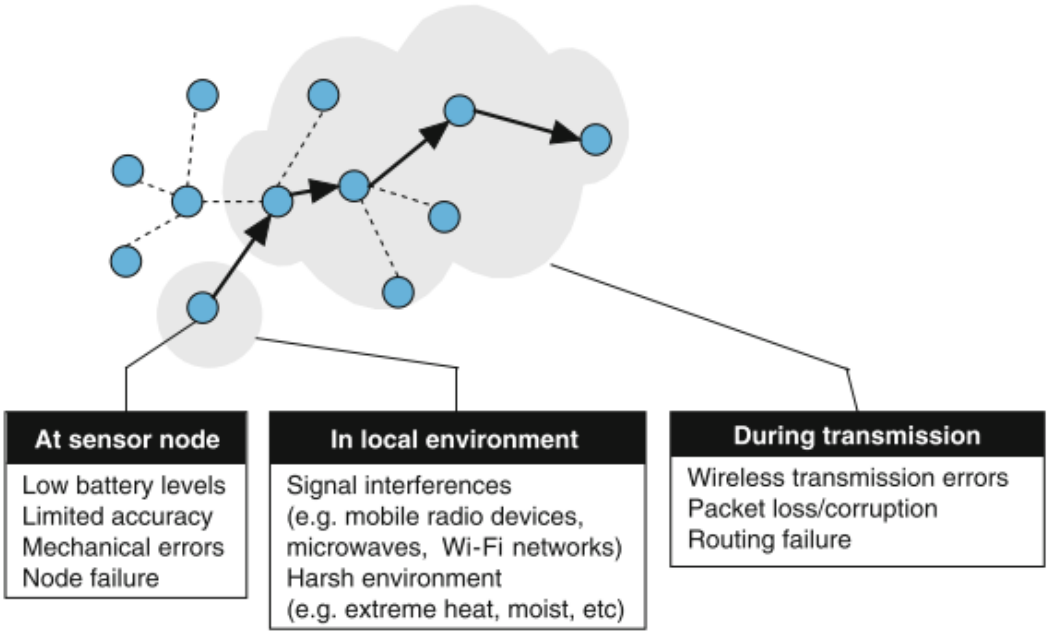


Figure 1.5: *Causes of losses in sensor data (Shuang-Hua, 2014).*

have less elements from what is expected. In this new scenario, an observation can be as in expression 1.1, where *nan* means the absence of a value (i.e. a missing sensor reading).

$$\mathbf{x} = \langle x_1, nan, \ldots, x_j, \ldots, x_{d-1}, nan \rangle. \tag{1.1}$$

Therefore, there is a need for IoT applications to adapt to such situation and keep responding properly. To elaborate on the limitations of current methods on such situations, consider the following example.

Suppose a classification task is being performed based on sensor readings from a network of two fixed sensors. This time, the task consist of identifying two groups of observations, which may represent optimal and suboptimal operation of a household appliance. Once training has been carried out, the training data set and the model $\hat{f}$ could look like illustrated in figure 1.6a, where the orange line represents the learnt model that can distinguish between the two groups of observations.

In a regular situation, a new observation $\mathbf{x} = \langle x_1 = a, x_2 = b \rangle$ arrives to the data centre and it is easy to see its location relative to the model (see figure 1.6b). This location is what defines whether the model say it corresponds to a group or the other.

Sometimes, however, a sensor may be under maintenance or could have gone out of range. In such condition, a particular observation arrives at a data center as $\langle x_1 = nan, x_2 = b \rangle$. Classifying such observation becomes a difficult task (see figure 1.6c). This situation has been named as the **incomplete data** or **missing data** problem: observations in a data set may contain values that are *missing* (i.e. values that were not recorded).

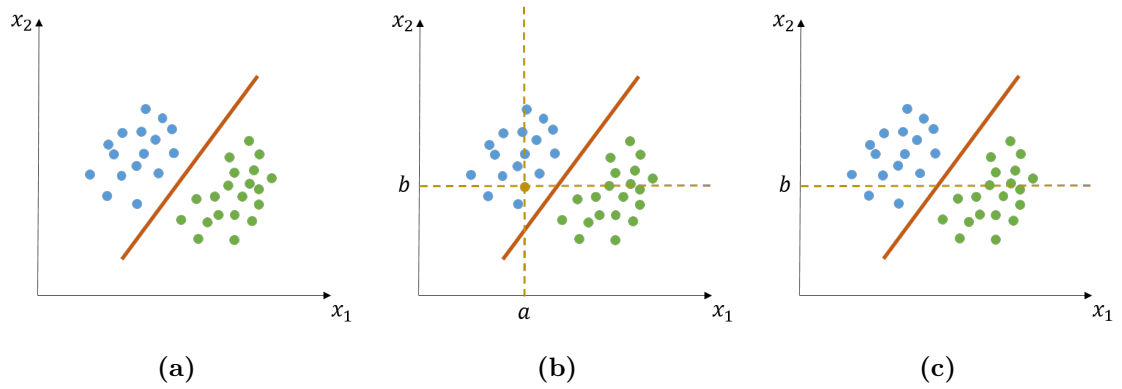


Figure 1.6: *Three situations in a classification task. (a) an example of a model $\hat{f}$ learnt in a classification task for a bivariate data set, (b) a complete observation to be classified, (c) an incomplete observation to be classified*

1.2.3 Naïve approaches for the missing data problem

The missing data problem is present in several other fields, like medicine (Barclay et al., 2018; Bell et al., 2014) and sociology (Ellison and Langhout, 2017). Therefore, solutions to this problem have been proposed (Little and Rubin, 2002) that are not tailored to the specific needs of the IoT. Here, I present some of those solutions together with an explanation on why those approaches are not suitable for IoT applications.

1.2.3.1 Omit features or observations

In social experiments, an interview is applied to a big enough sample set (size of the sample set will depend on the size of the population). Even if the interviewer properly selects the questions to be included in the interview, there might be some people that for several reasons do not provide answers for some of the questions. In such cases, data analysis of the interview's results can require handling of missing data in the following way:

- If there is an insignificant number of interviewees which has avoid answering one or many of the questions, the interviewer can simply delete such observations (i.e. pretend that the interview was applied to less people). Statistics derived from such a treatment may be less significant.
- Delete all features where missing values are present, which may cause the interviewer to answer a different set of questions (from the full set of questions that was originally of interest) about the population.

Consider the case of applying the first strategy in the context of the IoT: during training time, it may not be harmful to omit some observations (i.e. some training examples), which is specially true when the training size is big, just as it is expected from IoT data. On the other hand, the missing

data problem may happen for long periods; thus, a system that relies on machine learning predictions becomes unresponsive or useless if the first strategy is adopted.

During training time, the second approach may cause the learnt model to be inaccurate, because there is not enough information to build an accurate one: the models *underfit* the data. Moreover, in operation time, the system would be useless, as shown in figure 1.6c.

1.2.3.2 Imputation based approaches

A wiser and more useful way to deal with the missing value problem is to replace (impute) a missing value with some estimation for that value; nevertheless, imputation should be applied carefully. Here, I show naïve approaches, but I discuss more elaborated and applicable approaches to imputation in the IoT in Chapter 2.

Carry out last observed value One approach that may be applicable when there is a relatively small number of observations and features that may be missing; and when those features relate to time series data (such as sensor data), is to use the last observation of a variable as the value to impute. However, such estimation of a missing value results inappropriate when there are subsequent missing values of the same variable.

Imputation of the average values Another approach is to impute every missing value with the corresponding average value observed in a training set or a certain period of time. However, the context in which it is applied may have several different operation modes; thus, the mean of a certain variable during a period of time may be a bad estimate of that variable in a latter period (e.g. the mean of the temperature outside a home refrigerator is usually a lot higher at midday than at midnight).

1.2.4 Qualities of a *better* imputation method

The main subject of this project is to work towards the development of better imputation methods for the context of the IoT. This would require a consideration of the following:

I. Since the mechanisms that generate the missing data problem are different across different domains, more suitable imputation methods for the IoT should take into account a *precise description of the several mechanisms that would lead to missing data problems in the IoT.*

II. As explained in section 1.1, IoT applications may be the result of inputs of data from different environments and therefore, proper imputation methods should be *applicable to a great variety of cases*, by keeping assumptions about the data as little as possible.

III. Imputation methods should *adapt to changes in the process generating data*, that is, when the processes generating data change, estimations of the missing values must change accordingly.

IV. Since applications are expected to run smoothly during operation time, imputation methods should *provide estimates as they are needed* in order to avoid bottle necks.

The above brief description of supervised learning, thus, seems like a good starting point for a solution items II and III, since it has been applied to a great variety of cases and have been shown to extract what is relevant from the data instead of imposing conditions on it.

1.3 Objectives

As a consequence of the analysis presented in Section 1.2, I now present the general and specific goals pursued during this research project:

General objective:

To propose an imputation method for Internet of Things applications, through the use of supervised learning, to make accurate predictions of the missing values.

Specific objectives:

(I) To analyse how the problem of missing data problem presents in the IoT, in order to build a model of the problem and appropriately judge imputation methods in this context.

(II) To propose an imputation method to solve the missing data problem in the IoT.

(III) To evaluate the proposed method by comparing it with other solutions in situations that may be encountered in the IoT.

(IV) To evaluate the impact of different imputation methods (i.e. their performance) on the reliability of a smart application in the IoT.

1.4 Research Scope

Several aspects of application of advanced data analysis to the IoT have been described in many works such as Arsénio et al. (2014); Stolpe (2016); Gil et al. (2016); Wu et al. (2014). Among other aspects, there is the need to handle the gigantic amount of data that can be generated by the IoT, both in terms of features and observations; the noisy nature of data coming from sensor networks and the speed at which this data arrives to an analytics server; however, I will only consider the situation presented in Section 1.2.

The missing data problem could also be addressed by constructing better communication protocols, designing better sensors and platforms that integrate all available data. Those solutions are very important, but developers in the application level of the IoT have to integrate different data sources,

which very often operate under different protocols. Therefore, I will not consider those approaches in this work.

In the rest of this project I will assume that there is a maximum number d of variables to be considered (i.e. whether missing or not, the number of variables is defined from the beginning). Observations may have a number $d^* \leq d$ in the case of missing values. However, I will not consider the case of a growing network of devices (i.e. $d^* \geq d$). Traditionally, it will cause constant reformulation of the imputation method, as well as data analysis algorithms, but it would add a lot of computational complexity. This is a relevant challenge for the development of the field and further work should definitely consider it.

The problem of missing data is not only present in the context of the IoT. It is also present, for instance, in medicine (Masconi et al., 2015; Netten et al., 2016) and genetics (Eaton et al., 2017). However, very often the proposed solutions and methodologies are developed for their special context. Therefore, I developed the state of the art with solutions that were proposed in the context of the IoT.

1.5 Research Justification

Earley (2015) highlights that organizations still have the challenge of understanding how the inclusion of machine learning in the IoT can add value to their businesses; it is a fact in the academia world though. Wu et al. (2014) argued that "without comprehensive cognitive capability, IoT is just like an awkward stegosaurus: all brawn and no brains."[6]. So integration of machine learning to the IoT is required to fulfill its expectations.

Alam et al. (2016) evaluated eight machine learning algorithms and concluded that their application to the IoT showed promise; however, they evaluated the algorithms based on training time and prediction accuracy. In contrast, several authors have argued that traditional methods for machine learning should be adapted to face all the needs of IoT applications (Stolpe, 2016; Tsai et al., 2014; F. et al., 2015) and highlight the importance of handling missing data as it is ubiquitous in the context of the IoT.

Missing data in the IoT can be caused by physical limitations or by design (Tsagkatakis et al., 2016; Bijarbooneh et al., 2016). Interference, node/link failure, physical obstacles, environmental change, unstable communications, unreliable sensors, etc. make observations unpredictable (Nower et al., 2015; Yan et al., 2014; Sujbert et al., 2016; Zhao et al., 2016) by simple approaches. Using naïve approaches can lead to faulty conclusions and predictions from data (Srinivasan et al., 2016).

As I show in Chapter 2, current solutions to the missing data problem are limited in its imple-

[6]Here cognitive refers to the application of Artificial Intelligence to the IoT

mentation, since they are based on assumptions that are do not always apply to the great variety of IoT applications. For instance, some solutions assume a particular probability distribution generating the data and the missing values; others rely on manually chosen parameters. The imputation method I propose in Chapter 3 is consistent with the qualities expected from imputation methods in the IoT, so this project has considerable practical relevance. Moreover, the imputation method shown in Chapter 3 is intended to help developers of IoT applications by providing an imputation method that is applicable to a great variety of cases due to the comparably limited assumptions in its conception.

This project is also relevant from a theoretical perspective, because by building a model of the missing data problem in the IoT (see Section 3.1), fellow researchers will be able to reason about imputation methods that will be more suitable to the context of the IoT. Building this model also contributes in the definition of proper experimental studies that will help to select from different existing imputation methods.

1.6 Manuscript Organization

The rest of this manuscript is organized as follows. In Chapter 2 I present the relevant state of the art in imputation methods proposed in the context of the IoT. Chapter 2 also shows approaches to model the missing data problem and the way in which proposed imputation methods have been evaluated. Afterwards, Chapter 3 shows the imputation method I propose, together with three models of the problem of missing data in the IoT and their use to test imputation methods. I performed several experiments to test the validity of the proposed method of imputation and show their results in Chapter 4; the same chapter also presents the impact that different imputation methods have on the quality of a smart application. Finally, I present an analysis of the results of this research project and its conclusions in Chapter 5.

Chapter 2

Literature Review

Since the problem of missing data in the IoT has been reported significantly (Tsagkatakis et al., 2016; Zhao et al., 2016), several authors have proposed solutions. In this chapter I show:

- Approaches to model the missing data problem and why this is important (see Section 2.1).
- Current solutions to handle the missing data problem (i.e. imputation methods), their main features and limitations (see Section 2.2).

Often, the terms IoT, CPS and WSN appear together in scientific literature. Moreover, the missing data problem has been reported together with any (or more) of those terms, and the choice between terms is usually defined by the context of the authors. Therefore, the solutions I present in this chapter have been proposed in any of those terms. Nevertheless, I omitted solutions that were heavily dependant on information not available at the application layer of Figure 1.4 (e.g. signal strength in wireless communications). It is important to note that, on the application level, developers may only have their data and some information of its context.

In the end of this chapter, Section 2.3 presents the conclusion of what I found in the previous two aspects of the state of the art and highlight what is missing in current practices from the perspective of the IoT.

2.1 Models of the Missing Data Problem

Before describing solutions (e.g. imputation methods or algorithms) to the aforementioned problem and their limitations, it is first necessary to define the model of the problem itself. Such a model can be used to properly understand solutions and estimate some measure of their quality. Let me first introduce some notation so that the following discussion can be easily understood.

I call a *data set* a collection of several values of $\mathbf{x} \in \mathbb{R}^d$ that are placed vertically, like

$$X = \begin{bmatrix} x_{1,1} & x_{1,1} & \dots & x_{1,j} & \dots & x_{1,d} & x_{1,d} \\ x_{2,1} & x_{2,1} & \dots & x_{2,j} & \dots & x_{2,d-1} & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{i,1} & x_{i,2} & \dots & x_{i,j} & \dots & x_{i,d-1} & x_{i,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1,1} & x_{n-1,1} & \dots & \vdots & \dots & x_{n-1,d-1} & x_{n-1,d} \\ x_{n,1} & x_{n,1} & \dots & x_{n,j} & \dots & x_{n,d-1} & x_{n,d} \end{bmatrix},$$

where the index i is used to denote different observations, n is the number of observations and d is the expected number of variables per observation. Such data set is named *complete* when there is not a single *nan* in it. In contrast, an *incomplete* data set is one with at least one *nan* in it:

$$X^{nan} = \begin{bmatrix} nan & x_{1,2} & \dots & x_{1,j} & \dots & nan & x_{1,d} \\ x_{2,1} & x_{2,2} & \dots & x_{2,j} & \dots & x_{2,d-1} & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ nan & nan & \dots & x_{i,j} & \dots & x_{i,d-1} & x_{i,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1,1} & nan & \dots & x_{n-1,j} & \dots & x_{n-1,d-1} & nan \\ nan & x_{n,2} & \dots & x_{n,j} & \dots & x_{n,d-1} & x_{n,d} \end{bmatrix}.$$

Finally, a *reconstructed* data set $\hat{X}$ is one on which an imputation method has *successfully* been applied, therefore, it has no missing data in it. I do not use the word successfully as an indication of the performance of such imputation method; it rather means that $\hat{X}$ has no missing values.

The most commonly accepted way to assess the quality of an imputation method (see Liu et al., 2012; Nower et al., 2013; Pan et al., 2014; Yan et al., 2014; Li et al., 2015; Tsagkatakis et al., 2016; Zhao et al., 2016) is the following:

1. Take a complete data set.
2. Generate an incomplete data set by removing some entries of the one measured in the previous step, following a *model* for the process that causes missing data.
3. Impute with the imputation method to be tested, creating a reconstructed data set.
4. Compute the *reconstruction error*.

The reconstruction error (see Tsagkatakis et al., 2016) can be defined as any error measure between X and $\hat{X}$. One can say that imputation methods try to predict values $x_{i,j}$ for all pairs (i, j) for which the entry of the matrix X in position (i, j) is *nan*. Such predictions are labelled as $\hat{x}_{i,i}$. The reconstruction error is thus a measure of the difference between all $x_{i,i}$ and $\hat{x}_{i,i}$.

Not all authors have computed the reconstruction error in the same way, and there seems to be no preference among the different possibilities. I adopted the Mean Absolute Error (MAE), computed as

$$\frac{\sum_{l=1}^{m} |x_l - \hat{x}_l|}{m},$$

where m is the total number of missing values and l is just an index over the whole list of entries with missing values. For an argument about why this measure is preferred, see Willmott and Matsuura (2005).

The most significant difference I found in different works is the model the authors used to generate the incomplete data set. To have an idea of the type of results a model should produce, figures 2.1 and 2.2 show a picture of two public data sets that have been exposed in several articles regarding (see e.g Pan et al., 2014; Yan et al., 2014) the missing data problem. In each of these figures the gray parts represent a non missing values and the black ones represent missing values or consecutive missing values an the boxes highlight some patterns of interest. The reader is reminded that columns represent variables and rows represent observations.

Yan et al. (2014) defines three classes of missing data problems: i) when there is only one variable and an observation is missing at row i but neither row $i-1$ or $i+1$ are missing; ii) when there is only one variable and an observation is missing at row i and either row $i-1$ or $i+1$ is not missing; and iii) when there are two or more variables and there is a missing value at any position (i, j).

The first two cases do not generalize to the case of missing data in the IoT for two reasons: i) the number of variables in the IoT can be huge, ii) there can be subsequent missing values due to sources of data being out of range or out of power (Shuang-Hua, 2014). The last one has been adopted by other authors and is also expected in reality (see box b in figure 2.1), but the above description is certainly not concrete (i.e. there are a lot of situations that fit the description). Therefore, such description cannot be translated into a computer code that would consistently produce incomplete data sets. Descriptions of this type are found also in other articles (e.g. see Zhang and Yang, 2014; Pan et al., 2014; Zhao et al., 2016).

In the case of only one variable Sujbert et al. (2016) also showed three models for missing data[1] i) for every $i \in 1, 2, ..., n$ there is the same probability of observation i not being recorded (random independent data loss); ii) divide data into blocks and for every block there is a probability of that observation block being recorded (random block-based data loss); and iii) there are two probabilities of an observation not being recorded, use one of them when the previous observation was recorded and the other one otherwise (Markov model-based data loss).

If one considers a generalization of the first two models to any number of variables, it is clear that such models can generate patterns like some areas of figures 2.1 and 2.2. In contrast, although the

[1] I write in parenthesis the names given by the author; I do the same in latter descriptions.

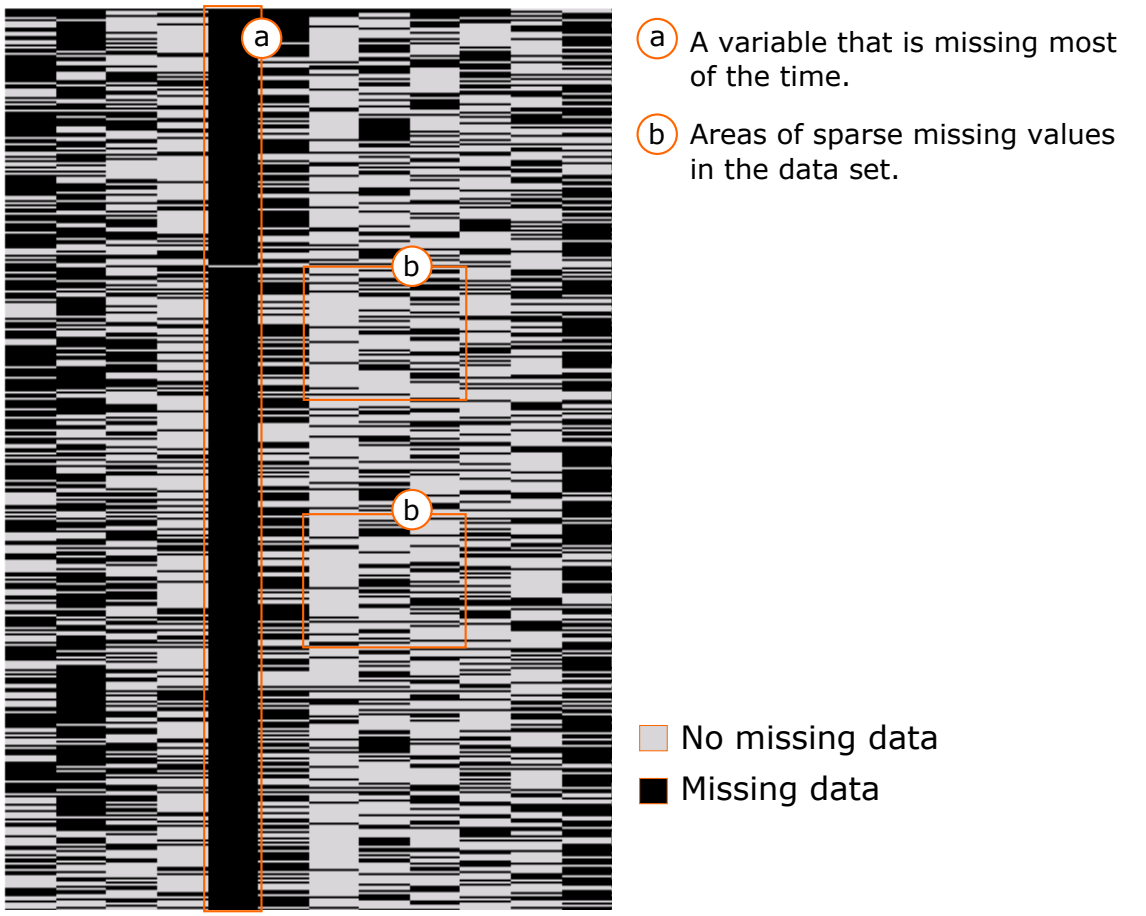


Figure 2.1: *A picture of a part of the Intel Lab data set downloaded from* `http://db.csail.mit.edu/labdata/labdata.html`.

third model makes intuitive sense from the perspective of sensor networks, the authors did not show the kind of pattern it would generate.

Li et al. (2015) propose two models: i) repeatedly choose a random time (row) and a random variable (column) (Random Missing Pattern); ii) after a certain number of rows of complete data, the rest will be completely missing in all variables (Consecutive Missing Pattern).

In this case, the first pattern seems very reasonable when looking into box *b* of figure 2.1, whereas the second seems to recreate some parts of the pattern shown in figure 2.2. Nevertheless, none of these models would cover the fact the missing data mechanism may affect some variables more than others (see box *b* of figure 2.2) or that blocks or missing and non missing data may alternate as shown in box *a* of figure 2.2.

Finally, Kong et al. (2013) defined four independent patterns of missing data: i) Element Random Loss, in which missing values are sparse across the data set; ii) Block Random Loss, in which the phenomena that causes the missing values affects equally some variables for a certain period of time;

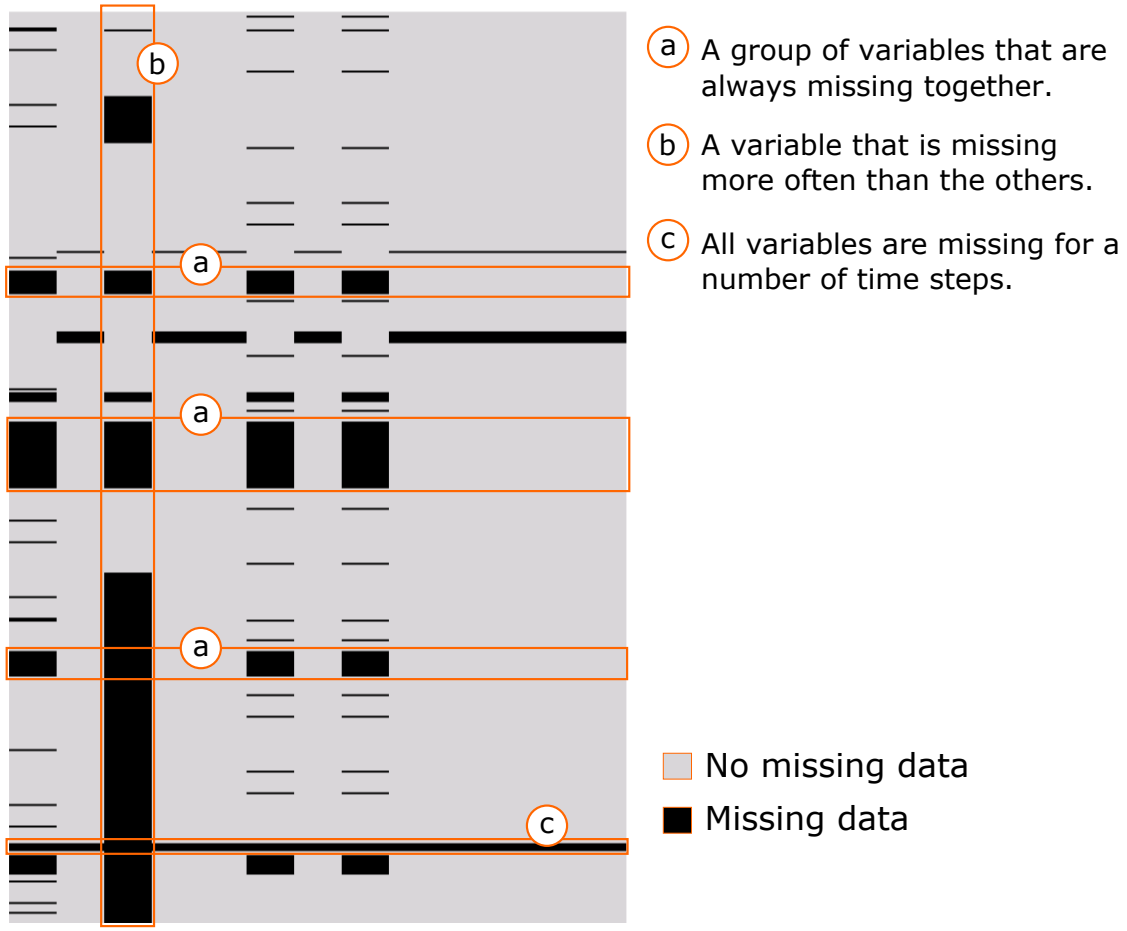


Figure 2.2: *A picture of a part of the Air Quality data set downloaded from UCI Machine Learning Repository (*`http://db.csail.mit.edu/labdata/labdata.html`*).*

iii) Element Frequent Loss, in which some variables are missing intermittently; and iv) Successive Element Rows, in which from a certain moment, some variables are lost and are not observed thereafter. This definitions were made by an analysis performed on real data sets, and they showed examples of these patterns. Indeed, looking at figures 2.1 and 2.2 reveals that the description match some areas of those figures. However, they did not provide precise descriptions to simulate such patterns. Moreover, as I show in Chapter 3.1, this patterns can be summarized into more concise descriptions for ease of simulation.

Other works did not emphasize the way in which authors modelled the missing data problem and simply mentioned that some entries, usually specified as percentage of the whole data set, were removed (e.g. see Pan et al., 2014; Zhao et al., 2016). An unfortunate consequence of such a poor model specification implies that different *experimental results are not reproducible across research groups*. Moreover, even if results were reproducible, obtained experimental results may be misleading,

because *they may not reflect the reality of missing data in the IoT.*

Another problem I found is that, although some authors tested their methods against others (see Nower et al., 2015; Tsagkatakis et al., 2016), the relative difference between competing imputation methods was not properly estimated. Usually, the authors made reported their results on simple plots of an experiment, which raises the question of whether the difference they found was statistically significant. Therefore, *there is a need for quantitative measures that allow proper hypothesis tests between competing imputation methods to be carried out.*

2.2 Approaches to Impute Missing Data In the IoT

In the context of the IoT there have been two main assumptions that underlie all the imputation methods that have been proposed:

- There is a strong association between the variables of a data set; in other words, the values that a variable may have are greatly influenced by the values of other variables in the data set (e.g. when the temperature in a room increases, the humidity may do as well).
- Data from the IoT is time series data, which means that all rows in a data set were collected sequentially, meaning that if row i was collected at time t, row $i+1$ was collected at time $t+1$. For this reason I often use t instead of i.

In what follows, I describe those assumptions in detail and show which of the review imputation methods relies on them. Finally, also describe a set of conditions that the authors defined so that their methods would work (e.g. assumptions about the data).

2.2.1 Strong variable association

On the sensor level, it is a common practice to locate several nearby sensors measuring the same variable, which results in a data set that contains copies of the same variable. In a data center this can be exploited by combining those measured variables into a single one and therefore one can safely ignore the missing value.

This simple strategy is not always possible and it is extended by Liu et al. (2012). For a missing value coming from a corresponding sensor, their approach was to compute a missing value based on the real distance between the *neighbouring sensors.* Likewise, Nower et al. (2013) computed the imputed value based on the neighbouring sensor, which measures the same variable, for which there is the highest correlation coefficient.

What is common in these approaches is that the authors assumed that nearby sensors were sensing the same variable, which is the same as saying that there is a *spacial relation between the variables.* This is clearly a rational assumption in WSN, but cannot be applied in the context of the IoT, where

a dataset is expected to have information from several independent WSN and other sources of data.

The previous assumption is often relaxed in other works, like Pan et al. (2013, 2014) that, although they expressed their ideas based on the spatial association between the variables, the values produced by their methods are the result of *linear* regression models with respect to the variables without missing values, which work even if the variables of the model do not represent the same physical parameter.

As opposed to assuming a linear relationship between the variables, other authors assumed a *non linear* relationship between the variables in a data set. For example, Gao et al. (2012) used a Support Vector Regression with a non linear kernel (see Vapnik, 1998, ch. 11) to estimate the values of the variables with missing values, based on the variables that do not have missing values.

Although the previous non linear approach is, in a sense, more general than the linear one, the choice of the kernel imposes certain form of relation between the variables (i.e. the one defined by the kernel). In contrast, Zhao et al. (2016) proposed the use of Stacked Autoencoders (see Goodfellow et al., 2015, ch. 14), which can be trained to produce the most suitable form of non linear relationship between the variables, while producing a lower space representation of them (i.e. a representation of the same data with a reduced number of variables). Their method uses such a lower representation to produce cluster of data points and change their imputed value until the clusters are stable.

2.2.2 Data from the IoT as time series data

A time series is defined as a set of quantitative observations arranged in chronological order (Kirchgässner et al., 2013). In most cases, such observations are equally separated by discrete time steps. Since the IoT collects data from a multitude of sensors, it is plausible to have all this data arranged properly.

In fact, this property has been exploited by several authors assuming some type of relationship (e.g. linear or non linear) between a variable and the same variable a few time steps ahead. By *linear* I mean that the value of a variable at time t is predicted as a linear combination of the previous values at other times, plus some other terms that do not include the variable. In contrast, I refer to a *non linear* relationship when the value of a variable at time t is computed from the values at other times, but using a different approach. Note that this does not imply that all non linear relationships are more appropriate than linear ones, simply that the term non linear refers to a broader set of relationships within a variable.

An example of a linear relationship is the work by Yan et al. (2014), whose method computes the missing value as the average of its subsequent and previous values. This approach can only be accurate in two conditions: i) the change between subsequent measurements of the same variable are *small*; and ii) there are very few and isolated missing values on given column (i.e. variable), which clearly cannot be expected from data in the IoT (see figures 2.1 and 2.2).

Assuming there is enough previous data, Nower et al. (2014) proposed the construction of an ARIMA model (see Box et al., 2015) to take into account the time dependency of the data. They integrated with the earlier model (Nower et al., 2013) by switching between the use of one or the other depending on the situation. Latter, in (Nower et al., 2015), the authors added a Kalman Filter (see Chui and Chen, 1998) to improve accuracy.

A *non linear* consideration of the time dependance in the data has been the approach by Tsagkatakis et al. (2016) who used Singular Spectrum Analysis (see KaltenbachHans-Michael, 2012) assuming that value of variables at time t can be estimated by the lagged values of the same variables. This approach certainly assumes that such lagged values are not missing. Although, it may seem like an strong assumption, it is relaxed by simply using predicted values up to a point in time as the lagged values of subsequent predictions.

2.2.3 Other features of the reviewed solutions

The reviewed solutions included more characteristics than just variable association and temporal relationships. In the following list I describe those other features, together with some examples and a comment on their applicability in the IoT:

1. A few methods are based on a *combination of models*. For instance, Nower et al. (2014, 2015) used a model for the relation between the variables and another for the relation within the variables across time steps, which they switched based on an online estimate of the performance of the two. A problem with such switching mechanism is that it may not be clear when to prefer a model over the other; moreover, better predictions may be achieved by producing predictions based on both features of the data from the IoT.
2. Many authors introduced one or several *hyperparameters* (i.e. parameters of the models that cannot be computed from data) in their methods. An example of hyperparameters are the so-called regularization coefficients in the work of Li et al. (2015), which control how much spatial an temporal correlation to include in their algorithm. This approach has the disadvantage that different values of the hyperparameters may result in highly different imputed values.
3. Several of the reviewed methods require that, to impute any values, one must firstly *train* a model; that is, compute some parameters of the models based on previously available complete data, just as it is the case with applications based on Supervised Learning. This is the case, for instance, of the linear regression approach to relate different variables that was used by Pan et al. (2014).
4. Most of the authors assumed that there is always a *complete part* in the data set; in some cases this complete part implied that there were only a few and sparse missing values in the data set (e.g. see Figure 2.1). For others, it may simply mean that to compute $\tilde{x}_{i,j}$ you need one or

some values $x_{k,j}$, with $k \neq i$. For example, Nower et al. (2013) assumes that, for a given row i there is at least one value that is not missing, therefore they could compute the missing value based on the nearby sensors available. This is a sensible assumption quite often; however, it breaks apart in situations like the one shown in Figure 2.2.

5. Finally, some methods were based on a particular *probability model*, such as the one proposed Zhang and Yang (2014), who proposed the use of a Bayesian Network Model. Assumptions of this type cannot generalise to the realm of the IoT due to the inherent heterogeneity in the sources of data.

2.3 Concluding Analysis of the Literature Review

Current models of the missing data problem in the IoT fail to mimic the conditions in which the problem occurs and fail to give enough detail as to be adopted by the community. This problem limits the conclusions that may be reached by independent experiments regarding imputation methods in two main ways: i) the different performance measures that the researchers use may be misleading; and ii) the reproducibility of the experiments is compromised.

Regarding the evaluation of the imputation methods, there is also a disregard in the inherent randomness of the missing data patterns that may arise in the context of the IoT. For instance, Figure 2.1 shows that a variable is much more affected than the others; a singular test of any imputation method in such a pattern, may reflect a bias of the imputation method over some particular values, instead of providing a measure that accounts for all possibilities. Therefore, testing in such a pattern, and indeed in other patterns, would require repeated experiments.

Consequently, I devote Section 3.1 to propose three procedures that consistently produce the kind of patterns that can be observed in figures 2.1 and 2.2.

With reference to imputation methods I classified the reviewed works based on how they exploit the properties of data in the IoT and what conditions they required to work. In the previous section I introduced such classification and provided a few examples of imputation methods that fit into a particular classification. I summarize the complete review of imputation methods in tables 2.1 and 2.2.

Proper imputation methods for the IoT should exploit the association between different variables as well as their temporal structure. Moreover, the heterogeneity of the IoT context, implies that there should be no assumption regarding the way in which this two aspects should be exploited. In Section 3.3 I present a method that exploit the two aforementioned properties of data from the IoT in their most general form and, at the same time, can in principle work on any of the patterns that I showed in this chapter and describe in Section 3.1.

Table 2.1: *Exploited data relationships by articles.*

	Temporal		Between variables		
	linear	non linear	linear	non linear	spatial
Guo et al. (2012)	•		•		•
Gao et al. (2012)				•	
Liu et al. (2012)	•				•
Pan et al. (2013)		•	•		•
Yan et al. (2013)		•	•		•
Kong et al. (2013)		•	•		•
Niu et al. (2013)		•	•		•
Nower et al. (2013)			•		•
Ni et al. (2014)		•			•
Pan et al. (2014)		•	•		•
Kong et al. (2014)		•	•		•
Nower et al. (2014)	•		•		•
Yan et al. (2014)	•				•
Zhang and Yang (2014)	•		•		
Li et al. (2015)		•		•	•
Nower et al. (2015)	•		•		•
Zhang et al. (2016)		•		•	
Zhao et al. (2016)				•	
Tsagkatakis et al. (2016)		•		•	

Table 2.2: *Other characteristics of the reviewed solutions.*

	Combined models	Hyperparameters	Trained	Complete part	Probability model
Guo et al. (2012)				•	•
Gao et al. (2012)		•	•	•	
Liu et al. (2012)	•			•	
Pan et al. (2013)	•		•	•	•
Yan et al. (2013)	•			•	
Kong et al. (2013)		•		•	
Niu et al. (2013)		•		•	
Nower et al. (2013)		•	•	•	•
Pan et al. (2014)	•	•	•	•	•
Ni et al. (2014)		•	•	•	
Kong et al. (2014)		•		•	
Nower et al. (2014)	•	•	•		•
Yan et al. (2014)				•	•
Zhang and Yang (2014)				•	
Li et al. (2015)		•		•	
Nower et al. (2015)	•	•	•		•
Zhang et al. (2016)			•		•
Zhao et al. (2016)		•		•	
Tsagkatakis et al. (2016)		•	•	•	

Chapter 3

Proposed Contributions

This chapter presents two contributions to the community of the IoT, in particular, to the community interested in the missing data problem. The first section of this chapter is intended to help in future tests of imputations methods (just as it is the case of Chapter 4). The second section analyses the use of regression methods to produce imputation methods. In the last section I present the method proposed method for imputation.

3.1 Simulation of Missing Data Patterns in the IoT

As explained in Chapter 2, the reviewed works lack a precise description of the way in which missing data patterns are generated in their experiments. Therefore, the aim of this section is to be precise about these matters. What I present in this section is intended to serve two purposes: i) make the experiments presented in Chapter 4 clear and reproducible; and ii) provide other researchers with a precise definition of the missing data patterns that arise in the IoT, which may help them in future simulated experiments such as the ones presented in the aforementioned chapter.

As a consequence of the missing data patterns, I also explain how to estimate the performance of imputation methods in the context of the IoT. Recall that, up to now, it is only possible to estimate the performance of imputation methods if there is a data set with no missing values in it, because this way assures an unbiased comparison between true values and imputed ones.

In summary, this section is concerned with the question of how to simulate the missing data patterns so that a proper evaluation of imputation methods can be made. This question is directly linked with the specific objective (I). Notice that the procedures described in the following subsections may not be related to the real causes of the missing data problem; nevertheless, they serve to replicate the appearance of the patterns I showed in Chapter 2.

3.1.1 Bernoulli independent missing data pattern

Consider a *complete* matrix of data X composed of d variables along n observations (i.e. $X \in \mathbb{R}^{n \times d}$). One can say that for every variable in X, there is a probability $P_j \; \forall \; j \in \{1, 2, ..., d\}$ of no recording that variable in any of the n observations, which can be seen as a Bernoulli process of length n for every variable in X (see Bertsekas and Tsitsiklis, 2000, ch. 5, p. 11). Using these d independent Bernoulli processes, one can construct an *incomplete* matrix X^{nan} that has the same entries as X except for a missing value indicator (e.g. *nan*) wherever the different Bernoulli processes resulted in success (i.e. the value of a particular variable in a particular observation is missing). In this way we can go from the complete data set represented in expression 3.1 to the one shown in 3.2.

$$X = \begin{bmatrix} x_{1,1} & x_{1,1} & \dots & x_{1,j} & \dots & x_{1,d} & x_{1,d} \\ x_{2,1} & x_{2,1} & \dots & x_{2,j} & \dots & x_{2,d-1} & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{i,1} & x_{i,2} & \dots & x_{i,j} & \dots & x_{i,d-1} & x_{i,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1,1} & x_{n-1,1} & \dots & \vdots & \dots & x_{n-1,d-1} & x_{n-1,d} \\ x_{n,1} & x_{n,1} & \dots & x_{n,j} & \dots & x_{n,d-1} & x_{n,d} \end{bmatrix}. \tag{3.1}$$

$$X^{nan} = \begin{bmatrix} x_{1,1} & x_{1,1} & \dots & x_{1,j} & \dots & x_{1,d} & x_{1,d} \\ nan & nan & \dots & x_{2,j} & \dots & x_{2,d-1} & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{i,1} & x_{i,2} & \dots & x_{i,j} & \dots & x_{i,d-1} & nan \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ nan & x_{n-1,1} & \dots & \vdots & \dots & nan & x_{n-1,d} \\ x_{n,1} & x_{n,1} & \dots & x_{n,j} & \dots & x_{n,d-1} & x_{n,d} \end{bmatrix}. \tag{3.2}$$

From now on, I refer to the result of this procedure as the *Bernoulli pattern of missing data*. Also, I call P_j the *missing rate* of variable j, instead of the *sample rate* referred by other authors (see e.g. Tsagkatakis et al., 2016) to avoid confusion with the rate at which observations are recorded. This model can be seen as a generalization of the random data loss model defined by Sujbert et al. (2016), and showed in Section 2.1, but applied to any number of variables. Figure 3.1 shows the result of applying the procedure just described.

The Bernoulli pattern can be summarised as follows:

Given a complete matrix $X \in \mathbb{R}^{n \times d}$ and $P \in \mathbb{R}^d$, (i.e. a set of constant probabilities for every variable in the matrix), generate an incomplete matrix $X^{nan} \in \mathbb{R}^{n \times d}$. For every variable $j \in \{1, 2, ..., d\}$, P_j is the probability of that variable being missing for any observation.

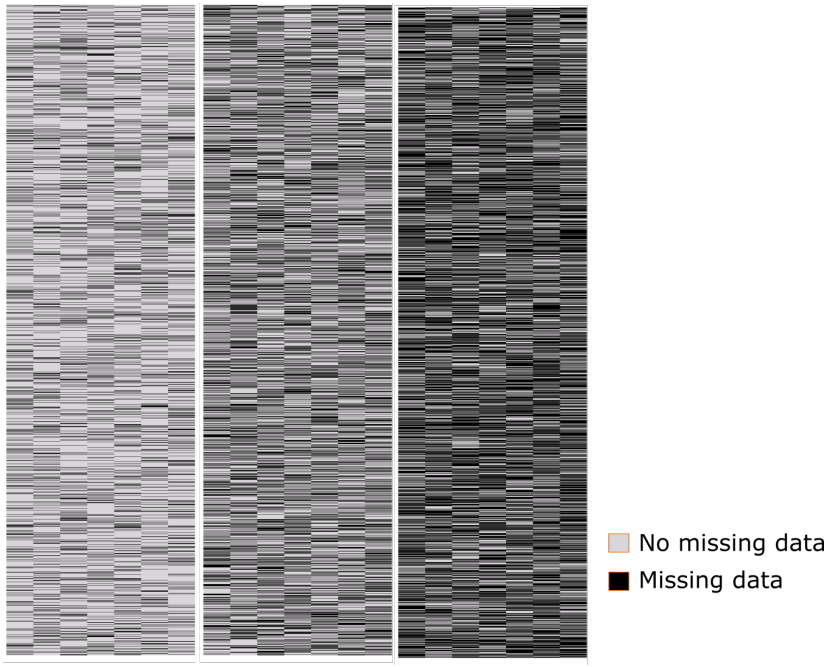


Figure 3.1: *Example of three Bernoulli missing data patterns generated by the procedure described here. The gray areas represent missing values.*

3.1.2 Column wise pattern of missing data

As seen in Chapter 2 there are situations in which the mechanism generating missing data affects some variables more than others. The resulting matrix of incomplete data may have continued and long columns with missing values and sparse missing values in the rest of the matrix.

Such situations can easily be simulated based on the Bernoulli pattern described in the previous section by means of the following procedure:

1. Randomly select a *small* subset of the complete set of variables (e.g. 20% of the variables).
2. Choose a *high* value of P_j (e.g. 0.8) for the subset of variables selected and a relatively *low* value (e.g. 0.1) for the rest of the variables. I call those missing rates P_h and P_l respectively.
3. Generate an incomplete data set using the previously defined missing rates using the Bernoulli pattern of missing data.

I chose the words small, high and low to stress the fact that the values they describe depend strongly on the context (i.e. the data, its source and application). See Figure 3.2 for three examples of what can be generated using this procedure.

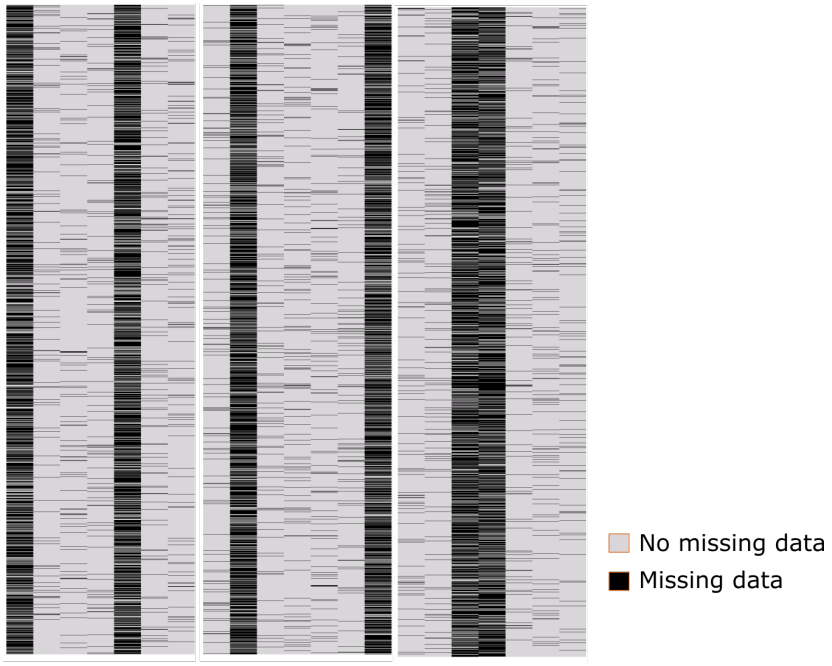


Figure 3.2: *Example of three column wise missing data patterns generated by the procedure described here.*

3.1.3 Row wise pattern of missing data

The last procedure simulates situations in which all or most of the variables are missing in some observations (rows). Such observations are usually immediate to each others, in other words, it is common to have blocks (i.e. continued observation for a set of variables) of missing data alternating with blocks of no missing data. In this case, start by defining the possible lengths for continued blocks of missing data and continued blocks of no missing data. I refer to these possible lengths by $l_{\not\varnothing}$ and l_O respectively. The symbol $\not\varnothing$ is intended to show that observations are not recorded. Also, define S_{miss} as a set of the indices of the variables (possibly random) that will be missed together. Based on that, apply Algorithm 1.

The choice of S_{miss} defines a *semi-row* (i.e. subset of the variables with unobserved values for a particular observation), unless one selects all variables instead. In accordance to the descriptions presented in chapter 2, S_{miss} would usually include almost all of the variables. To this procedure, one can additionally include a Bernoulli pattern to allow an occasional missing value appearing in the continued blocks of no missing data. This procedure can effectively simulate a row wise missing data pattern as shown in Figure 3.3.

Algorithm 1: Generate an incomplete data set with a row wise pattern.

inputs : A complete matrix $X \in \mathbb{R}^{n \times d}$, $l_{\varnothing}$ and l_O, S_{miss}

output: An incomplete matrix $X^{nan} \in \mathbb{R}^{n \times d}$

$X^{nan} \leftarrow X$;

$i =$`RandomChoiceFrom(`l_O`)`;

while $i < n$ **do**

  $i^* \leftarrow i + min(i - n,$ `RandomChoiceFrom(`$l_{\varnothing}$`)`$)$;

  foreach $j \in S_{miss}$ **do**

    $X^{nan}_{i \rightarrow i^*, j} \leftarrow nan$; `/* rows` i `to` i^* `*/`

  end

  $i \leftarrow i^*+$`RandomChoiceFrom(`l_O`)`;

end

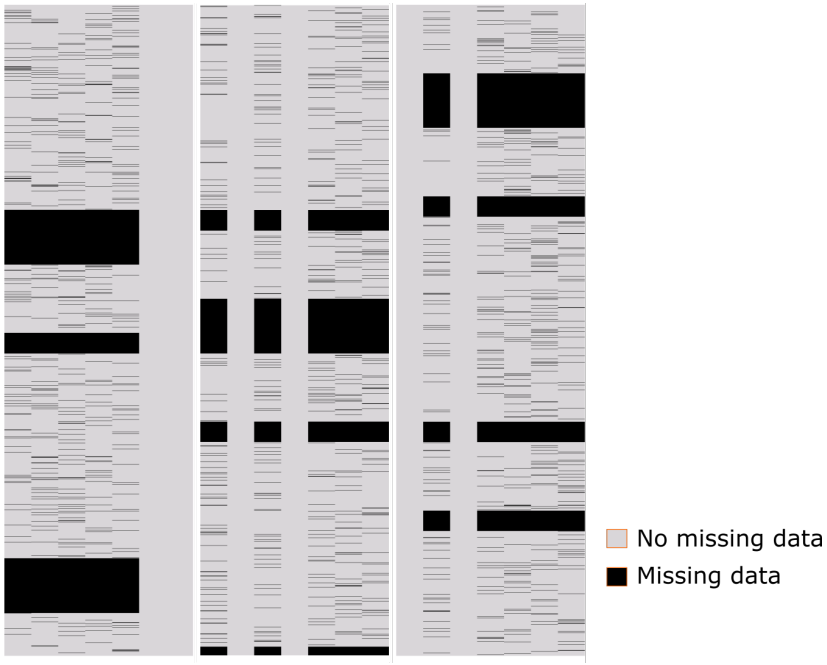


Figure 3.3: *Example of three row wise missing data patterns generated by the procedure described here.*

3.1.4 Estimating performance

In the case of the Bernoulli missing data pattern described in Section 3.1.1, one can set the missing rate of every variable to be the same (i.e. $P_j = k \; \forall j$), in which case the generated incomplete data set

will simulate the same conditions for missing values in all variables. If the data suggests a particular value of k, one can easily estimate the reconstruction error of any imputation method by comparing a *reconstructed* matrix from the original one.

Nonetheless, there may not be a particularly interesting value for k, in which case simulations may use increasing values of k and a qualitative performance measure can be done by plotting a line of how the reconstruction error varies along the different values of k for every imputation method. Indeed, this approach, without the precise description I gave, was used by some authors (see e.g. Tsagkatakis et al., 2016) to estimate the performance of their imputation methods.

In addition, a numeric metric may be of use in order to test the relative advantage, in terms of the reconstruction error, of using one imputation method over the other. This can be achieved by simply computing the Area Under Error Curve (AUEC) of the line explained above, which can be computed as in Algorithm 2.

Algorithm 2: Compute the AUEC based on the Bernoulli pattern.

inputs : A complete matrix $X \in \mathbb{R}^{n \times d}$, P_{start} and P_{stop}, $step$, $method$

output: AUEC

$K \leftarrow \{P_{start}, P_{start} + step, P_{start} + 2 \times step, ..., P_{stop}\}$; `/* e.g.` $0.02, 0.04, ..., 0.6$ `*/`

$error \leftarrow$ *empty array*;

foreach k *in* K **do**

  $X^{nan} \leftarrow$ `ProduceBernoulliPattern(`X, k`);`

  $\hat{X} \leftarrow$ `MakeImputation(`$X^{nan}, method$`);`

  `StoreIn(`$error$`,` `ComputeReconstructionError(`$X, \hat{X}$`));`

end

return `NumericIntegration(`$abscissa = K, ordinate = error$`)`

In contrast, computing any form of reconstruction error using the column or row wise missing data patterns immediately leads to a numeric mean of comparing any two or more imputation methods. Moreover, One can repeat the process a number of times with random choices for the variables most affected, which serves to perform statistical tests on the results and conclude whether the results are statistically significant.

On a data set composed of a handful of variables, the performance measures based on the column and row wise patterns can be evaluated over all possible selections of the small and big subsets respectively, but a feasible number of random selections should be adopted otherwise.

Dissimilarly, even if there are only a handful of variables, repeated estimation of the AUEC may be computationally prohibited. Unfortunately, it seems to be the most reliable way of comparing

several imputation methods using this metric. However, the results I present in Chapter 4 suggest that the performance based on the AUEC of different imputation methods does not vary considerably along the number of repetitions (see Section 4.4).

Finally, as stated in Chapter 2, a real world data set may combine all the aforementioned patterns; however, if one considers them all at once, the permutations are huge and the results may be confusing. Therefore, it is more practical and informative to use them separately.

3.2 Supervised Learning for Imputation

This section corresponds to an analysis of different techniques that will help to fulfil specific objective (II); nonetheless, this objective is completed in the following section.

As mentioned in Section 1.2, supervised learning is a powerful tool to extract patterns from data. Such patterns can also be exploited to make accurate imputation. Let me first explain why using supervised learning makes sense. As explained in Chapter 1, one of the most promising areas of the IoT is the development of smart applications or systems that analyse their environment or respond autonomously to it. Such systems can only be created by the introduction of artificial intelligence into products. In particular, I focus on applications that are based on supervised learning.

Any supervised learning application can be seen as composed by two stages: i) *development time* (or *stage*), when the developer trains one or several models and computes some estimate of their performance; and ii) *operation time*, which correspond to the application of one or any of the models that were trained in development time.

During development time the developer defines which variables will be used as input $\mathbf{x}$ and which variable y or, more generally, which variables $\mathbf{y}$ will be targets, which depends entirely on the application.

Proper training of the model requires the training data set $\mathcal{D}$ to have no missing values. Therefore, an application must start with a complete data set, which can not only be used to train for the application of primal interest, but also to train an imputation method to be used during the operation stage of the application.

To do this, simply remember that $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ and thus, such training data set can said to be composed of a complete matrix X and a complete vector $\mathbf{y}$:

$$X = \begin{bmatrix} x_{1,1} & x_{1,1} & \dots & x_{1,j} & \dots & x_{1,d} & x_{1,d} \\ x_{2,1} & x_{2,1} & \dots & x_{2,j} & \dots & x_{2,d-1} & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{i,1} & x_{i,2} & \dots & x_{i,j} & \dots & x_{i,d-1} & x_{i,d} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1,1} & x_{n-1,1} & \dots & \vdots & \dots & x_{n-1,d-1} & x_{n-1,d} \\ x_{n,1} & x_{n,1} & \dots & x_{n,j} & \dots & x_{n,d-1} & x_{n,d} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}.$$

To train imputation methods one simply takes the complete matrix X. In this chapter I also refer to this matrix as a training data set, in the rest of the manuscript the context in which I refer to the training data set should make clear whether I refer to X or $\mathcal{D}$. In the following discussion I use index t instead of i to emphasize the temporal structure in the composition of X and $\hat{X}$.

Clearly, when such complete data set is not available, one can use an approach that does not require training. See Table 2.2 in page 25 to identify possible methods for this.

3.2.1 Exploiting relation between variables

Let $\mathbf{b}$ be a vector of the same size of $\mathbf{x}$ to have ones in position j if variable x_j is missing from vector $\mathbf{x}$ and zeros otherwise. And let S_j be the set of all possible vectors $\mathbf{x}^*$ of length $< d$ that do not contain x_j: $S_j = \{\mathbf{x}^* : b_j = 1\}$.

One approach to impute missing values using supervised learning is to make multiple regressions. Each of these regressions uses one of the possible elements of S_j as the input vector to predict x_j. Mathematically, $x_j \approx \hat{\mathbf{f}}_j(\mathbf{x}^*)$ for all $\mathbf{x}^* \in S_j$. This time I used boldface $\hat{f}$ to indicate that there should be several models, one for every possible vector of S_j. The reason for having all such models is to be able to predict variable x_j regardless of the number of missing values present in an observation of $\mathbf{x}$, provided that there is at least one variable in $\mathbf{x}$ that is not missing.

Such approach should be repeated d times, one for every column of the data set. Therefore, although feasible for a small number of columns, its complexity increases rapidly. Specifically, the number of models that should be trained is given by

$$d \times \sum_{k=1}^{d-1} \binom{d}{k}. \tag{3.3}$$

Clearly, one can assume that there would be only one missing variable in $\mathbf{x}$ (see Pan et al., 2013, 2014), reducing the number of models significantly; however, I have already explained in Chapter 2 that such approach is unsatisfactory for what can occur in data from the IoT. Moreover, even if

computing all models is feasible, regardless of their accuracy, such approach would break when all elements in $\mathbf{x}$ are missing.

3.2.2 Exploiting temporal structure in the variables

One can assume that the value of $x_{t,j}$ can be approximated by a function of the first few previous values of the same variable, that is,

$$x_{t,j} \approx \hat{f}(x_{t-1,j}, x_{t-2,j}, ..., x_{t-p,j}). \tag{3.4}$$

where p is the number of previous steps that are used to estimate the value of $x_{t,j}$, sometimes referred as a *window size*. Such a expression can easily be written in a most compact form as

$$\mathbf{x}_t \approx \hat{\mathbf{f}}(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, ..., \mathbf{x}_{t-p}). \tag{3.5}$$

Note that the assumption of this model implies that there are several models, one for each variable j.

Clearly, another assumption of this model is that the first p rows of an incomplete matrix X^{nan} do not have missing values (or that the first p observations in operation time of a given application do not have missing values), afterwards, a missing value can be simply replaced by its prediction and be used for subsequent predictions.

3.2.3 Regression methods

I have explained how to produce imputation methods with a supervised learning approach. In particular, this way of applying supervised learning can be seen as a regression task, because the values to predict represent continuous variables (e.g. sensor readings). Consequently, I refer to the kind of methods that apply to the above explanation as regression methods.

Currently, there is a great deal of regression methods that are easily available in different computer languages. Even so, those can be easily divided in linear and non linear methods. A linear method is one that considers that points $(\mathbf{x}_i, y_i)$ are described by a *straight line* or a *hyperplane*. In other words, that the relation of $\mathbf{x}$ and y is given by

$$y = w_0 + \sum_{j=1}^{d} w_j x_j. \tag{3.6}$$

Finding the different values of w_j, called *weights*, is what the learning algorithm does. In fact, the difference between the several linear regression methods that are available today is the way in which the parameters w_j are calculated. For compactness, it is always assumed that vector $\mathbf{x}$ has a variable $x_0 = 1$ so that one can write the above relation simply as

$$y = \sum_{j=0}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}, \tag{3.7}$$

which is called dot product notation.

Consequently, a supervised learning approach for imputation, which only exploits the relation between variables, can be written as

$$x_j = \mathbf{w}_s^T \mathbf{x}_s^*, \tag{3.8}$$

where I used the index s to emphasize that a different set of weights is used for every vector $\mathbf{x}^*$ in S_j.

Similarly, a simple linear regression that exploits the temporal structure in the data can be written as

$$\mathbf{x}_t = \sum_{k=1}^{p} \mathbf{w}_{t-k}^T \mathbf{x}_{t-k}. \tag{3.9}$$

As explained in Chapter 2, this type of relationship may not be enough to model the relationships that may be encountered in real world data sets. In contrast, there are regression methods that are non linear, that is, the way they relate $\mathbf{x}$ and y is different from what I just showed. For example,

$$y = w_0 + w_1 x_1^2 + w_2 x_1 x_2 \tag{3.10}$$

is a particular form of Polynomial Regression. More generally, such a type of regression can be written as

$$y = \mathbf{w}^T \phi(\mathbf{x}), \tag{3.11}$$

where $\phi(\mathbf{x})$ is a polynomial transformation of the vector $\mathbf{x}$, that may include multiplication of different x_j, x_j^2, etc., which causes the resulting vector to be of length $> d$. Vector $\mathbf{w}$ has the proper number of elements: one element for each element of $\phi(\mathbf{x})$.

Another example of a non linear regression is Support Vector Regression (SVR) with non linear *kernels*. The kernel is basically the same as the ϕ transformation shown above, but imposes a definite form of the non linear expression. This, although more powerful than a simple linear regression, has the same limitation for the application in the IoT: due to heterogeneity in the possible data sources, a proper imputation method should no make any assumptions regarding the temporal structure of the data or the relation between the variables.

There are much more regression methods. However, they would all require one to choose between exploitation of temporal structure and relation between variables. One can obviously train two types of models, as did some of the authors referenced in the previous chapter, but the disadvantage of such approach has already been discussed.

3.3 Imputation method for the IoT

I now turn to particular method of supervised learning, which is ideal for making imputation in the IoT context. This section completes the treatment of specific objective (II).

3.3.1 Introduction to Recurrent Neural Networks

I start this section with an introduction to feed-forward neural networks[1] to then extend the idea to Recurrent Neural Networks (RNNs). Afterwards, I show the reason to propose RNNs as an imputation method in the context of the IoT. This section assumes a basic understanding of supervised learning, please refer to Subsection 1.2.1 if you are not familiar with the basics.

The reader is warned that feed-forward neural networks and RNNs are both sub fields of Artificial Neural Networks (ANNs), which is a very active area of research, and that proper understanding requires considerable machine learning background. Therefore, I only explain what is necessary to understand this manuscript and I try to show it in a way that is easily followed by a general audience.

3.3.1.1 Feed-forward Networks

Consider the diagram shown in Figure 3.4. This diagram represents a linear regression as I have already described ($y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$).

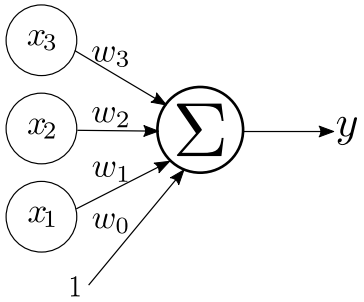


Figure 3.4: *A graphical representation of a linear regression with three input variables.*

Feed-forward neural networks extend this function by *i*) adding several stages of computation called *hidden layers* or *hidden states*; and *ii*) introducing non-linearities, called *activation functions* after the linear combination at each of the stages. Thus, a typical feed-forward neural network looks like the diagram in Figure 3.5. A network of this type can be built with any number of input variables, hidden layers and number of units per layer. In this representation $\phi^{(l)}$ is the activation function used at the l hidden layer. The connection of every unit at layer l with every unit of layer $l + 1$ has a different set of weights like those w_j shown in Figure 3.4; however, I omitted w_0 so that Figure 3.5 is easier to read.

The relation between the input and the output, also known as *hypothesis function*, for this feed-forward neural network is represented by equations 3.12 to 3.14. Note that there is now one $\mathbf{w}$ per hidden node in subsequent layers and they are stored in a matrix W; the number in parenthesis indicates the layer in the network; all vectors are column vectors; and $\phi^{(l)}$ is applied element-wise.

[1] I consider this a necessary step to understand the operation of Recurrent Neural Networks.

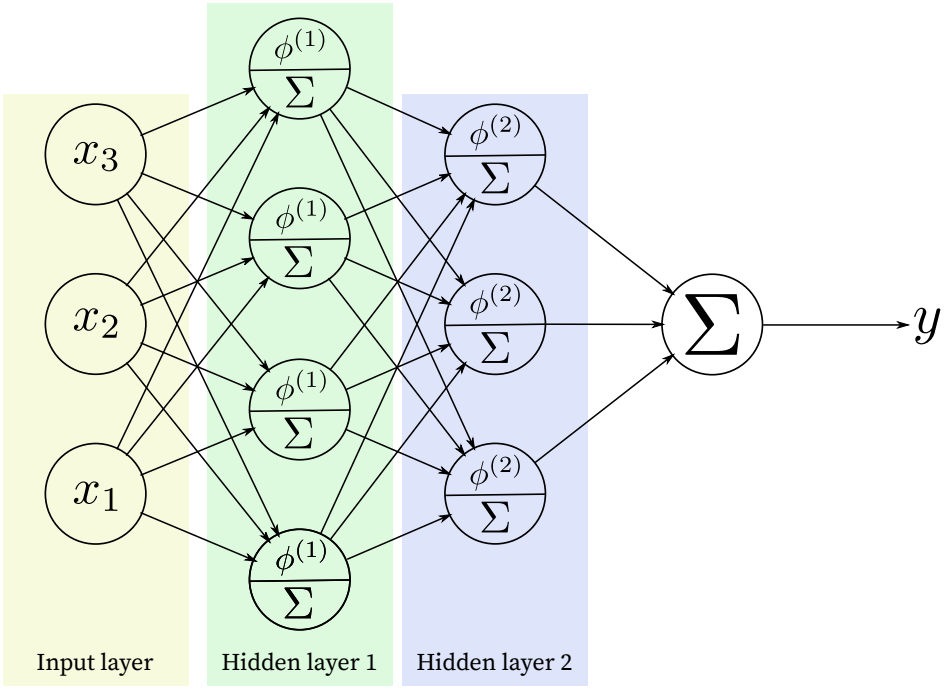


Figure 3.5: *A feed-forward neural network with three input variables and two hidden layers with four and three units each.*

$$y = W_{(2)}^T \, \mathbf{h}^{(2)} \tag{3.12}$$

$$\mathbf{h}^{(2)} = \phi^{(2)}(W_{(1)}^T \, \mathbf{h}_1) \tag{3.13}$$

$$\mathbf{h}^{(1)} = \phi^{(1)}(W_{(0)}^T \, \mathbf{x}) \tag{3.14}$$

The term feed-forward refers to the way in which information from the input goes to the output through the hidden layers. They are called neural networks for the way they are represented by connected elements of computation, called *neurons* or, more recently, *units*.

The connections with the units at one layer with the units of subsequent layers, together with the non-lineal activation functions is what makes feed-forward neural networks, and ANNs in general, capable of modelling complex relationships between the input and the output, as well as within the inputs (provided it helps to better map the inputs to the outputs).

3.3.1.2 Recurrent Neural Networks

If one feeds a feed-forward neural network, one vector at the time, its output would depend only on that input vector. In contrast, the output of a RNN would also depend on the output of previous input vectors, which is achieved by training the network with input vector $\mathbf{x}_i$ together with output

from previous training steps as shown in Figure 3.6. Note that I will continue to use a representation of a one layer RNN so that the image is clear, but it is easy to imagine subsequent layers on top of each layer at each step.

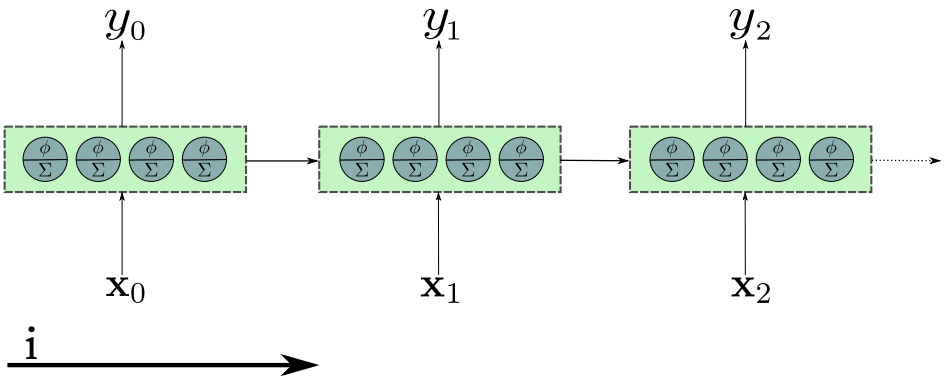


Figure 3.6: *A one-layer RNN.*

Equations 3.15 and 3.16 show the way in which the RNN of figure 3.6 computes the output given the input. Here, I used the index t to emphasize that outputs are computed sequentially. Note that there is a new set of weights W_h (implicit in the horizontal lines of Figure 3.6, which relate previous values of the hidden layer to future ones (the essence of RNNs); consequently, I used W_x (implicit in the vertical lines from the inputs to the hidden layer of Figure 3.6) to emphasize that this set of weights is only applied to the input.

$$y_t = W_y^T \, \mathbf{h}_t \tag{3.15}$$

$$\mathbf{h}_t = \phi(W_x^T \, \mathbf{x}_t + W_h^T \, \mathbf{h}_{t-1}) \tag{3.16}$$

Thus, RNNs have sequence-like connections within the input variables and between input and the output of the hidden layers, which is why they have shown outstanding results for applications in which the input or output at time $t-p$ somehow has useful information regarding the output at time t. For instance, doing text analysis and generation, where a word in certain location of a sentence greatly influences the words to be expected latter in the same sentence.

In the literature of RNNs, the different units in the hidden layers are usually called *memory cells*, which in general represent more complex computations that the traditional hidden layers of feed-forward neural networks. The memory cell shown in figure 3.6 represents one of the simplest types memory cells that have been developed. In this project, I also considered the use of Long-Short Term Memory (LSTM) and Gated-Recurrent Units (GRU) cells; I do not, however, explain their differences, concentrating only on the empirical results of applying them as an imputation method for the IoT. Their details can be found in Hochreiter and Urgen Schmidhuber (1997) and Cho et al.

(2014) respectively.

3.3.2 Recurrent Neural Networks as an Imputation for data from the IoT

As stated in Chapter 2, data from the IoT has the following two main characteristics: i) data is time series data. ii) there can be linear and non-linear relationships between the different variables to consider for a certain data analysis task. Such conditions, and the previous description of RNNs, suggest their use in the context of the IoT. Nevertheless, the introduction above needs to be re-formulated here, as was done previously for supervised learning methods in general.

I have presented RNNs as a way of learning a mapping from input vector $\mathbf{x}$ to a single output variable y. However, Figure 3.7 represents a mapping from input vector $\mathbf{x}_t$ to output vector $\mathbf{x}_{t+1}$, which shows the use of a RNN as an imputation method: use *explicitly* data measured at time t, and *implicitly* use measured data up to time t to predict data to be measured at time $t+1$. Note that I now use the index variable t instead of i to remind the reader that data is measured sequentially along time; also note in Figure 3.7 that I used a grey dashed arrow to distinguish between what is estimated and what is observed.

A data set with missing values is then filled if one replaces missing values at time $t+1$ with their corresponding predictions. Equations 3.17 and 3.18 are the operations represented in Figure 3.7.

$$\mathbf{x}_{t+1} = W_{x_{t+1}}^T \, \mathbf{h}_{t+1} \tag{3.17}$$

$$\mathbf{h}_{t+1} = \phi(W_{x_t}^T \, \mathbf{x}_t + W_{h_t}^T \, \mathbf{h}_t) \tag{3.18}$$

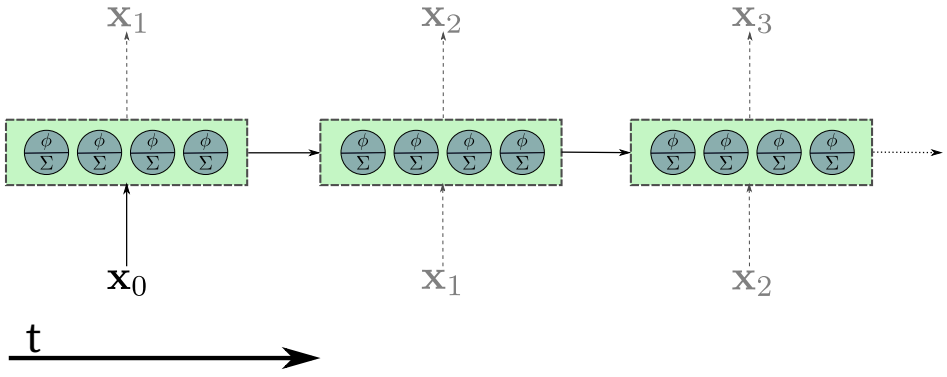


Figure 3.7: *A one-layer RNN that learns a mapping from* $\mathbf{x}_t$ *to* $\mathbf{x}_t + 1$.

Since RNNs explicitly use vector $\mathbf{x}_t$ to predict vector $\mathbf{x}_{t+1}$ it is assumed that there are no missing values at time t. Moreover, one can also explicitly use a time window of data from time $t-p$ to time t (see figure 3.8) and use it to predict the values at time $t+1$, in which case there should be no missing values in the first p observations in the data set. Furthermore, RNNs can be built with any

number of layers and nodes per layer. All these choices are known as *hyperparameters*, that is, a set of parameters that is not learnt by the training algorithm, but decided by the user of the ANN. In Chapter 4 I evaluate RNNs with a limited range of values for these two hyperparameters and show the argument for the particular choices.

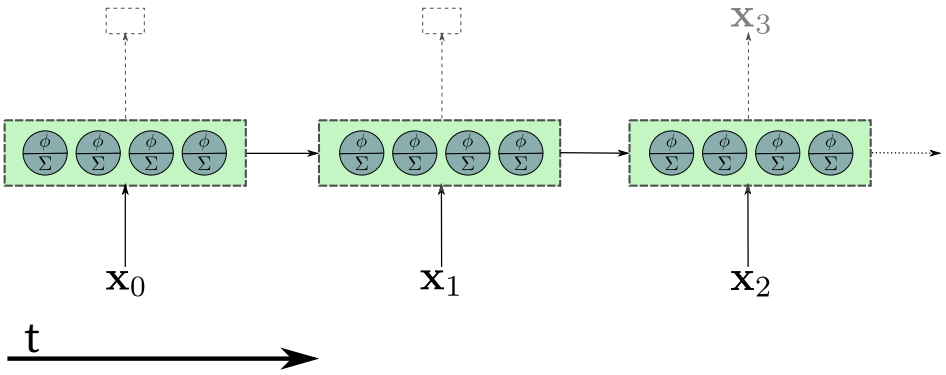


Figure 3.8: *A RNN that explicitly uses a time window of data with $p = 3$ to predict the following input vector. Note that the first $p - 1$ outputs are ignored.*

As with all supervised learning approaches, this way of using RNNs as an imputation method assumes that one can previously train the network using training data with no missing values. This assumption may be violated in several real world scenarios, but if the measured data is used in another prediction model (e.g. any of the examples of chapter 1) then a valid training data set must be available nonetheless. A method can be developed that uses RNNs without the restriction of having a valid training data set, but it is out of the scope of this work.

Note that RNNs exploit the temporal structure and the relation between variables at the same time. A particular prediction of x_j is thus influenced by i) the value of the same variable at previous time steps; ii) the value of other variables at the current time steps; and iii) the value of other variables at the previous time steps. In contrast to RNNs, other supervised learning methods cannot exploit all this features so seamlessly. Moreover, since at every time steps there are predictions of every variable, this method would not break even if there are continued missing values for all variables.

Chapter 4

Evaluation of RNNs as imputation method

In this chapter I show the empirical results obtained using RNNs as an imputation method on the three patterns described in Section 3.1. These results are divided in three sections. The first two sections show that the RNNs approach for imputation is competitive with other imputation methods, which is the main purpose exposed in the general objective presented in Section 1.3. The last section of this chapter demonstrates that accurate imputation methods effectively improve the reliability of smart applications in the IoT.

To say that RNNs are competitive with other imputation methods, I tested them against the following two open source data sets: i) individual household electric power consumption data set[1]; and ii) Air quality data set [2].

Since both data sets have the problem of missing data, no imputation method can be properly tested using the evaluation procedures explained in Section 3.1. On the one hand, this certainly stresses the justification for this whole project provided in Chapter 1. On the other hand, as mentioned in Chapter 2, this issue threatens the reproducibility of experimental results. Consequently, I performed certain preprocessing steps (besides scaling) for each data set and describe those in their corresponding chapters. Scaling each variable to have the same range (in particular I scaled every variable of each data set in the range $[-1, 1]$), is a necessary step in order to compute the reconstruction error properly. Otherwise the reconstruction error would be affected by the scale of the physical quantities of each of the measured variables.

[1] Dowloaded from `https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption`

[2] Downloaded from `https://archive.ics.uci.edu/ml/datasets/Air+Quality`

As explained in Chapter 3, RNNs have a number of hyperparameters; one can decide over the number of layers, the activation function in each layer, the number cells per layer and the number of explicit time steps to consider (a.k.a window size). Moreover, different training strategies may change the effect of each of these parameters. It is impossible to exhaustively consider all possible permutations of the hyperparameters mentioned above, which is indeed an infinite amount. Even a small but representative subset is unaffordable. Therefore, I decided to fix the following:

- I trained all RNNs using early stopping (see Goodfellow et al., 2015, p. 242).
- I used relu activation function (see Goodfellow et al., 2015, p. 174) for RNNs with basic cells, whereas LSTM and GRU cells used the default configuration as exposed in their original papers.
- Since deeper networks are harder to train, I limit the number of layers of all RNNs to just one.

Limiting the number of layers to just one reduces the capacity of RNNs, which means that they may not be able to learn highly complex non-linear relationships between the variables or their temporal relation; however, it is still possible that the RNNs learn useful relationships. Even in this case the following three questions arise:

- Which type of cell performs best?
- How many cells to use?
- How does the length of the time window of data affects performance?

One can understand every possible choice of memory cell as a different method (all based on RNNs) and evaluate the use of different number of cells and different sizes of time window separately. This combinatorial nature of all possible configurations of RNNs greatly limits the search for a definite answer of the previous questions, even for a particular data set. For this reason, any attempt to answer these questions must be bounded; specifically, I vary the following set of hyperparametrs (see Chapter 3), which lead to 300 different models:

- Type of cell: {Basic cell, LSTM cell, GRU cell}
- Number of cells in the hidden layer: $\{20, 40, \ldots, 200\}$
- Length of the time window: $\{10, 20, \ldots, 100\}$

In sections 4.1 and 4.2 I start by evaluating the aforementioned possibilities of RNNs using the AUEC described in Section 3.1. Latter, I test the relative advantage of RNNs as imputation method in the IoT. For this I use the three patterns of missing data described in the previous chapter and compute reconstructed matrices $\hat{X}$ with the following four methods (besides RNNs):

- Impute with the mean value of every variable.
- Impute with the median value of every variable.
- Impute the last recorded value of every variable.
- Imputation method based on Principal Components Analysis (PCA) as implemented in the

missMDA R's package.[3]

I found these are not the most modern alternatives for imputation methods in the context of the IoT, but programming more recent methods found in the literature turned out to be unaffordable for this project.

The reader must be aware of the exploratory nature of the experiments and results I present here. In the question of what is the right set of hypermarameters for RNNs to work as imputation methods, it is important to mention that any answer to this question is limited to an specific data set. Furthermore, the question of what is the most appropriated imputation method in the context of the IoT is as limited (or even more) than the previous one.

All the experiments of sections 4.1 and 4.2 are related to specific objective (III). Moreover, in Section 4.3 I present an experiment aimed to fulfil specific objective (IV), that is, an experiment on the impact of imputation methods on an smart application in the context of the IoT.

4.1 Data Set 1: Individual Household Electric Power Consumption

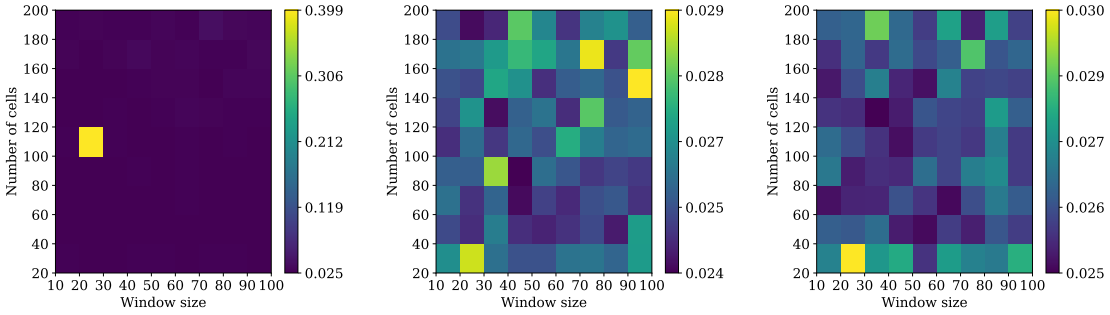
In this data set there are measurements of a household electrical equipment consumption averaged over every minute. The variables include the global active and reactive power, as well as the voltage and current intensity. There are also measurements in watt-hour of active energy of the kitchen and laundry room, and one dedicated to the water heater and air conditioner. There are 7 variables in total.

There is a total of 2,075,259 measurements gathered between December 2006 and November 2010. However, due to the presence of missing data, I only experimented with the values from 5:25pm December 16, 2006 to 8:51pm February 20, 2007 which gives a total of 95,249 measurements without missing values in any variable. Finally, I halved this data set and used the first part to train the RNNs, whereas I tested the the performance of the different imputation methods with the second.

4.1.1 Evaluation of RNNs Hyperparameters

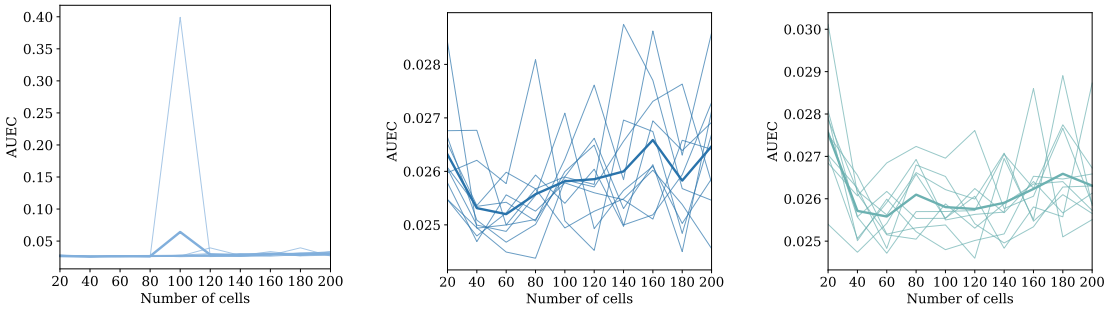
Figure 4.1 shows how the AUEC varied for all permutations of the hyperparameters mentioned at the beginning of this chapter. It can be seen that no obvious pattern is revealed: regardless of whether the number of nodes and length of time window increases (or decreases), there does not seem to be an effect on the error measure. This is further stressed in figures 4.2 and 4.3, which show, for every point on the x-axis, which corresponds to one of the parameters of Figure 4.1, ten different models (one for every possible value of the other parameter); the average is represented in a darker line.

[3] `https://cran.r-project.org/web/packages/missMDA/index.html`.



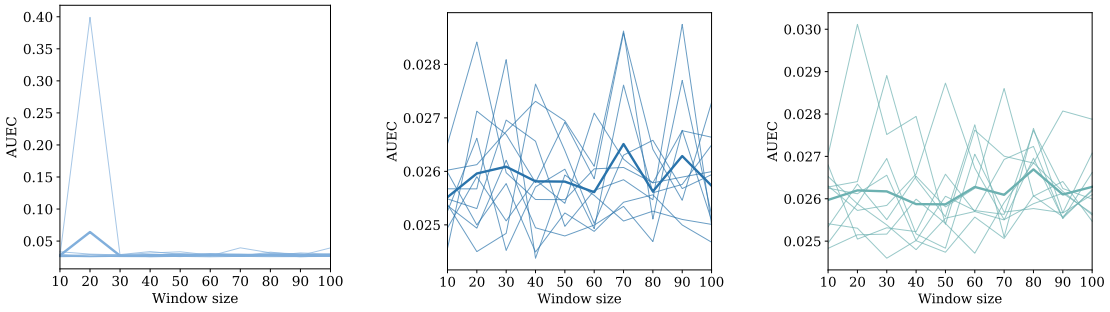
(a) *Basic recurrent cell* **(b)** *LSTM cell* **(c)** *GRU cell*

Figure 4.1: *AUEC with respect to the number of nodes and length of time window.*



(a) *Basic recurrent cell* **(b)** *LSTM cell* **(c)** *GRU cell*

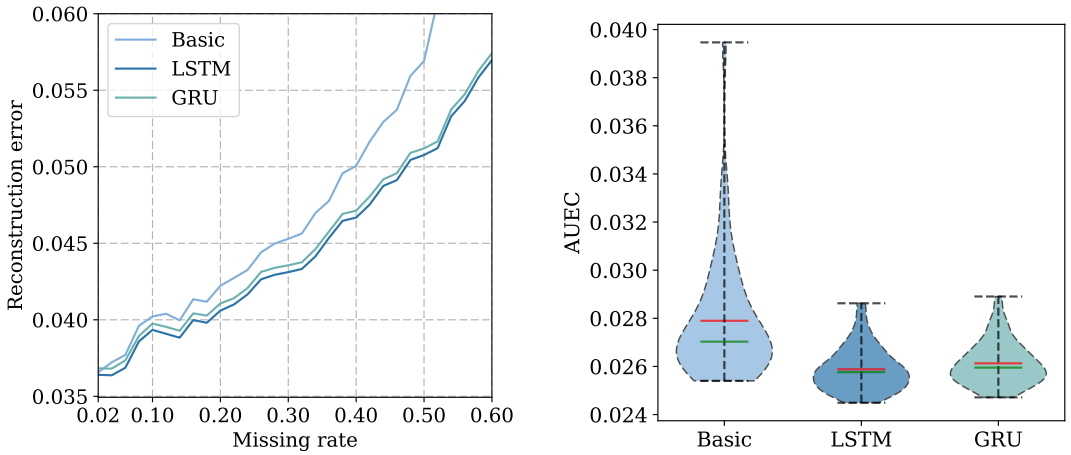
Figure 4.2: *AUEC with respect to the number of nodes.*



(a) *Basic recurrent cell* **(b)** *LSTM cell* **(c)** *GRU cell*

Figure 4.3: *AUEC with respect to the length of time window.*

There is a slight difference in the observed AUEC between the different type of cells, which is clear in Figure 4.4; note in this figure, I removed the most extreme values from the violinplot so that the outlier shown in Figure 4.1a does not affect the scale of the image. This result leads to the question of whether this difference is statistically significant.



(a) **(b)**

Figure 4.4: *(a) Mean of the error against missing rate. (b) Violin plot of the AUEC for the three cell types, the red and green lines represent the mean and median values respectively.*

To answer this question I used a Monte Carlo analysis as explained by (Gotelli and Ellison, 2004). The reason for this choice of statistical analysis is that some assumptions of parametric tests (see Larsen et al., 2012, ch. 5-14) are not met, namely [4]

1. Different models of each unit type do not correspond to a random representative sample of the population (i.e. I chose the hyperparameters by hand in a population of infinite size).
2. Since variations of the hyper parameters are discrete, the AUEC is not a continuous random variable.

The test statistic I chose for the Monte Carlo analysis is the difference between two of the median values obtained when using each type of cell. I chose the median instead of the most commonly used mean, because the median is more robust to outliers, and Figure 4.1a shows an obvious outlier in the results of models using the basic cell.

The results shown in Table 4.1 correspond to ten thousand Monte Carlo simulations. Note that they only suggest that the difference found between the medians of the groups is statistically significant, but they do not rule out the possibility of a RNN based on basic cells outperforms other networks if using the proper set of hyperparameters.

4.1.2 Comparison with other imputation methods

Here I compare four imputation methods and a RNN with the three different missing data patterns that may arise in the IoT explained in Chapter 2. Given that the best results in all RNNs were found

[4] And all assumptions that may follow from these.

Table 4.1: *Probability that the difference in the median value of pairs of cells' type is equal to or greater than the observed difference if the AUECs had come from the same cell type.*

	$P(DIF \geq DIF_{obs})$
Basic-GRU	0
Basic-LSTM	0
GRU-LSTM	0.21

using LSTM and GRU cells, I decided to randomly pick a model using GRU cells as a representative model of RNNs. The selected model has 40 cells and 30 steps of window size.

4.1.2.1 Comparison in a Bernoulli pattern

Figure 4.5a highlights the mean reconstruction error obtained in 30 repetitions along different values of missing rate. It also shows the variation of the reconstruction error (i.e. the minimum and maximum) obtained in the repetitions. However, due to the big difference between in performance of the different imputation methods, this variation is barely noticed. The corresponding AUEC is shown in figure 4.5b.

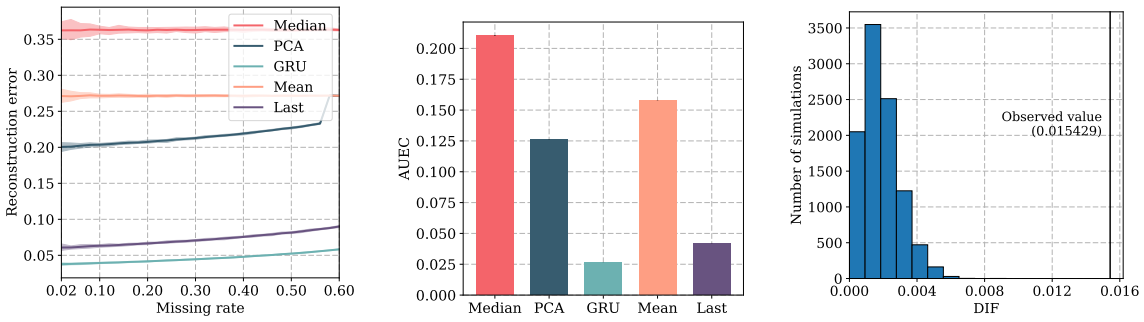
Table 4.2 shows the mean value of the five imputation methods tested, together with their standard deviation, minimum and maximum values observed in 30 incomplete data samples. Surprisingly, imputing the last recorded value in every variable, which is a simple method in terms of its formulation, shows lower reconstruction error than the Principal Components Analysis (PCA)-based method.

Table 4.2: *Summary of the comparisons with the AUEC metric.*

			Extreme values	
	Mean	Standard deviation	Minimum	Maximum
Last	0.0420	0.75e−4	0.0419	0.0421
Mean	0.1575	1.15e−4	0.1572	0.1578
Median	0.2106	2.52e−4	0.2100	0.2113
PCA	0.1259	0.96e−4	0.1257	0.1261
GRU	0.0266	0.40e−4	0.0265	0.0267

Since all five mean reconstruction error lines in 4.5a look very far apart, I tested for statistical significance only between the two methods that performed best, namely, imputation with last recorded value and imputation with the RNN. Again, I used a Monte Carlo analysis with ten thousand sim-

ulations (see Figure 4.5c); however, this time I tested the difference between the mean of the two methods, because there were not outliers in the results. The difference between the mean of the two methods was not observed in any of the Monte Carlo simulations; therefore, it is unlikely that the great difference between the two methods is due to randomness.



(a) **(b)** **(c)**

Figure 4.5: *(a) Reconstruction error against different values of missing rate of five methods. (b) AUEC. (c) Monte Carlo simulations testing the significance of the observed difference.*

4.1.2.2 Comparison in a column wise missing data pattern

For this test I chose $P_h = 0.7$ for two variables and $P_l = 0.05$ for the rest (see Subsection 3.1.2). Since there were only 7 variables in this data set, I was able to test performance against the 21 (7 choose 2) possible combinations.

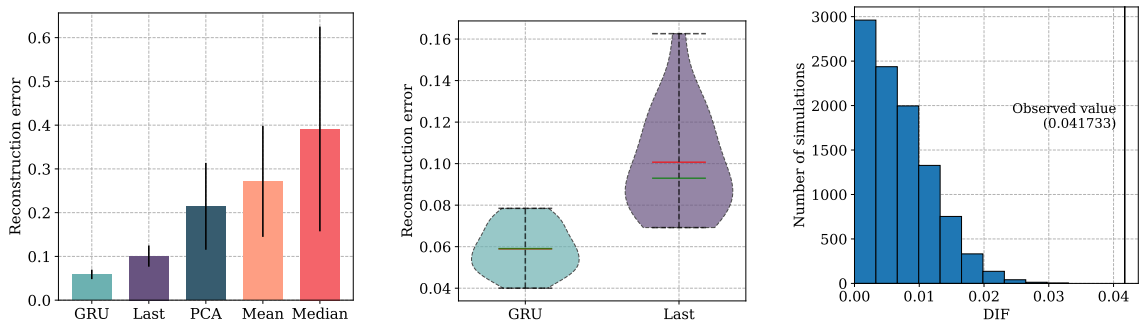
Figure 4.6 summarizes the results of the experiment over all 21 possible combinations, showing that the method based on RNNs with GRU cells outperformed all others. Moreover, a Monte Carlo analysis discards the hypothesis that the difference between the means of the best and second best performing methods in this experiment is due to randomness (see Figure 4.6c).

In addition, Table 4.3 shows the mean, standard deviation and extreme reconstruction errors obtained using each method.

In summary, it is evident that the method based on RNNs also outperforms all other four methods in a column wise missing data pattern.

4.1.2.3 Comparison in a row wise missing data pattern

In this test I simulated that l_O ranges from 5000 to 15000 observations, whereas $l_{\not O}$ ranges from 1500 to 4000 observations (see Subsection 3.1.3); I also simulated that five of the seven variables were part of the semi-rows, and reproduce the experiment for all possible 21 choices of those five variables. Finally, I added a Bernoulli independent pattern of missing data to all variables using $k = 0.05$

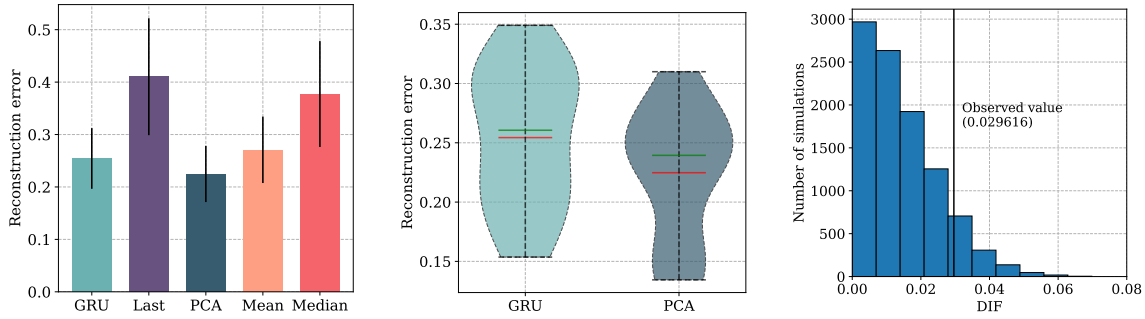


(a) **(b)** **(c)**

Figure 4.6: *Results in a column wise pattern. (a) Mean reconstruction error. (b) Results of the two best performing methods. (c) Significance of the observed difference.*

Table 4.3: *Summary of the experiments in a column wise pattern*

			Extreme values	
	Mean	Standard deviation	Minimum	Maximum
Last	0.1007	0.0242	0.0692	0.1626
Mean	0.2716	0.1270	0.1166	0.4946
Median	0.3913	0.2339	0.2297	1.0314
PCA	0.2144	0.0992	0.1139	0.3909
GRU	0.0589	0.0109	0.0400	0.0785



(a) **(b)** **(c)**

Figure 4.7: *Results in a row wise pattern. (a) Mean reconstruction error. (b) Results of the two best performing methods. (c) Significance of the observed difference.*

Figure 4.7 and table 4.4 show that this time, the method based on PCA outperformed all others, with the method based on RNNs with GRU cells being second. However, a Monte Carlo analysis

revealed that the difference between these two methods is not statistically significant at the 0.05 significance level (the obtained *P-value* was 0.100), which is a standard usually used in statistical hypothesis testing. From figure 4.7a it seems that even imputing with the mean of every variable showed similar results to the best two methods; the difference may not be statistically significant, but I did not address this numerically.

Table 4.4: *Summary of the experiments in a row wise pattern.*

			Extreme values	
	Mean	Standard deviation	Minimum	Maximum
Last	0.4103	0.1115	0.2153	0.5846
Mean	0.2708	0.0633	0.1588	0.3409
Median	0.3772	0.1010	0.2181	0.4912
PCA	0.2248	0.0536	0.1344	0.3099
GRU	0.2544	0.0579	0.1536	0.3491

4.1.3 Summary

The experiments made on the Individual Household Electric Power Consumption data set show that the approach based on RNNs is a promising alternative to other imputation methods. In particular, RNNsshowed a remarkable superiority to other imputation methods when tested in the Bernoulli and column wise patterns; moreover, although RNNs was the second best imputation method in the row wise pattern, the difference with respect to the best method in this pattern is not statistically significant.

4.2 Data Set 2: Air Quality

This data set contains hourly averages from an air quality chemical multi sensor device[5]. It has a total of 9,458 observations and the 13 variables.

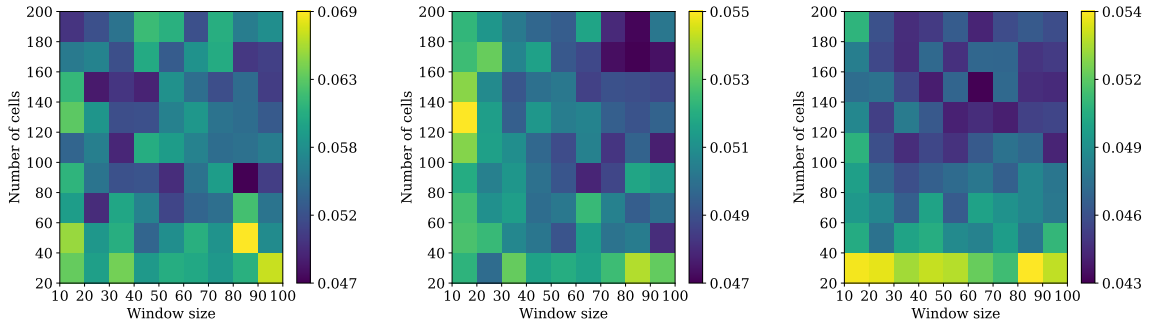
Among the 13 variables, there are 9 with the same amount of missing values, following a row wise missing data pattern. The rest of the variables have missing values much more often (column wise pattern), which correspond to the columns 1, 3, 6 and 8, so I removed them from this tests.

I took the last 1,242 rows of this data set as a test set and used all blocks of complete data to train the RNNs.

[5]Downloaded from `https://archive.ics.uci.edu/ml/datasets/Air+Quality`

4.2.1 Evaluation of RNNs Hyperparameters

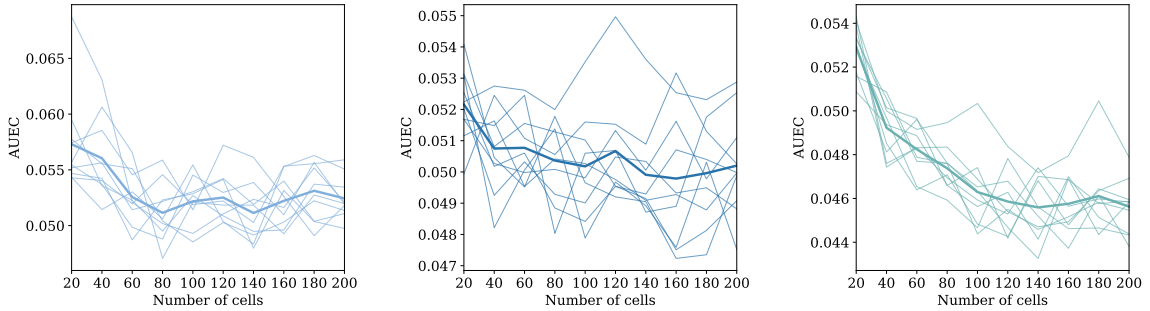
As with the previous data set, I plotted the heat map of the AUEC of all 300 hundred RNNs (see figure 4.8). Contrasting with the results of Section 4.1, this time the lower values of the AUEC are located in the top right corner of the heat map of RNNs using the GRU cell. The same is true, although in less extent, for RNNs using LSTM, but it is definitely not the case for RNNs using the basic cell. Specifically, increasing the number of cells and the window size showed a reduction in the AUEC.



(a) *Basic recurrent cell* **(b)** *LSTM cell* **(c)** *GRU cell*

Figure 4.8: *AUEC with respect to the number of nodes and length of time window.*

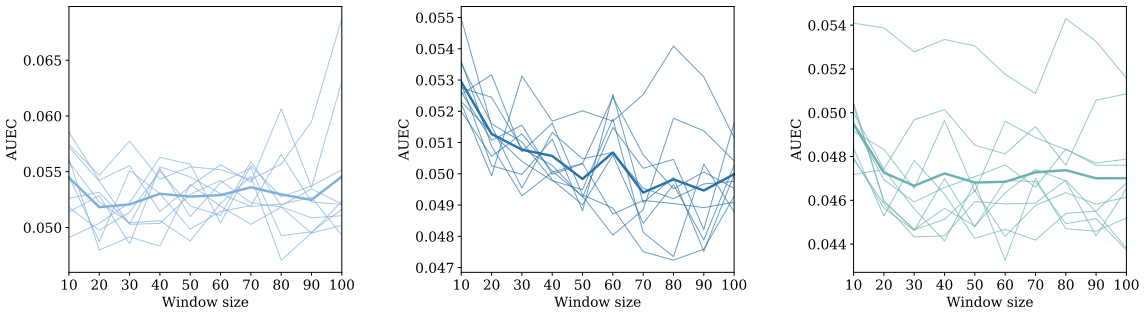
In fact, figure 4.9 reveals that, on average, there is a tendency towards lower values of AUEC when increasing the number of cells, regardless of which cell is used (although this tendency is marked more in the GRU cell type). The same is true when increasing the window size (see figure 4.10); however, the tendency is only strong for the LSTM type and null for the basic type.



(a) *Basic recurrent cell* **(b)** *LSTM cell* **(c)** *GRU cell*

Figure 4.9: *AUEC with respect to the number of nodes in the air quality data set.*

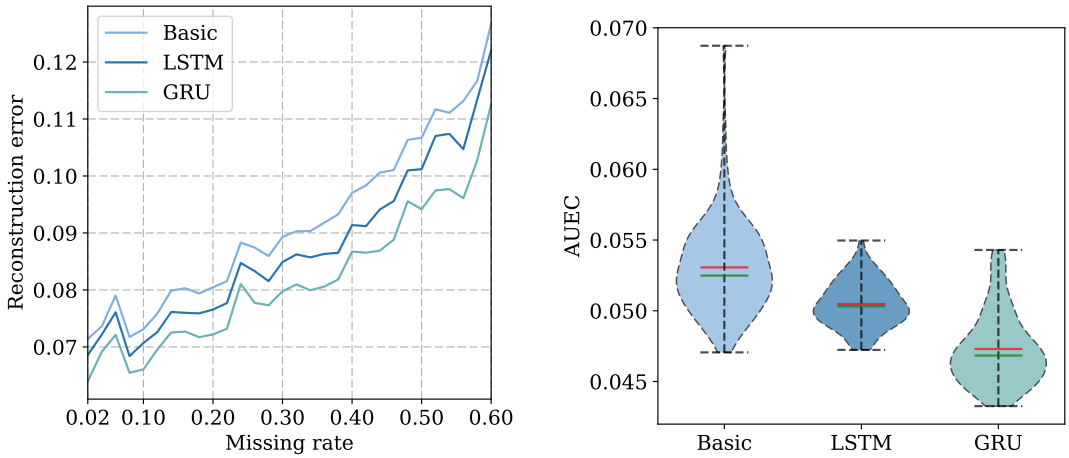
Once more, the basic cells were outperformed by the LSTM and GRU cells. This time, however, there is greater discrepancy between LSTM and GRU types (see figure 4.11). Again, I estimated the statistical significance of these results using ten thousand Monte Carlo simulations. Nevertheless, I



(a) *Basic recurrent cell* **(b)** *LSTM cell* **(c)** *GRU cell*

Figure 4.10: *AUEC with respect to the length of time window in the air quality data set.*

defined the test statistic to be the difference between the means rather than medians, because there were not outliers in the results obtained for this data set.



(a) **(b)**

Figure 4.11: *(a) Mean of the error against missing rate. (b) Violin plot of the AUEC for the three cell types, the red and green lines represent the mean and median values respectively.*

The results of the Monte Carlo simulations are shown in Table 4.5, which suggests that the observed difference between the types cannot be attributed to randomness, thus, the observed difference can safely be attributed to the type of cell.

4.2.2 Comparison with Other Imputation Methods

This time I randomly chose a RNN with GRU cells, 60 cells and 40 time steps of window size as a representative of the RNN approach to impute missing values.

Table 4.5: *Probability that the difference in the median value of pairs of cells' type is equal to or greater than the observed difference if the AUECs had come from the same cell type.*

	$P(DIF \geq DIF_{obs})$
Basic-GRU	0
Basic-LSTM	0
GRU-LSTM	0

4.2.2.1 Comparison in a Bernoulli pattern of missing data

Figure 4.12 shows that, in this data set, the method based on PCA outperformed all others, whereas the RNN using GRU cells was the second best performing method. Although Figure 4.12a does not show the line corresponding to using the mean for imputation, it is clear that it is being hidden due to its performance being similar to the method of using the median value for imputation (see Table 4.6).

The summary of the experiments computing the AUEC are presented in Table 4.6. Again, I tested for statistical significance of the results, specially between the results of the two best performing methods.

The results of the Monte Carlo simulations (see figure 4.12c) suggest that the observed difference between the RNN with GRU cells and the method based on PCA is statistically significant, since none of the ten thousand simulations showed the observed difference between the means across incomplete samples.

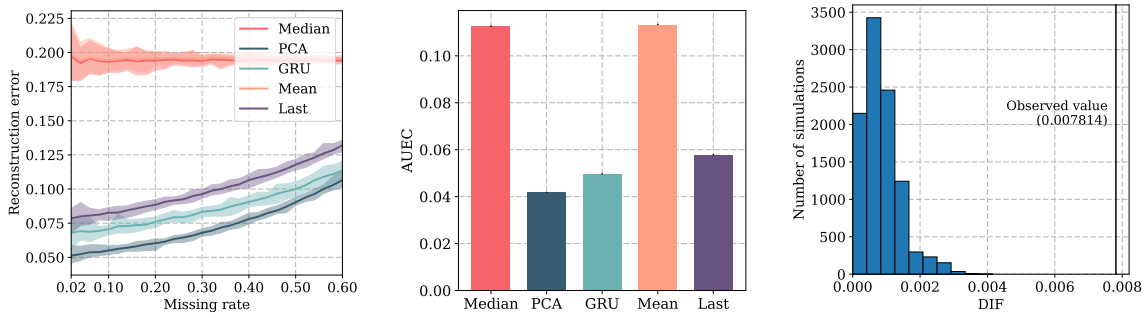


Figure 4.12: *(a) Reconstruction error against the missing rate. (b) AUEC . (c) Monte Carlo simulations testing the significance of the observed difference.*

Table 4.6: *Summary of the performance on the AUEC metric.*

			Extreme values	
	Mean	Standard deviation	Minimum	Maximum
Last	0.0579	2.29e−4	0.0575	0.0585
Mean	0.1133	2.77e−4	0.1130	0.1139
Median	0.1126	2.84e−4	0.1122	0.1132
PCA	0.0419	1.78e−4	0.0414	0.0423
GRU	0.0496	2.61e−4	0.0492	0.0502

4.2.2.2 Comparison in column wise missing data patterns

As in the experiment with the household data set, I chose $P_h = 0.7$ for two variables and $P_l = 0.05$ for the rest. Since this data set contains 9 variables, I performed experiments over all 36 (9 choose 2) possible combinations.

Figure 4.13 shows the main results of the 36 repetitions of column wise missing data patterns. A summary is shown in Table 4.7.

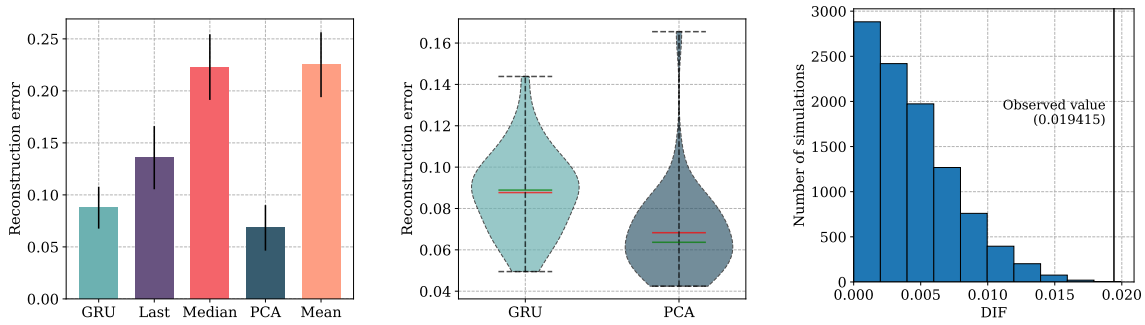
Table 4.7: *Summary of the experiments in the column wise pattern.*

			Extreme values	
	Mean	Standard deviation	Minimum	Maximum
Last	0.1358	0.0304	0.0762	0.1917
Mean	0.2252	0.0312	0.1705	0.3004
Median	0.2229	0.0316	0.1670	0.2989
PCA	0.0683	0.0219	0.0424	0.1655
GRU	0.0877	0.0201	0.0495	0.1438

This time, the imputation method based on RNNs with GRU cells showed the second best performance, whereas the method based on PCA performed the best (see figure 4.13). The Monte Carlo analysis reveal that the difference between these two methods was statistically significant with $P\text{-}value = 0.0002$.

4.2.2.3 Comparison in a row wise missing data pattern

Given that the available amount of test data for this data set (1,242 observations), in this experiment I defined that there could be from 100 to 200 continuous *observed* rows and from 30 to 60 continuous



(a) **(b)** **(c)**

Figure 4.13: *Results in a column wise pattern. (a) Mean reconstruction error. (b) Results of the two best performing methods. (c) Significance of the observed difference.*

unobserved semi-rows. Besides this settings, and the amount of variables (9) in this data set, there are no other differences between the process generating the missing data pattern of this experiment and the row wise missing data pattern experiment with the household power consumption data set.

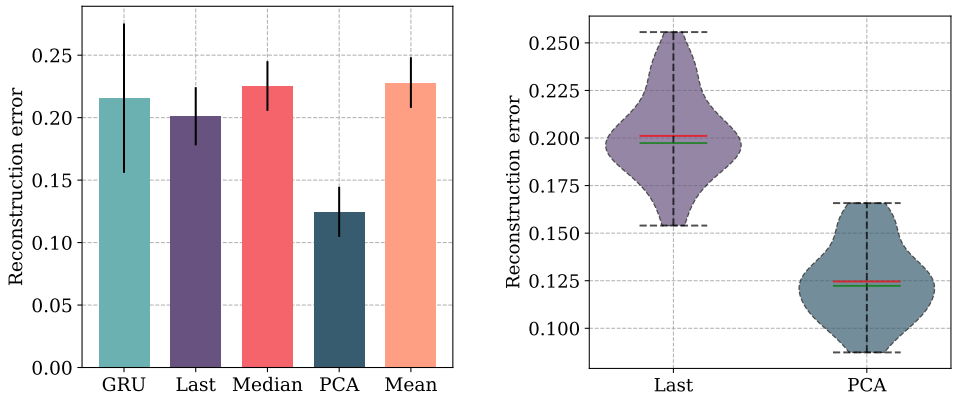
This experiment is summarised in Figure 4.8 and Table 4.8. As opposed to the rest of the experiments in this thesis, the method based on RNNsis not in the top 2. For this reason I did not performed the Monte Carlo analysis between the two best performing method as usual.

Table 4.8: *Summary of the reconstruction error in the row wise pattern of missing data.*

			Extreme values	
	Mean	Standard deviation	Minimum	Maximum
Last	0.2011	0.0232	0.1540	0.2557
Mean	0.2281	0.0203	0.1859	0.2718
Median	0.2253	0.0199	0.1815	0.2612
PCA	0.1246	0.0201	0.0873	0.1658
GRU	0.02156	0.0598	0.1432	0.3818

4.2.3 Summary

In the Air Quality data set there was no pattern in which RNNs were superior to all other methods. Instead, RNNs the second best method in the Bernoulli and column wise patterns, whereas they were third in the raw wise pattern. This result can be easily explained by the lack of training data available in the Air Quality data set, because it is well known in ANNs literature that the models need to be trained on a big enough data set. Moreover, since I did not vary the training procedure in the two



(a) **(b)**

Figure 4.14: *Results in a row wise pattern. (a) Mean reconstruction error. (b) Results of the two best performing methods.*

data sets, this result could also be explained by the choice of the strategy used during training of the RNNs.

4.3 Case Study: Impact of Imputation Methods on an IoT Application

In Chapter 1 I explained how the problem of missing values can harm the effectiveness of smart applications in the context of the IoT. In this section I show by a case study of a Fault Detection and Diagnosis (FDD) system for home refrigerators, how the imputation method (i.e. its reconstruction error) affects the performance of a smart application. This section is dedicated to fulfill specific objective (IV).

The exposition in this section is developed in the framework of a research project that is developed in alliance with a local home appliances manufacturer[6]. The industrial partner is interested in building a FDD system for a particular model of home refrigerators. Such a work is still under development and its focus is different from what is presented in this section; specifically, this section presents an example of how the missing data problem affects the performance of a smart application (i.e. the FDD system for home refrigerators), whereas the other project is focused on building a reliable and efficient FDD system for home refrigerators.

[6]The name is reserved by request of the company.

4.3.1 Online FDD of home refrigerators

The local home appliances manufacturer wants to build FDD system that can accurately report via the Internet the current status of certain model of home refrigerators. Creating a FDD system can be seen as a classification task in the perspective of supervised learning described in Chapter 1. Each faulty condition represents a label given to a set of data points that are measured for the duration of such condition. There is also another label representing the normal operation of the home refrigerator.

Diagnosing some faulty conditions may be time consuming and thus expensive (see table 4.9 for the approximated time spent by technicians to report the most common faults). Moreover, some faulty conditions require specialized tools that technicians cannot carry with them all the time. Therefore, an automatic FDD system for home refrigerators could substantially reduce maintenance costs.

Table 4.9: *Time for detection of most common faults in home refrigerators.*

	Time for detection (min)
Refrigerant leakage	30
Compressor fouling	25
Fan damage	20
Obstruction in pipes	30
Loose doors	30
Broken damper	10

If there were only two variables, and one only considered a faulty condition, together with the non faulty operation mode, this case study will be accurately described like the figures 1.6a and 1.6b in Chapter 1. This is just a representation; the reader will have to conceptually extrapolate what is presented there to the case at hand.

Currently, a FDD system for home refrigerators is under development by a team of several researchers, in which I am included. Such a work includes all relevant faults in home refrigerators and a great deal of experimentation. Furthermore, it is intended to be published in an indexed journal; consequently, I cannot publish here all the results. However, I can present here a reduced version of the work, in which I will consider only 3 of the faulty conditions shown in table 4.9: broken damper, loose door and refrigerant leakage.

For this task, 17 sensors were deployed in the refrigerator as shown in figures 4.15 and 4.16 (see Table 4.10 for the meaning of all names of sensors). Given these, data was collected during a day at one minute intervals. Before collecting data from every faulty condition, it was verified that the refrigerator could be taken back to a normal operation, so that the faulty conditions did not accumulate.

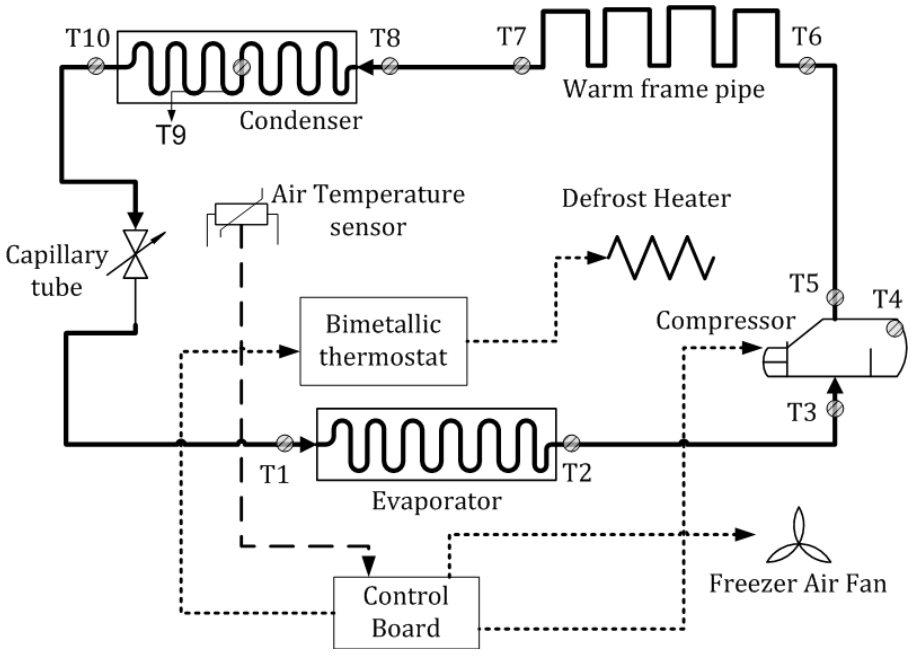


Figure 4.15: *Location of sensors in the system.*

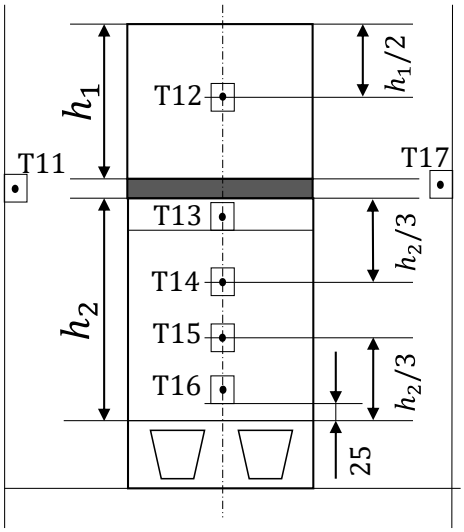


Figure 4.16: *Location of sensors in refrigerators compartments.* h_1 *and* h_2 *refer to the height of the freeze and conservation area respectively.*

As some sensors were placed inside the refrigerator, it had to be sealed (see figure 4.17), which implies that the potential FDD system would only be able to detect faults when the door is closed.

During the course of a day, data was collected from the refrigerator in a recreation of any of the faults considered. Afterwards, the refrigerator was fixed so that the faults could be considered as

Table 4.10: *Label and location of sensors.*

Sensor label	Location
T_1	Evaporator coil input
T_2	Evaporator coil output
T_3	Compressor suction input
T_4	Compressor frame
T_5	Compressor discharge output
T_6	Warmer frame pipe input
T_7	Warmer frame pipe output
T_8	Condenser coil input
T_9	Condenser medium section
T_{10}	Condenser output coil
T_{11}	Room temperature 1 (right side of the product)
T_{12}	Medium section of the freezer compartment
T_{13}	Quick chill compartment
T_{14}	Top section of fresh food compartment
T_{15}	Medium section of fresh food compartment
T_{16}	Bottom section of fresh food compartment
T_{17}	Room temperature 2 (left side of the product)

individual.

In the end, I had 1,660 observations for each of the conditions mentioned above, which I halved to create a training set using the first 830 observations of each condition and the rest was used to form the test set. I fed this data to three classifier, namely, Support Vector Machines, K-Nearest Neighbours and Logistic Regression, the results are presented in the following subsection.

4.3.2 Effect of reconstruction error on classification accuracy

In this section I show the performance of the three classifiers considered, dictated by the accuracy for the fault detection and diagnosis in the training data and the test data. Moreover, I show the performance on the reconstructed test data by each of the five methods that have been considered throughout this project (imputation with the mean, median, last value, the method based on PCA and

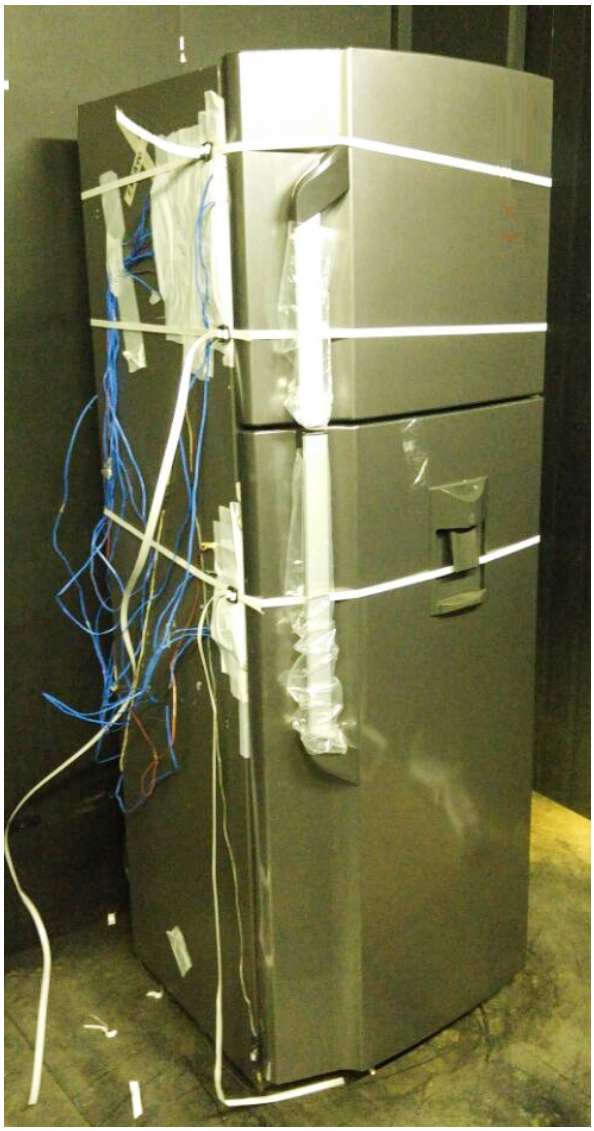


Figure 4.17: *A picture of the refrigerator used in the experiments.*

a method based on RNNs). In the case of RNNs I randomly chose a RNN with 120 GRU cells and 15 steps of window size.

Given that the purpose of this section is to show how an IoT application can be affected by the presence of missing data, it was not necessary to exhaustively evaluate the performance of imputation methods, that is, evaluate against all the missing data patterns. Rather, it is more interesting to see that the choice of imputation method (or the associated reconstruction error) affects the performance of the aforementioned application. Therefore, instead of computing the AUEC of every imputation method over several samples of incomplete data, I decided to test just under a simple condition for missing values, namely, that the probability of a value being missed for any variable in any observation

is 0.6 (or $k = 0.6$ seen as the Bernoulli process described in Section 3.1).

In that condition here is the reconstruction error I obtained by five imputation methods:

1. Impute values with the mean of every variable: 0.2110.
2. Impute values with the median of every variable: 0.1821.
3. Impute values with the last recorded value of every value: 0.0154.
4. Impute values with the method based on PCA from the R package missMDA: 0.1177.
5. Impute the values predicted by a RNN using GRU cells, 100 cells and 10 time steps of window size: 0.0287

Given the reconstruction error of the five methods, it is clear that the accuracy of the FDD system is affected by the error of the imputation method (see table 4.11).

Table 4.11: *Classifiers' performance on the training and test set, as well as the reconstructed data set by the five considered imputation methods.*

	Original data		Reconstructed data				
Method	Training	Test	Mean	Median	Last	PCA	GRU
K-Nearest Neighbours	100	86.6	53.8	57.8	86.6	84.6	87.0
Logistic Regression	100	99.7	59.7	53.1	99.2	78.6	95.5
Support Vector Machines	100	100	59.0	57.8	99.8	90.1	99.7

To have a clearer impression of the impact of the reconstruction error of different imputation methods (more precisely their reconstruction error) on the performance of applications of this type, I performed more tests like the above, using increasing values of missing rate in the generation of Bernoulli patterns, so as to produce increasing values of the reconstruction error (as it was observed in the previous chapter).

In figure 4.18 I show the results obtained in the two best performing classifiers for the case at hand for every one of the best three performing imputation methods in this data set. As a general rule, it can be seen that the greater the reconstruction error, the lower the accuracy, which justifies the search for better imputation methods.

At this point, it is pointless to try to define a mathematical relation between the reconstruction error and the accuracy of the classification methods used for this case study, because even if such a relation hold in this case study, there is no reason to believe that such a thing would generalize to other classification problems and imputation methods.

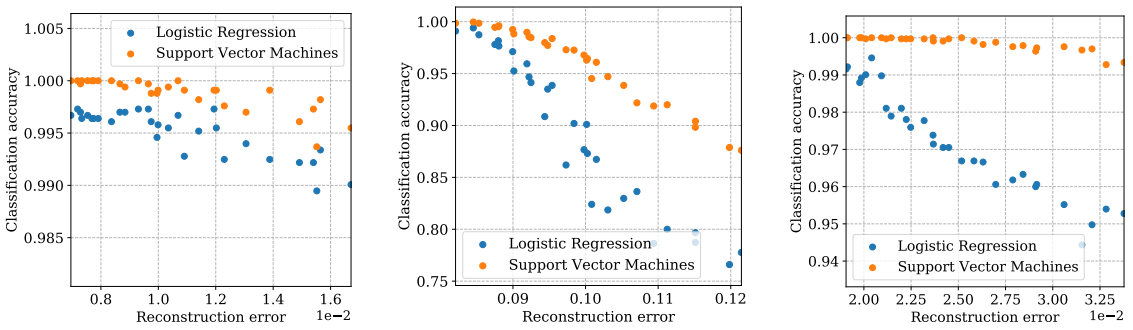


Figure 4.18: *Accuracy of two machine learning methods against the reconstruction error. (a) Imputing with last recorded value of every variable. (b) Imputation method based on PCA. (c) Imputation method based on RNNs with GRU cells.*

4.4 A note on the variability of the AUEC metric

In the end of Section 3.1 I commented, intuitively, on the computational cost of computing the AUEC for a number of repetitions. I also stated that the results I obtained suggested that this metric does not vary considerably among repetitions.

Although figures 4.5 and 4.12 show some evidence of the previous statement, I now present more dedicated evidence. For this I used a Bootstrap strategy (see Efron and Tibshirani, 1994): for every possible value in the number of samples (i.e. 1 to 30) I took 10 random samples with replacement. This served to calculate the standard deviation and the mean of the AUEC values as a function of the number of samples (i.e. 1 to 30).

It can be seen in Figure 4.19 that none of the methods showed big changes in the maximum standard deviation found among the bootstrap samples. Moreover, this measure of dispersion for the AUEC is relatively similar for the different methods when compared with the dissimilarity found in the mean AUEC (see figures 4.5 and 4.12).

Furthermore, Figure 4.20 shows how the difference between mean value of the AUEC found in the 30 repetitions differs from the mean value found in the bootstrap samples. Clearly, this difference is quite small when compared with the mean value found for the AUEC in any of the imputation methods.

This experiment clearly shows that several repetitions of the Bernoulli pattern may not make a significant difference on the computed AUEC. In other words, just one or few measures of the AUEC may be sufficient to stablish some degree of significance on the results. Nonetheless, this argument should be theoretically explored.

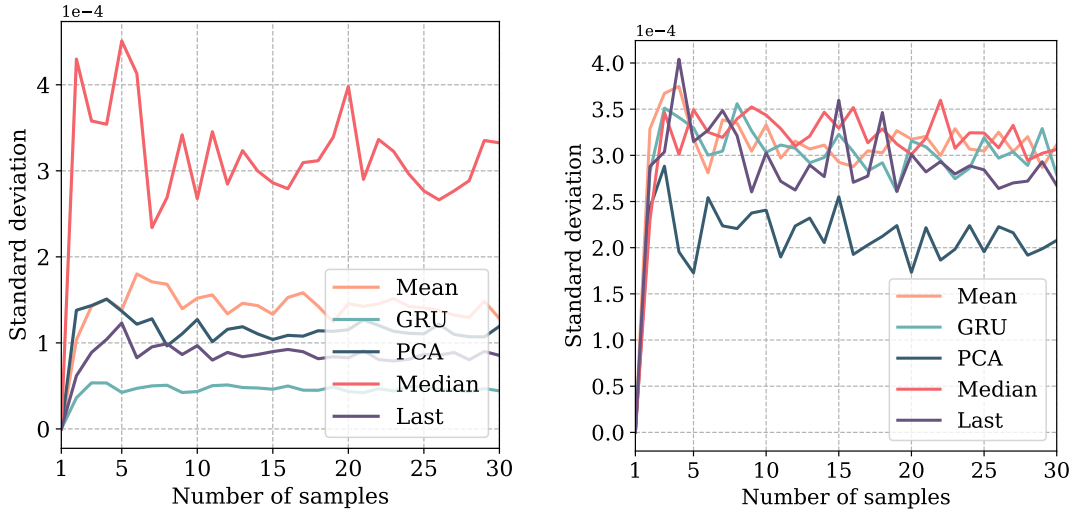


Figure 4.19: *Maximum standard deviation against the number of samples. (a) Household electric power consumption data set. (b) Air quality data set.*

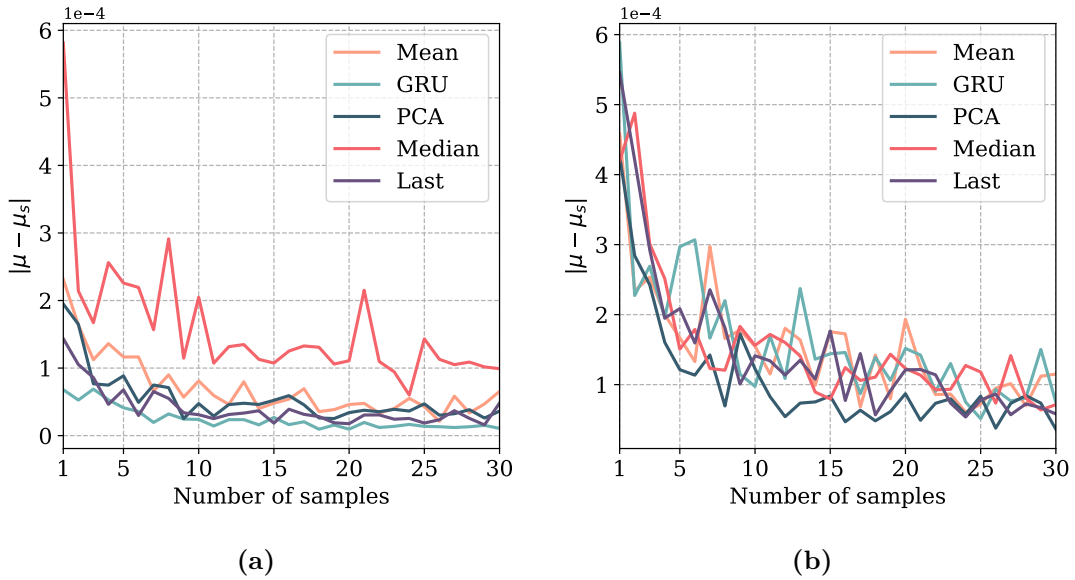


Figure 4.20: *Maximum error from the found mean of the AUEC against the number of samples. (a) Household electric power consumption data set. (b) Air quality data set.*

Chapter 5

Analysis and Conclusions

5.1 Analysis and Conclusions

The IoT opens up a new world for novel applications, which may combine different sources of data, gathered under different circumstances. Adversely, this variability in the sources of data creates new issues for the development of such applications. In particular, I dedicated this project to the treatment of the missing data problem.

When the aforementioned problem is not handled properly, it leaves to applications responding at lower quality, which can bee seen in Figure 4.18 on page 61; moreover, if it is not handled at all, it leaves to unresponsive applications. It is not possible to estimate a sensible detailed relation that explains how an application is affected by this problem, even for the particular demonstration of Section 4.3. Nonetheless, the results presented in Section 4.3 make it possible to assert that the lower the reconstruction error, the better the quality of the application, which in this case is measured by the accuracy of classifiers. For example, the performance of a FDD system for home refrigerators was affected by 42.5% using an imputation method of the median value, which had a reconstruction error of 0.1821; whereas the performance was only affected by 0.3% using an imputation method based on RNNs, which had a reconstruction error of 0.0287.

In the perspective of the variety and heterogeneity that is inherent in the context of the IoT, a proper solution to the missing data problem, and indeed for any other problem in this context, should arise from as very little assumptions as possible. Consequently, I proposed the use of RNNs as imputation method, which only assumes: i) a non-linear relation between the variables; and ii) a relation within the variables across a number of time steps.. This two assumptions are logical and have indeed been exploited, to some extent, by other authors (see Chapter 2). However, RNNs exploit them in their most general form, thus it is sensible to expect RNNs to work under a broader set of data sets, provided that they can be trained properly.

I showed in Chapter 2 that the missing data problem may occur following three patterns. Intuitively, the fact that RNNs do not rely solely on either the relation between the variables or the relation across time steps, implies that they would compensate and show good behaviour in any of those patterns. For example, in the column wise missing data pattern, the absence of a few variables can be predicted by its relation to other variables using RNNs; whereas a method that imputes every variable independently, has less information available for the imputation. This is why the method based on RNNs outperformed the methods based on the mean, median and last value for the Household Electric Power Consumption Data Set (see Section 4.1) and the Air Quality Data Set (see Section 4.2). Such result is also to be expected when comparing RNNs with methods that rely solely on time relations of each variable independently.

When comparing two or more imputation methods, it is important to whether define a missing data pattern of interest or to evaluate using the three patterns described Section 3.1. This is supported by the results I presented in sections 4.1 and 4.2, which show that, even for a particular data set, an imputation method that outperforms others in a particular pattern, does not necessarily perform as well in another pattern. For example, in the Household Electric Power Consumption Data Set, RNNs outperformed the method based on PCA in the AUEC metric (Bernoulli pattern) as well as in the column wise pattern; however, the method based on PCA outperformed RNNs in the row wise pattern. Therefore, if there is not a particular pattern of interest, a criterion for the advantage of one method over the other requires a consideration of all patterns.

Comparing several imputation methods in such a way is somewhat cumbersome task, because they basically multiply the computational complexity of each of the imputation methods being tested. However, it is the only reliable way to assert the superiority of any method over the other, at least in terms of the reconstruction error. Otherwise, the claim that a particular imputation method is superior to other (in terms of the reconstruction error) is biased to the particular choice of pattern on which the imputation methods are being tested.

I found that, for the five imputation methods I tested, the AUEC metric does not show much variation across the number of samples (see Figure 4.19 on page 62); in particular, the difference between the mean value obtained over 30 repetitions and the mean value obtained with smaller number of samples was found to be in the order of 1e−4 and decreased fast to zero as the number of samples increases. Thus, If one is using the average value of this metric to test two methods, one would get similar results regardless of the number of samples (see Figure 4.20 on page 62). Accordingly, repeated experiments may not be required in order to test the difference of two or more imputation methods, unless that for those imputation methods, the plots of the reconstruction error against the missing rate show crossing or close lines. This conclusion may hold true for any imputation method, because the methods I tested here had no relation in their formulation; however, it is evident that this should

be proved mathematically.

The results of RNNs across all experiments presented in sections 4.1 and 4.2 show that they are a promising alternative to missing value imputation in the context of the IoT. Specifically, RNNs were in the top-2 imputation in 83% of the experiments presented in those sections. These results did not show that a particular set of hyperparameters is more appropriated than any other in the two data sets used, which is to be expected from the literature of ANNs in general. However, regarding the cell type of RNNs, it seems that the basic type is less capable of learning useful temporal and between-variable patterns than the LSTM and GRU types (at least for the architecture and training procedures I defined in chapters 3 and 4). All this is perfectly expected by the deep learning literature.

The results of RNNs as imputation method obtained for different data sets, revealed a great limitation of this approach: they may have a clear superiority provided there is a big amount of training data, as was the case with the Household Electric Power Consumption Data Set, with a training set of 47,625 observations; but may not show outstanding results in cases where there is not a big enough training data set, like in the case of the Air Quality Data Set, which had less than 8,216 observations for training.

The last situation may be solved by data augmentation and regularization techniques, but the focus of this project was to show the possibility of RNNs being used as imputation methods, proper training of RNNs is a different subject.

The contributions of this work can be summarized as follows:

1. I showed that RNNs can be a great alternative as imputation methods, provided that they can be trained properly.
2. I presented a detailed description of procedures to simulate incomplete data based on a complete data set, together with ways to estimate the performance of imputation methods, which can be used to unambiguously specify the experiments that other researchers may carry and their corresponding results.

5.2 Further Work

This work leads to the following possibilities for future work:

1. To develop an algorithm based on RNNs that does not require a huge amount of complete training data (or no complete training data at all). Instead, such algorithm should work with whatever data is available, as some methods shown in Chapter 2 do.
2. Compare RNNs with at least some of the latest published imputation methods.
3. Evaluate the impact of the sampling rate of the variables in different imputation methods.
4. It is difficult to find a proper data set to compute the reconstruction error using the simulation procedures I presented in Section 3.1, and even if there were plenty of such data sets, it would

be more convenient to have metrics that leverage confident estimation of the performance of imputation method in operation time, so that at any point in the operation of the application, one can estimate how the imputation method is performing.

References

610.7-1995, I. S. (1995). *IEEE Standard Glossary of Computer Networking Terminology*.

Aeberhard, M., Rauch, S., Bahram, M., Tanzmeister, G., Thomas, J., Pilat, Y., Homm, F., Huber, W., and Kaempchen, N. (2015). Experience, Results and Lessons Learned from Automated Driving on Germany's Highways. *IEEE Intelligent Transportation Systems Magazine*, 7(1):42–57.

Alaa, M., Zaidan, A. A., Zaidan, B. B., Talal, M., and Kiah, M. L. (2017). A review of smart home applications based on Internet of Things. *Journal of Network and Computer Applications*, 97(July):48–65.

Alaba, F. A., Othman, M., Hashem, I. A. T., and Alotaibi, F. (2017). Internet of Things security: A survey. *Journal of Network and Computer Applications*, 88(April):10–28.

Alam, F., Mehmood, R., Katib, I., and Albeshri, A. (2016). Analysis of Eight Data Mining Algorithms for Smarter Internet of Things (IoT). In *InternationalWorkshop on Data Mining in IoT Systems (DaMIS 2016)*, volume 58, pages 437–442. Procedia Computer Science.

Arsénio, A., Serra, H., Francisco, R., and Nabais, F. (2014). *Internet of Intelligent Things: Bringing Artificial Intelligence into Things and Communication Networks*, volume 495.

Barclay, M. E., Lyratzopoulos, G., Greenberg, D. C., and Abel, G. A. (2018). Missing data and chance variation in public reporting of cancer stage at diagnosis: Cross-sectional analysis of population-based data in England. *Cancer Epidemiology*, 52(November 2017):28–42.

Bell, M. L., Fiero, M., Horton, N. J., and Hsu, C.-H. (2014). Handling missing data in RCTs; a review of the top medical journals. *BMC Medical Research Methodology*, 14(118):1–8.

Bertsekas, D. P. and Tsitsiklis, J. N. (2000). Introduction to Probability.

Bijarbooneh, F. H., Du, W., Ngai, E. C. H., Fu, X., and Liu, J. (2016). Cloud-Assisted Data Fusion and Sensor Selection for Internet of Things. *IEEE Internet of Things Journal*, 3(3):257–268.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, volume 4.

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time Series Analysis: Forecasting and Control*.

CERP-IoT (2010). *Vision and challenges for realizing the internet of things*. Publications Office of the European Union, Luxembourg.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.

Chui, C. K. and Chen, G. (1998). *Kalman Filtering with Real-Time Applications*. Springer.

E. Porter, M. and E. Heppelmann, J. (2017). How smart, connected products are transforming competition.

Earley, S. (2015). Analytics, machine learning, and the internet of things. *IT Professional*, 17(1):10–13.

Eaton, D. A., Spriggs, E. L., Park, B., and Donoghue, M. J. (2017). Misconceptions on missing data in RAD-seq phylogenetics with a deep-scale example from flowering plants. *Systematic Biology*, 66(3):399–412.

Efron, B. and Tibshirani, R. J. (1994). *An Introduction to the Bootstrap*. Chapman and Hall/CRC.

Ellison, E. R. and Langhout, R. D. (2017). Sensitive Topics, Missing Data, and Refusal in Social Network Studies: An Ethical Examination. *American Journal of Community Psychology*, 60(3–4):327–335.

F., C., P., D., J., W., D., Z., A.V., V., and X., R. (2015). Data mining for the internet of things: Literature review and challenges. *International Journal of Distributed Sensor Networks*, 2015(i).

Friess, P. (2013). *The Internet of things.*, pages 1–6. River Publishers.

Gao, S., Tang, Y., and Qu, X. (2012). LSSVM Based Missing Data Imputation in Nuclear Power Plant ' s. *International Conference on Advanced Computational Intelligence*.

Gil, D., Ferrández, A., Mora-Mora, H., and Peral, J. (2016). Internet of Things: A Review of Surveys Based on Context Aware Intelligent Services. *Sensors*, 16(7):1069.

Goodfellow, I., Bengio, Y., and Courville, A. (2015). Deep Learning.

Gotelli, N. J. and Ellison, A. M. (2004). *A Primer of Ecological Statistics*.

Guo, D., Liu, Z., Qu, X., Huang, L., Yao, Y., and Sun, M. T. (2012). Sparsity-based online missing data recovery using overcomplete dictionary. *IEEE Sensors Journal*, 12(7):2485–2495.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. springer.

Henning, K., Wolfgang, W., and Johannes, H. (2013). Recommendations for implementing the strategic initiative INDUSTRIE 4.0. *Final report of the Industrie 4.0 WG*, (April):82.

Henning, Kagermann. Wolfgang, Wahlster. Johannes, H. (2013). Recommendations for implementing the strategic initiative INDUSTRIE 4.0. *Final report of the Industrie 4.0 WG*, (April):82.

Hochreiter, S. and Urgen Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Izenman, A. J. (2008). *Modern Multivariate Statistical Techniques*. Springer.

Kaku, M. (2011). *Physics of the Future: How Science Will Shape Human Destiny and Our Daily Lives by the Year 2100*. Doubleday.

KaltenbachHans-Michael (2012). *SpringerBriefs in Statistics*.

Kirchgässner, G., Wolters, J., and Hassler, U. (2013). *Introduction to Modern Time Series Analysis*. Springer-Verlag.

Kong, L., Xia, M., Liu, X.-Y., Chen, G., Gu, Y., Wu, M.-Y., and Liu, X. (2013). Data Loss and Reconstruction in Wireless Sensor Networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1.

Kong, L., Xia, M., Liu, X.-Y., Chen, G., Gu, Y., Wu, M.-Y., and Liu, X. (2014). Data loss and reconstruction in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(11):2818–2828.

Larsen, R. J., Marx, M. L., and Town, C. (2012). *An Introduction To Mathematical Statistics*.

Li, C. Y., Su, W. L., McKenzie, T. G., Hsu, F. C., Lin, S. D., Hsu, J. Y. J., and Gibbons, P. B. (2015). Recommending missing sensor values. *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, pages 381–390.

Little, R. J. a. and Rubin, D. B. (2002). *Statistical Analysis with Missing Data*.

Liu, F., You, Z., Shan, W., and Liu, J. (2012). A Grey System Based Missing Sensor Data Estimation Algorithm. In *International Conference on Computer Science and Network Technology*, pages 482–486, Wuhan. IEEE.

Masconi, K. L., Matsha, T. E., Echouffo-Tcheugui, J. B., Erasmus, R. T., and Kengne, A. P. (2015). Reporting and handling of missing data in predictive research for prevalent undiagnosed type 2 diabetes mellitus: a systematic review. *EPMA Journal*, 6(1):7.

Netten, A. P., Dekker, F. W., Rieffe, C., Soede, W., Briaire, J. J., and Frijns, J. H. M. (2016). Missing Data in the Field of Otorhinolaryngology and Head & Neck Surgery : Need for Improvement. *Ear and Hearing*, pages 1–6.

Ni, J., Li, L., Qiao, F., and Wu, Q. (2014). A GS-MPSO-WKNN method for missing data imputation in wireless sensor networks monitoring manufacturing conditions. *Transactions of the Institute of Measurement and Control*, 36(8):1083–1092.

Niu, K., Zhao, F., and Qiao, X. (2013). A Missing Data Imputation Algorithm in Wireless Sensor Network Based on Minimized Similarity Distortion. 2:235–238.

Nower, N., Tan, Y., and Lim, A. O. (2013). Efficient Spatial Data Recovery Scheme for Cyber-physical System. pages 72–77.

Nower, N., Tan, Y., and Lim, A. O. (2014). Traffic Pattern Based Data Recovery Scheme for Cyber-Physical. *IEICE TRANS. FUNDAMENTALS*, E97-A(9).

Nower, N., Tan, Y., and Lim, Y. (2015). Incomplete feedback data recovery scheme with Kalman filter for real-time cyber-physical systems. *International Conference on Ubiquitous and Future Networks, ICUFN*, 2015-Augus:845–850.

P. Murphy, K. (2012). *Machine Learning: A Probabilistic Perspective*.

Pan, L., Gao, H., Gao, H., and Liu, Y. (2014). A Spatial Correlation Based Adaptive Missing Data Estimation Algorithm in Wireless Sensor Networks. *International Journal of Wireless Information Networks*, 21(4):280–289.

Pan, L., Gao, H., Li, J., Gao, H., and Guo, X. (2013). CIAM: An adaptive 2-in-1 missing data estimation algorithm in wireless sensor networks. *Networks (ICON), 2013 19th IEEE International Conference on*, pages 1–6.

Roser, M. and Ritchie, H. (2018). Technological progress. Retrieved on January 19, 2018 from `https://ourworldindata.org/technological-progress`.

Shuang-Hua, Y. (2014). *Wireless sensor networks: Principles, Design and Applications*. Springer.

Simonite, T. (2016). Moore's law is dead. now what? Rrieved on January 19, 2018 form `https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/`.

Singh, S. P., Kumar, A., Darbari, H., Singh, L., Rastogi, A., and Jain, S. (2017). Machine translation using deep learning: An overview. *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 162–167.

Srinivasan, K., Foe-Parker, C., Goebel, N., Herzl, R., Mehl, M. R., Gilligan, B., Heerwagen, J., Kampschroer, K., Canada, K., Currim, F., Ram, S., Lindberg, C., Sternberg, E., Skeath, P., Najafi, B., Razjouyan, J., and Lee, H.-K. (2016). Feature Importance and Predictive Modeling for Multi-source Healthcare Data with Missing Values. In *6th International Conference on Digital Health Conference - DH '16*, pages 47–54.

Stolpe, M. (2016). The Internet of Things: Opportunities and challenges. *SIGKDD Explorations*, 18(1).

Sujbert, L., Member, S., and Orosz, G. (2016). FFT-Based Spectrum Analysis in the Case of Data Loss. 65(5):968–976.

Tsagkatakis, G., Beferull-Lozano, B., and Tsakalides, P. (2016). Singular spectrum-based matrix completion for time series recovery and prediction. 2016(1):1–14.

Tsai, C. W., Lai, C. F., Chiang, M. C., and Yang, L. T. (2014). Data mining for internet of things: A survey. *IEEE Communications Surveys and Tutorials*, 16(1):77–97.

Vapnik, V. N. (1998). *Statistical Learning Theory*.

Vermesan, O., Friess, P., Guillemin, P., Sundmaeker, H., Eisenhauer, M., Moessner, K., Le Gall, F., and Cousin, P. (2013). *The Internet of things.* River Publishers.

Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, 30(1):79–82.

Wu, Q., Ding, G., Xu, Y., Feng, S., Du, Z., Wang, J., and Long, K. (2014). Cognitive Internet of Things: A New Paradigm Beyond Connection. *IEEE Internet of Things Journal*, 1(2):129–143.

Yan, N., Zhou, M.-z., and Tong, L. (2013). An Estimation Algorithm For Missing Data In Wireless Sensor Networks. *International Journal On Smart Sensing And Intelligent Systems (S2is)*, 6(3):1032–1053.

Yan, X., Xiong, W., Hu, L., Wang, F., and Zhao, K. (2014). Missing Value Imputation Based on Gaussian Mixture Model for the Internet of Things. *Mathematical Problems in Engineering*, 2015.

Zhang, H., Liu, J., Pang, A. C., and Li, R. (2016). A data reconstruction model addressing loss and faults in medical body sensor networks. *2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings*, pages 0–5.

Zhang, H. and Yang, L. (2014). An Improved Algorithm for Missing Data in Wireless Sensor Networks. *IET The Institution of Engineering and Technology*, pages 346–350.

Zhao, L., Chen, Z., Yang, Z., Hu, Y., and Obaidat, M. S. (2016). Local Similarity Imputation Based on Fast Clustering for Incomplete Data in Cyber-Physical Systems. *IEEE Systems Journal*, pages 1–11.

Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., Keller, C. G., Kaus, E., Herrtwich, R. G., Rabe, C., Pfeiffer, D., Lindner, F., Stein, F., Erbs, F., Enzweiler, M., Knoppel, C., Hipp, J., Haueis, M., Trepte, M., Brenk, C., Tamke, A., Ghanaat, M., Braun, M., Joos, A., Fritz, H., Mock, H., Hein, M., and Zeeb, E. (2014). Making bertha drive-an autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20.

Appendix A

Python Implementations Developed During this Project

Listing A.1: *Base class for imputation with a single constant value (different for every variable).*

```
import numpy as np
class ConstantValueImputer:
    """Provides an imputer that imputes the same values per column."""

    def __init__(self):
        """ Base class of imputers of constant values

        The __init__ method of any child class should include store constants
        in self._values. It should also define self._compute_values and the
        self._compute_kwargs in case self._values is computed based on some
        data.

        Returs:
            A ConstantValueImputer

        Raises:
            NotImplementedError: If values are not of proper type

        """
        raise NotImplementedError
```

```
    def fit(self, data):
        """ A wraper of numpy.nanmean function

        Calculates an stores the mean along the first dimension of data.

        Args:
            data (numpy.ndarray): Data to from which to learn the
                appropiate parameters to make imputations. It should
                normally be with shape (n_observations, n_variables).

        """

        self._values = self._compute_values(data, **self._compute_kwargs)

    def impute(self, data):
        return (np.nan_to_num(data)
                +self._values*np.isnan(data).astype(float))

    def fit_impute(self, data):
        self._values = self._compute_values(data, **self._compute_kwargs)
        return (np.nan_to_num(data)
                +self._values*np.isnan(data).astype(float))

    def get_params(self):
        pass

    def __repr__(self):
        return type(self).__name__
```

Listing A.2: *Class for imputation with the mean value of every variable.*

```
import numpy as np
class MeanImputer(ConstantValueImputer):
    """Provides an imputer that imputes the mean of every variable."""

    def __init__(self, means=None):
        """ Creates a mean imputer

        Args:
            means (numpy.ndarray): An array of shape (1, n_variables) with
            pre computed mean values to impute future data sets using the
            impute method.

        Returs:
            A mean imputer

        Raises:
            TypeError: If means are not of proper type
            ValueError: If means are not of proper shape

        """
        if means is not None:
            if type(means) is not np.ndarray:
                raise TypeError("means␣are␣not␣of␣apropiate␣type")

            if means.shape != (1, 2):
                raise ValueError("means␣are␣not␣of␣apropiate␣shape")

        self._values = means
        self._compute_kwargs = {'axis':0, 'keepdims':True}
        self._compute_values = np.nanmean
```

Listing A.3: *Class for imputation with the mean value of every variable.*

```
import numpy as np
class MedianImputer(ConstantValueImputer):
    """Provides an imputer that imputes the median of every variable."""

    def __init__(self, medians=None):
        """ Creates a median imputer

        Args:
            medians (numpy.ndarray): An array of shape (1, n_variables) with
            pre computed median values to impute future data sets using the
            impute method.

        Returs:
            A median imputer

        Raises:
            TypeError: If medians are not of proper type
            ValueError: If medians are not of proper shape

        """
        if medians is not None:
            if isinstance(medians, np.ndarray):
                raise TypeError("medians are not of apropiate type")

            if medians.shape != (1, 2):
                raise ValueError("medians are not of apropiate shape")

        self._values = medians
        self._compute_kwargs = {'axis':0, 'keepdims':True}
        self._compute_values = np.nanmedian
```

Listing A.4: *Class for imputation with the last recorded value of every value.*

```
import numpy as np
class LastValueImputer:
    """Imputes the last recorded value for every variable"""

    def fit(self, data):
        """ Does nothing.

        This method is kept just for consistency with the rest of the module

        """
        pass

    def impute(self, data):
        """ Impute the last recorded value for every variable

        Args:
            data (numpy.ndarray): A matrix with missing values that should be
            fill using this class.

        Returns:
            numpy.ndarray: data with missing values replaced

        Raises:
            ValueError: If the first row has missing values

        """
        data_ = np.copy(data)
        if np.sum(np.isnan(data_[0, :])):
            raise ValueError("Not enough steps of complete data")

        rows, cols =  np.where(np.isnan(data_))
        for i,j in zip(rows, cols):
            data_[i,j] = data_[i-1,j]
        return data_

    def fit_impute(self, data):
        """ A wraper of self.impute
```

```
        This method is kept just for consistency with the rest of the
        module

        """
        return self.impute(data)

    def __repr__(self):
        return type(self).__name__
```

Listing A.5: *Base class for imputation with the proposed architecture of RNNs.*

```
import numpy as np
import tensorflow as tf
class BaseRnnImputer:
    """
    This imputer class will impute values computed by a recurrent neural
    network.

    """

    def __init__(self, n_vars, n_steps,
                 n_neurons=None, lr=0.001,
                 meta_graph=None,
                 config=None):
        if meta_graph is None:
            raise NotImplementedError
        else:
            self._n_vars = n_vars
            self._n_steps = n_steps

            with tf.Session(config=config) as sess:
                self._saver = tf.train.import_meta_graph(meta_graph)
                self._saver.restore(sess, meta_graph[:-5])
                self._graph = tf.get_default_graph()
                self._model_params = _get_model_params()
                _restore_model_params(self._model_params)
                # The following just verifies that tensors can be restored
                outputs = self._graph.get_tensor_by_name("outputs:0")
                X = self._graph.get_tensor_by_name("X:0")
                y = self._graph.get_tensor_by_name("y:0")


    def _build_graph(self, n_vars, n_steps, n_neurons, lr, meta_graph=None):
        """
        Defines the graph operations to be used in the rnn model

        Args:
            graph: A new tensorflow graph object
```

```
    n_vars (int): number of variables in the data in which the
        imputer will be used
    n_steps (int): number of steps
    n_neurons (list):

returns: a defined graph object
"""

graph = tf.Graph()
with graph.as_default():

    X = tf.placeholder(tf.float32,
                       [None, n_steps, n_vars],
                       name='X')
    y = tf.placeholder(tf.float32,
                       [None, n_steps, n_vars],
                       name='y')


    layers = [self._factory(num_units=ne,**self._factory_kwargs)
              for ne in n_neurons]


    multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)

    rnn_outputs, states = tf.nn.dynamic_rnn(
            multi_layer_cell, X, dtype=tf.float32)

    stacked_rnn_outputs = tf.reshape(
                        rnn_outputs, [-1, n_neurons[-1]])
    stacked_outputs = tf.layers.dense(
                        stacked_rnn_outputs, n_vars)

    outputs = tf.reshape(stacked_outputs,
                         [-1, n_steps, n_vars],
                         name='outputs')

    loss = tf.reduce_mean(tf.square(outputs-y),
                          name='loss')
```

```
            optimizer = tf.train.AdamOptimizer(learning_rate=lr)

            #can I change it to local variables?
            self._training_op = optimizer.minimize(loss)
            self._init = tf.global_variables_initializer()
            self._saver = tf.train.Saver()
        return graph

    def fit(self, data, data_val=None,
            n_iterations=2000, batch_size=100,
            file_name=None, config=None,
            verbose=False):
        """
        Calculates an stores all parameters required to impute values
        whenever there is a missing value in data set of the same
        origin as data.

        Args:
            data (numpy.ndarray): Data to from which to learn the
                appropiate parameters to make imputations. It should
                normally be with shape (n_observations, n_variables).
            file_name (str): A file name to save the model.

        """
        if isinstance(data, np.ndarray):
            data = [data]
        if isinstance(data_val, np.ndarray):
            data_val = [data_val]

        best_val_loss = np.inf
        checks_since_last_progress = 0
        max_checks_without_progress = 20

        with tf.Session(graph=self._graph, config=config) as sess:

            X = self._graph.get_tensor_by_name('X:0')
            y =  self._graph.get_tensor_by_name('y:0')
            loss = self._graph.get_tensor_by_name('loss:0')
```

```
sess.run(self._init)
for iteration in range(n_iterations):
    d_train = random.choice(data)
    X_batch, y_batch = _next_batch(
                    d_train,
                    self._n_steps,
                    min(batch_size, d_train.shape[0]))
    sess.run(self._training_op,
             feed_dict={X: X_batch, y: y_batch})

    if iteration % int(n_iterations*0.1) == 0:
        mse_train_batch = loss.eval(
                    feed_dict={X: X_batch, y: y_batch})
        if data_val is not None:
            mse=0
            for d_val in data_val:
                X_batch, y_batch = _next_batch(
                        d_val,
                        self._n_steps,
                        min(batch_size, d_val.shape[0]))

                mse += loss.eval(
                    feed_dict={X: X_batch, y: y_batch})

            mse_val_avg = np.mean(mse)
            if mse_val_avg < best_val_loss:
                best_val_loss = mse_val_avg
                checks_since_last_progress = 0
                self._model_params = _get_model_params()
            else:
                checks_since_last_progress += 1

        if verbose:
            print(iteration)
            print('MSE','(train set)', mse_train_batch)
            print('MSE','(validation set)', mse_val_avg)
            print('Best validation MSE', best_val_loss)
```

```
            if checks_since_last_progress > max_checks_without_progress:
                print("Early stopping!")
                break

        _restore_model_params(self._model_params)
        if file_name is not None:
            self._saver.save(sess, file_name)


def impute(self, data, config=None,verbose=False):
    """
    Uses the parameters obtained by the fit method in order to
    impute values whenever there is a missing value in data.

    Args:
        data (numpy.ndarray): Data to from which to learn the
            appropiate parameters to make imputations.

    Returns:
        (numpy.ndarray): 2D array with imputed data where 'nan' was
            found.

    """

    # verify that data is complete for n_steps
    data_ = np.copy(data)
    if np.sum(np.isnan(data_[:self._n_steps,:])):
        raise ValueError("Not enough steps of complete data")

    indices = (
        np.arange(0,data.shape[0]-self._n_steps, 1).reshape(-1,1)
        +np.arange(self._n_steps+1)
        )

    with tf.Session(graph=self._graph, config=config) as sess:

        X = self._graph.get_tensor_by_name('X:0')
        outputs = self._graph.get_tensor_by_name('outputs:0')
        _restore_model_params(self._model_params)
```

```
            for idx in indices:
                if idx[-1] % int(data_.shape[0]*0.1) == 0 and verbose:
                    print("Imputing at row {}".format(idx[-1]))
                X_ = data_[idx[:-1],:].reshape(-1, self._n_steps, self._n_vars)
                y_pred = sess.run(outputs,
                                  feed_dict={X: X_})[:,-1,:][0,:]
                isnan = np.isnan(data_[idx[-1],:])
                np.place(data_[idx[-1],:], isnan, y_pred[isnan])

        return data_

    def fit_impute(self,data, n_iterations=2000, batch_size=100,
                   file_name=None, verbose=False):

        raise NotImplementedError


    def __repr__(self):
        return type(self).__name__
```

Listing A.6: *Class for imputation based on RNNs with basic cells.*

```
import tensorflow as tf
class BasicRNNImputer(BaseRnnImputer):
    """
    This imputer class will impute values computed by a recurrent neural
    network.

    """

    def __init__(self, n_vars, n_steps, n_neurons=(100,), lr=0.001):
        self._n_vars = n_vars
        self._n_steps = n_steps
        self._factory = tf.contrib.rnn.BasicRNNCell
        self._factory_kwargs = {'activation':tf.nn.relu}
        self._graph = self._build_graph(n_vars, n_steps, n_neurons, lr)
```

Listing A.7: *Class for imputation based on RNNs with LSTM cells.*

```
import tensorflow as tf
class BasicLSTMImputer(BaseRnnImputer):
    """
    This imputer class will impute values computed by a LSTM network.

    """
    def __init__(self, n_vars, n_steps, n_neurons=(100,), lr=0.001):
        self._n_vars = n_vars
        self._n_steps = n_steps
        self._factory = tf.contrib.rnn.BasicLSTMCell
        self._factory_kwargs = {'activation':tf.sigmoid}
        self._graph = self._build_graph(n_vars, n_steps, n_neurons, lr)
```

Listing A.8: *Class for imputation based on RNNs with GRU cells.*

```
import tensorflow as tf
class BasicGRUImputer(BaseRnnImputer):
    """
    This imputer class will impute values computed by a LSTM network.

    """
    def __init__(self, n_vars, n_steps, n_neurons=(100,), lr=0.001):
        self._n_vars = n_vars
        self._n_steps = n_steps
        self._factory = tf.contrib.rnn.GRUCell
        self._factory_kwargs = {'activation':tf.sigmoid}
        self._graph = self._build_graph(n_vars, n_steps, n_neurons, lr)
```

Listing A.9: *Function to generate Bernoulli patterns of missing data. The column wise missing data pattern can be produced with a proper choice of probabilities for each variable.*

```
def bernoulli(data, p=0.1):
    copy = np.copy(data)
    miss = np.random.binomial(n=1, p=p, size=copy.shape)
    np.place(copy, miss, np.nan)
    return copy
```

Listing A.10: *Function to generate row-wise patterns of missing data.*

```
def row_wise(data, rng_obs=(1,3), rng_miss=(1,3), p=0.1, omit_cols=[]):
    copy = np.copy(data)
    n_obs = copy.shape[0]
    rm_data = bernoulli(data=copy, p=p)
    ## correct to include rowwise
    rows_miss = [0]*np.random.choice(rng_obs)
    while len(rows_miss) <= n_obs:
        rows_miss.extend([1]*np.random.choice(rng_miss))
        rows_miss.extend([0]*np.random.choice(rng_obs))
    miss = np.zeros(copy.shape) + np.array(rows_miss[:n_obs]).reshape(-1,1)
    np.place(rm_data, miss, np.nan)
    rm_data[:, omit_cols] = copy[:, omit_cols]
    return rm_data
```