Alberto Gutierrez-Torre

Distributed Cloud-Edge Analytics and Machine Learning for Transportation Emissions Estimation

Thesis for the Doctoral Degree in Computer Architecture

Barcelona, July 2022

Universitat Politècnica de Catalunya - BarcelonaTECH
Computer Architecture Department

UPC
Universitat Politècnica de Catalunya
BarcelonaTECH
Barcelona, Spain

Thesis for the Doctoral Degree in Computer Architecture
Computer Architecture Department

Supervisors
Josep Lluís Berral García & David Carrera Pérez

Tutor
Yolanda Becerra Fontal

Cover depicts a view of San Vicente de la Barquera (Cantabria, Spain) from the port of La Barquera. A neural network transformed the original picture, specially the *Picos de Europa* mountain range, which was transformed into waves. Original image © Jon Chica (Photographer). The resulting image has been obtained from applying a Neural Network style transfer using Hokusai's The Great Wave off Kanagawa (Kanagawa-oki Nami Ura) as reference style.

Base thesis template courtesy of Andreas Liudi Mulyo.

This thesis was partially funded by a Severo Ochoa Center of Excellence grant SEV-2015-0493-16-5 during 4 years, by European Research Council (ERC) under the European Union Horizon 2020 research and innovation programme (grant agreement No 639595), the Ministry of Economy, Industry and Competitiveness of Spain under contracts TIN2015-65316-P, 2014SGR1051, 2017-SGR1414 and PID2019-107255GB, the Generalitat de Catalunya under project RIS3CAT GA.001-P-001662 and the European Union's Horizon 2020 research and innovation programme under grant agreement No. 952179 and No. 101004152.

To all the PhD students who are struggling

Keep on pushing

Nobody ever figures out what life is all about, and it doesn't matter. Explore the world.
Nearly everything is really interesting if you go into it deeply enough.

- RICHARD P.FEYNMAN

Abstract

In recent years *Internet of Things* (*IoT*) and *Smart Cities* have become a popular paradigm of computing that is based on network-enabled devices connected providing different functionalities, from sensor measures to domotic actions. With this paradigm, it is possible to provide to the stakeholders near-realtime information of the field, e.g. the current pollution of the city. Along with the mentioned paradigms, *Fog Computing* enables computation near the sensors where the data is produced, i.e. Edge nodes. This paradigm provides low latency and fault tolerance given the possible independence of the sensor devices. Moreover, pushing this computation enables derived results in a near-realtime fashion.

This ability to push the computation to where the data is produced can be beneficial in many situations, however it also requires to include in the Edge the data preparation processes that ensure the fitness for use of the data as the incoming data can be erroneous. Given this situation, Machine Learning (ML) can be useful to correct data and also to produce predictions of the future values. Even though there have been studies regarding on the uses of data at the Edge, to our knowledge there is no evaluation of the different modeling situations and the viability of the approach. Therefore, this thesis aims to evaluate the possibility of building a distributed system that ensures the fitness for use of the incoming data through ML-enabled Data Preparation, estimates the emissions and predicts the future status of the city in a near-realtime fashion. We evaluate the viability through three contributions.

The first contribution focuses on forecasting in a distributed scenario with road traffic dataset for evaluation. It provides a robust solution to build a central model. This approach is based on Federated Learning (FL), which allows training models at the Edge nodes and then merging them centrally. This way the models in the Edge can be independent but also can be synchronized. The results show the trade-off between accuracy versions training time and a comparison between low-powered devices versus server-class machines. These analyses show that it is viable to use Machine Learning (ML) with this paradigm.

The second contribution focuses on a particular use case of ship emission estimation. To estimate exhaust emissions data must be correct, which is not always the case. This contribution explores the different techniques available to correct ship registry data and proposes the usage of simple Machine Learning (ML) techniques to do imputation of missing or erroneous values. This contribution analyzes the different variables and their relationship to provide the practitioners with guidelines for correction and data treatment. The results show that with classical Machine Learning (ML) it is possible to improve the state-of-the-art results. Moreover, as these algorithms are simple enough, they can be used in an Edge device if required.

The third contribution focuses on generating new variables from the ones available with a ship trace dataset obtained from Automatic Identification System (AIS). We use a pipeline of two different methods, a Neural Network (NN) and a clustering algorithm, to group movements into movement patterns or *behaviors*. We test the predicting power of these behaviors to predict ship type, main engine power, and navigational status. The prediction of the main engine power is compared against the standard technique used in ship emission estimation when the ship registry is missing. Our approach was able to detect 45% of the otherwise undetected emissions if the baseline method was to be used. As ship navigational status is prone to error, the behaviors found are proposed as an alternative variable based in robust data.

These contributions build a framework that can distribute the learning processes and that resists network failures in low-powered devices. We propose different alternatives for the learning process depending on the requirements. The last two contributions focus on correcting the data of a particular use case. Nevertheless, the techniques applied are general enough to be used in other domains of application. With these corrections, the data post-process done in the Edge nodes, i.e. emission estimation, provides more accurate results as the fitness for use of the data is improved.

Acknowledgements

This chapter shows the human side of research, aside from all the technical details that we will cover later. Because, in the end, science is carried by researchers that are humans (at the present moment... who knows in 30 years!) and that are not always performing optimally. We are not machines, even though we learn. The journey of playing *The Game of Thesis* has been long and hard. Nevertheless, this manuscript shows that it was possible to carry on with the enormous task which a Ph.D. represents. Looking back we notice that this effort is not a thing done alone, even though the main work was done by the author. Therefore, this chapter is dedicated to the people that may not be authors in my papers but made them possible. Like P-Model, Underworld, Black Sun Empire, Björk, Lo-Fi Girl, Boards of Canada, The Flower Kings and so many more musicians and music collectives that have been with me during all the thesis. Also, I want to thank you, reader, for laying your eyes on this document. It takes a lot of courage and patience to go through a document like this. I hope this document brings you an idea or two that you can use.

First of all, of course, I want to say thank you to my advisors David Carrera and Josep Lluís Berral. David was the first contributor to this thesis as, thanks to him, I got a grant to do my research on this topic. He has always been there providing great insights on the conceptual side when we were focusing on the technical fun. From him, I have learned that it is better to do things that make sense than things that are technically novel and fun. Because we create things that will enable future advances.

On the other side, Josep Lluís has believed in me for years since the Bachelor's degree final project. He has endured my doubts, lack of performance at some points, and my procrastination. He has been an endless source of new ideas and general advice. If I am here today writing this thesis is because of the inspiration that I got from him. Thank you for being the best mentor and friend. Clai!

Also, thanks to Yolanda for being my tutor at UPC in the last years and managing all the related bureaucracy. Many thanks to Marc Guevara and Albert Soret from the Earth Sciences department from BSC for helping me with a topic I would have never thought I would research: Ship emissions estimation. They have helped me to get through the details and assumptions of the field, especially in the first years of my PhD, where I did not know the huge amount of work that these systems have inside.

Researching alone is a rare thing these days. This is why I have to also give my thanks to the fellow Ph.D. Students that I have met along the way. First of all, the BSC Data-Centric Computing team in which I have found mentors and partners in research: Felipe (Thank you for reviewing the thesis!), David B., Nicola, Marcelo, Aaron, Álvaro, Oscar, Tom, Jordà, Sergi, Paco, Anna, Claudia, Ferran, Gonzalo, Shuja, Kiyana, Juan Luís, Alejandro, Joan, Dani and Zoran. Thank you for being part of an amazing group. Moreover, I have found great people outside the group in the At Tudadis room: Jorge Carahuevo, Alberto M., Dani A., and Javier. Thank you for showing me in my early master years the hardships of the Ph.D. and motivating me to not pursue one. I am sorry that your advice was not followed.

Special thanks to the Doctoral Coffee Symposium members, Alexandre and Martí, for those afternoons full of science and procrastination which showed me that investing some time in relaxing and discussing the research and other topics can improve drastically the actual academic output. Also, thanks to Alan for our philosophical talks on those afternoons when we spoke about science, work, and all sorts of things.

Moreover, I want to say thanks to the workers of the Barcelona School of Informatics Bar for providing all those healthy meals and snacks during my 13 years of studies. The *dobles con queso* and

the *pitiflís* powered this work.

During my master's in Data Science, I have found inspiring people and bright minds. I especially want to say thanks to professor Valqui for his insights in what is science about[1] and how exciting and humbling it is. Those funny classes on how to do research and the Ph.D. comics strips[2] will not be forgotten. I am also very grateful to the professors Oscar Romero, Lluís Belanche, and Tomàs Aluja for being exceptional at teaching, inspiring, and helpful. Thank you for showing me what Data Science is about and how interesting can it be. Finally, I want to say thanks to Kaiser, Eduard, Iulian, Jonathan, and Poly for being great friends and helping me during that time.

I also want to thank the people from the Oasi magazine, from the UPC Kendo Club, and the students' associations from UPC for showing me other parts of life with discipline, knowledge, and, of course, pure fun. The list of names would be huge if I had to write it here. However, these experiences are not limited to the campus. The Associació d'Amics del MSX (AAMSX) has been a fountain of cool things, tin, and flux, all related to these magic machines that MSX are. Thanks to Maki, JamQue, DuHow, Chun, Aorante, DoraemonPPC, MsxKun, Marc el Seguero, Al, Julio and K0ga. Finally, after all this time you can call me doctor. Thanks also to Claudia A. and Laura for their classes full of fun and all the stress relief! And also thanks to Rubén and Mónica for being friendly and exceptional landlords in this Barcelona with evergrowing rents.

I also want to save a little spot for the medical staff and researchers who worked endlessly to fight the COVID situation. You have done great so far, we are almost there. Also, I want to give my thanks to those online video-conference platforms that enabled us to communicate during the lockdown.

Finally, this part is reserved for the family. First I want to thank my newly found Galician family, Begoña, Beatriz, Eva, Eduardo, and Helena, for taking so much care of me when visiting them. Thank you for everything, Begoña. On the other hand, moving to Barcelona with 17 years was very easy because I already had a great family to look after me. I want to especially thank Tía Pilar for all the *macarrones* and *mitjanes* on those Saturdays at her home, and all the company and care that she gave me. And the same did Lurdes, Toni, Fidel, Ana, and their children. You have always been there and I am very grateful that you have always watched over me!

An important spot is saved for my family from Cantabria. Thanks to my parents, Maribel and Fabio, who always said that what I am doing was complex but they still encouraged me to keep on working and took care that I was good and provided with great food like *chorizo* and *sobaos*. Obviously, without their dedication and help, I would not be here writing this. To my sisters Arancha and Lara, thanks for being there, taking care of me, showing me what computers were, (unwittingly) lending me things and taking me out when I was a kid... and also now because I still do not have a driving license. The Nintendo and the Visual Basic book took me far. I want to say special thanks to the smallest of the house, my niece Emanuchi, for being patient with me while I was working on holidays and for playing with me those old videogames like Bomberman. Yes, Emanuchi, I have been lying to you for years, I do understand english. Finally, I want to thank my grandparents, my godfather and great-aunt, who are not here to watch me finish this life stage but are always with me.

I want to mention the main contributor of this thesis, our cat Kim Bok Joo, who from time to time wrote in this document. Thank you for being my feline supervisor. Finally, I want to thank Dr. Antelo-Gonzalez, also known as Emma, for bearing with me during the Ph.D. We have both suffered and enjoyed the Ph.D. studies at the same time, which makes the Ph.D. a less lonely path. Having someone at home that understands perfectly what being a Ph.D. student feels is very helpful. I want to thank her for encouraging me and helping me organize and make sense out of what I was doing, even when I only could explain my research with unintelligible technicalities. More than once I have heard "Why are you doing that" while showing a new and shiny piece of code that did not have a clear purpose. Thank you for being there for me and I hope we keep on building a great future together.

[1] https://matt.might.net/articles/phd-school-in-pictures/
[2] https://phdcomics.com/

Contents

List of Figures

List of Tables

Part I

Preliminaries

Chapter 1

Thesis Introduction

1.1 Thesis context

In recent years *Internet of Things* (*IoT*) has become a popular term in both the academic and enterprise worlds. This term can be described in a simplified way as a new paradigm of computing that is based on network-enabled devices connected providing different functionalities, from sensor measures to domotic actions.

The *Smart City* movement makes use of the IoT paradigm as an enabler for better management of the cities, obtaining knowledge from the gathered data, and also using the connected actionable *things*. In this context, new architectures named Edge/Fog Computing have emerged. These architectures act as a standardized way to implement IoT and *Smart Cities*, e.g. VanLingen et al. [5]. These kinds of new Data-centric architectures are focused on where the data comes from, where it commutes, and where it is stored, providing data lineage and data quality assurance inside the given pipeline. Edge Computing refers to the architectures that enable the computation where the data is produced, i.e. in the edge devices which can be low-powered computers and sensor boards. Fog Computing is a term that covers both Cloud and Edge computing, as it makes use of the computation in the Edge devices and centralized Cloud resources. Moreover, it adds an intermediate layer of low-powered or mid-end computers that can act as aggregators or data processors in which intermediate operations can be executed, e.g. Machine Learning (ML). Figure 1.1 shows an example of an IoT network with intermediate edge devices infrastructure for data processing and aggregation.

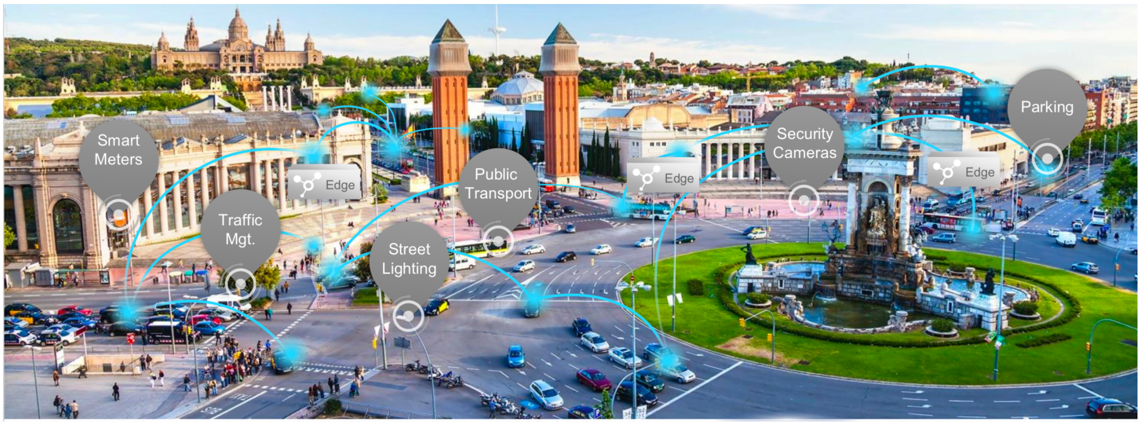


Figure 1.1. Example of different devices spread in a city that may be interconnected in the city of Barcelona. Intermediate edge devices may be set up to further process and aggregate the data produced in the devices. Image courtesy of David Carrera.

A research field that can directly benefit from IoT and Fog Computing is Air Quality monitoring. *Air Quality* is an important factor for the citizens' well-being, hence it is an important aspect to have in a *Smart City*. Regarding the societal impact of the *Air Quality*, in a recent study by Mueller et al. [6] performed in the city of Barcelona, it is shown that *Air Quality* is a cause, amongst other factors, of premature death. Moreover, another recent study by Bañeras et al. [7] shows that there is a correlation between air pollution and ST Elevation Myocardial Infarction (STEMI), which is a particular type of heart attack. In particular, this study shows that Particulate Matter (PM) and NO_2 pollutants increase the risk of having this kind of attack. This information has reached the citizens through the newspapers and television, making them more aware of the direct effects of pollution on their life.

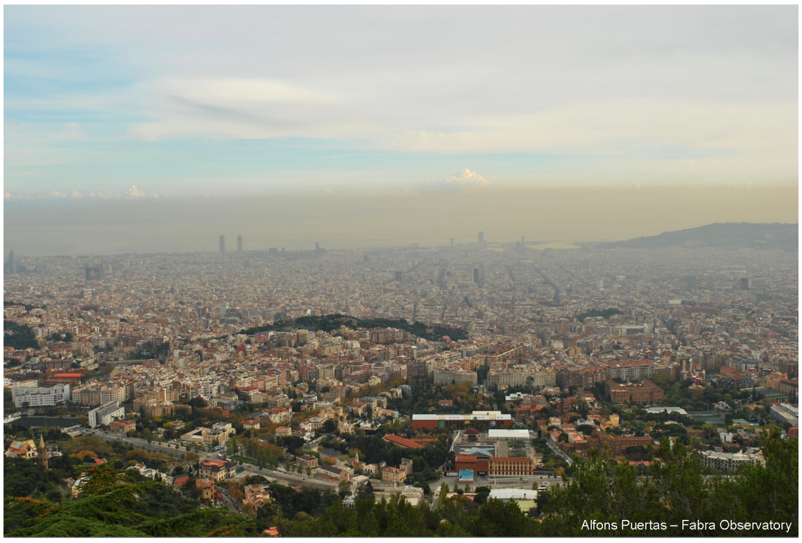


FIGURE 1.2. Pollution cloud over Barcelona. Photography was taken by the Fabra Observatory's meteorologist, Alfons Puertas.

Given the impact of *Air Quality* on our well-being and also on the environment itself, knowing how the pollution propagates has a key role in helping the governments to know their cities' status and make policies for future improvement. To evaluate, preserve and improve the state of the atmosphere, air quality modeling is a necessary tool that provides a complete description of the problem (i.e. meteorology, emissions, and atmospheric chemistry), which complements the information obtained from air quality monitoring networks.

In particular, the CALIdad del aire Operacional Para España (CALIOPE) air quality forecasting system[1] is a state-of-the-art modeling framework that integrates a meteorological model, an emission model, a dust model, and a chemical transport model to simulate air quality concentration with a high spatial (up to 1km^2) and temporal (1 hour) resolution for Europe and Spain. The air quality results are continuously evaluated with a system based on measurements from the European Environment Information and Observation NETwork (EIONET), and the performance of the system has been previously tested in the different evaluation and air quality management studies [8].

From the four different models that this system has, this work focuses on the emission models. The High-Elective Resolution Modelling Emission System (HERMES) model is the emission core of the CALIOPE system and has been fully developed by the Earth Science department of the Barcelona Supercomputing Center (BSC) [9], which gathers information about different areas that generate pollution (e.g. agriculture, road traffic) and estimates the pollution for each of them, as the first step of the Air Quality System.

As recorded in the Barcelona Air Quality Improvement Plan (2015-2018) [10], in 2013 in Barcelona the 46% of the NOx (Nitrogen Oxides) are produced in the port and 33% in road traffic. Similar numbers are seen for PM10 (Particulate Matter of less than 10 μm in diameter), as 52% is originated in the port and 37% in road traffic. Notice that this is the total amount of generated pollution which, depending on the wind, might not affect the city fully, especially for the port emissions. As observed in these numbers, transportation takes a key role in the emissions produced in the city. However, only ships provide Global Positioning System (GPS) traces and other attributes to compute directly the emissions.

[1]http://www.bsc.es/caliope/es

In particular, according to the European Community Shipowners Associations (ECSA) in 2015, maritime traffic has become a key component of the European economy [11]. According to a recent report by the International Maritime Organization (IMO), it is expected that this form of transport will continue increasing in the future due to globalization and the increase of global-scale trade [12]. The main reason for this is that sea transportation is more fuel-efficient than other modes of transport (e.g. trucks and trains). At the same time, it is considered an important contributor to primary atmospheric emissions in coastal areas [13] and subsequently to European coastal air quality degradation [14], especially in the North Sea and the Mediterranean basin. Maritime traffic is also responsible for about 2.5% of global greenhouse gas (GHG) emissions and it is expected to grow in the future for economic reasons. For instance, CO_2 emissions are expected to increase between 50% and 250% by 2050 [12]. Given the impact of the emissions in coastal areas and the expectations of growth of shipping emissions, it is very relevant to have as many details as possible of this type of emission origin.

Inside HERMES, there is a module that calculates the emissions generated by both road, ship traffic, and other sources by distributing the total pollution measured provided by the government using emission measures, usage profiles, routes, and other information. However, it can benefit from GPS based systems that can improve its spatio-temporal granularity, as is the case of road traffic in the work of Rodriguez-Rey et al. [15]. This kind of approach is based on having data points distributed over the map and aggregating them to the resolutions required. Using this methodology, not only we can estimate correctly the emissions but also we can know with greater detail where are they located, improving the precision of the whole Air Quality system.

It is important to know the current status of a city in a fine-grain detail and in a near-realtime fashion to make short-term policies and be able to raise alarms to the citizens. Moreover, it is also important to be able to evaluate possible policies for port management. The STEAM [1, 16] methodology enables having a system that has both features. This methodology uses data from ship registries containing static ship data, e.g. ship size or installed engine power, along with GPS enabled data, Automatic Identification System (AIS) data, to estimate the emissions in the precise place they are produced. Automatic Identification System (AIS) is a GPS-based tracking system used for collision avoidance in maritime transport, as a supplement to marine radars. AIS provides information such as a unique identifier for each transport (IMO identifier), the position as latitude and longitude (GPS positioning), the course and speed. Such information is used by maritime authorities to track and monitor vessel movements, from AIS base stations located along the coast, and transmitted through standardized VHF transceivers. According to the IMO's Convention for Safety of Life at Sea, AIS equipment is required to be installed in all international voyaging ships with more than 300 Gross Tonnage (GT) units, and all passenger ships [17], leading to the high availability of data.

Given that the information can be gathered by antennas distributed along the coast and other similar devices, Fog Computing provides a well-fitted framework for this problem. Figure 1.3 represents the conceptual infrastructure. First, in each node, the data is received and then the fitness of the data for the application is assured. Then, emissions can be computed and forecasting can be performed. After that, the derived data is obtained. This way the emission estimation is produced where the data is gathered and therefore the estimations can be provided with the minimum latency possible. This low latency provision is what enables the near-realtime emission estimation framework that provides precise geolocation. Moreover, if only statistics about pollution are required, they may be also computed in the nodes and only the required information would be sent. This approach provides storage and network efficiency as fewer data have to be stored and sent.

1.1.1 Thesis statement

Even though this system is used as a vital component in ship maneuvering and cruising, it is not free of machine and human errors [18, 19]. For example, ships can report unrealistic speeds when they

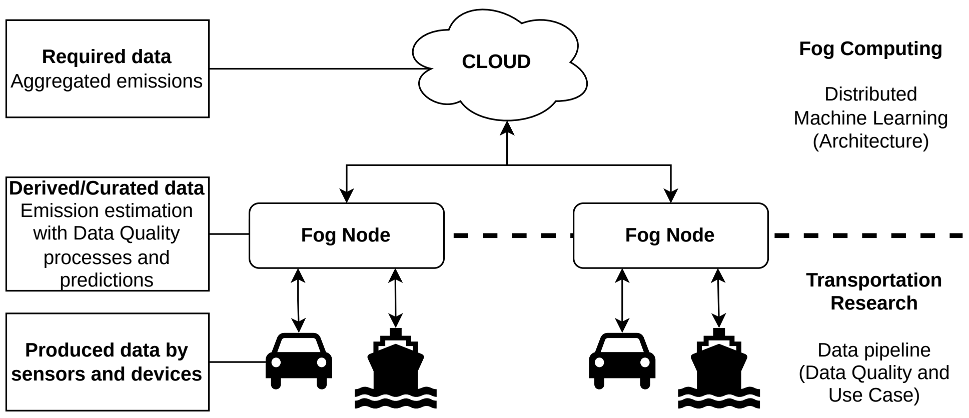

Required data
Aggregated emissions
CLOUD
Fog Computing
Distributed Machine Learning (Architecture)
Derived/Curated data
Emission estimation with Data Quality processes and predictions
Fog Node
Fog Node
Transportation Research
Produced data by sensors and devices
Data pipeline (Data Quality and Use Case)

Figure 1.3. Representation of how the transportation means interact with the system. The lower part represents the specific transportation research use case, which contains the data preprocessing process to improve the fitness of data (Data Quality), prediction, and emission estimation pipeline. The upper part represents the Fog Computing system which enables distributed Machine Learning. The Fog Node falls in between as the use case is elaborated using both parts.

are in the port premises [20]. Moreover, ship registries might be incorrectly filled, and incomplete and some ship data may be unavailable [3]. Given that there are many shortcomings of using this data, a methodology for cleaning it should be established to be able to use it for purposes like emission estimation, which requires accurate data to be able to correctly estimate emissions.

This work aims to demonstrate the feasibility of the following thesis: **It is possible to build a distributed system in a Fog Computing environment to apply complex data processes (i.e. ensure data quality, estimate emissions and predict future values) in a smart city setting with results in a near-realtime fashion.**

1.2 Thesis contributions

To cover the objective of this thesis, we need to provide computation with low latency to make the system near-realtime that is able to manage data. In particular, there is a need to adapt the Data Science pipeline to the IoT environment. This pipeline must be able to receive, curate and make use of data at the *Edge* level, where the data is produced, or intermediate levels. Moreover, the processes required for traffic prediction and emission estimation can be complex and also require of data preparation steps, i.e. correct data so that the errors in it do not propagate to derived results. Finally, as there is a need to provide a near-realtime system, the proposed architecture must be able to provide derived results as the data comes into the system.

Therefore, a *Fog Computing* framework is needed to provide minimal network transmission of the data and computation where the data is produced. Moreover, we need to have a system that can clean data and predict the next status of the city using Machine Learning (ML) techniques. In some cases the data will not be trivially cleaned, therefore we need mechanisms to extract new features from available data to restore the required data for the emission modeling process.

The requirements of such a system can be summarized in three questions:

- Can we distribute Machine Learning used for transportation use cases in a Fog/Edge Computing framework?

- Can we improve emission estimation by correcting the input data with data preparation processes?

- Can we extract new features that can be used when the required data is not available?

To answer these three questions, this thesis includes three parts:

- **First contribution**: Proof that it is viable to use Machine Learning on the Edge enabling near-realtime processing. In particular, we apply Federated Learning (FL) which is a kind of distributed Machine Learning framework. With this, we show the possibility of building a central model by merging local models trained at the Edge. Moreover, the structure of the Data Science pipelines in the IoT environment is provided to show how to translate from a classical Data Science pipeline to the IoT world.

- **Second contribution**: Improve the fitness of the data for the application (Data Quality) with a real ship dataset with simple Machine Learning techniques, so that non Machine Learning experts can take profit from the work and that low-powered devices can make use of them. This contribution is focused on the data preparation step of the Data Science pipeline and provides a guideline on how to correct ship static registry data to be used later for derived results.

- **Third contribution**: Reconstruct missing ship data and generate of new features based on the methodology used in the other contributions. Again, this is related to the data preparation step of the Data Science pipeline, however, the focus here is to reconstruct the ship registry data that is required and is not available. This is done using another dataset that will be introduced later in this work. Moreover, the same process can be used to derive new features that can help with future problems.

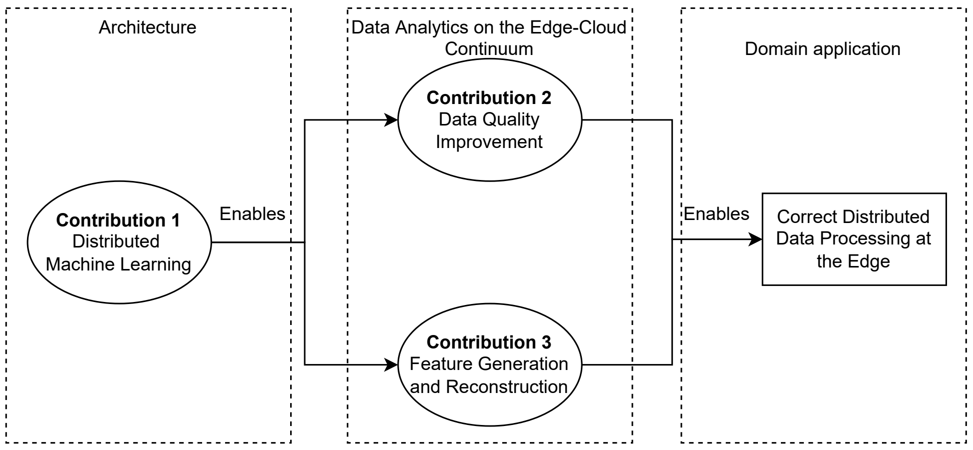

Architecture
Data Analytics on the Edge-Cloud Continuum
Domain application
Contribution 1
Distributed Machine Learning
Enables
Contribution 2
Data Quality Improvement
Contribution 3
Feature Generation and Reconstruction
Enables
Correct Distributed Data Processing at the Edge

FIGURE 1.4. Thesis contribution diagram. The first contribution is centered around the exploration of distributed Machine Learning architectures. The second and third contributions explore data processing techniques to improve the fitness of data for the application. The three contributions together propose a framework to improve and process data that can be distributed in a Fog computing scenarios.

Figure 1.4 represents the thesis contributions and their relationships. The first contribution acts as an enabler of distributed processing. The other two contributions build processes to improve the fitness of data for the application (Data Quality), in particular, focused on the data preparation step using Machine Learning techniques for data imputation/correction. Both contributions can be distributed by using the first contribution results, especially the third contribution, which depends completely on streaming data. With these contributions, an initial framework for using Machine Learning in a Fog Computing setting is established for transportation research, with a special focus on ship emission estimation as the main driving use case. This framework will be used to build a system that is able to receive streams of ship data at the edge, curate them and then estimate the emissions for those ships. All of this is performed at the *Edge* without requiring the raw data to leave the *Edge* devices and providing derived results, e.g. emission estimations, with low latency.

1.2.1 First contribution: Machine learning distribution for traffic prediction

This contribution explores how to adapt the Data Science pipeline to IoT and how to create central models for prediction and data preparation in a distributed way, i.e. models trained with all the data available from a city. This is done using Federated Learning (FL) framework, which is a way to distribute Machine Learning (ML) model training. Models are trained at the *Edge* with the data they have and then the models are sent back to the *Cloud* for merging. This merging process only requires the models, so no data is sent back. Finally, the merged model is sent back to the *Edge* so it can be used or retrained. In this work, we show how this is done and the impact of part of the hyper-parameters when training a model for traffic prediction.

This contribution works towards enabling the computation at the edge nodes so that there is no necessity to upload all the data from the *Edge* nodes to the *Cloud*, providing network fault resistance and consuming less power because of the cut on network usage. In this sense, it also enables near-realtime, as the data is computed in the edge nodes and then it can be directly consumed, instead of having to wait for the data to be transferred and computed in the *Cloud*.

This work is performed with a Gated Recurrent Unit (GRU)-based Neural Network model and extends the work done by Gutierrez-Torre et al. [21] including Federated Learning (FL) synchronization rounds and other relevant details to build a successful distributed model. We test the training of the algorithm in different conditions and simulate training at the central node vs. training in a distributed environment. Three different hardware are compared: two low-powered devices, Raspberry Pi 3B and NVIDIA Jetson Nano, and a Xeon server-class machine. The batch size, i.e. the number of processed samples at the same time, is studied to check if the Graphics Processing Unit (GPU) inside the NVIDIA Jetson Nano can take profit from packing samples.

Overall, the performance of the distributed training is close to the centralized training and, in some cases, the model produced is better. For this problem, both low-powered devices show acceptable training times compared to the server-class machine. In this case, the GPU is not an advantage to have due to the format and amount of the data, being the Raspberry Pi more suitable for this problem.

1.2.2 Second contribution: Estimating missing data for emission modeling

The second contribution revolves around the process of emission estimation for ships. The STEAM [1, 16] model makes use of AIS data and ship registry data to compute the pollutants with high spatio-temporal precision. This can be done by leveraging the GPS coordinates and timestamps that are provided by ships equipped with AIS. Moreover, the ship registries provide ship engine details, e.g. installed engine power, that enable to compute the power used by a ship during their cruising and maneuvering.

However, as in every process with real-world data, the data may be missing or incorrectly recorded. In the case of AIS data, there are works like Jeon et al. [2] that propose complete pipelines to perform data preparation, i.e. cleaning, for the ship traces with clear results. Yet, the same does not happen for the ship registry data, where the proposed solutions are vague and a clear comparison of methods is not provided.

In this contribution, we propose to use simple ML techniques as the data preparation step of the Data Science pipeline proposed in the previous contribution. The objective is to clean this kind of data and provide a comparison of methods from the state of the art to do so. The methods used are standard and simple to implement so that researchers and practitioners that are not experts in ML can test and implement them. On the other hand, simple methods can be used in commodity hardware or low-powered devices, e.g. Raspberry Pi or MicroController Unit (MCU).

A review of the state of the art methods is given for AIS and ship registry data cleaning. Experiments to correct each variable required for the emission estimation process is done iteratively: first, all the other available variables are used, and then the importance of every variable for predicting each variable is computed. With this, a subset of two variables is selected to predict each of the variables and it is compared with the results of using the same subset plus the ship type. Finally, a subset with the most important variable for each variable is tested. In this way, we offer a comparison of the algorithms in the described setting and a comparison between them.

The methods that performed generally best were *Random Forest* and *Gradient Boosting Trees*. *K Nearest Neighbours* offered the easiest way to implement a cleaning method, as it can effortlessly predict multiple variables at once and offers a good baseline result. Moreover, a list of important variables to predict the others is established, providing a guide on which variables to use for each case. With this work, a common comparison of methods is established and guideline to treat the ship registry data is offered, so that practitioners can apply these methods.

1.2.3 Third contribution: Feature extraction and missing data estimation from alternative data sources

As seen in the previous contribution, there is an important part of the data that is incorrect and needs to be estimated. AIS particularly provides the ship type so that the ship can be minimally identified, navigational status so that we can know what is the ship doing at each moment, e.g. cruising or fishing, and also provides the position, rotation, and speed, among others. This last set of variables is crucial for identifying the movement of the ship. However, sometimes ship type is missing, and navigational status has human errors as it is manually managed. Moreover, if the required ship static data for emission estimation is completely missing it is not possible to perform the corrections done in the second contribution. Therefore, there is a need for a method to extract them from the GPS traces, as it is the most reliable source of information in AIS.

In this work, a method to group similar movements in a ship trace is presented. The movements are represented by the speed and the rotation of the ship and the position in terms of bathymetry, i.e. how deep is the sea in the current ship position. These groups of movements are named behaviors, as they are linked with what the ship is doing at each moment. These newly found behaviors are then used as input variables to estimate the ship type, the main engine power of a ship, and linked with the navigational status variable. This is done using the Conditional Restricted Boltzmann Machines (CRBMs) which encodes a time window of attributes previously mentioned in a numeric array. This codification is then used for prediction and also to find the behaviors by clustering similar codifications. The behaviors are then correlated with the navigational status to relate what is observed. Then, profiling of each ship type is done using the amount samples from each behavior found.

The methodology proved to be useful for enhancing AIS datasets by correcting and expanding their features, toward producing better estimations when using AIS-based emission models. Experiments show that ship type and navigational status can be corrected on missing data scenarios. Moreover, they show that navigational status can be expanded with new uncovered behaviors. Finally, experiments have proved that our method can estimate the power consumption of the ship better than the standard methods when there is no ship registry data available. The methodology is able to detect around 45% of the undetected emissions when the main engine power specification is not available. Therefore, this contribution not only provides a mechanism to extract new features that can be used for future research on the topic but also provide a methodology for the data preparation step of the Data Science pipeline that is able to reconstruct ship registry data that is not available from a stream of incoming AIS data.

1.3 Document organization

This thesis is organized as follows. Chapter 2 introduces the required knowledge in Emission Estimation and Machine Learning (ML) to understand the contributions and the datasets used. Chapter 3 reviews the related work in IoT with centralized and distributed ML, and transportation research. Chapter 4 studies how to transfer the Data Science pipeline to the IoT environment and shows how to build models in the *Edge* devices and synchronizes them in the central node using Federated Learning (FL). Chapter 5 provides an end-to-end process to correct ship registry data required by the emission model and studies the relationship between variables to help these corrections. Chapter 6 proposes a method to extract patterns from AIS traces and shows how to apply them for data correction and feature generation. Finally, Chapter 7 presents conclusions extracted from the contribution and possible future lines of research.

Chapter 2

Background

2.1 Introduction

This chapter introduces all the background required for understanding this thesis, aside from basic knowledge of Machine Learning (ML). The section is divided into three parts: the first part describes the datasets used in the following chapters. The second part covers the non-basic Machine Learning (ML) algorithms and methodology used. Finally, the third part contains all the information required about ship emission estimation. Nevertheless, the key concepts will be repeated in each section to make the content as clear as possible.

2.2 Machine Learning

In this subsection we present the most relevant non-classical ML methods to understand the contributions presented in this thesis. It mainly covers two neural networks models: Conditional Restricted Boltzmann Machine (CRBM) [22] and Gated Recurrent Unit (GRU) [23] networks. The objective of the first one is to build codification of the data including time dependencies and predict time series, whereas the objective of the second is just for the latter functionality. CRBM is used as key building block in Chapters 4 and 6. GRU networks are used as learning algorithm in Chapter 4.

2.2.1 *Conditional Restricted Boltzmann Machines*

As mentioned before, one of the objectives of the Conditional Restricted Boltzmann Machine (CRBM) network, aside from predicting, is to codify the input data. In this work it is used codify the input data considering the time dependencies of the samples. This enables non time aware methods to use the time series method seamlessly. Before explaining the CRBM, first Restricted Boltzmann Machine (RBM) is introduced as a building block.

Restricted Boltzmann Machines

An Restricted Boltzmann Machine (RBM), or more concretely Gaussian Bernoulli RBM (GB-RBM) is a key building block of the Conditional Restricted Boltzmann Machine (CRBM). A GB-RBM is an undirected graphical model with binary hidden units and visible Gaussian units that models the joint log probability of a pair of visible and hidden units $(\boldsymbol{v}, \boldsymbol{h})$. This means that it models the relationship between the input and the resulting numbers of the hidden neurons, i.e. hidden neurons activations. These activations act as a code that allows us to reconstruct the original data. It is modeled as:

$$\log P(\boldsymbol{v}, \boldsymbol{h}) = \sum_{i=1}^{n_v} \frac{(v_i - c_i)^2}{2\sigma_i^2} - \sum_{j=1}^{n_h} b_j h_j - \sum_{i=1}^{n_v} \sum_{j=1}^{n_h} \frac{v_i}{\sigma_i} h_j w_{ij} + C \tag{2.1}$$

where σ_i is the standard deviation of the Gaussian for visible unit i, c is the bias of the visible units, b is the bias of the hidden units, w_{ij} is the weight connecting visible unit i to hidden unit j and C is a constant. Notice that n_v and n_h refer to the dimension of $\boldsymbol{v}$ and $\boldsymbol{h}$ respectively. In practice, we normalize the data to have zero mean and unit variance. Moreover, σ_i is fixed to 1 because it empirically works well as shown in the work of Taylor et al. [24].

This kind of undirected graphical model is trained with a process called Contrastive Divergence [24] which enforces that the activations act as a code to recover the original data, as mentioned before.

Conditional Restricted Boltzmann Machines

The Conditional Restricted Boltzmann Machine (CRBM) is a GB-RBM that models static frames of a time series modified with some extra connections used to model temporal dependencies. The CRBM keeps track of the previous n visible vectors in a $n \times n_v$ matrix which we call the history of the CRBM, where n_v is the number of different variables. The learned parameters of the CRBM are three matrices **W**, **A**, **D**, as well as a two vectors of biases **c** and **b** for the visible and hidden units respectively. $\mathbf{W} \in \mathbb{R}^{n_v \times n_h}$ models the connections between visible and hidden units, as in the previous section. $\mathbf{A} \in \mathbb{R}^{(n_v \cdot n) \times n_v}$ is the mapping from the history to the visible units. $\mathbf{D} \in \mathbb{R}^{(n_v \cdot n) \times n_h}$ is the mapping from the history to the hidden units. Training and inference in the CRBM are performed using the contrastive divergence method, as with RBMs. Figure 2.1 shows how the input data and the CRBM interact in terms of reconstruction.

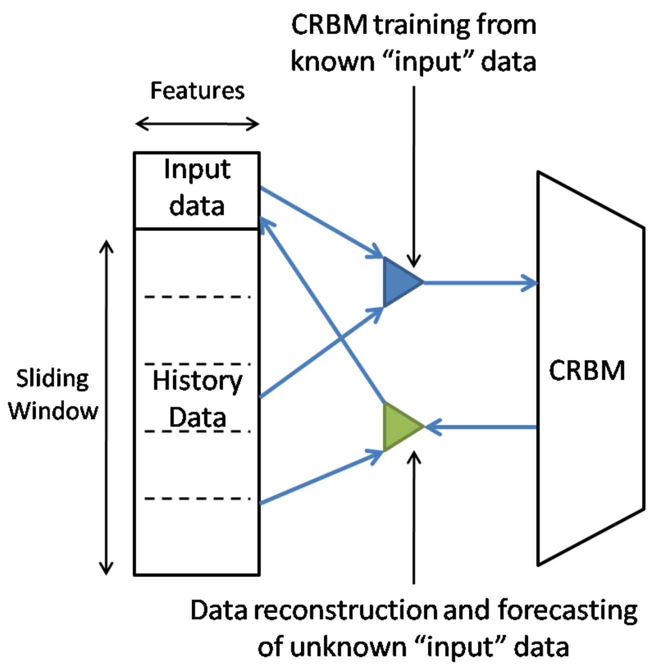


Figure 2.1. Schema of CRBM training and prediction

Figure 2.2 shows a graphical representation of a CRBM referencing the previously mentioned variables. In the case of this thesis, we have an interest in using the activations produced by the hidden units when fed with a sample of traffic trace plus the n steps window. Then we use this vector of activations as a time-aware code that represents a sample with a time window for algorithms that are not thought to handle time-series, enabling them to manage temporal dependencies. Notice that even though CRBMs suffer when doing long-term predictions, therefore an alternative method is also studied for prediction.

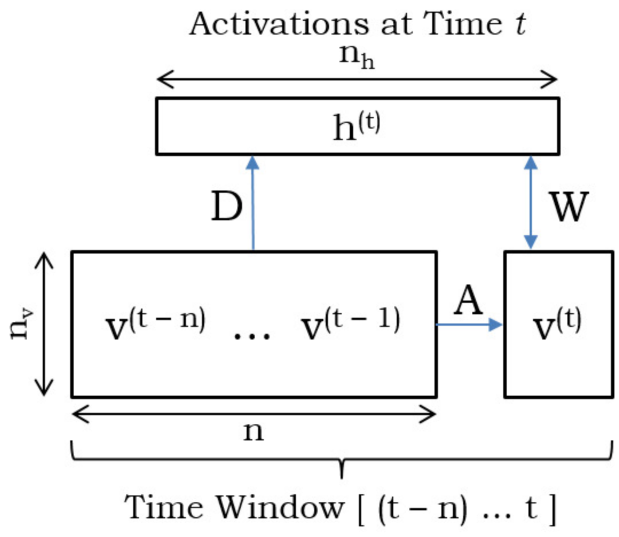


FIGURE 2.2. CRBM with window size n and n_h hidden units

2.2.2 *Gated Recurrent Unit (GRU) Network*

Another technique that we use for traffic modeling are Recurrent Neural Networks (RNNs), specifically a configuration known as *Gated Recurrent Unit (GRU)* network. This network is formed by one or more layers of *GRU* and other neural network layers, e.g. densely connected layers for prediction. With this kind of network we can predict a long period of time series.

This kind of layer can deal with time dependencies using a gating mechanism. This gating mechanism is composed of two different gates for each *GRU* layer: the reset gate and the update gate. These two gates affect the value that the next hidden status S_t will have. This S_t is used for internal calculations as the state of memory and it is also used as the output of the layer. Figure 2.3 provides a graphical representation of this kind of layer. The data flow represented in this figure is a simplified version of the following equations. Biases and weights are omitted for the sake of clarity. The box containing r_t is the reset gate, the box containing u_t is the update gate and the one containing $\widetilde{S}_t$ is the proposed new state. In the following equations, x_t represents the input at time t.

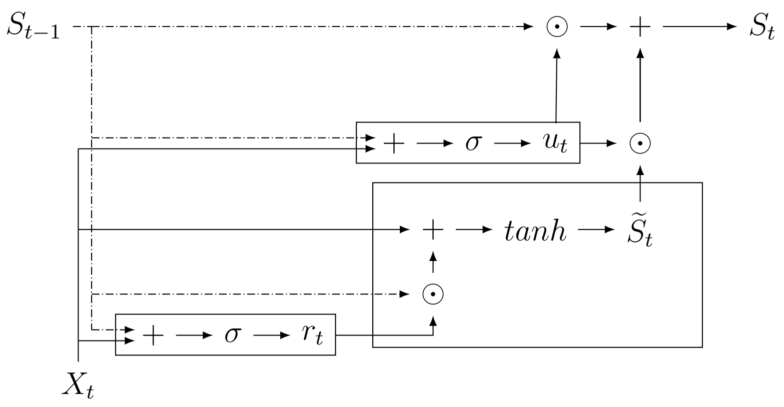

S_{t-1}
$\odot$
$+$
S_t
$+ \rightarrow \sigma \rightarrow u_t$
$\odot$
$+ \rightarrow tanh \rightarrow \widetilde{S}_t$
$\odot$
$+ \rightarrow \sigma \rightarrow r_t$
X_t

FIGURE 2.3. Schema of a unit of a GRU layer

First, the reset gate controls the access to the previous hidden state S_{t-1} and is used to compute the new proposed state $\widetilde{S}_t$ as can be seen in Equation 2.4. The reset gate is computed as shown in

Equation 2.2.

Second, the update gate is in charge of how much the state is updated, i.e. which interpolation between S_{t-1} and $\widetilde{S_t}$ is desired, between the extremes of maintaining the previous state or updating it completely. This interaction is shown in Equation 2.5 The update gate is computed as shown in Equation 2.3.

Finally, the proposed state $\widetilde{S_t}$ is the new hidden state calculated regarding the previous state and the current input. This status is calculated in Equation 2.5. Notice that $\odot$ is the Hadamard or element-wise product and σ the activation function. After the gates and the proposed state is computed, we do an interpolation between the current state S_{t-1} and the proposed next state $\widetilde{S_t}$, which represents how much do we change our memory given the current input x_t.

$$r_t = \sigma(x_t W^{xr} + S_{t-1} W^{sr} + b_r) \tag{2.2}$$

$$u_t = \sigma(x_t W^{xu} + S_{t-1} W^{su} + b_u) \tag{2.3}$$

$$\widetilde{S_t} = tanh(x_t W^{xs} + (r_t \odot S_{t-1}) W^{ss} + b_s) \tag{2.4}$$

$$S_t = (1 - u_t) \odot S_{t-1} + u_t \odot \widetilde{S_t} \tag{2.5}$$

Each layer has a given number of units as the one represented in Figure 2.3 that have memory, i.e.feedback loop. The more units we add, the more elements the layer will be able to remember.

2.3 Emission estimation

Air Quality Modelling Systems (AQMS) are systems built to estimate and analyze the quality of the air. These systems describe what pollutants there are in the air, where they are located, and when they are emitted. These systems are built of several models that take into account where, when, and how much of each pollutant is produced, how is it transported in the air, and how the pollutants react with other chemicals that are in the air. In this work, we are particularly interested in the origins of the emissions. Inside the AQMS the module that performs this task is the emission estimation module. This module makes use of an activity factor, e.g. the movements of ships in the sea, and the emission factor, i.e. for a given amount of energy produced how much pollutant is produced. With these two factors, the emissions can be computed. Specific details on this computation for a specific model are given in Section 2.3.2.

In case that the emission origin can be tracked and continuous activity measurements are extracted, the emission estimation model we will be able to estimate the emissions produced continuously in a near-realtime fashion. This model is a key component to build a system that provides the status of the emissions in the air constantly so that policymakers can know the status of an area. In particular, this part of the background is required to understand the contributions of Chapters 5 and 6 as it is the main driving use case.

2.3.1 Top-down vs Bottom-up emission estimation

In general, there are different types of emission estimation approaches: Top-down and Bottom-up. Even though there may be mixed methodologies. The top-down estimation approach estimates the emission using a global measurement for an area and then disaggregates it using activity factors, e.g. known traffic in a given set of roads. With this process, a grid of emissions is built, which represents the status of the area as a raster, i.e. a map of cells of a given size ($1km^2$ for example) in which each contains the amount of pollutant for that particular region. On the other hand, bottom-up approaches like HERMESv3 [25], which is an estimation methodology included in the CALIOPE Air Quality System built in the Barcelona Supercomputing Center, make use of punctual estimations measured or estimated in the area of interest and then use them along their positions to build the emission grid. With this kind of approach, the emissions can be located and tracked. Compared to Top-down approaches, Bottom-up require access to detailed information which may be harder to obtain. However, these methodologies provide finer spatio-temporal granularity. For this work, Bottom-up methodologies are selected, as they enable the creation of a framework that can estimate emissions in a near-realtime fashion and provide better spatio-temporal resolution.

2.3.2 STEAM

The Ship Traffic Emission Assesment Model (STEAM) model is a bottom-up ship exhaust emission model developed in the Finnish Meteorological Institute by Jalkanen et al. [1, 16, 26]. This model makes use of ship traces to estimate and geographically locate the pollution. STEAM2 [16] is an upgraded version of the STEAM model with improved engine characterization and the possibility of estimating different pollutants not covered in STEAM. However, in presence of missing data STEAM act as a fallback of STEAM2. Therefore this work only covers STEAM [1] and a small part of STEAM2 [16].

Ship traces can be obtained from the Automatic Identification System (AIS), the GPS-based tracking system used for collision avoidance in maritime transport as a supplement to marine radars. This system is used to track and monitor vessel movements from base stations located along the coast used in all the ships to prevent collisions. It is transmitted through standardized Very High Frecuency (VHF) transceivers. AIS provides, for each vessel, its unique IMO identifier and Maritime Mobile Service Identity (MMSI) number, GPS positioning, course, and speed among other information. This system is mandatory, according to IMO's *Convention for Safety of Life at Sea*, for all ships with gross tonnage greater than 300 tons, and all passenger ships [17]. This data provides basic details like position and speed which can be used, along with a ship registry, to estimate emissions as Figure 2.4 depicts.

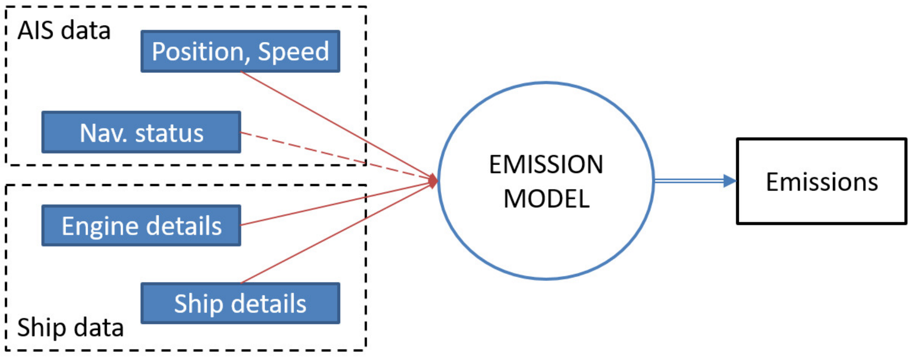


FIGURE 2.4. Estimation of Emissions from Ship Traces

Our approach is a methodology based on the STEAM model, proposed by Jalkanen et al. [1]. STEAM is an AIS-based emission estimation model that uses the traces of the ships and their characteristics to provide information about the pollution with GPS positioning precision. Emissions are calculated using the current power consumption of the ship at a given time and the emission factor for that ship regarding a pollutant. Conceptually, the formula is the following (units in brackets):

$$E_{s,p,x,t}[g/h] = P_{s,x,t}[kW] \cdot EF_{s,p}[\frac{g}{kWh}]$$

Being P the current power consumption of the ship and EF the emission factor, s ship characteristics (mainly engine), p the pollutant to estimate, x position of the ship, and t the current time. Therefore, the emission of a given pollutant p for a ship s that is in the position x and time t is conditioned by the ship characteristics for calculating the actual power that this engine is using and the ship characteristics plus the pollutant constants for the installed engine.

Power estimation

To estimate the emissions, first, we have to know how to calculate the power consumption of a ship. We have to take into account that generally there are two kinds of engines installed in ships: main engine and auxiliary engine. The first is mainly in charge of the movement of the vessel and the latter is in charge of the onboard electrical power devices but may be used for other tasks, e.g. maneuvering.

Main engine The power used by the main engine at a given time t is estimated using the current vessel speed, the design speed and the installed power. In particular, the formula to calculate the transient power is the following:

$$P_{transient} = \frac{V_{transient}^3}{(V_{design} + V_{safety})^3} * \varepsilon_p * P_{installed}$$

Being $V_{transient}$ the current speed provided by AIS, V_{design} the maximum speed that the ship can reach by design, V_{safety} a safety offset as ships may report speeds slightly greater than V_{design}, ε_p engine load at Maximum Continuous Rating and $P_{installed}$ the actual power installed in kilowatts. This formula acts as a ratio between the current speed and the maximum speed to find the actual power, along with an efficiency multiplier. Following the instructions from STEAM methodology, V_{safety} is fixed to 0.5 knots (2,57 m/s) and ε_p is set to 0.8.

Auxiliary engine The auxiliary engine is in charge of tasks like providing electrical power to the ship, e.g. to the sockets in the passenger rooms, or providing power for maneuvering in ports. As the usage of the auxiliary engine is not entirely reflected in any data available, we have to make some assumptions. The proposed assumptions in the work of Jalkanen et al. [1] are the following:

- If the ship is a passenger ship, RoPax (ferry) or a cruiser: constant 4000kW assumed
- For other ship types it depends on the current status:
  - Cruising (speed > 5): 750kW
  - Maneuvering (5 >= speed > 1): 1250kW
  - Hoteling (1 >= speed) : 1000kW

Notice that if the assumed output power is greater than the actual installed power, the output power is lowered to match the installed power. The current status of the ship is extracted from the speed that the ship has, as reflected in the list above (in knots). There is a strong seasonal usage of the auxiliary engines for air conditioning. This is considered to be compensated as boilers are used in winter and air conditioning in summer. Finally, we must take into account that there is data that can be used for a more precise auxiliary engine estimation. However, this is covered only in STEAM2 [16].

Engine Load and Relative Specific Fuel-Oil Consumption (SFOC) As STEAM does not cover Particulate Matter (PM) emissions, we need to include some characteristics from STEAM2 [16] regarding engine characterization. In particular, we need to add two concepts from STEAM2: Engine load (EL) and SFOC relative to the load ($SFOC_{Relative}$). With these two components, we can track the engine load at each moment and its particular fuel consumption, which is linked directly with the PM. To calculate the load of the engines we need to know how many engines there are and their rated power.

$$EL = \frac{P_{Total}}{P_E * n_{OE}}$$

$$n_{OE} = \frac{P_{Total}}{P_E} + 1$$

Being:

- P_{Total} = Actual power being used
- n_{OE} = Number of operative engines
- P_E = Power of one engine

In STEAM2 [16] the SFOC is composed of an SFOC base value which is set according to the ship specifications and another part that acts as a multiplier of this base value. $SFOC_{Relative}$ was extracted from a regression analysis using real engines and it is computed from the engine load as follows:

$$SFOC_{Relative} = 0.445 * EL^2 - 0.71 * EL + 1.28$$

Emission Factor

As shown in the equation, to calculate the exhaust pollution emitted we also need to have a value that explains how much of a given component is emitted per unit of fuel used or, as in this case, per kW produced. In this case, we will use $\frac{g}{kWh}$ as a reference unit for the emission factor. All the factors are extracted from the supplemental materials of STEAM [1] and STEAM2 [16].

NOx The amount of NOx is obtained from a regression study made by IMO as seen in the work of Jalkanen et al. [1]. The NOx has a direct link with the Revolutions Per Minute (RPM) of the engine, as can be observed in the curve of Figure 2.5. The formula is the following:

$$EF(NO_x) = \begin{cases} 17, RPM < 130, \\ 45 * RPM^{-0.2}, 130 <= RPM < 2000, \\ 9.8, RPM <= 2000 \end{cases}$$

SOx The amount of SOx, i.e. amount of every sulfur oxide, can be estimated by calculating the amount of SO2 as other sulfur oxides are found in much lower concentrations. We start from the idea that the number of mols of sulfur is the same as the number of mols in SO2, as it only has one atom of sulfur. First, we calculate how many mols of sulfur there are and then we relate them to SO2 to obtain the actual mass, as defined by the following equations:

$$n(S) = \frac{m(S)}{M(S)} = \frac{SFOC * SC}{M(S)} = \frac{SFOC * 0.001}{32.0655} = SFOC * \frac{0.001}{32.0655}$$

$$n(S) = n(SO_2)$$

$$EF_{SO_2} = m(SO_2) = M(SO_2) * n(SO_2) = 64.06436 * SFOC * \frac{0.001}{32.0655} = SFOC * 0.001998$$

Being:

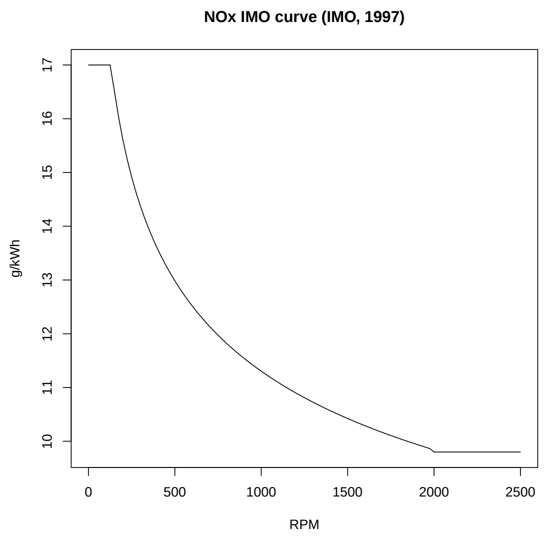


FIGURE 2.5. NOx curve as established in IMO 1997 [1]

- $SFOC$ = Specific Fuel Oil Consumption (g/kWh)
- SC = Sulphur content of fuel (mass %)
- $M(S)$ = Molar mass of sulphur (g/mol)
- $n(S)$ = number of mols of sulphur (mol)
- $m(S)$ = mass of sulphur (g)
- $M(SO_2)$ = molar mass of sulphur dioxide (g/mol)
- $n(SO_2)$ = number of mols of sulphur dioxide (mol)
- $m(SO_2)$ = mass of sulphur dioxide (g)
- EF_{SO_2} = emission factor for SO_2 (g/kWh)

Notice that we assume that SC is 1% as it is the current regulation with the Sulphur Emission Controlled Area (SECA).

CO2 To calculate the mass of CO2, we follow a similar approach to what we have seen in the previous pollutant. In this particular case, the CC is 85% as shown in STEAM [1].

$$n(C) = \frac{m(C)}{M(C)} = \frac{SFOC * CC}{M(C)} = \frac{SFOC * 0.85}{12.01} = SFOC * \frac{0.85}{12.01}$$

$$n(C) = n(CO_2)$$

$$EF_{CO_2} = m(CO_2) = M(CO_2) * n(CO_2) = 44.00886 * SFOC * \frac{0.85}{12.01} = SFOC * 3.114699$$

Being:

- $SFOC$ = Specific Fuel Oil Consumption (g/kWh)

- CC = Carbon content of fuel (mass %)
- $M(C)$ = Molar mass of carbon (g/mol)
- $n(C)$ = number of mols of carbon (mol)
- $m(C)$ = mass of carbon (g)
- $M(CO_2)$ = molar mass of carbon dioxide (g/mol)
- $n(CO_2)$ = number of mols of carbon dioxide (mol)
- $m(CO_2)$ = mass of carbon dioxide (g)
- EF_{CO_2} = emission factor for CO_2 (g/kWh)

PM Particulate Matter is in fact an emission that is composed of SO_4, H_2O, OC (Organic Carbon), EC (Elementary Carbon) and ashes. The following formulas compute the emission factor and were extracted from Jalkanen et al. [16]:

$$EF(PM) = SFOC_{Relative} * (EF_{SO_4} + EF_{H_2O} + EF_{OC} * OC_{EL} + EF_{EC} + EF_{Ash})$$

Where:

$$EF_{SO_4} = 0.312 * SC$$
$$EF_{H_2O} = 0.244 * SC$$

$$OC_{EL} = \begin{cases} 3.333, EL < 0.15 \\ \frac{a}{1+b*e^{-c*EL}} \end{cases}$$
$$a = 1.024, b = -47.66, c = 32.547$$

$$EF_{EC} = 0.08$$
$$EF_{OC} = 0.2$$
$$EF_{Ash} = 0.06$$

Being:

- $SFOC_{Relative}$ = Specific Fuel Oil Consumption multiplier relative to engine load
- EL = Engine Load (load %)
- EF_{SO_4} = Emission Factor for SO_4 (g/kWh)
- EF_{H^2O} = Emission Factor for H^2O (g/kWh)
- EF_{OC} = Emission Factor for OC (Organic Carbon) (g/kWh)
- OC_{EL} = OC multiplier in terms of Engine Load (combustion efficiency)
- EF_{EC} = Emission Factor for EC (Elementary Carbon) (g/kWh)
- EF_{Ash} = Emission Factor for ashes (g/kWh)
- SC = Sulphur content of fuel (mass %)
- a, b, c = regression coefficients as defined in STEAM2 [16]

Limitations

Notice that the ships may have installed devices that provide abatement techniques, i.e. techniques that reduce pollution, however, this information is not provided in the data we have available. The contribution of the sea waves is also not included for the sake of simplicity, but can be included using the work of Jalkanen et al. [1].

2.4 DATA

In this section the required datasets for the work of this thesis are introduced. Three datasets are covered for this thesis: Floating car dataset and two ship related datasets. The first dataset is used in the first contribution, in which we apply forecasting techniques to predict the amount of cars and the average speed. The other two datasets are used to estimate ship emissions, but require techniques to ensure the fitness for use of the data.

2.4.1 Floating Car Data

Floating Car Data (FCD) is a real traffic log from one week in the city of Barcelona, provided by one of the largest road-assistance companies in Spain, comprising thousands of vehicles from their fleet only in the city. The dataset comprises data collected over one week between 10/27/2014 and 11/01/2014 across the Barcelona metropolitan area. Figure 2.6 shows a heat-map of the vehicle tracking data, comprising over 890,000 data samples and a fleet of more than 100 cars moving simultaneously around the city at some times.

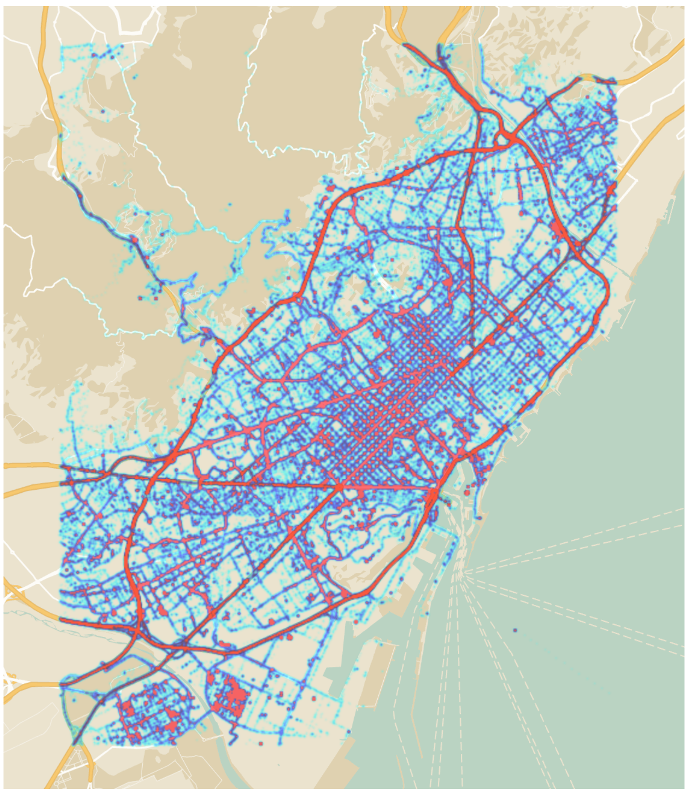


FIGURE 2.6. Barcelona metropolitan area map, combined with a heat-map overlay of the FCD dataset used for the simulations presented in this work. The dataset contains more than 890,000 data samples of road-assistance cars moving around the city.

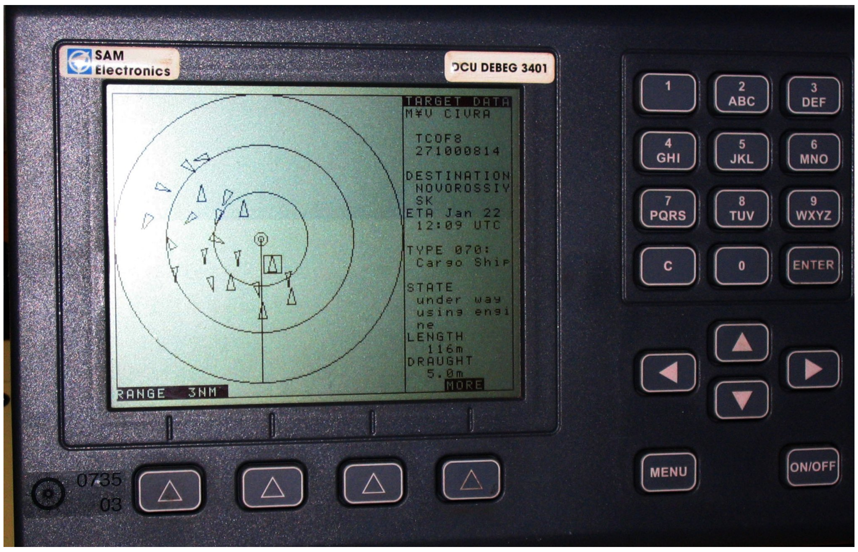


Figure 2.7. AIS Transponder showing AIS data in the range of the antenna. Image obtained from Wikipedia.

FCD represents geo-localized timestamped data of moving vehicles, collected and analyzed for various applications, including smart cities, traffic engineering, and traffic management. Typically, FCD is received through antennas deployed in the town representing a large urban zone, a localized neighborhood, a street, or a street segment, depending on which granularity is required for the specific application. That data is provided to the Edge analytics indicating the received timestamp for each vehicle transmission and its speed. Data like vehicle position, e.g. GPS, is not provided for privacy and security reasons; only the Edge node position is provided.

The FCD arrives asynchronously to our Edge nodes and is aggregated periodically into summaries of traffic information, i.e., the average speed of vehicles surrounding the node (in Km/hour) and vehicles' count, considering that vehicles will be reported once for each aggregation window time. An example of this aggregated dataset can be seen in Table 2.1. With 1-minute aggregation interval as lower bound, we aggregate the incoming data into data entries containing latitude, longitude, number of cars, speed average, and timestamp. Before performing the analytics, the Edge nodes independently collect and aggregate the FCD into a specific time interval.

latitude	longitude	n.cars	μ.speed	timestamp
41.362	2.095	4	17	1414365180

Table 2.1. Example of 1-minute aggregated FCD record. Notice that position is from the receiving antenna/Edge node, not the vehicle.

2.4.2 *Ship datasets*

AIS dataset

Automatic Identification System (AIS) is a system used by ships to prevent collisions among them and to help locate and manage all the ships by the port authorities. This system provides GPS data along with other information, making it possible to position ships and know other relevant variables like speed in near-realtime. This fact makes this data very useful for emission estimation, as we will see in Section 2.3. An example of an on-board AIS display can be seen in Figure 2.7.

The current dataset has been provided by the Spanish Ports Authority (*Puertos del Estado*), from their vessel monitoring database collecting the AIS signals from all registered ships navigating national

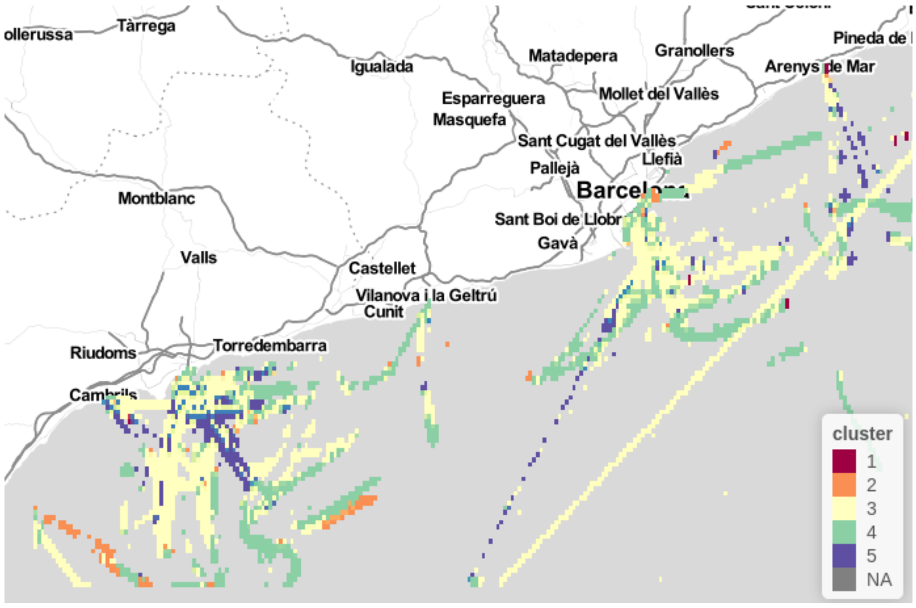


Figure 2.8. A week of AIS data from the year 2014, showing the cells that have more than 10 samples during the week. These can be understood as common routes. Cluster variable can be understood as the most common ship behavior found in that particular cell/pixel. More details regarding this figure can be found in Chapter 6.

waters. Such database collects the information periodically sent from all registered vessels and can be used by local port authorities. The dataset used for our experiments is a slice of data concerning the coastal area of Barcelona. The relevant variables of the dataset for this study will be introduced later on. In particular, the used dataset for Chapter 6 is a week worth of data from 2014 containing a total number of 1579393 samples with 19 variables.

Puertos del Estado has deployed a network of AIS base stations through the whole Spanish coast, with the dual objective of obtaining maritime traffic information (especially at the port area) and applying the AIS capabilities to navigation aid[1]. Each AIS base station is responsible for receiving the AIS data within its coverage area and sending it to the central hub for processing, storage, and subsequent distribution to other AIS networks or interested users. An example of the coverage of the coast of Catalonia can be seen in Figure 2.8.

Each vessel is identified by 1) name of the ship, 2) the IMO number, given by the IMO, 3) and the MMSI number. There are two AIS device classes (A and B) differing in transmission power and capabilities, being Class B smaller and short-ranged compared to Class A. As ships transmitting with a Class B device are not required to have an IMO number, it might be missing and marked as *Not Available* values (NAs) in the data. MMSI is used as an identifier if IMO is missing. Moreover, AIS devices are periodically transmitting *static* attributes, properties of the ship that do not change on time, e.g. length, beam, or draught, so authorities and other ships can know the size of the vessel. These last attributes are not considered for this study as they are unreliable for the current task.

From the dynamic data provided by AIS, the following subset is used in this study:

- Time-stamp of the transmission
- GPS Coordinates in latitude-longitude
- Speed over Ground (SoG): Speed of the boat, measured as effective over the ground, by taking into account the tidal drifting or speeding up/down the ship, measured in knots.

[1]http://redais2.puertos.es

- Navigation Status (navstatus): A standardized identification of the current status of the ship. This feature is manually set by the crew. This denotes the susceptibility of such features to errors and missing values.
- Type of ship and Cargo (typeofshipandcargo): A combination of two integer values, encoding the type of ship and materials that it is currently transporting.

Additionally, the AIS provides information like the ship rotation (Course over Ground (CoG)), the rotation speed, and compass heading. These features have proved to be unstable to perform accurate predictions. The information from every single vessel is collected in their navigation trace along time. Table 2.2 shows a sample from our dataset.

ID	size_{a, b, c, d}	length	beam	draught	sog	cog	rot	heading	navstatus	type	lat	lon	timestamp
1	62 , 126 , 13 , 15	188	28	7	5.50	317	127	326	0	70	40.91	2.47	2014-04-13 23:59:32
2	17 , 19 , 7 , 1	36	8	3	0.00	170	0	47	8	37	41.53	2.44	2014-04-13 23:59:31
3	4 , 16 , 4 , 2	20	6	4	10.00	220	-128	511	7	30	41.30	2.19	2014-04-13 23:59:33

TABLE 2.2. Sampled data from the dataset. Identifiers are surrogates from the real identifiers.

Cleaning and Normalizing AIS Data

Working with time series implies having data regularized in time, as many techniques interpret samples as steady and regular, more than sparse, occasional, or even redundant. When using CRBMs with time as a conditioner, each position in the *delay* (the window of data history) is supposed to be given a set of weights towards the hidden layer, then data values slide through the window facing new weights based uniquely on their position in history. This way, each position in the history window discretizes time in equal segments, so sparse data needs to be densified, and missing data must be interpolated or predicted. To do this, linear interpolation is applied to adjust data points to a regular time scale, as performed in the previously mentioned studies by Jalkanen. Even though more advanced interpolation algorithms can be used, we have chosen to follow the linear interpolation procedure described by Jalkanen so that the results are standard.

Static characteristics dataset

The second dataset contains the static properties of the vessel, e.g. main engine power, length of the ship. This dataset was obtained from the company IHS Fairplay but can be obtained from others like Lloyd's registry. This dataset shares variables in common with the previous dataset, e.g. length. However, as the data comes from a curated register it is more reliable. The original dataset contains 3819 samples, each representing a ship with over 50 variables.

This work focuses on the variables shown in Table 2.3 as those are the basic characteristics required for the basic emission estimation. In case that other methods of emission estimation are used, e.g. STEAM2 [16], the number of variables required is higher and, therefore, the amount of uncertainty increases. Further details of this data will be given in the Chapters 5 and 6.

2.5 DATA SCIENCE PIPELINES

Data Science in general implies having processes from both Data Management and Data Analytics. In a simplistic view, Data Management processes import, merge and ensure that the data is good and ready to be used. On the other side, Data Analytics objective is to extract knowledge out of the data, whether it be with classical SQL queries, OLAP, statistics or ML. Depending on the required procedures, both parts can become quite complex. Therefore, there is a need to establish a way to order and map the transformations of the data. This is where the concept of data pipeline appears. With the pipelines it is possible to easily observe where the data is coming from and which transformations are being done on

Attribute	Description	Origin
Type	Ship type	AIS/IHS
Main Engine (ME) power	Installed main engine power	IHS
Auxiliary Engine (AE) power	Installed auxiliary engine power	IHS
ME RPM	Main engine RPM	IHS
AE RPM	Auxiliary engine RPM	IHS
Eng. type	Type of engine installed (Main engine)	IHS
L	Ship length	AIS/IHS
B	Ship beam	AIS/IHS
T	Ship draught	AIS/IHS

TABLE 2.3. Variables available in the registry. Only shown the variables relevant for the contributions of this thesis.

it to fulfill the requirements of our applications. Even though data pipelines can be quite complex, in our case it can be summarized in 6 stages, as depicted in Figure 2.9.

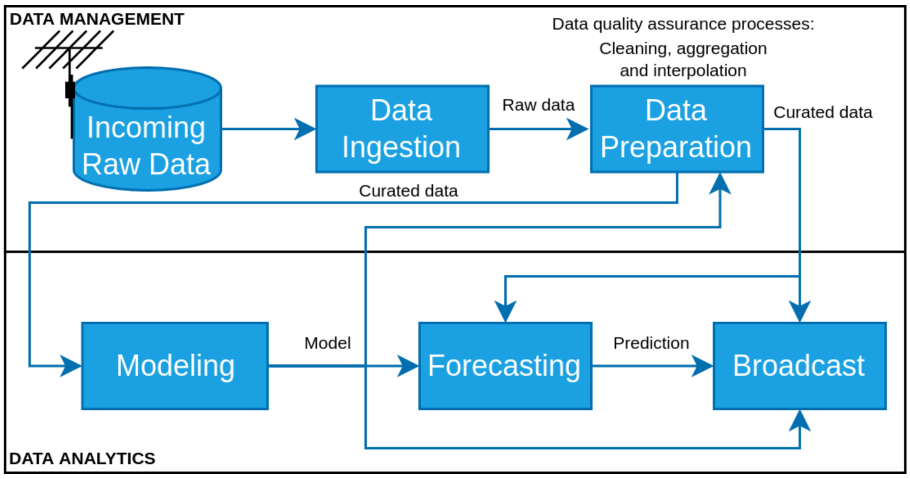


FIGURE 2.9. A general data pipeline for IoT environments with analytics on the edge.

Each stage is defined as follows:

- Incoming Data: Represents the physical parts that are required to capture data up to the point where it reaches the edge device.
- Data Ingestion: Transform the input data into a data structure that can be used by the following stages. This stage includes the integration of different data sources.
- Data Preparation: Ensures the fitness of the data to the application by cleaning, transforming, aggregating and interpolating the data and generating new features. Notice that this stage may use the models from the next stage in order to perform operations like cleaning.
- Modeling: Trains a model from the available data for a defined purpose. This model can be a multivariate statistical model or a ML model, to list some examples.
- Forecasting: Transform the incoming data using the previously trained model to obtain derived data from the original curated data, e.g. the prediction of the number of cars in 5 minutes using the current number of cars or a derived alarm if the number of cars will pass a given threshold.
- Broadcast: Propagates the aggregated data, models and predictions/alarms to the next tier in the Fog Architecture, i.e. the next aggregating level or the cloud.

As it can be seen, this pipeline is an actual simplification or subset of the complete data pipelines that we may have in traditional Cloud-based services. Notice that the data may not be completely stored in databases like data lakes as storage in edge devices is limited. The data that is processed in this pipeline is mostly dynamic. We may, however, store or maintain in memory a given time window of data for the purposes of the analytics. The uncovered parts in the IoT part of the pipeline can be covered in the Cloud level treating the incoming data as the input of the ingestion phase. In fact, this way the resource usage for processing is effectively being distributed over the Fog Architecture, therefor off-loading the Cloud and making scalability more viable, as the data producers can also be the data processors.

Regarding the contributions of this thesis, each part of them fit in different stages of this pipeline. First contribution covers the whole pipeline from end to end, providing an architectural framework to work with. Moreover, it specifically focuses on the possibility of distributing the modeling phase over different devices to build a common model. The second contribution may be understood as a data preparation process as the focus is on processing data to improve the fitness for use for the applications. The modeling phase of this process can be both trained in the Edge or in the Cloud, as the particular pieces of information required are static and usually available. Finally, the third contribution focuses on both data preparation and modeling stages, but may also be used in the Cloud level for further analysis. As this contribution is built over dynamic data, the models can be trained at the Edge level to then be used in the data preparation stage. This mechanism allows to have models that are resilient to Concept Drift if trained on-line. This means that the models would be able to handle changes in the underlying distribution of the data generator.

2.5.1 Data Preparation and Data Quality

Aside from enabling the processes to be run, this thesis has a focus on the data preparation aspect. As seen before, *data preparation* is the phase of the pipeline that is in charge of ensuring the fitness for use of data, i.e. the data quality. From the definitions found in the work of Zaveri et al. [27], it is possible to extract that data quality is a wide concept that is in fact a multidimensional construct. Each dimension is measured with both qualitative and quantitative metrics which are used to measure each potential issue on data and, in the end, provide a global score of how fit is the data for an application. Notice that there is no static or golden standard for metrics, as each application has its requirements, e.g. one application may be tolerant to missing values or even use them whether other application may lead to wrong results.

In particular in this work the process described by Saha et al. [28] data quality management process is followed:

- Discover rules: Find logical rules or statistical rules with which it is possible to find inconsistencies in the data and that might be used to repair them. In this case, we might apply data filters to find potential outliers or complete statistical or ML models.
- Check for inconsistencies: Applying the previously found rules or models, find elements that do not comply to later evaluate them. For some cases, this may require manual checking.
- Repair: With the modeling techniques used, repair the data that was inconsistent.

This thesis has a focus on the following data quality dimensions, as defined by Zaveri et al. [27]:

- Semantic accuracy: How accurate is the usage of the variables and fields, the precision of the values and annotations and the presence of outliers. In particular, the focus is set on outliers as they represent a potential set of incorrect values. Notice that the outlier detection process can be completely automated, however the process of testing whether they are erroneous or not might not be automatic if rules are not defined. This may require a human expert.

- Completeness: How much missing values, columns or even classes we have in our date. Specifically, the focus of this work is on column completeness, which refers to the amount of missing values there are per variable/column.

In summary, by improving the metrics from the relevant dimensions the data quality is improved. From now on when we refer to ensure data quality, we will speak about improving the fitness for use of the data for the applications defined.

Chapter 3

Related Work

3.1 Centralized and distributed Machine Learning on IoT infrastructures

The Cloud has been widely used to address the emerging challenges of big data analysis in many smart city ecosystems such as smart houses, smart lighting, and video surveillance [29–31]. However, IoT scenarios usually require low latency between sensors/actuators and usually there are scarce computing resources. With these restrictions, avoiding unnecessary north-south bound communication of data is desirable. In this framework, the data can be processed on the Edge or intermediate nodes, reducing the amount of information to be processed. Location awareness is also a must in several Smart Cities IoT architectures providing immediate in-place services. As IoT services in Smart Cities are being increasingly used, Cloud services alone can hardly satisfy the mentioned requirements of this ecosystem.

Several cities around the world are involved in projects towards smart-city management. Platforms designed for management of smart cities exist in cities like Nice, France, where the Connected Boulevard [32] platform has been developed to optimize all aspects of city management, including parking, traffic, street lighting, waste disposal, and environmental quality. Also in Santander, Spain, the project SmartSantander [33], focuses on a European facility for research and experimentation of architectures, technologies and applications for smart cities, but without focusing yet on Fog computing. Further, other cities like Songdo (South Korea), Masdar City (Abu Dhabi, UAE), Paredes (Portugal), Manchester (UK), Boston (US), Tianjin (China) and Singapore, announced smart-city related projects [34]. Although approaches differ on each city, resilient and secure analytics between the Edge and data centers are a hard requirement, revolving around a coherent and affordable way of management [35].

Fog computing, the paradigm combining the Edge and Cloud capabilities, can handle the significant data treatment, including acquisition, aggregation, analytics and pre-processing, while reducing transportation and storage, and even balancing computation power among intermediate nodes [36]. In previous a work [37], there was a focus on how Fog computing architectures can improve the deployment of distributed commercial solutions on smart cities where cloud models fall short. This scenario was explored through a Barcelona Supercomputing Center and Cisco Systems joint initiative towards a Fog computing deployment in the city of Barcelona. In addition, transforming this data into actionable knowledge and adapting to changing dynamics of modern cities, requires *intelligent* modeling techniques not only accurate but adaptive. ML techniques enable *smartness* in Smart Cities by modeling, predicting and extracting useful information from collected data, through advanced statistics and artificial intelligence algorithms. Deep Learning, an ML subfield based on multi-layer neuronal networks, is becoming an important tool to city-modeling challenges across many areas such as forecasting [38], self-driving research [39], image processing [40, 41], or object recognition [42], useful to manage public services, detect hazardous scenarios or to guide emergency services among others. This kind of techniques for management on cities has been already implemented in Cloud systems and could easily benefit from Edge computing: from management of power grids using machine learning in big cities from grid monitored data [43], to reduction of data transmission in health-care monitoring wearables by performing pattern mining on the Edge [44], to illustrate some examples.

A significant amount of effort and research has been devoted to tackling the challenge of training huge data sets through building large models with more parameters and parallelization or distribution methods based on the Cloud computing infrastructure. For example, Google implemented a distributed framework for training neural networks over Central Processing Unit (CPU) based on the DistBelief framework [45, 46] which makes use of both model parallelism, and data parallelism. This model has also proved useful for computer vision problems, achieving state-of-the-art performance on a computer vision benchmark with 14 millions of images. To scale up the training phase of learning, researchers utilize accelerators such as a single or cluster of GPUs [47, 48]. Recently, Facebook [49] announced achieving 90% scaling efficiency in training visual recognition model, using data parallelism combined with the use of GPUs. This is not limited to Cloud solutions. K. Hong et al. [50] proposed a fog-based opportunistic spatio-temporal event processing system to meet the latency requirement. Their system predicts future query regions for moving consumers, and starts the event processing early to make timely information available when consumers reaches the future locations. Yu et al. [51] proposed a Deep Reinforcement Learning based system that is able to share execution of tasks in Edge nodes taking into account the battery, quality of service and other details.

However, the increasing amount of data to be processed, along with the computational demands of sufficiently-accurate neural network algorithms, have led to bigger computational and memory resource requirements. Accelerating neural networks training to competitive accuracy within a sufficiently short time is a major challenge that may lead to increase computational demands. Seeking solutions that assure scalable and efficient learning has given rise to the notion of "distributed ML". Federated Learning (FL) [52] is a promising solution when both data and resources are scattered along in the architecture, with the added challenge of the near impossibility of having all data in the same place, and the cost of constantly offloading computation to the Cloud. FL aims at keeping the data near where it is produced [52, 53]. This solution can be understood as allowing the Edge devices, the clients, to produce a predictive model with their own local data, and then coordinate with a central node, the server, for model merging. In particular this is interesting in the contexts where data privacy is an issue as in the work of McMahan et al. [52], as the only data exchanged between the data producers and the central server are the weights, i.e. the configuration, of the neural network. On the other hand, there have been efforts like in the work of Hu et al. [54] that focus on having a model that works properly on both sides, client and server. Moreover, it has been proved Stochastic Gradient Descent (SGD) converges in this scenario [55], proving the suitability of Neural Networks for this particular task. This approach brings properties that are desirable for Edge Computing architectures, like the ability to keep on working without network connectivity when the system fails.

Given that at the Edge nodes computing power is scarce, works such as Marchisio et al. [56] study how to perform ML inference in ultra-low powered devices, and review the usage of Neural Networks (NNs) with this kind of device. This approach minimizes both power usage and hardware costs. Sudharsan et al. [57] proposed a methodology to train a kind of Convolutional Neural Network (CNN) and then adapt it to run in different MCUs to do prediction. Their approach reduces the size of the trained network to the 10% of the original. In the same direction, TinyML [58] enables training a NN with TensorFlow and then convert it to it can be run using TensorFlow Lite on ultra-low power MCUs. Neither of these approaches handle training on the device, but other approaches like Neuro.ZERO [59] enable training on the device by means of hardware acceleration. However, FL has yet to be covered on this kind of setup with MCUs, so that it enables to train different models and average the model configuration among nodes.

Even though the methodology per se has already been described, there still is a knowledge gap regarding the actual applicability of FL on a Fog Computing architecture using low-powered devices and the effectiveness distributed models versus a centralized one. This work aims to fill this gap applying the methodologies described in Chapter 4.

3.2 Floating Car Data

The appearance of Floating Car Data (FCD) as data source is expected to provide support to many practical use cases in the near future, leveraging Intelligent Transportation Systems telemetry. To complement the current lack of sensorization in cars and communications infrastructure, works like Briante et al. [60] propose the use of smartphones and Wi-Fi hotspots, also Ali et al. [61] proposes crowdsourcing architectures to collect data from smart devices on vehicles for these same purposes, or Ancona et al. [62] studies vehicle-to-vehicle networks to handle the expected escalation of FCD data volume. In the field of treating Floating Car Data, we find works like Yu et al. [63] where the framework RTIC-C presents a high level architecture to deploy traffic analytics, using Map-Reduce approaches for distributing modeling and processing algorithms. The RTIC-C authors defend the use of big data analytics on traffic data due to its increasing volume and complexity, then they focus on distributing received data for processing on anomaly detection and traffic trend prediction. Works like Moretti et al. [64] and Xia et al. [65] present different traffic modeling approaches, ensembles using bagging and Feed-Forward MLP Neural Networks the prior and Spatial-Temporal Weighted k-Nearest Neighbor the later, producing general models from the aggregated datasets and distributing computation on the Cloud. Also, works like Lv et al. [66] present a methodology where a Stack of AutoEncoders is applied for traffic flow prediction at different granularities with good results for t+1 forecasting. As presented in our work, localized models on the Edge can create in-situ specialized predictors adapted to their coverage area, using re-trainable machine learning models.

Chapter 4 focuses on the data transmission architecture towards receiving traffic data streams properly for being ingested by analytics methods, i.e. localized re-trainable CRBM prediction mechanisms, and could complement high level frameworks like those named here on generalist model scenarios. All the analytic process done is mainly done in the Edge, using dedicated High Performance Computing (HPC) only to process results. To the best of our knowledge, there currently is no evaluation of this kind of problem with FL using RNNs with server-class hardware and low-powered devices. Moreover, mechanisms are needed to stop training as soon as a reliable-enough model is obtained. We believe that FL distributed learning can be highly beneficial for data analytics over scenarios like smart cities.

3.3 Ship Characteristics and Emission Estimation

Both AIS and ship characteristics data are a source for potential applications in industry and academic research as shown in the work of Svanberg et al.[67] and Miliuše et al.[68]. There is also an interest on standardizing the data access for interoperability. Lensu and Goerlandt[69] established a framework to build an AIS database for research, and apply it in particular to a use case related to navigation in ice. AIS-assisted emission estimations can be effectively used to assist policy design and corrective measures of a specific shipping sector (e.g. cruises and ferries) [70] and to improve the efficiency of ships [71]. Jalkanen et al. [1, 72] show that AIS data can been used for the estimation of high spatial and temporal resolution maritime emissions. Compared to traditional emission estimation methodologies, the use of AIS data provides information of instantaneous speed, position and navigation status of vessels and subsequently allows for more accurate estimations of vessels' activities and the improved reliability of emissions and fuel consumption estimations [73]. Navigational status is included on AIS data and with this attribute the current engine usage can be estimated along with other attributes like speed, however in some cases it is incorrectly set as this attribute is manually set.

Both sources of data may have missing or inconsistent data, which can make the derived applications unreliable. Harati et al.[18] identify that attributes reported by AIS like *draught* are prone to error due to human errors. In their study, they conclude that 1 out of 14 messages is wrong. Moreover, Goldsworthy and Goldsworthy[19] highlight that there may also be errors on the onboard equipment, e.g. unrealistic high speed reported. This fact is also shown in the work of Miola and Ciuffo[74] where they find speeds higher than 20 m/s inside the port of Rotterdam, which is not reasonable. On the other hand, ship registers may not have the full information about a ship. Peng et al.[3] claim that missing data is one of the main problems when estimating exhaust emissions as some information may not be recorded. Huang et al.[4] report that in their case out of 90,000 ships only 50,000 had correct values for rated power and design speed. Both of these attributes are essential to estimate exhaust emissions using this kind of methodology as they are used to estimate the current ship power.

3.3.1 Other approaches to emission estimation using Machine Learning and Statistics

In this case scenario where the data might be incorrect, statistics and ML provide a good solution. Some approaches try to directly estimate the emissions using ML without any physical model, e.g. STEAM [1], behind it. Fletcher et al.[75] propose an ML method to predict the exhaust emissions of 2 ships using as input data the Shaft Speed (RPM) and the Engine Power (kW) in time. However, this solution requires sensors in the engine. Finally, they propose to mix this methodology with STEAM [1] to improve emission location as future work. Si et al.[76] also predict the emissions produced by one cogeneration unit using several engine-specific variables, which are not commonly found in ship registers.

A different approach to this problem was described in the study by Peng et al.[3]. The method, based on STEAM [1], deals with missing data by sampling and extrapolating. The ships that have complete data are selected and then from those a stratified sampling methodology is followed. In the paper, they show that with 10% of the ships they have enough data to estimate the total emissions with statistical extrapolation. It is also shown that using high temporal granularity produces a more accurate estimation of the pollution as the model can catch the speed variation at each time step. However, this approach loses spatial granularity as the emissions are estimated by areas and not all the AIS traces are used.

Other approaches try to estimate engine-specific characteristics intended to help with the design of the engines and that may be used to estimate the emissions. Chan and Chin [77] predict the engine power performance using the shaft power, shaft rpm, and shaft torque. Noor et al.[78] models predicting engine performance, i.e. output torque, brake power, brake specific fuel, and exhaust gas, using engine speed, engine load, fuel flow rate, and air mass flow rate. Fuel consumption in container ships is a well-studied topic. Le et al.[79] predicted the fuel consumption of this attribute using variables like sailing time and average trip speed using neural networks. Also, Uyanık et al.[80] did a similar study using noon reports, logbooks, and sensors in the vessels. This fuel consumption can be used later to estimate the exhaust emissions but this approach lacks GPS precision. Liang et al.[81] studied the prediction of propulsion power with neural networks using transient speed, weather, and other ship attributes and later on added AIS data to improve the prediction positioning. This approach provided better results overall compared to a physical model for container ships.

3.3.2 Data Quality and Pattern Mining for AIS Data

AIS traces are GPS-based, therefore there are many techniques that can be applied to study the movement of ships. For example, pattern mining for GPS traces is a common practice in very different fields, looking for specific patterns in movement and behavior. Works like Qiu et al. [82] describe a methodology for mining patterns through Hidden Markov Models, producing semantic information to feed frequent pattern mining methods. Such work is also based on discovery of frequent episodes in time series [83], with the goal of discovering patterns series of events. Use cases for such techniques are social mining and recommendation [84], animal movement patterns [85], or elder care [86]. In Chapter 6 we propose to find common patterns using CRBMs as a base for time windows, feeding them from GPS and other input sources, for discovering discriminating behaviors on a geographical space. The CRBMs, as probabilistic models derived from RBMs [87, 88], are used in a wide range of problems like classification, collaborative filtering or modeling of motion capture, developed by the team of professor Geoffrey E. Hinton at the University of Toronto [24, 89–91]. Such models are usually applied for problems where time becomes a condition on data, i.e. time-series. Other works like X.Li et al. [92] and Lee et al. [93] use the models for multi-label learning and classification. Based on their experiences and techniques, we are taking advantage on CRBMs time-series learning capabilities.

These analysis require that AIS data has good quality, however this is not always the case. There are mainly two problems with this data: time gaps and outliers. Usually, time gaps are filled using interpolation techniques [1, 19]. In the case of outliers, Goldsworthy and Goldsworthy[19] propose to mark as outliers those values that have values bigger than the 50% of the average of the trace. In the case of finding an outlier, it is removed and then the point is linearly interpolated. Velasco and Lazikis[94] performed an extensive literature review on missing data imputation of sensor time-series data in vessels. In the study, they compared classic univariate autoregressive methods with multivariate methods for different output variables, e.g. current engine power. They found out that the Auto-Regressive Integrated Moving Average (ARIMA) outperformed the rest, showing low Mean Squared Error (MSE). However, this study has its limitations, as there is no time window provided for the multivariate methods, hence missing the time correlation of the data. Jeon et al.[2] proposed a preprocessing pipeline for AIS and onboard equipment data to find and correct anomalies and denoise the data. After that process, a dimension reduction is performed so that the regression algorithms are used over variables without any colinearity problem. To predict the engine performance, three different models (Generalized Additive Model (GAM), Gradient Boosting Regressor (GBR), and Multivariate Adaptive Regression Splines (MARS)) are trained with the best hyperparameters found using Bayesian optimization and used in an ensemble weighting the predictions with learned coefficients from a trained linear regression. They show that this pipeline works well for AIS data and particularly that the classic technique Kalman Filter works well for denoising this data.

3.3.3 *Data Quality in ship register data*

In the report of the emission estimation used by National Atmospheric Emissions Inventory (NAEI) from the United Kingdom [95] is shown that some static data attributes, i.e. ship register data, provided by registers are not reliable, e.g. the auxiliary engine is missing in half of the ships in their study. When the data is missing a linear regression of the attributes is done, analogously to what the IMO shows in their studies [96, 97].

Johansson et al.[26] propose in the STEAM3 model to use the Most Similar Vessel (MSV) search method, which is a k Nearest Neighbors (KNN) with k=1 with a custom distance function. This method requires knowing at least the length, the design speed and the ship type to be used. In case they are not available they use a crawler to get this information. Design speed can be estimated as the 95th percentile of AIS Speed to avoid outliers. Finally, if the required information is not found, a small boat of 300 GT units is assumed. Instead of doing this assumption, Chapter 6 shows that it is possible to use a neural network plus a clustering method to extract patterns from the AIS traces and use them along the ship type to predict the main engine power. This technique may be also applied to other attributes and it will be shown in the next chapter of this thesis. Liu et al.[98] propose to use statistical and ML methods to regress the missing data using the other data available. Gradient Boosted Regression Trees are finally selected because of their performance, however, there is no detail on how the process is taking place nor results. Huang et al.[4] regress the main engine power for container and tanker ships using polynomial regression using as predictive variable the length times the breadth of the ship. To regress the design speed they use the method described by Cepowski [99]. This method is based on extensive work with regressions for the same type of ship.

As shown, there are several approaches to improve the fitness for use of this kind of data, but there is no comparison nor any work that studies the relationship of the variables at a level that can be used to correct the data.

Part II

CONTRIBUTIONS

Chapter 4

Distributed Machine Learning on Fog Computing systems

4.1 Introduction

In IoT and smart cities environments, data is produced by sensors scattered in the environment. Cloud provides extremely scalable resources from a remote data center. In contrast, in IoT scenarios, e.g., smart cities, domotics, or e-healthcare, proximity, and quick reaction are required while generating big amounts of data. As the number of sensors grows, the requirements for Cloud computing power and network bandwidth increase drastically. Cloud architectures scalability of resources that can be dynamically managed, however, there is a limit on responsiveness. The more sensors are installed in a city or environment, the more network bandwidth will be required. If a continuous analysis of the sensors is required, a constant stream of data would reach the Cloud for each sensor, overflowing the network of the system and reducing its responsiveness. Given these issues, *Fog* Computing architectures have emerged as an alternative for this type of environment.

Fog Computing architectures, also called the *Edge-Cloud continuum*, make use of all the computing devices that are available in a network, instead of off-loading every process to the Cloud. This kind of architecture is the natural evolution from the classical IoT paradigm, which includes net-enabled sensors into a network so that they can communicate their readings and expose actuators. This layer, from now on called *Edge* is the place where the data is regularly produced. As mentioned before, the data has to be sent to the Cloud for further processing if a classical Cloud approach is used. However, using *Fog* Computing, this data can be processed and consumed in any intermediate point between the *Edge* and the Cloud, without requiring to send raw data directly to a Cloud.

This architecture is represented by Figure 4.1, where the 3 levels of *Fog* Computing can be observed. The first level is the *Edge* where only low-powered devices are generally found, then the intermediate levels where mid-end computing nodes can be found, and finally the Cloud, where the resources are highly scalable.

With this paradigm, critical components that require to be active at any time can remain independent of network and infrastructure failures [100] if the required analytics are pushed to the device. Specifically, Oil & Gas [101], power grid systems [102], smart cities, smart industries, and IoT applications [103]. In these environments, local analytics are required as they need a low latency QoS [104]. Given the importance of exploiting the data at the *Edge* level, considerable research effort was devoted to establishing a common framework to cope systematically and effectively with the restrictions proposed by this kind of environment [105]. Furthermore, some applications do not require to have fine-grain metrics from the sensors if alarms are programmed. An analytic process that constantly checks and aggregates the data could be deployed at the *Edge* level. This process can raise alarms while analyzing the fine grain data and only send to the Cloud those alarms and aggregations, saving network and storage usage in the Cloud.

Machine Learning (ML) is a field that provides a big part of the analytical functions that are being currently implemented for data analysis, e.g. time series prediction, object recognition, or hardware failure time estimation. Given the compute-intensive nature of training ML models, so far this process has been pushed to the Cloud. This strategy has the advantage of using powerful computing machines, however, it has several drawbacks: it adds a cost of additional network dependency, increases latency,

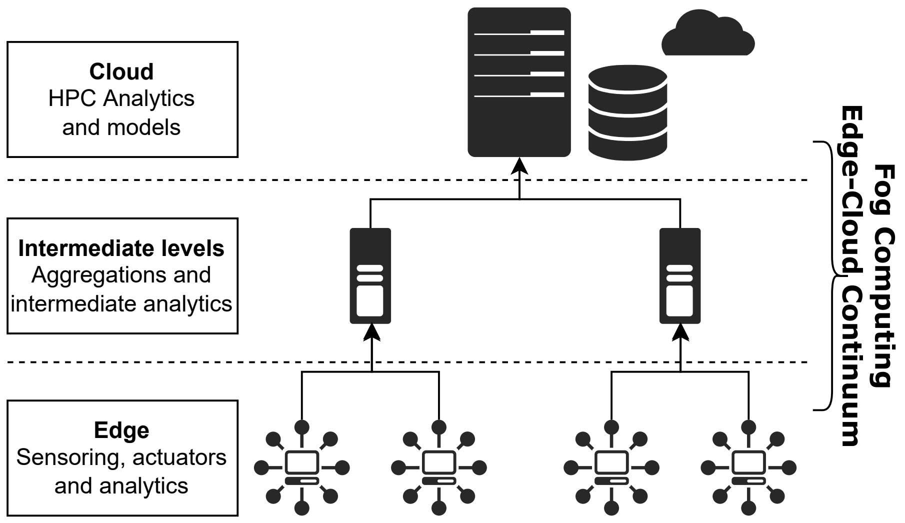


FIGURE 4.1. Schema of a *Fog* Architecture. The *Edge-Cloud continuum*, from *Edge* to Cloud.

and moves the processing away from the data producers, as defined before in the general IoT case. In contrast, using limited computing power available at *Fog* nodes is interesting for training ML models efficiently. Recent work shows the importance of training an ML model on *Edge* infrastructure. For example, Plastiras et al. [104] show the importance of doing the computation for training Deep Learning models on *Fog* nodes for Computer Vision tasks like object detection. The authors remark on the importance of privacy, performance, latency, and power efficiency in this kind of application, which can be a perfect fit for *Edge* and *Fog* Computing. In particular, this work focuses on using models for inference, i.e., applying the model to the input data. However, training on the *Edge* is also possible and desirable to enable mechanisms that protect the model from degrading, e.g. Concept Drift [106]. Considering the computing power available, models must be light enough to be able to be trained in a sensible time.

In the particular case of this thesis, we require a system that is able to gather data from transportation means sensors and use it for analysis and prediction. This would provide a mechanism for city councils and policymakers to study the traffic and gather information about what is happening at that precise moment in the city. One of the requirements of such a system is that it is a near-realtime system. With *Fog* computing, we are able to receive the data and process it at the same time, reducing the latency between receiving and having a result ready. In this way, we avoid sending raw data, and instead, we can send the already processed data aggregated, lowering bandwidth usage and off-loading all the processes to small and power-efficient devices. Furthermore, adding more devices to the network does not saturate Cloud Resources as installing a new device generally implies installing a computing node in the *Edge*, as it is required to send the gathered metrics.

In our previous works (Perez et al. [100], Gutierrez-Torre et al. [21]) we explored the idea of training light enough models in *Edge* devices and the difference between using local models and centralized models. These centralized models were trained using all the data in the central node [100] and using Federated Learning (FL) [21]. This last approach is based on the idea that models can be trained in each *Edge* node and then merged in a central node without transferring the data. In this chapter, we extend those works using FL in a distributed *Fog* Computing environment. The previous approach [21] was limited to only one FL round, i.e., training at the *Edge* and then merging but not retraining the merged model again. Moreover, it did not provide a common initialization for all the models. This limits the accuracy when increasing the number of machines or data splits. This idea is tested in a similar setup exploring further the hyper-parameter space. It also includes a comparison of Xeon-based server and two low-powered devices, i.e., Raspberry Pi 3 and Jetson Nano, to simulate real *Edge* devices for

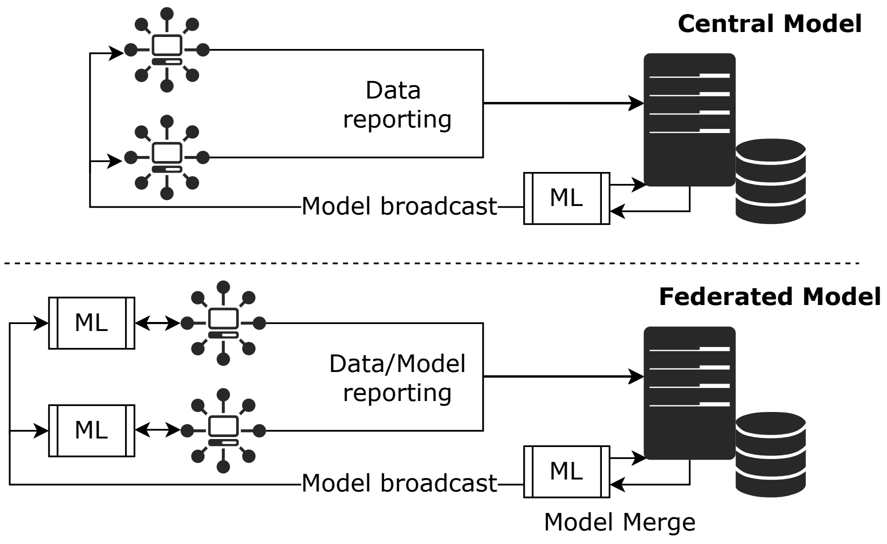


FIGURE 4.2. Federated model compared to having a centralized model and then broadcasting it.

training. These two low-powered devices are used to simulate a real *Edge* node where computational resources available are scarce. The experiments related to this are also contained in Gutierrez-Torre et al. [21], but are included in this chapter for the sake of completeness.

We present an extension of the system proposed in Gutierrez-Torre et al. [21] to automatically distribute the time-consuming task of training deep learning models on a *Fog* computing network consisting of low-powered and resource-constrained computing devices. The proposed approach is based on FL, which leverages the work of McMahan et al. [52] and Bonawitz et al. [53].

The framework enables training models in any level of the *Fog* computing architecture. In particular, we focus on making use of the *Edge* devices for training. We extensively evaluate the proposed system using the FCD dataset, designed for city-wide traffic modeling and prediction running on the *Fog* computing paradigm. The methodology can make use of any kind of NN by distributing the training on *Edge* devices. In particular, GRUs are selected as the model of choice to produce short/medium-term traffic predictions using the FL approach. Our evaluation investigates the effect of the FL rounds on accuracy and time requirements. The evaluation is based on real traffic logs from one week of Floating Car Data (FCD) in Barcelona. The data was provided by one of the largest road-assistance companies in Spain and comprises thousands of vehicles. The approach was tested in a Smart City environment, but it can be applied to any other field that requires distributing ML. This is especially interesting for the healthcare industry, as the collected patient data does not have to leave the *Edge* device or the healthcare center premises to be used in an anonymous and secure way.

The experimental results show that the models produced using a FL approach are similar to the models using a classical central training approach. Also, they show the relationship of the hyper-parameters and the error measure with a FL approach. Moreover, using this kind of light model architecture is fitting to be used with low-powered computers.

In summary, this contribution expands a previous contribution [21] that showed the viability of using GRU with one round of FL. The list of contributions is the following:

1. A system for distributed modeling for city-wide applications using the *Fog* computing paradigm for predictive analytics using low-powered and resource-constrained devices with Federated Learning (FL).

2. Adaptation of the Data Science pipeline to a use case with a *Fog* Computing architecture.

3. An exploratory analysis of FL over a real Floating Car Data (FCD) and guidelines on how to select relevant parameters, e.g. number of FL rounds.

4. Evaluation and comparison of time required to model the deep neural network using the proposed solution on *Fog* (low-power and resource-constrained) vs. Cloud (high-performance) environments.

5. A comparative analysis of resource usage vs. accuracy on training models for real Floating Car Data (FCD) compared with existing baseline methods.

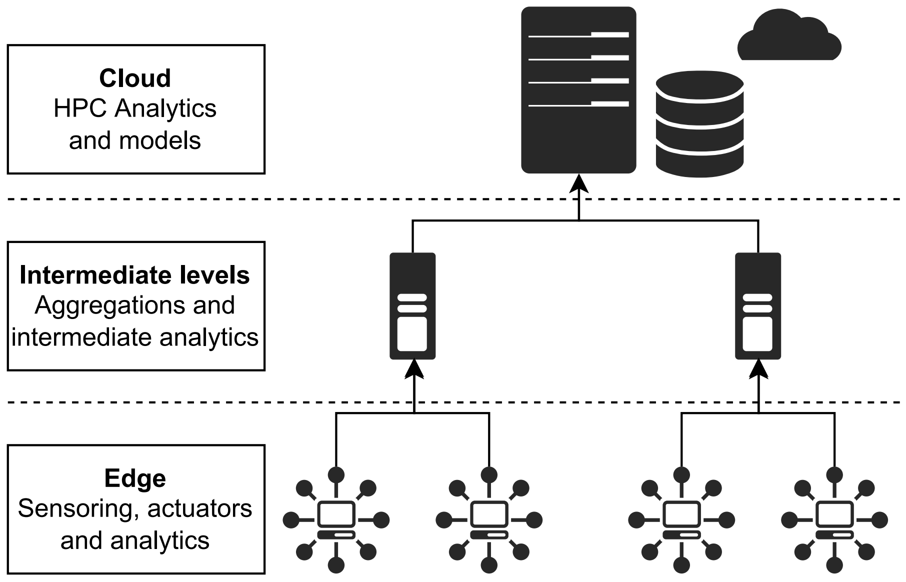
The document is organized as follows: Section 4.2 introduces the *Fog* Computing architecture and how the Data Science pipeline fits in this kind of approach. Section 4.3 introduces the methodology, covering the model used Gated Recurrent Unit (GRU) networks and how to distribute the training using Federated Learning (FL). Section 4.4 covers the experimental results. Finally, Section 4.5 presents the conclusions and limitations.

4.2 ARCHITECTURE

In this section, the basic concepts of Fog computing are introduced and then applied to the actual use case. After that, the data analytic pipeline is described along with general ideas regarding the data processing in this architecture.

4.2.1 Fog computing: Edge-Cloud continuum

Fog computing is composed of three different layers: Edge, intermediate, and Cloud. The first layer is composed of Edge computing devices that collect data and have enough computing power to process them for simple analytics. The intermediate layer is composed of mid-range computers that can aggregate data from different Edge devices and perform more powerful analytics. Finally, Cloud is the layer where resources can be easily obtained and, therefore, it is possible to perform heavy analytics and other computing-intensive processes. These layers can be seen in the diagram from Figure 4.3.



Cloud
HPC Analytics and models
Intermediate levels
Aggregations and intermediate analytics
Edge
Sensoring, actuators and analytics

FIGURE 4.3. Schema of the Fog Infrastructure, from Edge to Cloud.

Fog computing extends the Cloud paradigm [107], allowing computation in any step of the computing pipeline. This is also called Edge-Cloud continuum to represent the possibility of doing the computation at any point between the Edge and Cloud. This is especially interesting for use cases that require that the sensors and actuators located in the Edge remain autonomous if there is any connectivity issue [100] and when we build near-realtime systems [21].

In our case, we require a system that is able to perform analytics as soon as the data is produced. These analytics ranges from ML to predict future outcomes from a sensor to applying physical models to estimate the pollution produced by the transportation means. If this requirement of low-latency on analytics is fulfilled, it is called *near realtime system*. This kind of system produces the results almost instantaneously so that the users can see the system data and derived information (e.g., the pollution produced in the city by cars) with a small delay. With this kind of system in a Smart City environment enables the city council or the policymakers to evaluate the status of the city and check if a short-term policy is correctly working and its effects. Moreover, having almost immediate information can be used to detect issues that may arise. Compared to a classical Cloud system, the advantage is that the system produces the derived result as soon as it receives the data, whereas in a Cloud system the data would take time to be moved to the Cloud and be processed [108]. The more devices there are producing data, the more computing power would be required in the could. Moreover, it also requires scaling in network capabilities as more data would be required to be transmitted. On the other hand, in a Fog computing system the Edge device would always be present, so leveraging this device comes for free as most of the times it is already part of the system and with this, we do not require to scale up the Cloud. Also, in case the raw data is not required in the Cloud, the network requirements fall drastically as the Edge layer can perform aggregations, and also analytics can be sent whenever they are required, e.g., sending an alarm after the analytics result overpasses a given threshold. Notice that in scenarios with a low amount of data per node, ML may underperform if a critical mass of useful examples is not met or reaching it takes a considerable amount of time. In these cases, it is better to rely on the Cloud or intermediate nodes. Other scenarios where data is continuously being produced benefit from the Fog computing approach.

Another aspect of analytics is the trade-off of making use of online analytics versus using offline analytics. While on off-line ML methodologies we require a training dataset to be built, on online ML the model is continuously being trained with each arriving data point. Having an online learning system overcomes the problem of *Concept Drift*, i.e., changes in the underlying producing mechanism of the data, which makes the models more robust to changes. However, a hybrid approach with updating policies is more stable.

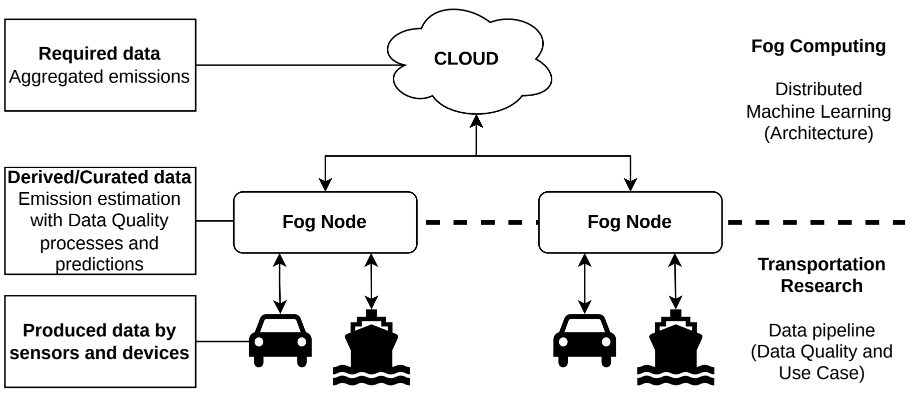

Required data
Aggregated emissions
CLOUD
Fog Computing
Distributed Machine Learning (Architecture)
Derived/Curated data
Emission estimation with Data Quality processes and predictions
Fog Node
Fog Node
Transportation Research
Produced data by sensors and devices
Data pipeline (Data Quality and Use Case)

Figure 4.4. Schema of the Fog Infrastructure applied to transportation research.

In this work, we want to focus on applying ML at the Edge computing layer. If the analytics required in the system are computed sensor by sensor and do not require information about neighboring sensors, they can be performed in the Edge. On the contrary, if the analysis requires neighboring information, this computation should be moved to an intermediate node that aggregates that information. In the particular case of this thesis, all the analytics performed are focused on samples from the same sensor, therefore we focus on pushing all the computation to the Edge layer. Removing the intermediate layer would result in the simplified architecture depicted in Figure 4.4. The Fog nodes are the Edge devices available near where the data is produced. These devices are in charge of receiving the data from sensors and performing predictions, as will be seen in this chapter, and processes to improve the fitness of the data for the application, i.e., ensure *Data Quality*, and estimate the pollution produced by the transportation means, as will be seen in further chapters.

Usually, the Edge devices or Fog nodes are meant to be low-power, therefore the computing capacities available are scarce. To simulate this, we include in our experiments a Raspberry Pi 3B (ARM processor) and a Jetson Nano (ARM processor + GPU) [109] micro-computers to evaluate how the ML performs in this kind of device and to compare the difference between using CPU or GPU to perform the analytics.

4.2.2 *Data pipeline*

As mentioned in previous sections, data can be processed in any tier of the Fog computing architecture. Therefore, the whole data processing pipeline can be distributed over the infrastructure, i.e. the Data Science pipeline over the infrastructure. In this particular case, we focus on pushing all the computation to the Edge level, so that there is no computation required from upper levels aside from aggregations and models over those aggregations. By doing so, it is possible to offer the end results directly in the Cloud without further processing, enabling near-realtime visualizations of the complete system. Part of the processing could be done in intermediate nodes, however for the sake of simplicity only Edge devices are used.

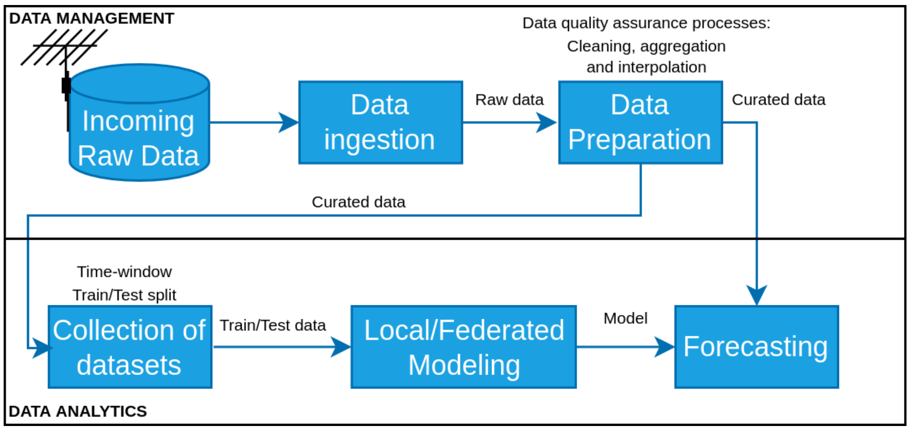


Figure 4.5. Data pipeline including the management and the analytics parts. The data wrangling processes include data cleaning, aggregation, and interpolation. These processes can be performed by classical techniques or using ML models, specially in the data cleaning process as will be seen in further chapters. This figure is an implementation of a subset the Figure 2.9 modules.

Figure 4.5 represents the processing done for each data point. Here, the data is ingested from the signals received as incoming raw data and then loaded into the system. This system represents a purely stream system in which data is processed as it comes. After being received, it is cleaned, interpolated and aggregated to the desired granularity. With this we start two processes: a collection of the data in a time window to be used for training (model update) and the forecasting with the current model. Notice that for this work we depart from a dataset and simulate the streaming.

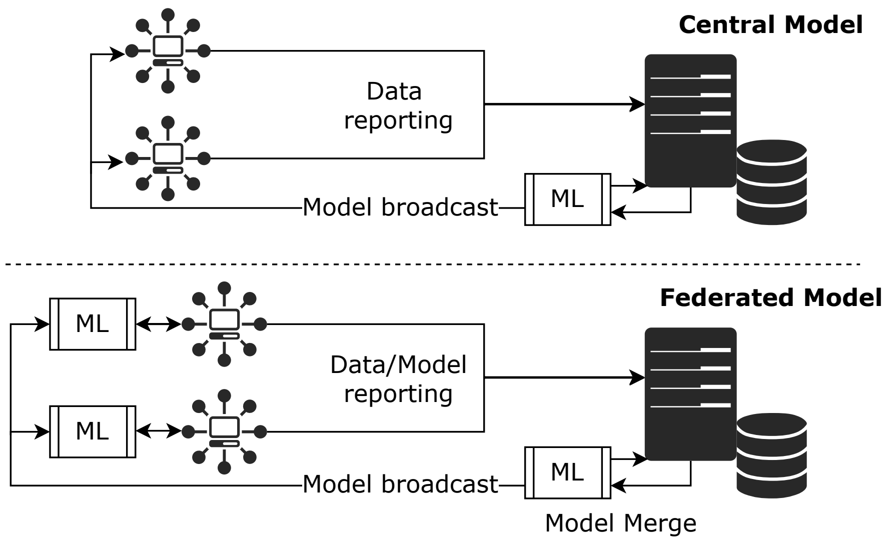


Figure 4.6. Federated model compared to having a centralized model and then broadcasting it. Notice that it is mandatory to broadcast the model using the FL approach, however, data broadcasting is optional, i.e. it is not necessary for FL to work.

4.3 Methodology

In this section, we cover the methodology used to perform FCD time-series forecasting and distributed modeling on the Edge.

4.3.1 GRU for traffic prediction

This problem targets the prediction of two variables: the number of cars and the average speed. Using the previous d elements, i.e. d is the *delay*, we try to predict from $t + 1$ up to $t + N$, where N is the size of the testing data (approximately 1 day). Because the series have been aggregated the forecast goal is to predict the next aggregated period of traffic.

GRUs are used given their capabilities to do medium-term forecasting. This is possible because GRUs are *generative* and can generate predictions by using their last prediction and status as input/memory for the next prediction. This means that we can use the predictions as input for the next steps in forecasting.

In the case of this work, we are using a model composed of one layer of GRU and one fully connected layer. This model has the focus to be as easily trained as possible by low-powered computers while getting the best accuracy possible.

4.3.2 Distributed Machine Learning: Federated Learning

Federated Learning (FL) is a way to distribute ML proposed by McMahan et al. [52] which focuses on training local neural network models and then merging them centrally. This idea matches with the Fog computing paradigm, as in FL only the local data is used to train each model, and then the model, i.e. the neural network weights, are sent to the central coordinating node. In this sense, none of the data is required to travel through the network and, therefore, network bandwidth is saved. Notice that there is the possibility to also apply this kind of technique with other types of models, e.g. Random Forest [110].

This kind of modeling framework is especially good when computing nodes are required to be as autonomous as possible as shown in our previous work (Perez et al. [100]). In that case, it was seen that having local models performed better than sending all the data to the central node and training globally when network connectivity issues happen. In this sense, we go a step further and extend this idea of training local models with local synchronization, as proposed in Gutierrez et al. [111]. In this work, models were trained locally and then averaged to obtain a synchronized model. This way the nodes can both stay autonomous but synchronized whenever it is possible and the data remains where it is produced. This makes the framework privacy compliant and also saves bandwidth, which is desirable for Fog computing systems. Notice that in the cases where local models perform better than centralized models, it is possible to train Graph NN that can take these considerations into account.

The FL can be seen in Figure 4.6 and it works as follows:

1. Initialize model and distribute it to the Edge nodes
2. Train the model with the required epochs and configuration
3. Send the models back to the central node
4. Average the weights of the models
5. If more FL rounds are required, go back to step 2

By using this procedure what is obtained in the end is a central model that has been trained with all the data available. This data has not left the Edge nodes and the training process has happened completely in a distributed way. As mentioned before, this has the advantage of the nodes being able to adapt to network connectivity issues, respond with less latency and still provide a global model. There are different approaches for model merging, however, we will focus only on *FedAvg* without weighting, which consists in doing the average of the weights without taking into account the number of samples as the samples of each node will be of the same size. This merging process requires that all the models have the same structure, so the weights for each neuron can be assigned to their corresponding place, as represented in Figure 4.7.

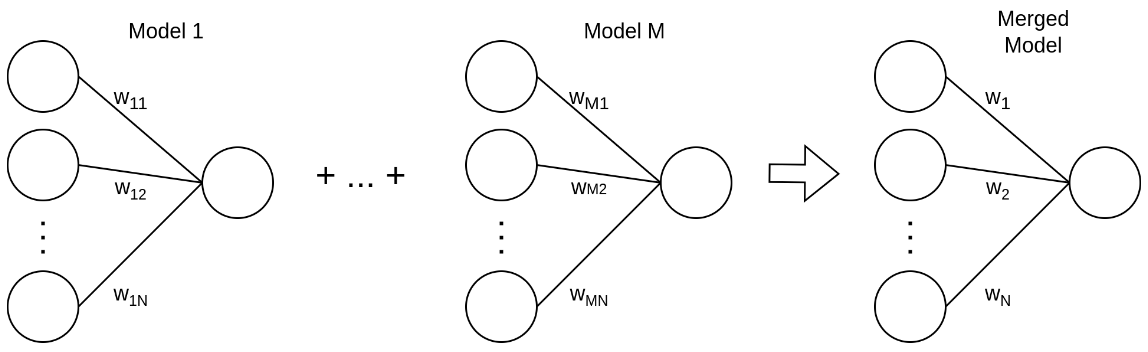


FIGURE 4.7. Example of merging weights in a single neuron model. All the weights are averaged taking into account the network structure. For example $w_1 = (\sum_{i=1}^{M} w_{1i})/M$.

On the other hand, the data available at each node may not come from the same producing mechanism, i.e. it can be non-Independent and Identically Distributed (IID). In the work from McMahan et al. [52] the IID vs non-IID is tested. In their results, it is seen that non-IID lead to converging to less accurate models but really close to the model it would be obtained from central training. On the other hand, Linardos et al. [112] show that this effect is completely dataset dependent, as in their work they show that in the non-IID cases the FL model can perform better than the centrally trained one.

In this regard, the work of Izmailov et al. [113] follows the same line of thought. In particular, they show that using what they define as Stochastic Weight Averaging (SWA) it is possible to obtain models that generalize better. This technique is based on merging the final model with models generated during the training (checkpoints). In this sense, FL follows a similar approach as we are merging models that have a common structure but are being trained with different data.

In summary, Federated Learning (FL) provides a framework to build models autonomously and in a distributed way that can be synchronized. It also provides low latency as models are always being trained locally and, therefore, can react to new situations if needed. It also provides low bandwidth usage, as only models are required to be transferred and, generally, they should not be big. In our previous work [111] the model initialization and the FL rounds effect was not taken into consideration, therefore this work will cover it.

4.4 Experiments

This section explores the fitness of GRU for traffic prediction comparing it with other methods, the effect of the model synchronization (federated training rounds), the learning rate and the common initialization of models, and the effectiveness of the approach in low-powered computing devices.

4.4.1 *Comparison with baseline methods*

First, we provide a comparison of the proposed learning method with a previously used method [100] and a classic time series model used for estimating the traffic data. We compared our solution with Vector Auto-Regression (VAR), a classic time-series analysis method, and CRBMs.

As we can see in Table 4.1, the proposed solution based on GRU outperforms VAR and provided comparable performance with CRBM when the granularity is set to 5 minutes. As it can be seen, GRU is better at predicting the number of cars, but worse when predicting the speed. In particular, Figure 4.8 shows GRU is slightly better than CRBM when granularity is finer, compared to the results from Gutierrez-Torre et al. [21]. Both kinds of neural network perform well in our framework. However, due to our particular interest in finer granularity, GRU is the chosen method for this work, as we are focused on 10-minute granularity. For other experiments with different data sets, both methods should be compared in order to select the final model.

Method	RMSE	
	Cars	**Speed**
VAR	4.99	7.44
CRBM	2.03	5.68
GRU	1.76	6.17

Table 4.1. Comparison with baseline models VAR and CRBMs as N = 1 (5 min aggregation). Notice that the results are for the overall best configurations found.

4.4.2 *Single versus distributed models*

With the following experiment, we explore the differences between training in the classical way (one single model) versus using an Federated Learning (FL) framework. In particular, we focus on the effects of the federated rounds, i.e., how many times the models are synchronized, and the effect of the learning rate with respect to the number of nodes used, i.e., the number of models to federate.

The hyper-parameter grid was selected taking into account the hyper-parameters that worked best in Gutierrez-Torre et al. [21] and extending it to the parameters that were not available. In particular the hyper-parameters are the following:

- **Number of nodes**: Number of models to federate. *Values*: 1, 2, 3.

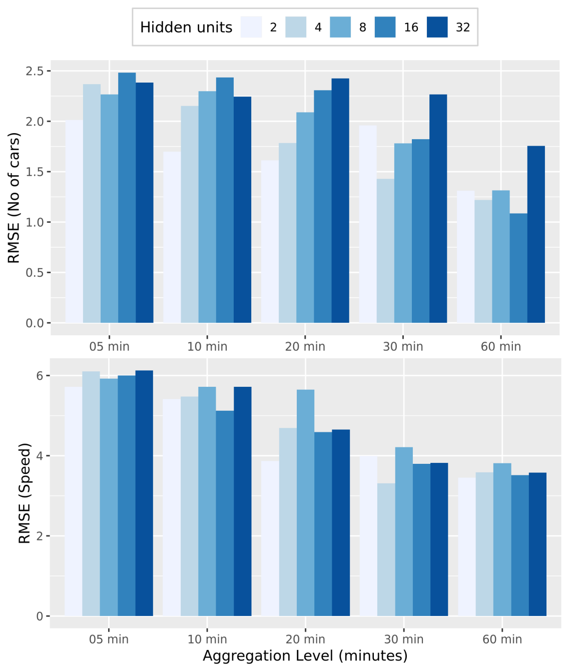


FIGURE 4.8. Error vs. Hidden units vs. Time aggregation for the number of cars and speed estimation on CRBMs.

- **Hidden units**: Number of GRU units that the GRU layer has. *Values*: 4, 6, 8.
- **Batch size**: Number of elements to process for each gradient update. *Values*: 4, 32, 512.
- **Epochs/FL rounds**: Combination of number of times that the dataset is reviewed by the network and how many times the whole FL training is done. As these two parameters are related, their values are selected together. In case the number of nodes is 1, a number of epochs is 94 used [21]. *Values* (e, r): (94, 5), (31, 15), (1, 100).
- **Learning rate**: How much the NN weights have to be updated for each step initially. In the experiments, RMSProp is used as the optimizer, which has an adaptive learning rate mechanism. Notice that it will reset after each FL round. *Values*: 0.01, 0.001, 0.0001.

Model evolution in Federated Learning rounds

First, we compare the effect of the number of FL synchronization rounds and, consequently, the number of epochs with how many models we have to train in parallel. In order to do a fair comparison, instead of adding new data for each new federated model, we split the total data we have into N pieces so that all combinations of N use the same data, therefore making the experiments comparable.

Figure 4.9 upper part shows the average error of all hyper-parameter configurations grouped by the number of nodes and number of rounds. On average, having 100 rounds with 1 epoch performs best. This result shows that having high synchronization between the models leads to stable models. On the other hand, models that do more epochs tend to provide higher errors.

Figure 4.9 lower part shows the error of the best configuration found for each case. In this case, it can be seen that 100 rounds are still a good choice. However, for $N = 3$ the other two options provide a lower RMSE in the number of car prediction. Overall it can be observed that with the FL approach it is possible to obtain models that are close to the centralized model. Moreover, in some cases, the model can be better.

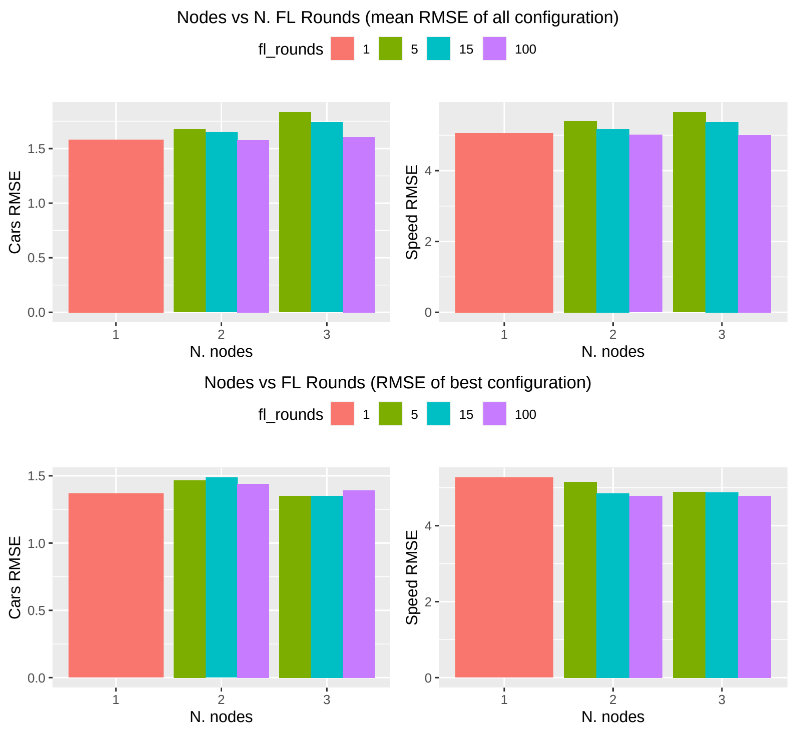


Figure 4.9. Comparison of the Root Mean Squared Error (RMSE) regarding the number of nodes (federated models) and the number of FL rounds with (top) the average error of all the tested model configurations and (bottom) the best model configuration. Notice that the different number of rounds have an associated number of epochs.

Learning rate on Federated Learning

Now the focus is on the effect of the learning rate when using FL. The upper part of Figure 4.10 shows the average error of all hyper-parameter configurations with respect to the learning rate and the number of nodes.

When the learning rate is small enough, splitting does not have a high impact on the error. However, if we increase the learning rate, the error increases with respect to the N. This effect may happen because the local models are overfitting and the merged model is not able to fall in the local minima and lead to divergence. Figure 4.11 is an example of this. When the learning rate is high, the models are overfitted to the local data as is the case of the local model 2. As the data from local models 1 and 3 might be more similar, the merged model favors them as they have two-thirds of the voting when averaging. When using a smaller learning rate, the learning curve is smoother and converges to a better solution that fits all the data splits, i.e. produces a more generalized model. This can be observed in the lower part of Figure 4.10, which shows the error of the best configurations found for each parameter combination.

4.4.3 *Effect of initialization when using Federated Learning*

It is known that using a common initialization of the models before training is required to find a good solution (McMahan et al. [52]). However, it is not clearly shown what happens when not using the initialization in the long run. Therefore, in this experiment, we show the effect of using a common random initialization versus random initialization for each model. The experiments are performed over the best configuration found for the cases of two and three models from Section 4.4.2. For each training process, random initialization and common initialization are tested.

In Figure 4.12 it can be observed that initially, the non initialized setting produces a model that

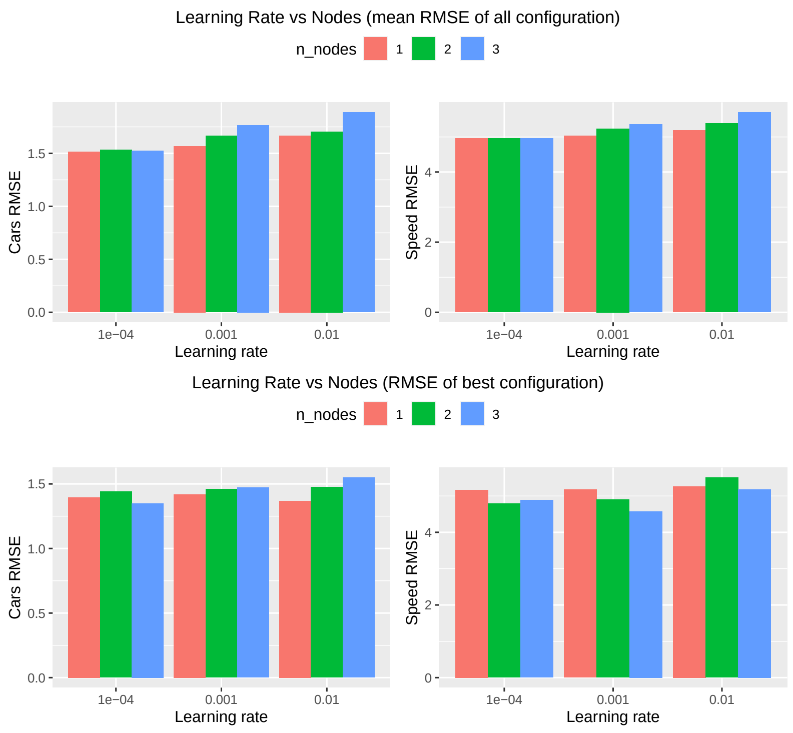


FIGURE 4.10. Comparison of the RMSE regarding the learning rate and the number of nodes (federated models) with (top) the average error of all the tested model configurations and (bottom) the best model configuration.

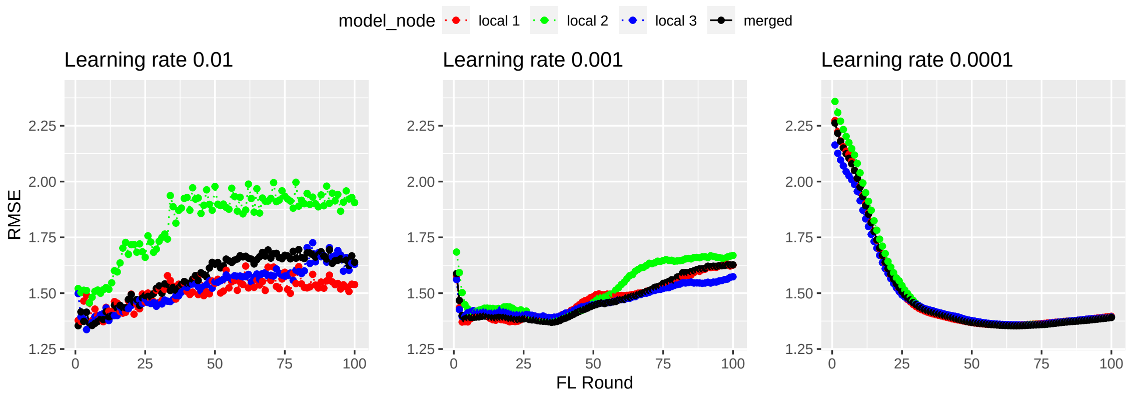


FIGURE 4.11. Cars RMSE with respect to the learning rate and the FL rounds. Notice that when the learning rate is 0.01 the merged model is overfitted towards the blue local data from models 1 and 3. When the learning rate is smaller, the model obtained is more general and is able to cover all the data splits. Also notice that for the four models, the test data is the same, therefore the data used for training in the local model 2 is more different than the other two splits.

initially has a higher error. This is because the two models from the non initialized setting are initialized differently, producing different weight distributions. This makes that, for example, the hidden unit 1 recognizes one part of the input data while the unit 1 of the other model recognizes another part of the data. This makes the initial solution produced in the first step is close to random. However, when synchronization starts, both models get to similar error metrics. Finally, in this case, the speed is adjusted better in the first rounds of the non-initialized model by chance.

Figure 4.13 shows a similar trend for the experiments federating 3 models. In particular, initialization provides a good solution in the first step from the start as we are training 34 epochs before synchronizing, a similar setup to what was proposed in Gutierrez-Torre et al. [21]. Notice that the first

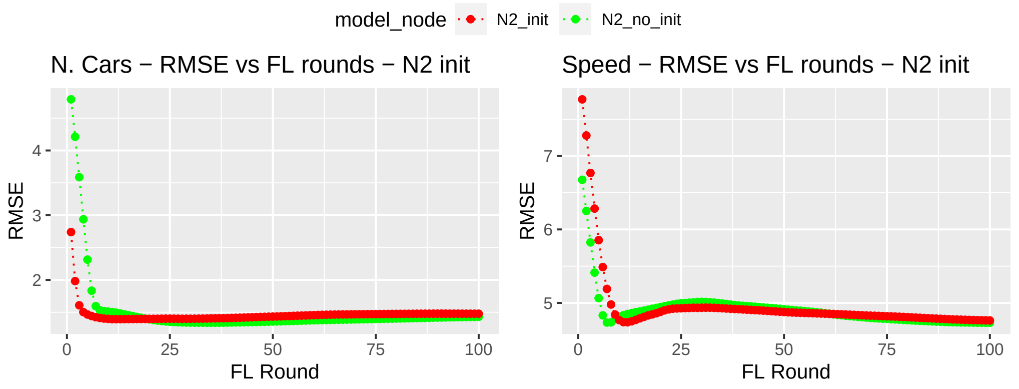


Figure 4.12. RMSE of the merged model for the number of cars and speed with 2 federated models comparing having a common initialization (N2_init) versus random initialization (N2_no_init). Configuration: Hidden units 6, batch size 32, epochs 1, federated rounds 100, and learning rate 0.0001. RMSE calculated with the test set.

point of the green line represents the proposed method in Gutierrez-Torre et al. [21]. In that previous work, as federated rounds were not included, the error increased linearly with the number of models to federate. Moreover, we see that the non-initialized setup reaches a lower error in speed. This is not because not initializing the process is beneficial but to the stochastic nature of the model initialization. Different initialization leads the network to different configurations and then different results if the training falls into local minima.

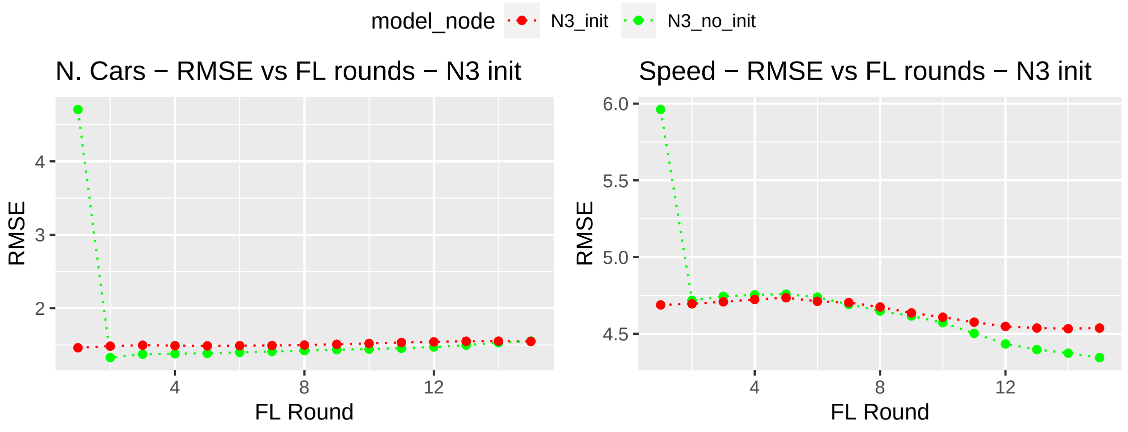


Figure 4.13. RMSE of the merged model for the number of cars and speed with 3 federated models comparing having a common initialization (N3_init) versus random initialization (N3_no_init). Configuration: Hidden units 8, batch size 32, epochs 31, federated rounds 15, and learning rate 0.0001. RMSE calculated with the test set.

In conclusion, providing a common initialization is helpful to have a usable result from the first synchronization step. However, it is not completely required, as after the second step the models will share the same structure because of the process.

4.4.4 *Evaluation on low-powered devices*

In this experiment, we compared a FL round of the proposed solution's effectiveness on low-power and resource constraint devices designed for the Edge, like the *Raspberry Pi* model 3B and the *NVIDIA Jetson* model Nano. Such devices are built for consuming less than 12W and embed low CPU and GPU computing resources. Raspberry Pi is used for general purposes while the Jetson integrates a GPU for AI and neural network computing on the Edge and smart devices. We measure the time of doing one FL round with the given configuration and 94 epochs to simulate the maximum time required to process one round.

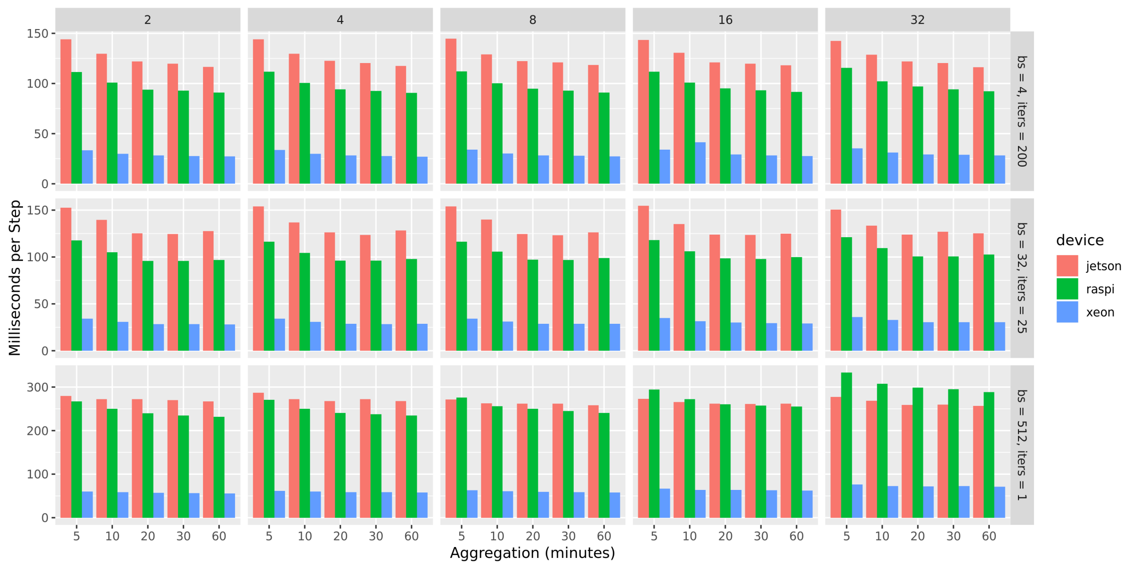


Figure 4.14. Time comparison for configurations in low-power devices vs. single CPU Xeon ref., for each amount of hidden units.

Testing the grid configurations for *Time Aggregation* vs. *Hidden Units* on the Raspberry Pi and the Jetson Nano, we observed a noticeable increase in execution time in comparison with the single-CPU Xeon. Still, the training plus validation time is below 30 minutes for nearly a week worth of data. We tested 4, 32 as 512 Batch Size (BS), i.e., the number of samples used for each training step in the neural network to check the Jetson GPU's possible advantages due to data bandwidth. The bigger the batch size, the more we profit from the GPU's parallelism up to a certain point. The number of epochs is fixed at 94, to compare the performance of identical training processes, and the steps (iterations) per epoch are proportional to the batch size (200 steps/epoch for BS = 4, 25 steps/epoch for BS = 32, 1 step/epoch for BS=512). The objective was to test the method's performance on low-powered devices with different properties while maintaining the error (that may vary when modifying the batch size). As a comparison metric, we show the milliseconds per step and the seconds per epoch. When computing the average milliseconds/step, the first epoch was excluded as it carries the overhead on warm-up around ×4 the average epoch.

Figure 4.14 shows the performance in times per step for the different configurations of the GRU in the different used technologies, for the training time with a common and proper configuration found for the GRU on the single-CPU Xeon, the Raspberry Pi ARM-based CPU, and the ARM-based and GPU enhanced Jetson.

From this experiment, we concluded that our method is fully fit for use on low-power or resource-constrained devices, as training times take at most half an hour for a model representing around six days. Moreover, we noticed that the GPU at the Jetson Nano does not provide improvement for the kind of data until the batch size reaches larger sizes.

Figure 4.15 shows the absolute training time for the 3 devices, where the single Xeon outperforms the low-powered ones but for no more than a factor of 4, and how in scenarios requiring large memory bandwidth (low data aggregation and large batch sizes), the GPU starts chasing CPU execution times.

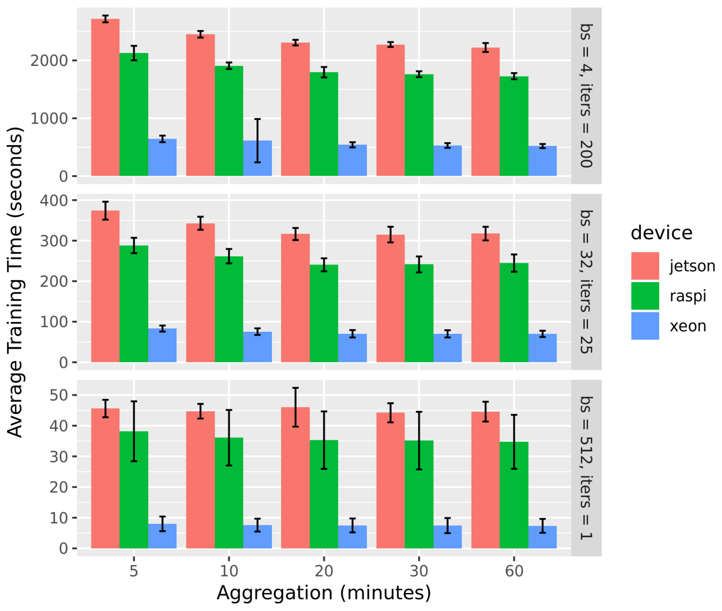


Figure 4.15. Average modeling time on different Edge devices.

4.5 Conclusions

In this chapter, we have explored what Fog computing architectures are and how they are used to build distributed systems for near-realtime applications. Moreover, the Data Science pipeline was adapted to this kind of architecture, and Federated Learning (FL) framework was applied to it. Using this system it is possible to build centralized ML models training them where the data is produced. Doing this process on the Edge, allows the system to avoid sending unnecessary data to the backend which saves network bandwidth. Moreover, in case of connectivity failures, the training process can be continued unaffected, which provides a more robust system. In this chapter, we proposed to train the models in the Edge instead of uploading the data to the Cloud and performing the training there. This leads to two main advantages: the system can be trained online while receiving data without any additional latency and with this, a system that detects *Concept Drift* can be set up so that models are retrained instantly when they start failing, e.g. detecting changes with statistical mechanisms like Kullback-Leibler divergence. The architecture was evaluated with road traffic, but the same architecture applies to further chapters in which the main topic is maritime traffic and emission estimation.

The experiments showed that this kind of system is possible and have extended the work from Gutierrez-Torre et al. [21]. First, the FL framework produces models that are similar in accuracy to the ones that would be produced centrally, contrary to our previous work. Moreover, in some situations the models are better which may be related to the improvement in generalization that merging models can have [112, 113]. The initialization of the models was also tested and the experimental results showed that a common initialization of the models is recommendable but not vital for the process, as the first synchronization performs the same effect as initializing. Mind that not initializing the models leads to losing computation time of the first step. Finally, the experiments show that this approach can be used with the proposed model as it is light enough to be trained in low-powered devices.

There are also limitations and future work to cover for this chapter. First, the optimization algorithm *RMSProp* is used, which includes an adaptive learning rate mechanism. This is generally good when training this kind of model. However, when doing a new FL round the learning rate is reset due to how the current code works internally. It will be positive to create a mechanism that takes this into account and saves the last learning rate applied to avoid jumps in the error function. Moreover, online and batch learning are not compared, only the results for batch learning are provided. As mentioned before, it is also required to study the *Concept Drift* in this context, however, this would require extra data and, if possible, online streaming data. Finally, other techniques like Graph Neural Networks (NNs) could be explored and also other hardware platforms like MicroController Units (MCUs).

Chapter 5

Ship properties missing data estimation and evaluation with Machine Learning

5.1 Introduction

Maritime traffic is a key component for the European Community [11] because of its fuel efficiency compared to other means of transportation for goods, according to the European Community Shipowners Associations (ECSA). As the global economy expands, the relevance of maritime traffic grows, especially in terms of exhaust emissions. The Third IMO Greenhouse Gas study [96] showed that due to the global-scale trade the contribution to global Greenhouse Gas will increase between 50% and 250% by 2050. Given this situation, governments and organizations like the World Health Organization are interested in the advances of emission detection to create law enforcement for Air Quality Standards.

Given the current technological advances, detecting the exhaust emissions from ships in near real-time is possible. By international regulation, most of the ships are equipped with the AIS system. This GPS-based tracking system is used to prevent collisions between ships, however, the data can be correlated with the ship characteristics to estimate how much pollution the ship is producing at that precise moment.

The estimation of the exhaust emissions with this data is covered in the scientific literature, like in the work of Jalkanen et al.[1, 16] with their STEAM methodology. Other methodologies based on different data sources like the HERMESv3 model [25] can benefit from this data to accurately position the emissions, improving therefore their spatial and temporal resolution. AIS-enabled methodologies make use of the AIS speed, timestamp, and position to evaluate how, when, and where the engine is being used.

Along with this data a static dataset containing the characteristics of the ship is used. This dataset is crucial as it contains attributes used to compute the emissions, e.g. the total installed engine power. This dataset is required for the calculations inside the model, defining characteristics like the boundaries of how much power and fuel is being used, the type of fuel required in the ship, or the amount of auxiliary engine power at a given point, among others.

Even though the methodology provides high spatio-temporal accuracy compared to other approaches [98], it is highly dependant on the quality of data, i.e. how fit is the data for a given problem as defined in Section 2.5.1. On one hand, AIS data may be incorrect due to human or machine errors or samples may be missing [18, 95]. On the other hand, a similar situation happens with static data provided by ship registries. The data may be missing due to multiple registries managing the same area [3] or attributes that are commonly missing or incorrectly set [4, 95].

IMO reports showed that important features for the emission estimation can be missing and that the methodologies for estimation should be aware of this phenomenon [96]. This is the case of RPM (91% of availability), the main engine installed power (99.1% of availability), or the length of the ship (43.2% of availability).

Since vital data for maritime traffic analysis can be missing, there has been research in this direction. This process is called in the literature as error correction, data infill or missing data estimation and is part of the data preparation process in the Data Science pipeline, which is in charge of making sure that the data is fit for use for a given application. For example, Johansson et al.[26] proposed a mechanism to estimate the missing data with ships that are similar to the one being treated. There have been other efforts using standard Machine Learning [98] and specific studies of a particular ship type [4]. However, to our knowledge, there is no comparison of methods nor a guideline for practitioners that want to use this kind of data for maritime studies or emission estimation. Moreover, scenarios with low-powered compute devices like Smart Cities or Edge Computing settings can benefit from exploring traditional fast ML techniques. The missing data estimation process can be done in the devices near where the data is produced, e.g. AIS antennas, so that the data is only transmitted after correction.

Therefore, the contributions of this study are the following:

- Provide a review of the methods that are currently being used for this task in the field.
- Provide a comparative analysis of missing data estimation techniques applied to ship characteristics.
- Apply standard ML techniques that are easy to use and fast to train.
- Close the gap between practitioners and data analytic techniques providing a guide on how to successfully apply standard ML techniques.

The structure of the chapter is the following: Section 5.2 presents the background and methodology to reproduce this work. Section 5.3 shows the results and the discussion of experiments. Finally, Section 5.4 contains the conclusions and limitations of this study.

5.2 Methods and data

In this section all the required methods and data is presented. Some specifics on emission estimation are provided as context along with the available data required for the process. Then, the manual preprocess of the data is introduced to ensure the fitness for use of data. The basic ML and state-of-the-art techniques required to understand this chapter are introduced. Finally, the used validation protocol and metrics are introduced. All these elements define the prerequisites and key building blocks for this contribution.

5.2.1 Emission estimation

Ship exhaust emissions can be estimated with different approaches and points of view. Primarily, there are two main approaches: Top-down and Bottom-up. Top-down starts from the global emission measure and tries to disaggregate the value to estimate where it comes from. On the other hand, Bottom-up approaches use positional data, e.g. GPS, to accurately position the origin of the emission. With the available characteristics from that origin, this kind of methodology can estimate how much pollution is produced by that origin. This approach is better than the former in precision and temporal resolution [3]. Notice that there might be methodologies that fall in between the two main approaches.

In particular, this work makes use of the second approach using the STEAM methodology proposed by Jalkanen et al.[1, 16]. STEAM2 from Jalkanen et al.[16] improves STEAM from Jalkanen et al.[1] in terms of the ship characterization, adding details on water friction, engine load balancing, and other factors that affect the engine usage.

Both versions of the methodology make use of the data coming from the AIS and static information from the ship to position the emissions and calculate the amount of pollution that is produced.

Following this methodology, we have to calculate two factors to estimate the emissions: the activity factor and the emission factor. The activity factor is measured in kWh and refers to how much power are the engines producing at a given time. The emission factor is measured in g/kWh and refers to how many grams of each pollutant is emitted for each kWh generated. Equation 5.1 represents the two parts needed for the calculation. Notice that with AIS we have the timestamp and the position of each sample, therefore we can locate spatio-temporally the data with precision.

$$Emissions\ (g) = EmissionFactor\ (g/kWh) * ActivityFactor\ (kWh) \tag{5.1}$$

For this purpose, we require that the data is available and correct. In STEAM2, whenever a key attribute is missing, the STEAM methodology is used as a fallback. This is because the first version is more general and simple and, therefore, it requires fewer attributes to do the calculation. In this work, we focus on STEAM so that we make as few assumptions as possible about data.

To estimate the activity factor the methodology needs the current speed of the ship and the installed power on main and auxiliary engines. For the emission factor the SFOC, the engine's RPM, and the fuel type are required. In both STEAM and STEAM2 the SFOC is fixed to a given value. If more accuracy is desired the approach followed by United Kingdom's NAEI [95] can be followed. This approach uses other studies to estimate reference values by using the engine type and fuel type. In this case, we consider SFOC as constant following STEAM as we are already in a case where data is missing. Given this information, this work will focus on the requirements from STEAM as STEAM2 uses it as a fallback when data is missing [16].

Figure 5.1 shows a simplified data pipeline with every step needed for a correct emission estimation. Static and AIS data cleaning process and data inferencing are the data preparation steps mentioned in Section 2.5.1 that ensure the fitness for use of data for computing emissions, i.e. they ensure the data quality. In the following subsections all the processes are described.

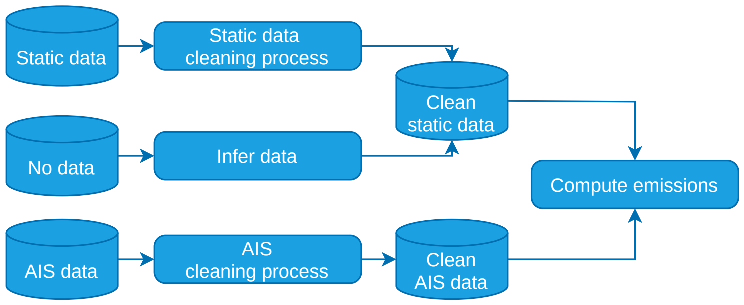


Figure 5.1. Description of the data pipeline from raw data to the emission computation. AIS cleaning process is a simplified version of Jeon et al. [2]. Data inference when there is no data is covered on Section 5.2.6. This work covers the *static data cleaning process*.

Notice that the complete process can be performed on-line. It is possible to use the pipeline in the system described in Chapter 4 so that the data preparation is performed when the data arrives and then the emissions are computed. However, the correction of static data is also well suited for the cloud or central system as this kind of data does not evolve over time. However, it is interesting to use the approach of Chapter 6 (data inference in the figure) in the Edge as the data required for the process is collected there.

5.2.2 *Data description*

There are two datasets required to estimate emissions: the data provided by the AIS and the data provided by a ship registry. Both datasets are constrained to the area of the port of Barcelona and refer to the year 2017.

AIS dataset

The first dataset is a time-series dataset that contains the position, speed, and other ship characteristics. This dataset was obtained from the local port authorities. Each ship emits this data to the others by radio frequency using the AIS. This technology was devised to avoid ship collisions and to help the ships in their navigation. However, this data may be gathered from land to know the actual position and other data of the ships. However, as in every data problem, this data might not be clean due to machine or human errors. As seen in Section 3.3.2, there is plenty of work done regarding this issue in the literature. In particular, Jeong et al.[2] propose a pipeline to find anomalies and completely denoise the data. For this work, we have simplified this pipeline and only used the Kalman Filter step whenever it is required as our AIS dataset is already well-curated.

Static characteristics dataset

The second dataset contains the static properties of the vessel, e.g. main engine power, length of the ship. This dataset was obtained from the company IHS Fairplay but can be obtained from others like Lloyd's registry. This dataset shares variables in common with the previous dataset, e.g. length. However, as the data comes from a curated register they are more reliable.

This work focuses on the variables shown in Table 5.1 as those are the basic characteristics required for the basic emission estimation. In case STEAM2 wants to be used, the number of variables required is higher, and, therefore, the amount of uncertainty increases.

Attribute	Description	Origin	Missing
Type	Ship type	AIS/IHS	0%
Main Engine (ME) power	Installed main engine power	IHS	0.03%
Auxiliary Engine (AE) power	Installed auxiliary engine power	IHS	37.11%
ME RPM	Main engine RPM	IHS	2.57%
AE RPM	Auxiliary engine RPM	IHS	100%
Eng. type	Type of engine installed (Main engine)	IHS	0%
L	Ship length	AIS/IHS	0%
B	Ship beam	AIS/IHS	0.03%
T	Ship draught	AIS/IHS	0.21%

Table 5.1. Availability in our dataset of the variables required for STEAM emission estimations. Notice this scenario is not general as the ones seen in the work of Peng et al. [3] and Huang et al. [4].

In Table 5.1 the actual missing values of our data can be observed. Even though there are not many missing values in our case, it is still a concern to be tackled as suggested by the scientific literature as there may be missing data due to having multiple administrations for the same space [3] and that some attributes are prone to be missing [4].

5.2.3 *Data filtering*

As in every data-related field, data correction may not be warranted. Therefore, before doing any analysis with data, we are required to do an error analysis of it. Doing this filtering and correction fully automatically can be dangerous as we could be removing outlier data, i.e. data that is correct but has uncommon values. Because of this, in this work we have used a combination of automatic and manual analysis.

To find errors we first need to know which row of our data are outliers. To do so we apply Rosner filter [114] to find the possible univariate outliers. Notice that we are not using a multivariate outlier detection technique because we are interested in punctual incorrect values introduced by human error or device failures. After applying this filter we observe that there are 152 ships with possible outliers. After building the tables of possible errors, we manually check every row and compare it with the rest of the ships of the same type and also further information available on the Internet, e.g. MarineTraffic. We found that 15 of the 152 ships have erroneous values. The most common erroneous value was Revolutions Per Minute (RPM), reporting unrealistic value. It was followed by Auxiliary Engine (AE) power and Main Engine (ME) power. The values are marked as *not available* so that we can use the in-fill techniques to correct them afterward.

5.2.4 Statistical and Machine Learning techniques

Missing data can be estimated using statistical and Machine Learning (ML) techniques. These techniques will use the relationship between the variable to be estimated (target) and the variables used in the process (predictors or features). In this case, all the target variables are numeric, therefore we will focus on regression algorithms. Most of the algorithms presented below are well-known. If further information is needed, please consider checking the work of Bishop [115]. All the methods were implemented using *python* and the ML library *scikit-learn*. Gradient boosting trees were implemented using the *XGBoost* package. Data is scaled subtracting the mean and dividing by the standard deviation, i.e. standardizing, to avoid pitfalls in the algorithms and improve prediction, e.g. Neural Networks.

Linear and polynomial regression

We use the well-known linear regression and, in some cases, quadratic regression as a baseline. These methods are a good starting point as there is no hyper-parameter to be tuned and regression coefficients can be interpreted. We will use the coefficients to assess the predictors' importance. Moreover, the risk of memorizing the training data is low because of the simplicity of these methods, i.e. the risk of overfitting.

k Nearest Neighbors and Most Similar Vessel

k Nearest Neighbors (KNN) is a method that uses distances like the Euclidian or Mahalanobis distances to find the k data points that are closer to the one we want to estimate. This distance is computed using the desired predictors. In the case of regression, the data from those k points are usually averaged to obtain the estimation. As the data points are the model, it will always compute the same estimation for the same input data and point to estimate.

Most Similar Vessel (MSV) is a particular case of KNN proposed by Johansson et al. [26]. In particular, this is the case when $k = 1$ and the following custom metric is used:

$$s = \sqrt{a\left(\frac{v - v_c}{v_c}\right)^2 + \left(\frac{l - l_c}{l_c}\right)^2} \tag{5.2}$$

Where s measures the similarity of the ship with characteristics to estimate with a candidate vessel c. The lower the value s is, the more similar is the current ship. Particularly, this approaches uses the design speed of the ship (v) and the length of the ship (l) as variables. Moreover, the α weight parameter was added and optimized to 0.35 using a large database of ships and with a Monte Carlo simulation.

Both KNN and MSV were implemented with *scikit-learn KNeighborsRegressor*, which allows to use a custom distance function.

Neural Networks

Neural Networks are a set of learning techniques for regression and classification based on interconnected units. In particular, we use the Multi-Layer Perceptron (MLP) which is made of units that can be regarded as a linear model each. Their output is connected to other units and these units are interconnected in layers so what the first layer extracts from the data is used in the following layer as input. For this problem, only one layer and two layers will be used for simplicity. The L2 regularization parameter α (from 10^{-4} to 5), the initial learning rate (from 10^{-3} to 0.5), and the number of neurons per layer (from 1 to 100 in each layer) are tuned.

Support Vector Machines

Support Vector Machines in regression work similarly to linear regression but provides a mechanism to regularize (i.e. generalize better), the margin. Moreover, this method can use *kernels*, which provide the method to define non-linear regressions. However, as it increases too much the computation complexity in our case this mechanism is left out, only making use of the linear support vector regression. The C regularization parameter (from 10^{-3} to 1000) will be tuned.

Random Forest and Gradient Boosting Trees

Random Forest (RF) [116] and Gradient Boosting [117] are both based on the idea of using decision/regression trees together in an ensemble. The primary idea in Random Forest is to generate a variety of small trees that can be considered *weak learners*. These learners are trained using a different subset of predictors each time, therefore never seeing the complete data and generating, thus, different classifiers. Moreover, the data offered to each tree is a bootstrapped sample of the training data. With these two mechanisms, the trees can be more independent one from the others, which is a desired property in ensemble methods. In particular, the number of trees (from 10 to 100) and the maximum depth of each tree (from 2 to 20) will be tuned.

On the other hand, Gradient Boosting Trees uses the same idea of *weak learners* but focuses on hard to predict data. When the first model is trained, the prediction from the data points is checked. Using that, the next model will be trained to make more emphasis on the data points that are incorrectly predicted. This way an ensemble of *weak learners* that focus on different parts of the data is obtained. In particular, we use the python implementation XGBoost. From the available parameters, number of trees (from 10 to 100), subsample for each tree (from 20% to 100%), number of columns for each tree (from 66% to 100%), and a maximum depth of each tree (from 2 to 20) will be tuned.

5.2.5 *Validation methodology and metrics*

Data with missing values were discarded as we need all the attributes to be present for the validation of the approach. This is because we calculate the validation metrics by comparing the real value with the predicted value. After that, the data was split into a training dataset and a test set, in a 70%/30% split. This split is done using a stratified sampling procedure, which ensures that the train and test sample contains a similar proportion of ships regarding the ship type. The same testing split is used on every experiment. This is shown in Figure 5.2. Moreover, we can see how the experiments are done in terms of data input and results.

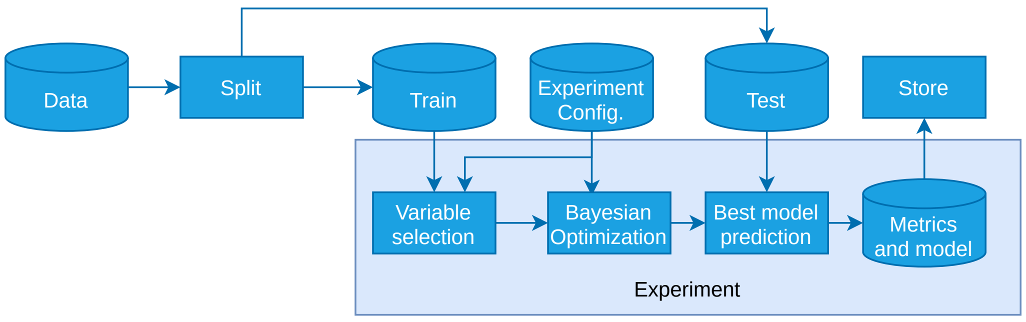


Figure 5.2. Experimental pipeline diagram. The configuration is used to define the variables used for prediction and the target and also define the model to be used and the hyper-parameters to be tuned. The metrics and the models are stored in the file system for model comparison.

Whenever there are free available parameters from the algorithms of Section 5.2.4, i.e. *hyper-*

parameters, they are tuned. To do this, the classic Grid Search can be used. With this approach, a grid is built with all the possible parameter combinations to be tested. However, this exploration may not be efficient or optimal. In this work Bayesian Optimization is used as it is a more effective way compared to Grid Search [118].

To use this approach, the user has to define the range of values to be explored. With this space, the algorithm builds a surrogate model to search for the best hyper-parameters in a fast way, evaluating only promising parts of the space, instead of every possible combination. The hyper-parameters and the range of values for each hyper-parameter are defined in Section 5.2.4.

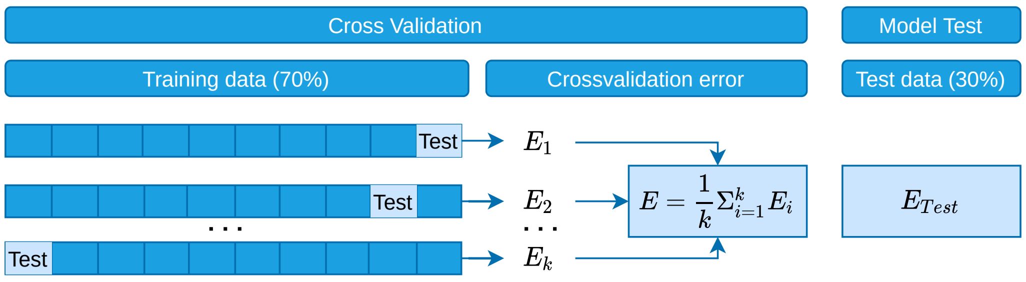


FIGURE 5.3. k-fold Cross-validation and model testing processes. The data is divided into k fragments and then k models are trained. These models use 9 folds as input and then the other one as validation data set. Each model uses a different partition as test. Afterward, the best model configuration found is tested with the test partition.

Inside of the Bayesian Optimization, the classical cross-validation procedure is used. As seen in Figure 5.3, in cross-validation k models are trained with the same hyper-parameters to ensure that the error estimated is global, i.e. how we split the data does not affect the error estimation. In this case we use 10 fold cross-validation. This means that for each combination of hyper-parameters we train 10 models leaving outside a different partition of data (fold) and compute a validation metric with the remaining fold. The best hyper-parameter combination is obtained from the combination that on average leads to models with less error. The optimization was implemented using the python package *skopt*.

For the internal evaluation of the model the R^2 is used. R^2 shows how much the prediction adjusts to the real values, being 1 the maximum value. In the experiments, we also show Mean Absolute Error (MAE) and RMSE. MAE will show on average how much the value of the predictions is wrong, either by predicting more or less than the actual value. On the other hand, RMSE will penalize bigger errors. Both MAE and RMSE are measured in the original variable unit.

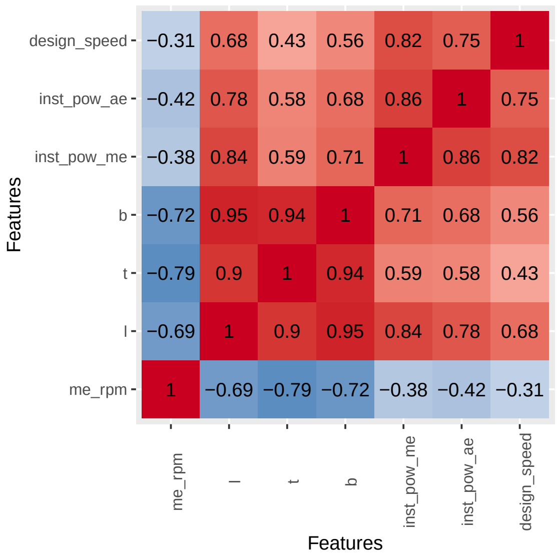
5.2.6 *Absence of static data*

Notice that when there is no record of a ship in the static characteristics dataset nor correct static information in the AIS dataset, the attributes are usually computed using the mean of the attributes by ship type. If that attribute is also missing, a small ship can be assumed [1]. Another approach is to use the traces defined in the AIS system to build movement patterns and use those patterns to estimate the missing attributes as will be seen in the next chapter.

5.3 Results and discussion

In this section, we present the experiments to evaluate the proposed corrections. First, a correlation analysis is done to see which are the linear relationships of the variables, that will be exploited later. Then we perform the prediction of the variables using all the other variables, then the relevant variables, and then with just one variable. Finally, we compare this approach with state-of-the-art methods. The results presented in this section are summarized. For the complete tables and figures please check the supplemental material found in Appendix A.

5.3.1 *Correlation analysis*



	me_rpm	l	t	b	inst_pow_me	inst_pow_ae	design_speed
design_speed	−0.31	0.68	0.43	0.56	0.82	0.75	1
inst_pow_ae	−0.42	0.78	0.58	0.68	0.86	1	0.75
inst_pow_me	−0.38	0.84	0.59	0.71	1	0.86	0.82
b	−0.72	0.95	0.94	1	0.71	0.68	0.56
t	−0.79	0.9	1	0.94	0.59	0.58	0.43
l	−0.69	1	0.9	0.95	0.84	0.78	0.68
me_rpm	1	−0.69	−0.79	−0.72	−0.38	−0.42	−0.31

Figure 5.4. Variable correlation. *me* is main engine, *ae* is auxiliary engine, *inst_pow* means installed power, *b* is beam, *t* is draft, *l* is length.

First, we perform correlation analysis to show how the variables are related. Figure 5.4 shows two groups of variables regarding correlation: size variables and engine variables. These groups are well correlated among themselves. Notice that this correlation reflects only the linear relationship between variables.

Design speed has the highest correlation with *installed main engine power*, as the more power the engine has, the higher speed it could potentially get. On the other hand, it is important to take into account also the size of the ship, because bigger ships have higher engine power but may be slower. Therefore, *length* and *installed main engine power* are key components for predicting *design speed*. *Installed main engine power* is highly correlated with the *length* of the ship. This fact is reasonable as the bigger the ship is, the more engine power is needed to move it. *Installed auxiliary engine power*, the attribute that is most commonly missing, is very correlated with *installed main engine power* and *length*.

Regarding the size variables (*lenght (l)*, *beam (b)* and *draft (t)*) are very correlated among themselves. *Beam* is a good candidate to predict the other two. *Length* is a good candidate to predict the *beam*. Finally, *main engine RPM* is very correlated negatively with *draft*, as ships with smaller draft have high *main engine RPM*. This could be the case of smaller and faster ships like the yachts.

These attributes hold sound relationships that may be used for prediction. As strong correlations are found between variables, it may be possible to predict an attribute with just one of the variables. This fact is explored in Section 5.3.4.

5.3.2 *All variables*

In this section, we explore the results of modeling each attribute with the other available predictors.

In Table 5.2 it can be seen that from all the available algorithms, the two tree-based algorithms worked best. In every variable, both algorithms differ very little in error but *Random Forest* tends to

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam	XGBoost	1.0000	0.9921	0.0053	0.9476	0.0026	0.4590
	RF	0.9985	0.9918	0.4180	0.9644	0.2094	0.4951
Design Speed	RF	0.9854	0.9504	0.4849	0.8881	0.2387	0.5133
	XGBoost	0.9728	0.9470	0.6611	0.9182	0.4487	0.6052
AE Power	RF	0.9823	0.9097	477.5738	1109.8918	210.8580	496.8095
	XGBoost	0.9954	0.8969	242.0613	1185.4034	103.0322	530.4132
ME Power	XGBoost	0.9999	0.9852	134.5394	2112.8145	22.7920	779.6078
	RF	0.9979	0.9850	768.2021	2130.9932	319.0958	844.8177
Length	RF	0.9984	0.9909	3.0077	7.2957	1.5252	3.7745
	XGBoost	0.9999	0.9900	0.7402	7.6552	0.1760	3.8023
ME RPM	RF	0.9903	0.9233	45.2185	125.5659	18.6240	53.1499
	XGBoost	0.9995	0.9178	10.0946	130.0155	4.6482	52.2184
Draft	XGBoost	0.9997	0.9882	0.0592	0.4033	0.0349	0.2209
	RF	0.9974	0.9870	0.1891	0.4238	0.1018	0.2420

TABLE 5.2. Error measures for training and test for the target variable predicted with all the other variables. Only the two best algorithms are shown for each variable. The full table is available in the supplementary material. Ordered by test R^2.

work better in this case. Notice that *XGBoost* usually has less training error than *Random Forest* but test error is higher. This is because gradient boosted algorithms tend to perfectly fit the training data even though there are regularization mechanisms. This is why we always have to test the algorithms using a separate test set. Overall it can be seen that the R^2 is between 0.92 and 0.99, which is a very good result. *ME RPM* is the attribute that appears to be harder to predict, whereas *Beam* and *Length* are the easiest. This is probably due to the high correlation between the size attributes.

Predicting a variable with the others available is not the best situation in a deployed solution. This is because for a given entry we may have more than one variable missing. Therefore in the following section, we explore the importance of each variable in prediction so that it is possible to reduce the number of used variables.

Extracting relevant variables

Given that the variables can be well predicted using the others, now the objective is to reduce the number of variables necessary for prediction. To find which variables are relevant for prediction, we analyze the linear model coefficients, the variable importance from *Random Forest* and *XGBoost* and the correlations from Figure 5.4. The importance of the variables of the *linear model* is calculated by sorting the t statistic of coefficient relevance in absolute value (the higher, the more important). In the case of *Random Forest* and *XGBoost*, the importance is calculated as how many times is a variable used in the tree splits. Notice that the linear model coefficients and the variable importance refer to the importance of the variables for the algorithms and that it may not be extrapolated directly as the importance of the variables for the data. However, in this work, we propose to use the three different importance values to extract which variables are relevant in general, regardless of the model that is being used. Table 5.3 shows the relevant variables for each model and the finally selected ones. Figure 5.5 shows the importance of each variable for predicting the *beam*. All the other plots for each variable are available in the supplementary material provided in Appendix A.

Variable	Linear Model	Random Forest	XGBoost	Final
Beam	Draft, Lenght, Type	Length, Draft	Length, Type, Draft	Length, Draft, Type
Design Speed	Type, ME Power	ME Power, Draft, Type	Type, Pow. ME	Type, ME Power, Draft
AE Power	ME Power, Type, Beam	ME Power, Length, Beam	Type, Length, ME Power	ME Power, Length, Type
ME Power	Length, Draft, Type	Length, Design Speed, Beam	Design Speed, Type, Length	Length, Design Speed
Length	ME Power, Beam, Draft	Beam, ME Power, Draft	Type, Draft, Beam	Beam, ME Power
ME RPM	Type, Draft, ME Power	Draft, Beam, ME Power	Type, Draft, Length	Draft, Type, ME Power
Draft	Beam, Length, Type	Beam, Length, Type	Beam, Type, Length	Beam, Length, Type

TABLE 5.3. Variable importance for each model and the final selection of variables.

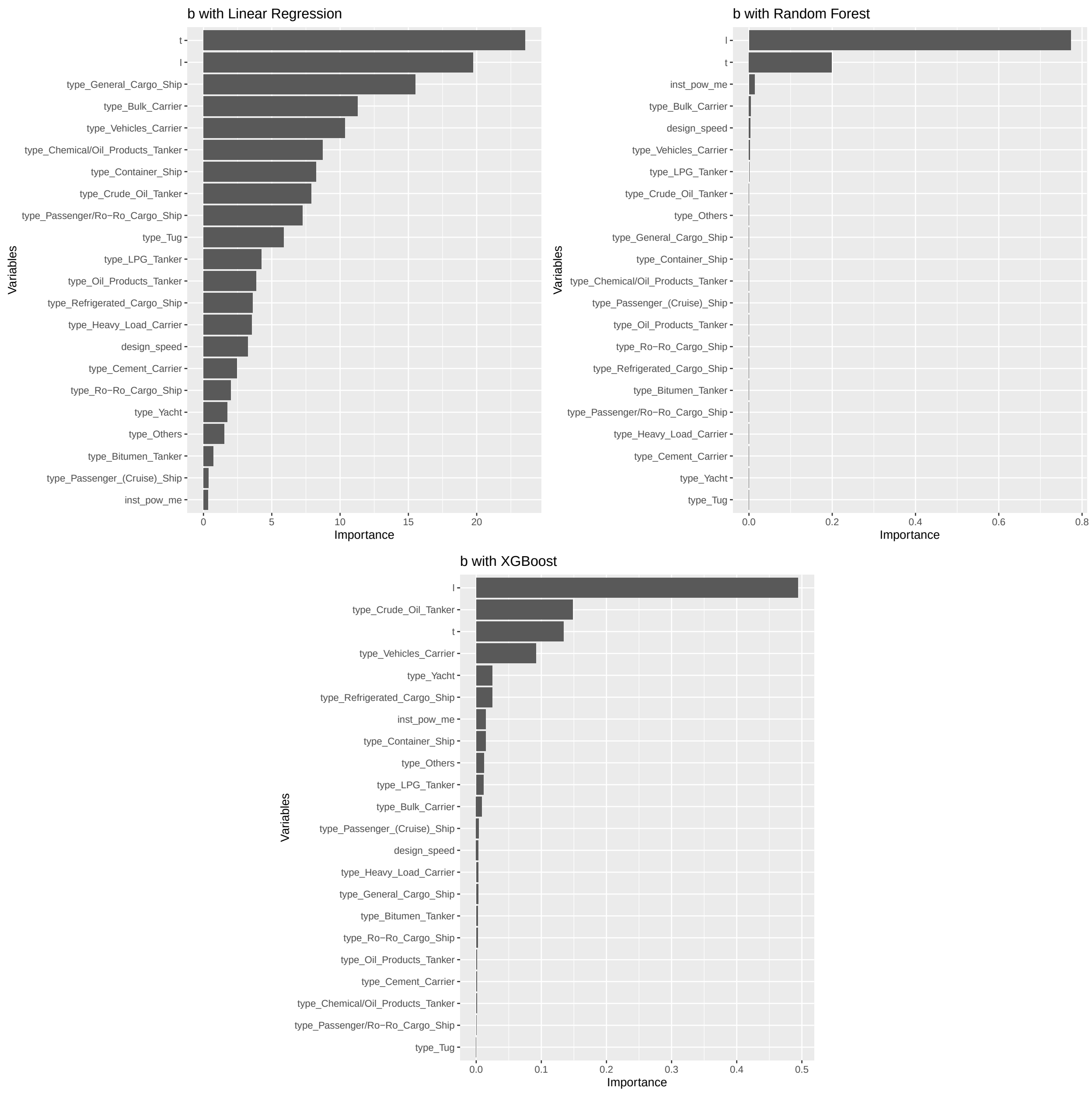


FIGURE 5.5. Relevance of variables to predict the beam.

Notice that the variable selection matches closely with the correlations shown in Figure 5.4. The final set of variables is used in Section 5.3.3.

5.3.3 *Relevant variables*

Here we test the performance of using the relevant variables from Section 5.3.2. In particular, we wanted to explore the importance of the *type* variable because it was shown that for some attributes it was very relevant, e.g. for design speed. Therefore this experiment was done with two sets of variables: the relevant variables without *type* (Table 5.4) and the relevant variables with *type* (Table 5.5), even for variables where *type* was not marked as relevant.

Table 5.4 shows the results using only two variables. From Table 5.2 we can observe that *Beam* and *AE Power* prediction stay in the same level of precision. *ME Power* (Test R^2: 0.985 $\longrightarrow$ 0.98), *Length*

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam	RF	0.9975	0.9900	0.5377	1.0659	0.2618	0.5651
(Length, Design Speed)	XGBoost	0.9976	0.9893	0.5214	1.1075	0.3092	0.6100
Design Speed	XGBoost	0.9563	0.9193	0.8381	1.1323	0.5812	0.7296
(ME Power, Draft)	RF	0.9739	0.9132	0.6471	1.1747	0.4019	0.6501
AE Power	RF	0.9761	0.8981	554.7271	1178.8827	257.6787	550.5837
(ME Power, Beam)	XGBoost	0.9934	0.8821	290.2708	1268.1226	85.3203	511.4801
ME Power	RF	0.9969	0.9802	933.4769	2443.5418	407.8315	967.2775
(Length, Design Speed)	XGBoost	0.9979	0.9781	776.8909	2571.8108	524.7569	1056.9163
Length	RF	0.9966	0.9854	4.3831	9.2533	2.3068	4.9576
(Beam, ME Power)	XGBoost	0.9907	0.9800	7.2951	10.8370	5.2606	7.2009
ME RPM	RF	0.9853	0.8721	55.6414	162.1726	22.6584	64.8916
(Draft, ME Power)	XGBoost	0.9892	0.8675	47.8553	165.0145	22.8113	66.3813
Draft	RF	0.9900	0.9622	0.3721	0.7214	0.1892	0.3871
(Beam, Length)	XGBoost	0.9893	0.9612	0.3849	0.7305	0.2378	0.4133

TABLE 5.4. Error measures for training and test for the target variable predicted only with the relevant variables. The variables used to predict are written below the variable to predict. Only the two best algorithms are shown for each variable. The full table is available in the supplementary material. Ordered by test R^2.

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam	RF	0.0005	0.0011	-0.0506	-0.0611	-0.0313	-0.0613
(Length, Design Speed)	XGBoost	0.0015	0.0013	-0.2103	-0.0700	-0.1832	-0.1047
Design Speed	XGBoost	0.0277	0.0269	-0.3309	-0.2082	-0.2798	-0.1867
(ME Power, Draft)	RF	0.0220	0.0280	-0.3925	-0.2078	-0.2702	-0.1159
AE Power	RF	0.0004	-0.0009	-4.9752	5.2422	-8.7550	-16.7539
(ME Power, Beam)	XGBoost	-0.0151	0.0051	237.2241	-27.9569	250.9958	43.4501
ME Power	RF	-0.0011	-0.0012	150.1328	73.8664	248.2645	137.4452
(Length, Design Speed)	XGBoost	0.0016	-0.0003	-381.3468	18.2249	-276.7815	-62.3720
Length	RF	0.0019	0.0062	-1.4683	-2.2352	-0.8114	-0.9614
(Beam, ME Power)	XGBoost	0.0069	0.0097	-3.5487	-3.0396	-3.3472	-3.1494
ME RPM	RF	0.0115	0.0253	-29.7687	-16.9290	-11.2778	-7.2151
(Draft, ME Power)	XGBoost	-0.0049	0.0297	9.7729	-19.6946	0.5686	-6.1110
Draft	RF	0.0068	0.0259	-0.1616	-0.3174	-0.0728	-0.1559
(Beam, Length)	XGBoost	0.0092	0.0251	-0.2393	-0.2967	-0.1553	-0.1761

TABLE 5.5. Difference of error when adding *type* variable as a predictor with respect to Table 5.4. The variables used to predict (excluding type which is used in every experiment) are written below the variable to predict. Increase in R^2 and decrease in RMSE and MAE means better fit. Only the two best algorithms are shown for each variable. The full table is available in the supplementary material. Ordered by test R^2.

(Test R^2: 0.99 $\longrightarrow$ 0.985) and *Draft* (Test R^2: 0.988 $\longrightarrow$ 0.96) show a slight decrease in precision. On the other hand, *Design Speed* (Test R^2: 0.95 $\longrightarrow$ 0.919) and *ME RPM* (Test R^2: 0.92 $\longrightarrow$ 0.87) precision fall drastically. In particular, we are increasing the mean error in 0.21 knots in *Design Speed* and 11.7 RPM for *ME RPM*. Notice that the results provided by *RF* and *XGBoost* are still better compared to the other algorithms, even using just two variables.

Table 5.5 shows the results of using the previous two variables plus *type*. Overall the increase of precision is not increased except for *Design Speed*, *ME RPM* and *Draft*, which were the variables that lost more precision when reducing the variable set. *Type* is an important variable, however, it may not as relevant as the others, or the same information is contained in the other variables.

5.3.4 *Unique variable*

In the extreme case where there is little information, it is still possible to predict the variables with a certain degree of accuracy. In this case, we have selected the variable that had more relevance from 5.3.2, which matched with the variable with more correlation from Figure 5.4. In Table 5.6 we can observe the results from this experiment. Compared to the relevant variables without *type* outcomes (Table 5.4) we can observe an overall decrease in precision. In particular, it can be seen that KNN and 1 layer

Neural Network appear now as good solutions to the problem. This is because RF is strong when there are more variables to sample from. When there are few variables, the trees built are more related. With this, we lose the tree independence requirement that makes RF work.

Regarding the R^2, *Beam* can be predicted with a 1.5% of decrease in the metric, which impacts the average error (MAE) as 0.5 more meters on average incorrectly estimated. In the case of *Design Speed*, it can be seen that the decrease is 6%, however, the impact on the actual values is an increase of 0.3 knots in error, which is acceptable for this task. On the other hand, *ME Power* falls only 2.5%, but this means that on average we increase the error by 900kW. Notice that regarding RMSE the big errors grow more than the average. *Lenght* also suffers from a decrease of 4%, adding 6 meters of error on average in the estimation. *ME RPM* keeps a stable error regarding R^2 and on average it only adds 30 RPM to the error. Finally, *Draft* falls a 4%, but only adds about 0.3 meters to the average error.

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam	XGBoost	0.9868	0.9744	1.2264	1.7085	0.8013	1.0654
(Length)	RF	0.9805	0.9727	1.4892	1.7648	0.9385	1.1449
Design Speed	XGBoost	0.8695	0.8574	1.4473	1.5057	0.9991	1.0932
(ME Power)	KNN	0.8922	0.8561	1.3157	1.5125	0.7975	0.9997
AE Power	XGBoost	0.9026	0.8624	1119.2508	1369.8897	651.3921	713.7344
(ME Power)	KNN	0.8767	0.8505	1259.3877	1427.8572	611.7369	707.7976
ME Power	KNN	0.9665	0.9537	3070.5436	3742.5165	1407.5615	1874.7986
(Length)	XGBoost	0.9693	0.9517	2937.0998	3819.8798	1682.1618	2032.9765
Length	XGBoost	0.9559	0.9473	15.8575	17.5739	9.8878	10.9411
(Beam)	RF	0.9553	0.9471	15.9653	17.6154	9.9571	11.0681
ME RPM	1 layer NNet	0.8671	0.8622	167.5842	168.3189	96.4330	95.5028
(Draft)	RF	0.9073	0.8597	139.9690	169.8230	78.0129	89.9753
Draft	RF	0.9367	0.9216	0.9344	1.0391	0.6337	0.7072
(Beam)	XGBoost	0.9345	0.9211	0.9510	1.0420	0.6555	0.7124

Table 5.6. Error measures for training and test for the target variable. The variable used to predict is written below the variable to predict. Predicted only with the most relevant variable. Only the two best algorithms are shown for each variable. The full table is available in the supplementary material. Ordered by test R^2.

As observed, with only one variable the variable prediction is still doable with some shortcomings, e.g. the big increase of error of ME Power. Attributes with strong correlation are logically still easy to predict. In the case of *ME RPM*, a conclusion is that ships with less *Draft* tend to have higher *ME RPM*, e.g. small yachts, and the other way around, e.g. Cruisers and Cargo ships with low RPM. Another example is *Design Speed*, where the more ME Power a ship has, the higher the design speed is. However, it is obvious that in this case, other characteristics have to be taken into account, like for example *Draft* or any of the size measures, as bigger ships need more power to move.

5.3.5 Comparison with existing solutions

MSV comparison

Johansson et al.[26] propose MSV which will use the characteristics of the ship that is most similar using the metric shown in Section 5.2.4. As seen there, this method is a KNN with a custom metric over *Design Speed* and *Lenght* and a parameter a previously adjusted using a Monte Carlo procedure. To compare we use KNN with $k = 1$ and $k = 3$. $k = 1$ is the same as doing the MSV with the following metric (euclidean distance):

$$s = \sqrt{(v - v_c)^2 + (l - l_c)^2} \tag{5.3}$$

In Table 5.7 the experiments of using *Design Speed* and *Lenght* as predictors are shown. For every variable, it can be observed that the results are consistently worse than the ones in Table 5.4, which also use two variables. Moreover, tree-based algorithms are always superior. One thing to notice is that

Target	Algorithm	R^2		RMSE		MAE	
		Train	Test	Train	Test	Train	Test
Beam	RF	0.9957	0.9842	0.6994	1.3413	0.3479	0.7555
	XGBoost	0.9966	0.9823	0.6188	1.4219	0.3228	0.8451
	3NN	0.9874	0.9823	1.1967	1.4226	0.5906	0.8152
	1NN	0.9979	0.9805	0.4920	1.4902	0.0512	0.6413
	MSV	0.9973	0.9730	0.5527	1.7562	0.0598	0.7836
AE Power	RF	0.9729	0.8890	590.2442	1230.4580	281.1902	580.1873
	XGBoost	0.9679	0.8847	642.5920	1253.9018	314.9681	597.1411
	1NN	0.9848	0.8825	441.6284	1265.9871	83.2525	541.7500
	3NN	0.9292	0.8789	953.9023	1285.2560	449.6223	632.9286
	MSV	0.9886	0.8669	382.5517	1347.0451	75.5772	554.3957
ME Power	RF	0.9963	0.9799	1025.3560	2465.9112	428.6885	978.8303
	3NN	0.9871	0.9755	1901.8463	2721.2939	768.5421	1143.6786
	1NN	0.9995	0.9717	368.2261	2927.1558	77.7506	963.1471
	MSV	0.9995	0.9710	365.3108	2963.1109	71.6225	1041.1629
ME RPM	XGBoost	0.9742	0.8710	73.7837	162.8481	36.6714	82.0743
	RF	0.9759	0.8700	71.3819	163.4616	38.1288	83.3670
	3NN	0.9297	0.8557	121.8638	172.2444	59.0172	91.1690
	1NN	0.9963	0.8223	28.0424	191.1055	3.0852	75.2486
	MSV	0.9957	0.7816	30.0128	211.8606	3.5080	85.6100
Draft	RF	0.9890	0.9541	0.3890	0.7946	0.2411	0.4687
	1NN	0.9963	0.9486	0.2264	0.8413	0.0377	0.3893
	3NN	0.9670	0.9410	0.6747	0.9009	0.3497	0.5210
	MSV	0.9970	0.9311	0.2029	0.9740	0.0325	0.4515

Table 5.7. Comparison of the best algorithms found, KNN and MSV. The full table is available in the supplementary material. Ordered by test R^2.

MSV is consistently worse than both KNN. This fact shows that the $a = 0.35$ from Johansson et al.[26] is not a global constant, i.e. it is overfitted to the data, and should be tuned for each database.

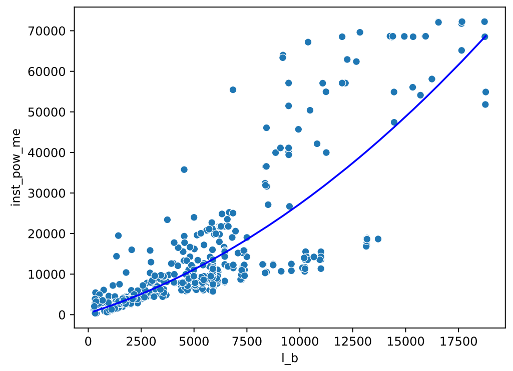
Length*Beam

As shown by Huang et al. [4], the *ME Power* can be predicted using the product of the *Length* and *Beam*, i.e. the area of the ship if it was rectangular, for Tankers. In our case, we compare this metric for all the ship types. Contrary to the situation shown in Huang et al. [4], the correlation of *ME Power* with *Length* times *Beam* we obtain that it is 0.8377, lower than the correlation of *Length*, which is 0.8428.

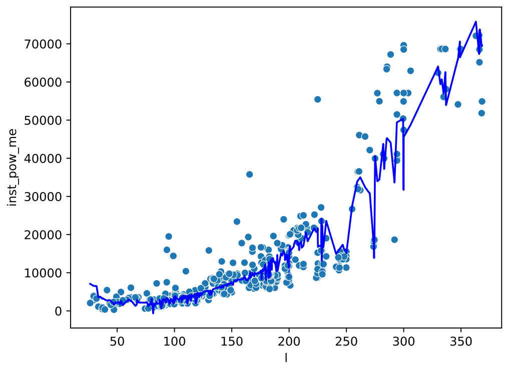
Target	Algorithm	R^2		RMSE		MAE	
		Train	Test	Train	Test	Train	Test
ME Power (Length*Beam)	RF	0.9787	0.9307	2444.9302	4576.5184	951.1324	2006.6138
	XGBoost	0.9485	0.9257	3803.4448	4739.4473	2081.3133	2576.2111
	poly2	0.7225	0.7473	8831.9435	8740.2747	5310.2186	5273.8748
	LM	0.7017	0.7180	9155.9720	9232.4103	6155.3972	6231.0306
ME Power (Length, Beam)	RF	0.9934	0.9699	1360.2618	3015.9177	627.2374	1302.5491
	XGBoost	0.9917	0.9648	1523.9397	3262.3938	942.4226	1515.6529
	poly2	0.8995	0.9008	5315.7676	5474.9995	3124.0154	3245.4752
	LM	0.7924	0.7847	7637.8210	8067.4960	5432.4080	5712.9536

Table 5.8. Installed power main engine predicted with Length*Beam and (Length and Beam). The full table is available in the supplementary material. Ordered by test R^2.

Table 5.8 shows the two best results of applying all algorithms and also the linear model and squared polynomial regression for comparison with the results from Huang et al. [4]. It can be observed that in all the cases it is better to leave both variables instead of building the joint *Length* times *Beam*, as with this classifiers have more freedom to adjust how the variables are related. This fact is clear in Figure 5.6, where the curves produced by the polynomial regression are shown. Figure 5.6a shows the polynomial regression for *Length* times *Beam*. It can be seen that the dispersion along the curve is high, showing why the model is not that good. On the other hand, Figure 5.6b shows the same data points with different x axis and the polynomial regression for *Length* and *Beam*. In this case, the surface produced by the polynomial regression is not a line but a 2D plane, which allows fitting better the data.



(A) *ME power* versus *Length* times *Beam* scatterplot. Blue line is the result of the polynomial regression by using *Length* times *Beam* as predictor.



(B) *ME power* versus *Length* scatterplot. The blue line is the result of the polynomial regression produced by using *Length* and *Beam* as predictors.

Figure 5.6. Polynomial regression scatterplot for each case.

Given these results, we can conclude that the results from Huang et al. [4] are not extrapolable to other ship types and that this metric needs to be studied before being used.

5.4 Conclusions

AIS-enabled emission estimation methodologies are growing because of their ability to locate spatio-temporally the exhaust emissions produced by vessels. By combining AIS data and vessel technical databases it is possible to produce an estimation. However, both data sources may contain human and machine errors. If this data is used directly, the estimation produced will be erroneous as well. There are efforts in the literature to clean AIS data. Nevertheless, there are few approaches to provide clean data in vessel characteristic databases and there is no comparison among them.

In this work, we have provided a review and a common framework of the current methods that are being used to in-fill vessel characteristics data and a comparison of the methods. Moreover, we have explored the relationship between variables to then exploit them with predictive algorithms. And finally, we have shown how to apply standard techniques for this kind of use case obtaining good accuracy.

The studied variables have shown strong correlations inside of two groups: variables that are related to the size of the ship and variables related to the engines. Moreover, there are relationships between those groups, e.g. installed ME power with the length of the ship. As the variables are very correlated, the prediction methods provide good accuracy. Finally, we have found which variables are relevant to predict each variable by analyzing the linear model coefficient and the tree-based algorithms variable importance. The findings with these methods were closely related to the linear correlations computed and provided good predictive results. The *type* variable is a relevant variable for most of the target variables. However, the experiments show that it is not a vital part of the information for the prediction in our case.

Regarding predictive methods, *Random Forest (RF)* and *XGBoost* are the methods that work best in the case of predicting the target using two or more predictor variables, confirming that the selection done in the work of Liu et al. [98] is a good approach (*XGBoost*). When using only one predictive variable other methods may outperform them. Methods like *KNN* or *MSV* can provide all the required attributes from the ship information as they are based on using the actual data directly as the prediction. This can also be done using *Neural Networks* if trained with a multivariate output, i.e. trained to predict an array instead of one element. Even though of the convenience of this approach, experiments show that picking the correct set of predictors leads to better results. The standard *KNN* with $k = 1$ has provided better results than *MSV* for every variable. This may happen because the α parameter from *MSV* may be overfitted to the data used in Johansson et al.[26] and could need retuning. Nevertheless, adopting *KNN* as a first baseline method is highly recommended because of its simplicity and availability in ML software.

We have seen that the accuracy of the predictive models decreases with the number of predictor variables. In this case, what we propose to implement is a system in which can use different trained models regarding which variables are available in the vessel to correct. This way ships with more information available benefit from better estimations, i.e. using the *All Variables* model whenever it is possible and if not, fall back to the others.

This study is also subject to some limitations. We have not performed the analysis of other interesting variables, e.g. build year of the ship. However, the methodology to perform this analysis would remain the same. Moreover, *Neural Networks* could be trained to predict a vector of targets, providing the same advantage that *KNN* and *MSV* have. Future work should cover more ship-related variables. For example *deadweight* as covered in the 4th IMO Greenhouse Gas report [97] or the variables used in emission estimation methodology STEAM2 [16]. Finally, as this data is going to be used in further data processes, e.g. emission estimation, an uncertainty measure could be provided for each prediction so that each estimation has an error margin.

Chapter 6

Improving maritime traffic emission estimations on missing data with CRBMs

6.1 Introduction

Maritime traffic is considered an important contributor to primary atmospheric emissions in coastal areas [13] and subsequently to European coastal air quality degradation [14], especially in the North Sea and the Mediterranean basin. It has become a key component for European economy [11], according to the European Community Shipowners Associations (ECSA) in 2015, being sea transportation more fuel-efficient than other modes of transport (e.g. trucks or trains). Nevertheless, according to recent reports by the IMO, this form of transport will continue increasing in the future due to globalization and global-scale trade [12], increasing between 50% and 250% its contribution to the global Green-House Gas (actually 2.5%) by 2050. World governments, specially the European Union and the World Health Organization, are specially interested in advances on detection of emissions, for proper law enforcement towards the Air Quality Standards.

Given this growing tendency, *Smart Cities* need to be enabled to know how much pollution do the citizens suffer and act accordingly. In this case, ships can be seen as connected things like in the *IoT* paradigm, informing about position and other characteristics with which exhaust emissions can be estimated. With all this information the cities will be able to evaluate pollution and propose more informed measures in order to have a healthier city.

Data from ship positioning and maneuvering is required to compute the emission components produced by maritime traffic and to understand levels of pollution from Environmental Research modeling techniques. This data is obtained from the AIS, a GPS based tracking system used for collision avoidance in maritime transport as a supplement to marine radars, providing for each vessel its unique identifiers, GPS positioning and speed among other information.

Well known and validated state of the art techniques to estimate emissions are using this information plus ship engine characteristics databases, like Jalkanen et al. [1, 16]. Doing so enables spotting sources and amounts of emitted pollutants. Such estimations result in large scale simulations and complex physics models, that feed from features like *speed* and *installed engine power*. Other techniques like the HERMESv3 [25] model in CALIOPE project [119], one of the trusted sources of Air Quality estimations by Spanish and Catalan Governments, performed at the Barcelona Supercomputing Center (BSC), use emissions reports, ship estimated routes and profiles by ship type for calculating ship emissions. While the HERMESv3 uses estimated routes, AIS-based estimations can place the pollutants estimations accurately as it uses the actual ship placement, therefore improving the overall precision of the system.

However, AIS data may be incomplete or faulty, e.g. information that could be used to enhance physical models like the ship operational mode (e.g. cruising) is incorrect, missing or poorly detailed too often, so are usually discarded when modeling emissions. This also may be the case of commercial databases that provide the required ship engine characteristics.

Further, dealing with AIS data in populated regions is not trivial, as the average frequency of signal emission is of one message per each six seconds on average. Only in the coastal region of Barcelona represents 1.5 million registers per week. Processing this data in a periodical basis requires employing *Big Data* techniques, understanding *Big Data* as those situations where big volumes of input overwhelm our commonly used methodologies, making us to change them for techniques designed towards automation, scalability or approximation. Applying the complex physical models and simulations over such amount of data makes the problem more complex, requiring supercomputing infrastructures on a daily basis for periodical estimations, also predictions for public health and interest (CALIOPE computes 48-hour forecasts for all European continent).

To enhance such estimations, allowing better enabling features, we have available Data Mining and Machine Learning (ML) techniques, to refine, correct and fill missing data, allowing better accuracy on current air quality methods, also allowing experts on using once discarded features on physics models with higher confidence on the results. Data Mining provides consolidated techniques for analyzing such data, extracting relevant values, frequent and rare patterns, and also model behaviors. Most of those wanted patterns are not trivial or present at simple sight, even they can be found across huge amounts of data, unable to be handled exclusively by human experts. Considering the AIS obtained profiles for each ship of any size and characteristics as multi-dimensional time-series representing their behavior, we can discover new latent features that can enhance the AIS data-set towards modeling emissions. There are several approaches for mining patterns on time series, e.g. stream mining methods for time-changing data [120], series-aware neural networks as CRBMs [121] or *Recurrent NNs*, even Hidden Markov Models for time-series modeling [122].

This work provides a methodology to enhance the obtained AIS data-sets by cleaning, treating and expanding some of its features using domain knowledge, to produce better emission models and correct emission inaccuracies. The proposed methodology focuses on using CRBMs to boost clustering and prediction algorithms, to improve the correctness of features like ship main engine power, navigation status and ship category, from each ships navigation traces. The decision of using CRBMs is based on their capacity to deal with AIS as multi-dimensional time-series [89], also encouraged by the methodology proposed by Buchaca et al. [123] used for detecting phase behavior patterns on time-series. The CRBMs are used to extract and cluster temporal patterns, also to expand features from the time series, allowing non-time-aware predictors better accuracy. Our methodology combines CRBMs with clustering techniques (i.e. k-means) and prediction techniques (e.g. Random Forests, Gradient Boosting, Lasso) towards predicting and characterizing the engine installed power, the navigation status and ship types from their traces, for vessels do not have any of those attributes or that provide them incorrectly.

To summarize, the contributions are the following:

1) Generate feature representations of local behavior of ship movements using CRBMs. This new feature representation is a key building block for 2) and 3).
2) Ship type and main engine installed power missing values correction for better emission estimations using the previously generated features plus machine learning algorithms.
3) Provide an initial step to correct and improve Navigational Status AIS feature, using the generated features plus a clustering technique.

In particular the second contribution of this chapter links with the work done in Chapter 5, where the focus was on correcting data when other static ship attributes are present. When there is no static ship attribute available for a ship that has AIS data available, this other approach has to be used. In this sense, we reconstruct this static data out of the available dynamic data, enabling the model to work with more data, i.e., doing less assumptions about the ship. Further, this contribution can be computed completely ant the Edge following the contribution in Chapter 4 as the AIS data is produced there.

The current approach has been tested using real AIS data provided by the Spanish Ports Authority (*Puertos del Estado*), complemented by ship and engine characteristics coming from the Ship database provided by IHS-Fairplay. We validate and test our approach to enrich and complete data by comparing it against the current state of the art approaches with a methodology based on Jalkanen et al. [1, 16] in scenarios of missing data, in a supervised manner to get emissions computed with real and complete data. After the aggregation of the emissions, we show that when enhancing the data with our method we are able to detect a 45% of the previously undetected emissions (152.95 tonnes out of 343.47) when applying the proposed standard procedure. We also show how other faulty features like Navigation Status and Ship Type can be corrected or improved.

This chapter is structured as follows: Section 6.2 describes the data-set, its features and details. Section 6.3 explains our methodology and machine learning pipelines. Section 6.4 details the experimentation and validation of our methodology. Finally, section 6.5 summarizes this work, and presents the future work.

6.2 Data Preparation

Cleaning and interpolation

First of all, rows with incorrect time-stamps are removed if there is no possibility of repairing them. Names, if missing, are obtained from third party ship databases, freely providing such information, such as VesselFinder [124]. Next step is to retrieve each ship time series and then processed it for time regularization and interpolation when required. Data goes through a two step procedure: 1) Produce the time-steps for the desired time granularity, e.g. a sequence of steps of 5 seconds in between; and 2) Using the available data, linearly interpolate the steps generated in the previous operation if the time difference between samples is less than 72 hours[1].

In order to avoid bias or over-fitting on locality when searching for patterns, a new feature is added indicating the relative movement, by obtaining the difference in Latitude/Longitude between each consecutive points. This way we register the movements between registered observations instead of absolute values, having a movement feature free of geographical information. Also, the same procedure can be performed over rotation features, having as result relative rotation movements. However, rotation attributes from AIS are not always available, hence here we created a rotation variable calculated from the GPS traces as they are more reliable.

Another generated feature is the zone location of vessels. Following the information provided from CEPESCA [125], we consider that sea is divided in three zones: coast, fishing area and high sea. These zones respond where bathymetry is below 50 meters, not suitable for fishing and close to coast, between 50 and 1000 meters, where fishing vessels labor, and beyond 1000 meters as high sea.

After pre-processing we have a time series for each ship, with regularized time-steps between observation, and new derived features indicating relative positions and movement, allowing us to compare ships for their positioning and maneuvering, independent of the origin port or coastal point, even from length of some pattern repetitions.

The final features, from now on called *original features*, used for training the CRBM and for comparison in the experiments are the following:

- Ship ID: IMO and MMSI number
- Ship type
- Relative GPS rotation
- Speed over Ground
- Bathymetry as a 3 value categorical variable: coast, sea and high sea

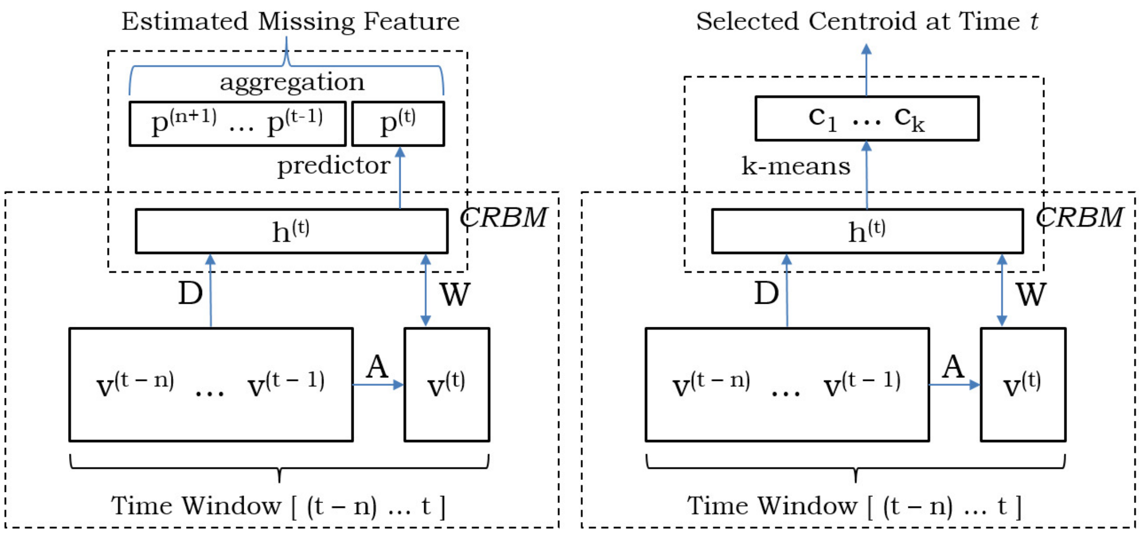
In Section 6.4.2 IHS dataset is used for that particular task along with the previously defined data.

6.3 Learning Models

Our methodology proposes a set of learning pipelines towards improvement of emission estimation. First, we attempt to reconstruct the missing data on ship type and engine power, required for emission estimations. The pipeline for this estimation consists on using the CRBMs to produce a new set of latent features, to be ingested by classification (ship type) and regression (engine power) algorithms. Second, we have the pattern mining ensemble, using the CRBMs along clustering algorithms (i.e. k-means), to find the NavStatus patterns and behaviors on ship traces, intended for future emission models where NavStatus and latent sub-type ship data can be used.

6.3.1 *Data Pipelines*

The CRBMs require as input a time-series window, fed by the traces for each ship. The data-set is composed by time-windows of size $n+1$, considering at each time t an Input $v^{(t)}$ of dimensions $n_v \times 1$, and a history record $v^{(t-n)} \ldots v^{(t-1)}$ of dimensions $n_v \times n$. This data-set is produced by sliding the time-window from time n to T, where T is the total number of recorded steps. Notice that this implies to *burn* the first n records in order to create a history record for Input $v^{(n+1)}$. The outputs of the CRBMs are a vector of size n_h, where h is the total size of generated latent features. This CRBM is connected to a prediction algorithm that will feed from those features, from now on called *Activations*, and compared with the real output variables (ship type and engine power) in a supervised learning fashion. For the scenario of correcting the NavStatus and characterize ships for further usage, we connect the CRBMs to a clustering method *k-means*, fed by the CRBM activation vector. The idea of using CRBMs rely on the fact that those mentioned classical learning algorithms are fed with implicitly time-aware data. Figure 6.1 shows the schema for both of the pipelines.



Estimated Missing Feature
aggregation
$p^{(n+1)} \ldots p^{(t-1)}$
$p^{(t)}$
predictor
CRBM
$h^{(t)}$
D
W
$v^{(t-n)} \ldots v^{(t-1)}$
A
$v^{(t)}$
Time Window [(t – n) … t]
Selected Centroid at Time t
$c_1 \ldots c_k$
k-means
CRBM
$h^{(t)}$
D
W
$v^{(t-n)} \ldots v^{(t-1)}$
A
$v^{(t)}$
Time Window [(t – n) … t]

Figure 6.1. Pipeline for CRBMs, Prediction and Clustering

6.3.2 *Training the Pipeline*

The CRBM is trained with sample series of data, structured as explained previously. Providing the history record matrix to a CRBM as the *Conditional element*, provides the notion of time. This allows training it through data batches without forcing any particular order between batches, as the notion of order is present within each batch. Best practices in modeling and prediction require to split training data with validation and testing data, to prevent the auto-verification of the model, so for this reason we performed this training process with a subset of the available time-series. Also the splits have been performed using the *ids* from the traces (each *id* identifies a single time-series), therefore none of the splits shares a single time step from the other split. Each instance passing through the CRBM is encoded into an activation vector of size n_h. This way, the ship tracking information and history are codified by a n_h-length vector, knowing that as far as a CRBM reconstruction misses the original data by little, such vector contains a compressed or expanded version (depending on the values of h and n) of the current and historical status of such ship.

After training the CRBM, using the activation data-set we train the prediction and clustering algorithms. The principal hypothesis is that ships with similar properties will produce similar activations over time. For the prediction scenarios, we are using well known algorithms like Random Forests, Gradient Boosting, Multi-layer perceptron networks, Logistic Regression and Lasso. We chose those models as the ones performing better or the best models for baseline comparison, discarding those performing worse on accuracy and slower on training stages. As we have a set of activations, from time $n + 1$ to T, for each output value P, each ship traces pass through the pipeline creating a vector of predictions $\hat{p}^{(n+1)} \ldots \hat{p}^{(T)}$, then aggregated and compared against P. For classification we used the fashion (top vote), while for regression we used average and median, as explained on the experiments. For the unsupervised scenarios, we focused directly on k-means algorithm for its simplicity of use and interpretation. We experimented with these models and their hyper-parameters using cross-validation and grid search, on a 6 Intel Xeon 40-core and 128GB RAM cluster.

6.4 Experiments

When installed power and design speed are missing, Jalkanen et al. [1] propose to use the average of those characteristics for the given ship type, usually available in AIS data. However, the installed power has high variance giving an estimator of the installed power with low accuracy. As design speed can be obtained directly from installed power, we focus on the main engine power.

In the case of auxiliary engine, the methodology defines 3 stages: hoteling (moored), maneuvering and cruising. These three operational modes have a power consumption associated, different by type, as there is no other information available for the power estimation of this type of engine. The operational modes are assigned to the trace using the current speed of the ship. Regarding that for each stage we have a different value for a given ship type, it is very important to know the actual ship type. In some cases this type is incorrect or missing in AIS data.

Also, AIS data provides the navigational status of the ship. However, this attribute is not reliable as it is manually set by the crew and it can have delays or be incorrectly set, e.g. in fishing ships the status is always *engaged in fishing* even if they are hoteling. This attribute can potentially give more information about the current usage of the engine than the simpler division done using the speed of the ship, however it needs to be fixed in some cases.

Therefore, the experiments focus on the different pipelines used towards the improvement on emission estimations, as follows:

1. Discriminate ship types: predict the type of a ship given its behavior at every time step.

2. Improving main engine emission modeling: predict installed main engine power when missing values for emission estimations.

3. Navigational status pattern mining: determining the status of vessel directly from reliable GPS coordinates, potentially correcting badly input *NavStatus* values and finding uncovered behaviors.

After several experiments with feature selection and refinement of aggregated features, we selected as input features the *bathymetry*, the *SoG*, and the *GPS-rotation* (the rotation calculated from the GPS positioning, as the feature *rot* is frequently missing or with incorrect values), as mentioned in Subsection 2.4.2. Bathymetry is indicative of the geographical zone where vessels are navigating, if coastal zones, fishing zones, and open sea. Speed and rotation provide the vector of the vessel movements.

Following the proper methodology for training models, we have separated the data-set into training and test, by a random split $0.66 - 0.34$ of the ship series. To measure the CRBM errors (minimal error at data reconstruction), we used the testing series and performed a *simulation*: passing the whole series through the CRBM for activation and reconstruction (i.e. the process of generating the input features from the activations), then computing the MSE of the inputs and reconstructions.

During the experiments we attempted different CRBM hyper-parameters, with a wide range of hidden units in the hidden layer, and different *delay* or history window length. For most of the experiments, we concluded that 10 hidden units with 20 observations of history (1 per minute), provided us the best reconstruction results and differentiating clusters. For the experiment in Subsection 6.4.2 we concluded that expanding features (from 60 visible units to 70 hidden units) produced better prediction results.

6.4.1 *Ship Type Prediction*

In the emission estimation process it is required to know the type of the ship as the emission model use it to determine how much power is the auxiliary engine producing given the navigation status. For instance, in STEAM [1], cruisers are always assumed to constantly use 4000kW of auxiliary engine power. On the other hand, other ship types may be estimated to use 750kW during cruise, 1250kW during port maneuvers and 1000kW while hoteling. Therefore it is very important to know which type the ship is to correctly asses the auxiliary engine consumption.

The following experiment consists on using the CRBM classification pipeline for classifying ship types on a supervised learning scenario. A train/test split of $70\% - 30\%$ is used along with cross-validation for hyper-parameter search. The target class attribute *type* is retrieved from the *typeofshipand-cargo* AIS variable, that encodes it on its first digit. We are dismissing the type of cargo at this moment, and focusing on the type of ship. However this makes some classes to be more difficult to predict, e.g. class 3 “Special Category” has a range of different ships from fishers to tugs, therefore this is a hard classification problem due to the heterogeneity of ships within one class.

As classifying algorithms we used Logistic Regression, also Multi-Layer Perceptron network with different values of neurons at the hidden layer, from 100 to 500, obtaining the best results with 500 hidden units, a rectification layer and the *Adam* optimization method, with 0.9 momentum. Models have been trained and evaluated with three different sets of features as its input: 1) the original features (*bathymetry*, *SoG*, *GPS-rotation*); 2) the original features appended by a time window of 20; and 3) the 10 hidden unit activations of the CRBM.

Figure 6.2 shows the accuracy result on the train and test sets. The CRBM features helped both models to identify ship types and produced more balanced models, e.g. in the case of logistic regression the other features get better results in training, but in test the activations are better and closer to the training error. This can also be seen in Figure 6.3, in which it can be seen that the proportion of *k-means* clustering result, i.e. the proportion of the found patterns inside one class, is different in some ship type, therefore it seems that CRBM enhances data separability for this case.

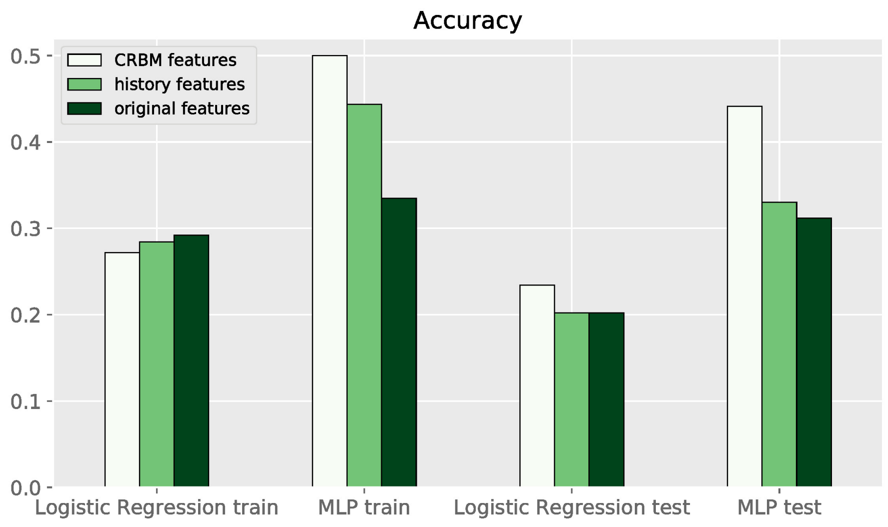


FIGURE 6.2. Accuracy of the ship classification pipeline depending on the features.

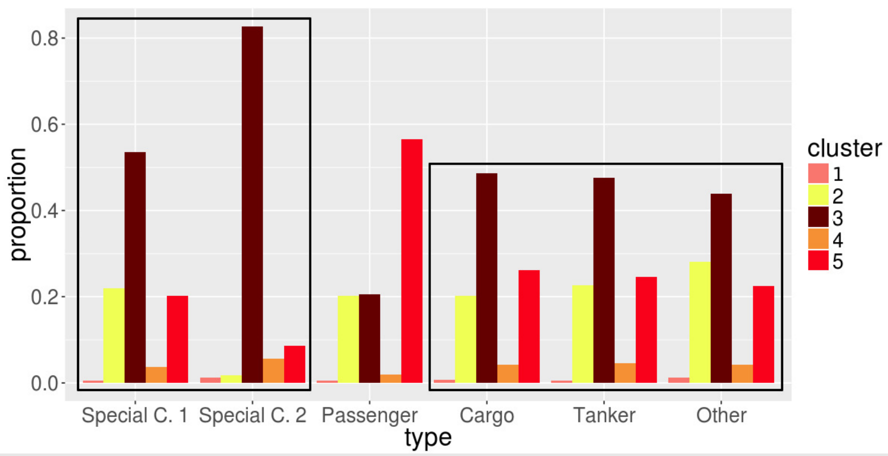


FIGURE 6.3. Proportion of patterns by ship type. It can be seen that some ship types have similar pattern frequency, but others differ drastically. For example, both special categories 1 and 2 (class 3 and 5 respectively) are quite different as 3 is more fishing oriented than the service oriented class 5. On the other hand, carrier ships and others are similar.

6.4.2 *Improving main engine emission estimations on the presence of missing data*

In order to estimate the emissions that a ship produces some characteristics of the engine are required, as shown in the work of Jalkanen et al.[1, 16] with their STEAM methodology. In particular, one of the most important attributes is the main engine power. This data is available in commercial databases provided by companies, e.g. IHS Fairplay [126], however the data may be missing for some ships.

The aim is to provide correct attributes to an estimation model results that have been already validated when all the information is correctly given. When there are missing attributes, assumptions need to be made. In case that some attributes are missing but other static ship attributes are available, the approach defined in Chapter 6. However, in case that there is no data about the ship aside from the data given by AIS, the suggested approach in STEAM methodologies for main engine power is to use the average by ship type. It is a simple and effective solution but not the best as the variance in installed power by ship type is high. We want to use the ship trace along the ship type, data provided by AIS to estimate the ship characteristics needed for pollution estimation, in this case the already mentioned main engine power.

The experiment has the same setup as the use case of Subsection 6.4.1 in terms of validation framework. In this case, the IMO number is used as identifier, as the IHS data-set only contains ships with “class A” transceivers and does not provide an MMSI field, hence we have to discard all the ships that do not provide an IMO number for this experiment. We predict the main engine and we validate it using a separated test set. We have trained a new CRBM model specifically for this use case. The best results were obtained with 70 hidden units and with ensemble vote function *median*, which is more stable than the mean in the experiments using the CRBM activations.

For this experiment we tested different algorithms, e.g. Random Forests, Gradient Boosting and Lasso Regressing, the average of values per type as proposed by Jalkanen et al. [1], and the global average. The model that produced the best result was Random Forest with 200 estimators and no limit in number of features and depth, as can be observed in Table 6.1.

	Median		Mean	
Model	Train	Test	Train	Test
Global avg	8815.28	13279.71	8815.28	13279.71
Type avg	7904.08	11870.83	7904.08	11870.83
Lasso act.	8562.75	12744.20	8562.75	12744.20
Lasso hist.	8430.93	12933.68	8237.30	12616.83
GB act.	6365.63	10817.70	6652.80	11130.79
GB hist.	6336.83	11288.47	6269.50	11185.10
RF act.	**854.42**	**8866.62**	**1328.32**	**9228.00**
RF hist.	2178.95	10001.44	2499.74	9790.29

Table 6.1. Main engine installed power regression error (Mean Absolute Error) for the best configuration found for each model. Votes aggregated with mean and median.

To see the actual impact of the approach, the emissions are estimated with a methodology based on the STEAM model[1, 16] using the power estimated by the best regressors found before (history and activations), the mean of the ship type and the real value. In Table 6.2 it can be observed that our approach is closer to the estimated with the real values than the best model with the original input plus history and the average by type. In fact, the proposed methodology is 152.95 tonnes closer than the estimation using the average, detecting around 45% of the otherwise undetectable emissions, also 62.15 tonnes than the best model using the original data with history. Notice that this data-set covers 1 week of data and the emissions are evaluated over 31 ships from the test set.

In terms of the overall percentage of pollution regarding the real, we can see that there is between an 7.9% and 10.2% of improvement from plain prediction method and between a 23.7% and 25% from the baseline in all the pollutants, as can be observed in Figure 6.4.

There is still room for improvement as around a 31% of pollution is yet to be covered, however this is not a trivial task as the variability of installed power is high.

Used value	SOxME	NOxME	CO2ME	PM ME
Real engine values	0.38	13.31	598.13	0.16
Prediction with activations	0.26	8.82	412.27	0.11
Prediction with history	0.23	7.79	351.19	0.10
Type average engine	0.17	5.64	262.63	0.07

Table 6.2. Estimated pollution in tonnes for each component, using the test-set individuals with the different input values.

6.4.3 *Navigational Status pattern mining*

The *NavStatus* feature (Navigation Status) is a value manually introduced by the vessel crew. In regular cruisers or passenger boats, it is expected to be updated in a regular procedure, while most of the fishing

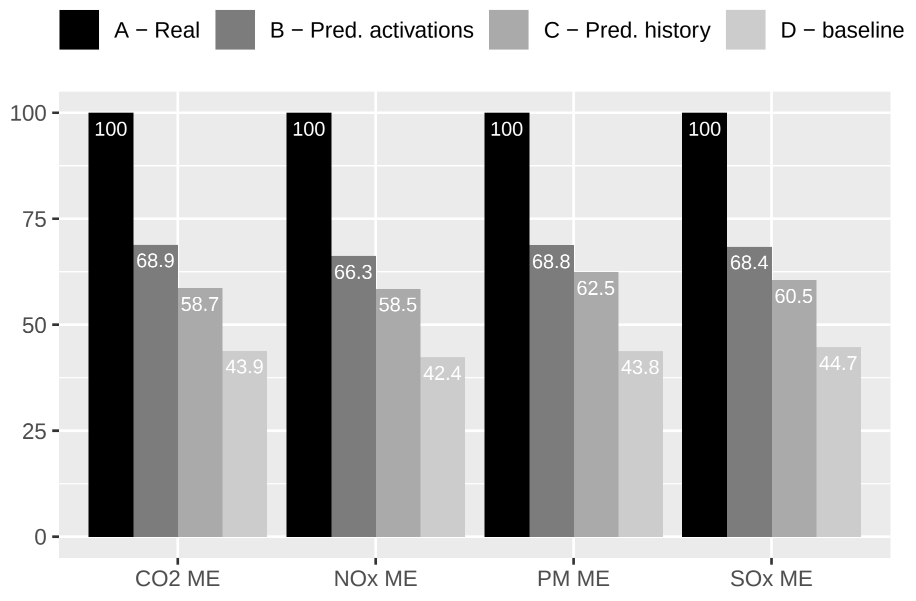


Figure 6.4. Percentage of pollution covered for each method and pollutant with the predicted main engine power. The real data marks the 100%.

ships do not update it and keep the same value always even though they change of operational mode.

This attribute is essential to estimate the power usage of the auxiliary engines of the ships which are not reflected in the ship's speed, contrary to what happens with the main engine power. The ideal situation would be to use this attribute as proposed in Figure 2.4, however it is not being used directly in the emission modeling literature as it is not reliable. Instead, a 3 level operational mode surrogate variable is used. This variable is based on speed limits which define three states: moored, maneuvering and cruising. Navigational status provides more information about the usage profile of the ship, therefore it is interesting to explore this attribute and expand.

This use case proposes to focus on using the cluster labels as a surrogate for the *NavStatus* indicator, to be compared to the real one and correct it when unavailable or considered more reliable. For this, we feed the *k-means* algorithm with the CRBM activations as previously mentioned. In this case, we selected $k = 4$ as hyper-parameter (not considering *burned* samples for initial history, marked as $Cluster 1$), as lower k only detected differences between movement and resting, and higher k produced very similar clusters.

	At anchor	Engaged in fishing	Moored	Not under command	Restricted maneuv.	Undefined	Under way using engine
1	0.03	*0.18*	0.09	0.00	0.00	0.03	0.67
2	0.07	*0.17*	**0.50**	0.00	0.00	0.04	0.22
3	0.02	*0.26*	0.11	0.00	0.00	0.06	**0.55**
4	0.11	*0.22*	**0.47**	0.01	0.00	0.05	0.14
5	0.06	*0.24*	**0.32**	0.00	0.00	0.04	**0.34**

Table 6.3. Clusters vs. NavStatus labels. Values are normalized per row. Notice that Cluster 1 refers to the *delay* data not classified

Table 6.3 shows the NavStatus vs. clustering, grouping those stopped due to anchoring and mooring, those stopped due to fishing, and those in movement. Such results allow us to validate the cluster labels: Cluster 1, as mentioned before, is the status for the data used as initial history, not classified; Cluster 2 refers principally to vessels mooring and in minor measure moving with their engines started, considering this maneuvering; Cluster 3 indicates those that are moving or fishing, and we visually detected that it is assigned to those moving towards fishing positions, or it is mixed with cluster 4 in trawlers; Cluster 4 refers to those moored or fishing, and we visually detected that such status is given to those trawling, moving much slower compared to other speeds (1/4 to 1/10 of regular moving

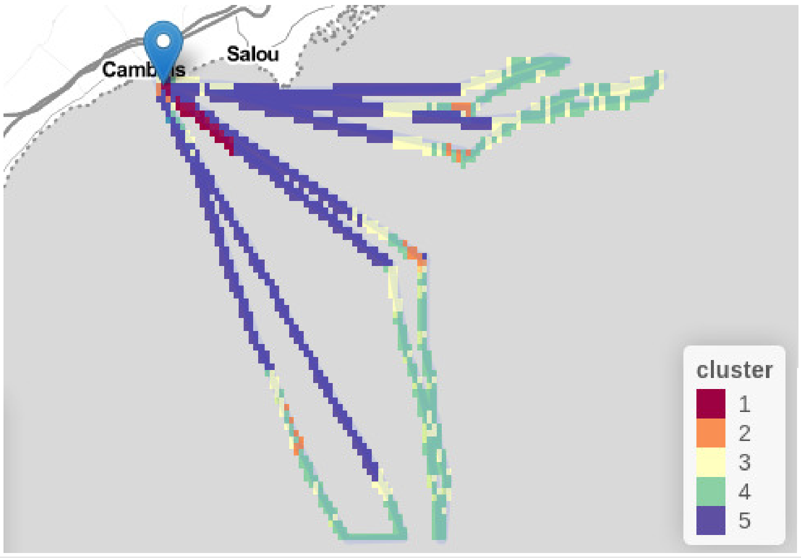


Figure 6.5. Example of status classification by clustering, on a trawling ship. Visualization tool at http://patrons.bsc.es.

speed); Cluster 5 is split between moving, fishing or moored, but by visualization we observed that those labeled as 5 are actually sailing towards fishing positions or returning to port.

As fishing vessels usually set their NavStatus to "engaged in fishing" always, even when sailing or moored, we can determine their "real" status with this classification. Also, for those without status ("undefined"), we can use the assigned cluster label as expected status, and applying approximate NavStatus labels by using the majority label for each cluster: indicating as "moored" if Cluster classifies it as 2, "under way using engine" if Cluster is 3 or 5, "moored OR slow fishing" if cluster is 4.

Even though the correlation of these clusters with the NavStatus is not clear, we can identify new latent behaviors. As an example, we can identify patterns for ships performing trawling, not present in other fishing ships, cargos and passenger boats. In this example, shown in Figure 6.5, we can identify a first cluster (n.2) indicating the maneuvering in port and when shifting trajectories before and after trawling; two clusters (n.3 and n.4) identifying the movements during trawling, slower that regular sailing, that potentially can consume more energy thus more emissions, as they are trawling fishing nets; then a cluster (n.5) for ships speeding towards or from the fishing regions and the port.

For this identification pattern exercise, the validation has been done by expert visual recognition of ship movement traces and location according to port maps [125], and by identifying the vessels registry indicating whether they possessed trawling equipment on board. In future research this extra status found may be used in new emission models.

Finally, this approach shows potential to be applied in other contexts for uncovering behaviors, e.g. analyzing patterns on road traffic mining or other kind of data-set containing GPS.

6.5 Conclusions

Computing the pollutant emissions from maritime traffic is an important issue for coastal cities air quality, as indicated by research in environmental sciences, also a major concern for world governments and global health organizations. Current state of the art techniques to model those emissions are based on processing AIS data, from ships traces, to complement large air quality simulations currently performed in supercomputing centers. The principal problem comes when usually that data is incomplete or incorrect for a large amount of vessels.

In this chapter we presented a methodology for enhancing AIS data-sets by correcting and expanding its features, towards producing better estimations when using the latest emission models.

Our proposed methods focus on using CRBMs to boost prediction and clustering algorithms, used for complete crucial missing data required for producing those estimations. Experiments show that ship type and navigational status may be corrected on missing data scenarios. Moreover, they show that navigational status can be expanded with new uncovered behaviors. Finally, Experiments have proved that our method is able to estimate emissions than those proposed by the current emission models, detecting around 45% of the usually undetected emissions when the required features are not available. As the CRBM architecture and the pipelines are computationally light, it is possible to use them along with the Chapter 4 contribution, the distributed ML system. It provides the near-realtime system with online missing data correction, along with an inferencing process containing the Chapter 5 contribution, so that the emission estimation process can be performed with more accuracy.

Part III

EPILOGUE

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis has aimed to explore the possibilities of Fog computing systems with Machine Learning to enable near-realtime analytics and data processing. This aim was described with the following thesis: **It is possible to build a distributed system in a Fog computing environment to apply complex data processes (i.e. improve data fitness for use, estimate emissions and predict future values) in a Smart City setting with results in a near-realtime fashion.** Figure 7.1 shows how this thesis is explored. The first contribution provides a framework to address Data Science problems in a distributed environment like the one presented. This framework enables to do complex computations like ML or emission estimation where the data is being produced, i.e., the *Edge*. In it we saw how to adapt the Data Science pipeline in a *Fog* computing environment, also called the *Edge-Cloud* continuum. This kind of architecture can be effectively applied to IoT problems. It is shown that Federated Learning (FL) is fitting for this setting and proven that it works for predicting road traffic. From this point, we are able to distribute computing at the *Edge* and build centralized models by training models locally and then merging them. This approach has the advantage that the data does not leave *Edge*, reducing network usage and increasing privacy. The other two contributions work towards improving the fitness of the data received and estimating emissions with the improved data, i.e. addressing the data preparation step. These two contributions can be seen from the pure Data Science point of view. In this sense, both contributions address correcting data from different starting points.

If the whole system is used, the users are able to improve the fitness for use of data, predict future outcomes, and perform other processes like emission estimation at the *Edge*. The first functionality (contribution 1) enables knowing the status of the city and the status that we will have in the short term so that the monitoring agencies can act accordingly. The second functionality (contributions 2 and 3) helps to correct the incoming AIS and the registry data, as both sensor data and human input data can be error-prone. Without correct data, it is not possible to produce correct derived information like predictions or emission estimation. The more uncertainty we have in our data, the bigger the uncertainty we will have in the derived results, and errors will increase producing a snowball effect.

The three contributions of this thesis not only propose techniques, solutions, and challenges that may be found to start building a system that can manage the incoming data in an IoT/Fog computing scenario but are also interesting for other data systems and architectures. This system is able to gather, correct, and make use of data where it is produced. Compared to a classic *Cloud*-based system, this approach provides lower latency and distributes the computational requirements.

In the following sections, the conclusions for each contribution will be summarized, final remarks about the overall contribution will be given and future work will be described.

7.1.1 *Contribution 1: Machine Learning distribution for traffic prediction*

The first contribution explored the possibility of using ML in a distributed system setting and how can we set up a Data Science pipeline in this kind of environment. When we speak about ML is not

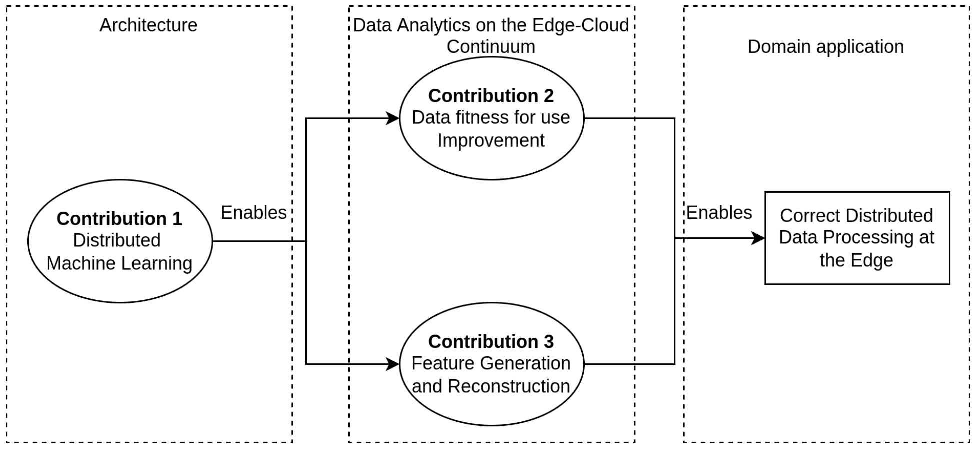

Architecture
Data Analytics on the Edge-Cloud Continuum
Domain application
Contribution 1 Distributed Machine Learning
Enables
Contribution 2 Data fitness for use Improvement
Contribution 3 Feature Generation and Reconstruction
Enables
Correct Distributed Data Processing at the Edge

FIGURE 7.1. Thesis contribution interaction diagram. Contribution 1 offers an architectural framework for distributed ML that can be used in the other two contributions.

uncommon to think about a centralized application that learns from all the data available. However, this may not be the best approach for all the different scenarios. In previous works we explored the concept of local models and training local models and merging them [21, 100], however, in this work we went a step further.

Whenever a central model is required, Federated Learning (FL) approach can be used. FL framework provides a mechanism to train models locally and then merge them by a central authority. With this approach, we can avoid network failures, as the models are trained and used locally. When a network failure occurs, the *Edge* node can remain independently trained and using the current model it has and when the network is again available it can be synchronized again. Also, the network bandwidth usage is reduced, as only the models are sent through the network. The data, if not required to be saved completely or in aggregations, stays in the Edge node. In a mixed setting, we can train with the finer granularity data and report aggregations to the central node. Even though the model in a global sense has not seen all the data, the local models do. Merging models provide a model comparable to a centralized model trained with all the data and, in some cases, this model is better than the original due to the regularizing effect of the merging [113]. In this contribution, a brief guideline on how to select the hyper-parameters was provided to have a successful FL training process, especially for the number of epochs and training rounds. The approach was also tested with two low-powered devices and a Xeon machine. The two low-powered devices produced good enough results in a reasonable time for this scenario, confirming the viability of the approach in resource-constrained Edge nodes. The GPU-enabled low-powered device performance was lower than the CPU-based one as the overhead of transferring the data from RAM to video RAM was higher than the performance using the GPU. This may not be the case for other workloads as, for instance, the image data workloads could perform better in this kind of device. This is due to the number of operations and the amount of data is managed at each CPU/GPU operation.

7.1.2 *Contribution 2: Estimating missing data for emission modeling*

Chapter 5 provided a review of methods to estimate ship missing registry data, a comparative analysis of these methods and standard machine learning methods, the variable importance for each attribute to be fixed, and a guideline on how to fix this kind of data.

This chapter provided a key element to enable the emission estimation process in general, as incorrect data lead to incorrect estimations. This contribution wanted to fill the gap between the different proposed methods for the ship registry data and the practitioners, establishing a fair comparison of the methods. Moreover, with models that enable knowing the variable importance to predict the other variables, a table of variable relevance is given. With this, practitioners will know which variables are good to start with for doing this process.

Because of the high availability of the ML algorithms used, implementation in their systems will be easy enough to have at least a baseline running in a short period of time. As these methods are also fast to produce outputs if they have a favorable configuration, we can utilize them in low-powered edge devices instead of in the backend, correcting erroneous data locally.

Finally, as the study is done departing from using all the variables available to using only one variable, a multi-tier prediction method can be built using different models regarding which data is available and correct. Notice that this contribution can be located both in the *Edge* and *Cloud* in the final system. However, as the data records are highly static, it is more suitable for the *Cloud*.

7.1.3 Contribution 3: Feature extraction and missing data estimation from alternative data sources

Chapter 6 showed how to build new features from an available time series. In particular, from ship trajectories, movement patterns are extracted by grouping codes derived from available data time windows. These patterns are considered the behaviors of the ships at each time point. These patterns were used to predict the type of ship, generate a new variable related to the navigational status of the ship, and to predict the main engine. Even though this contribution is exploratory, it provides useful results for the emission estimation process. This method can be used when no registry data is available. It can be used as part of a methodology along with the processes discussed in Chapter 5 to provide a complete missing data estimation solution that considers the availability of data for each ship (from no data available to one missing field). Considering that this approach is based on the one used in Chapter 4, this methodology is well fitted for Fog computing settings.

Regarding the application for emission estimation, the method showed that it can outperform the general assumptions when ship registry data is not available. Even if the complete method is not used, using the ship traces to estimate the engine power is better than assuming the average of engine power for the ship type. With the full methodology, we were able to estimate 45% more pollution. Given the variability of the main engine attribute, it is difficult to estimate it with a high level of accuracy. However, this method is a good start towards this task.

A final remark about this approach is that with these new uncovered behaviors we may be able to generate a new more refined Navigational Status value that has a finer granularity than the one already available in the AIS data. In particular, most of the fishing ships emit the *Engaged in fishing* even though they may be hoteling at the port. With this new variable, we can see what these kinds of ships are doing. Finally, with respect to the *Fog* architecture, this data preparation process is well suited to run in the *Edge* as the input data is the AIS data which comes in streams directly from the ships.

7.1.4 Overall conclusions

The three contributions work in different directions towards the same objective: providing a suitable framework and techniques to build a near-realtime system with error correction, forecasting capabilities, and feature generation. In a system like this, the fitness for use of data is relevant as derived information will carry the original errors and even multiply them, losing then accuracy. Therefore these contributions have to be understood holistically having two parts: A way to distribute computing with *Fog* computing (Computer Architectures) and a way to adapt the data processing pipeline to this environment with a special focus on data preparation (Data Science).

From the user perspective, policymakers like the city councils need to have actionable knowledge to properly address the city issues. Forecasting provides them with a glimpse of a possible future so that they can act accordingly in the short term. Nevertheless, this is not only useful for policymakers, as the companies maintaining the infrastructure may also benefit from this capability, e.g. predicting when a device or the network is going to fail and deploying countermeasures. Finally, the feature generation procedure enables the development of new markers that can be useful to generate new information or to refine the actual data that the system has. This can help provide extra information for future tasks that the system tenant may have.

These functionalities were tested in a low-powered device distributed system with network fault tolerance, mimicking a realistic system that may be available in the real world. This work provided an analysis of alternative ways to make use of distributed ML and its application in data preparation. The capabilities were evaluated with two real use cases: road and ship traffic. In particular, for ship traffic, there was an emphasis on correctly evaluating the data to provide good exhaust emission estimation. Both use cases were satisfied by the proposed methods and architecture. The data preparation processes produced data ensuring that is fit for the application, enabling the use of the application. Moreover, when there is no ship registry data these approaches provided a better solution than the standard methods leading to the detection a 45% more of the undetected pollution. Considering what is mentioned before, this thesis offers a framework, references, ideas, and details on how to build a system that can clean data, estimate emissions and predict the future status of the city in a data-centric approach.

Finally, the thesis has opened collaborations inside the Barcelona Supercomputer Center between Computer Sciences and Earth Sciences departments. Moreover, it has also opened collaborations with the Spanish Port Authorities, University of Padua, the Punjab University, and the company NearbyComputing, where a patent extracted from this work is currently in industrial use. The emission estimation module produced for this thesis has been used effectively in the FemIoT project (SIFECAT/RIS3CAT 001-P-00166). The research on FL has been a starting point to important architectural contributions in the INCISIVE and CALLISTO EU2020 Horizon projects (grant agreement No.952179 and No.101004152), in which our expertise in FL is being used for medical and satellite data security and privacy.

7.2 Future work

This work has also its limitations, especially due to time constraints. This section covers possible future lines of research and the application of this thesis.

Complete system integration Even though the emission estimation pipeline is completely implemented, the work done has been performed in an experimental setting. A possible future line of research would be to gather all the knowledge contributed to this thesis and build the complete near-realtime system in real edge nodes distributed in the city or different places on the coast of Spain. We made a Raspberry Pi-based edge device with AIS gathering capabilities which were used for experimental data gathering, therefore it is possible to make such a device and create a network as described in this thesis. Moreover, in a real environment, the failures, in general, must be taken into account. Sensor failures can be addressed using the defined ML at the edge nodes and network errors are also regarded in the present work. However, other ways to mitigate errors can be devised, e.g. node redundancy at the edge.

Ship emission estimation In this work STEAM [1] and STEAM2 [16] were implemented. Also, considerations from other methodologies like HERMESv3 [25] or NAEI emission inventory [95] were included. However, to provide a reliable system more effort should be done in this direction regarding the evaluation of the estimated pollutants Moreover, Chapter 5 work can be expanded to cover more ship-related variables. In particular, it needs to evaluate the usage of the *deadweight* variable as used in the Fourth Greenhouse Gas study performed by the IMO [97]. In terms of emission estimation methodology, STEAM2 [16] makes use of a bigger amount of input variables to produce better emission estimations. These variables should be verified as this work does with the ones from STEAM. Even though STEAM is the default backup method for STEAM2 when there is data missing, it is interesting to apply the methodology to the missing variables so that STEAM2 can be used. Finally, as the corrected data is going to be used in further data processes, e.g. emission estimation, an uncertainty measure could be provided for each prediction so that each estimation has an error margin. This would enable users to evaluate how trustable are the derived values.

Concept drift In the development and execution of such a system, new questions should be addressed. For example, this kind of temporal data is sensitive to concept drift as mentioned in the work of Bifet [106]. Therefore, this should be studied and addressed by building policies on how and when to update both local models and federated models.

Minimization of power Even though the present work explores the usage of low-powered computing resources, in extreme cases we may be required to have devices that consume even less electric power. There is a new interesting trend in the ML area towards using MCUs to perform predictive tasks, e.g. TinyML [58]. Even though there are already systems that can learn from the data in these devices, there is no work to our knowledge that makes use of FL along with several devices.

Further exploration of Federated Learning and Swarm Learning Federated Learning (FL) provides a framework in which learning tasks can be done where the data is produced and without it leaving the system. This reduces avoids sending the data over the network and enables using learning methods with privacy in multi-tenant environments. However, in this work, we have applied the most basic way of merging the learning models. In the work of Li et al. [110] they propose a regression/classification tree-based approach that can establish weights for each local model to be merged. As can be seen in the work of Perez et al. [100], local models may be better fitted to their region because each region may have a different data distribution behind. Regarding this, we may be able to build a set of centralized models that take into account these distributions but remain general enough to predict bigger areas or use models like Graph NN that can take into account this kind of relationship between the data and their position.

On the other hand, using similar ideas to the ones found in the work of Li et al. [110], Warnat et al. [127] propose the *Swarm Learning* framework. This framework is similar to the FL framework but the synchronization mechanism is not centralized and it happens in each node. This way in case of losing back-haul connectivity, the devices that remain connected between them may keep collaborating in their learning task independently.

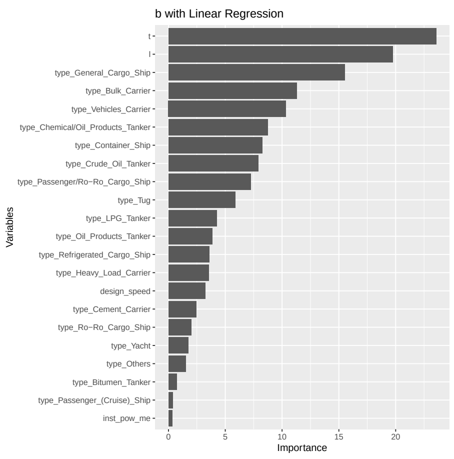
APPENDIX A

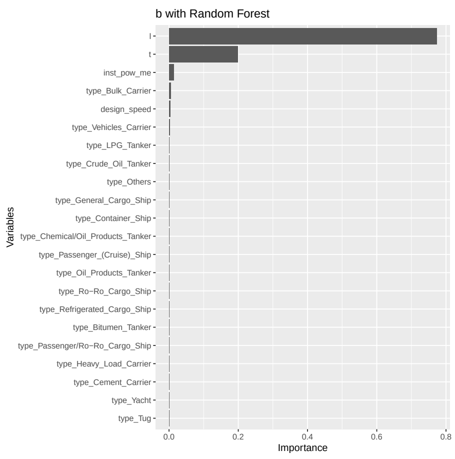
Supporting information for Chapter 5

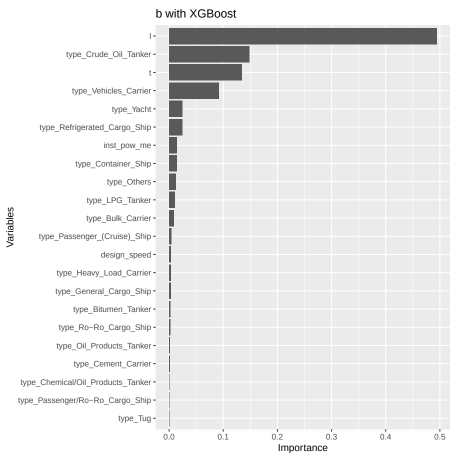
A.1 VARIABLE IMPORTANCE

The importance of the variables in linear regression is shown using the absolute value of the t statistic for coefficient relevance. The importance of the variables in Random Forest and XGBoost is measured as the percentage of how many times they are selected for a split in a tree.

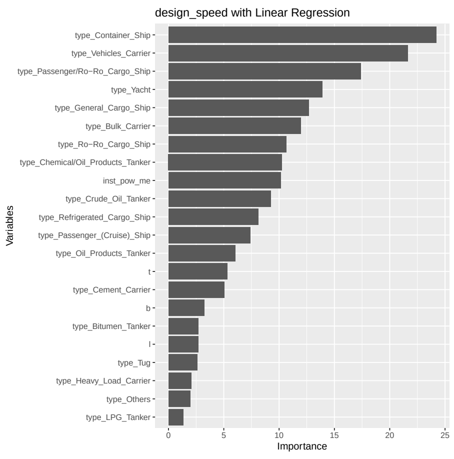
A.1.1 Beam

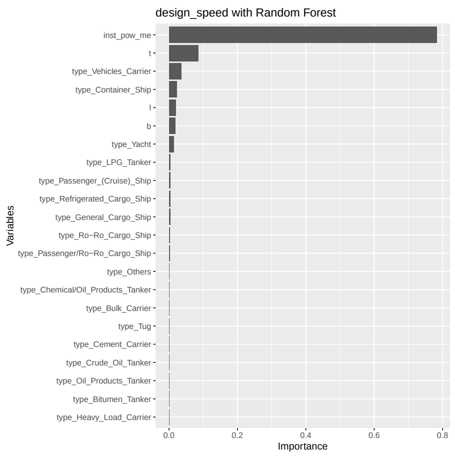


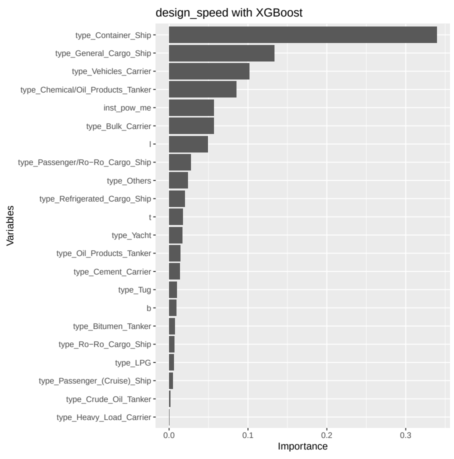




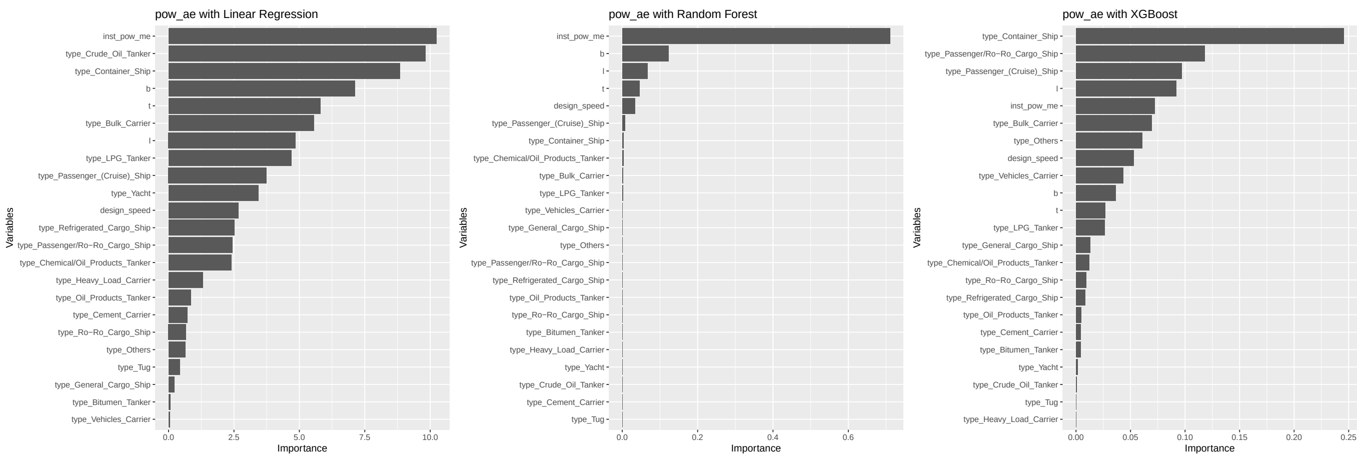
A.1.2 Design speed



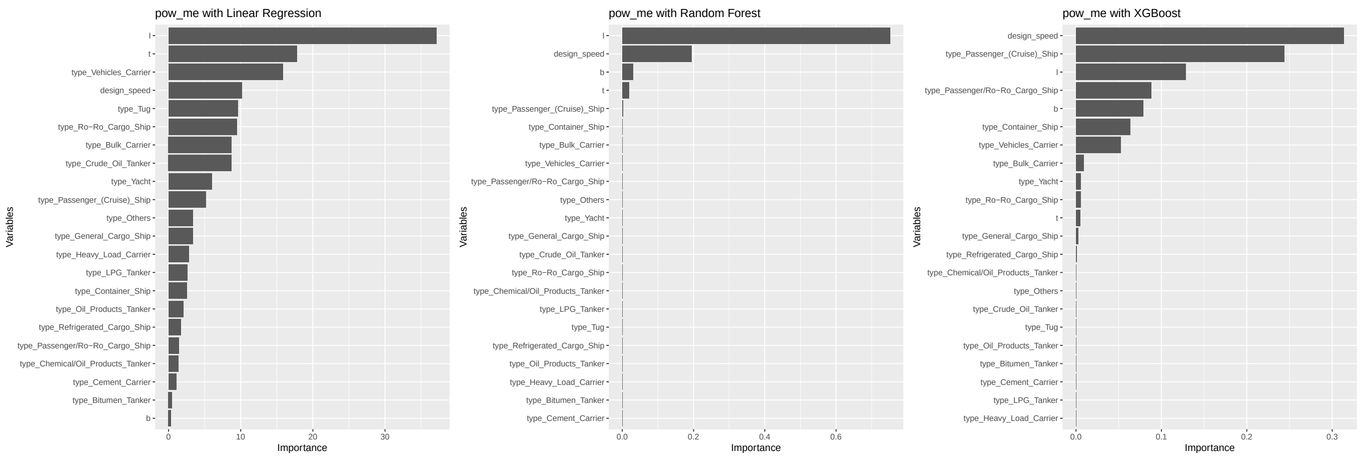




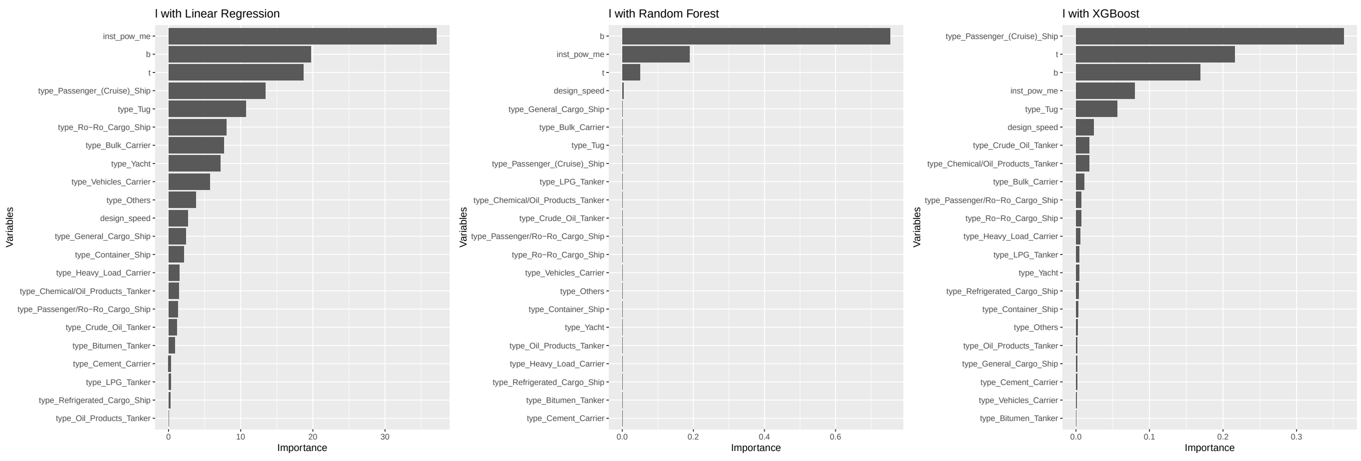
A.1.3 Auxiliary Engine Power



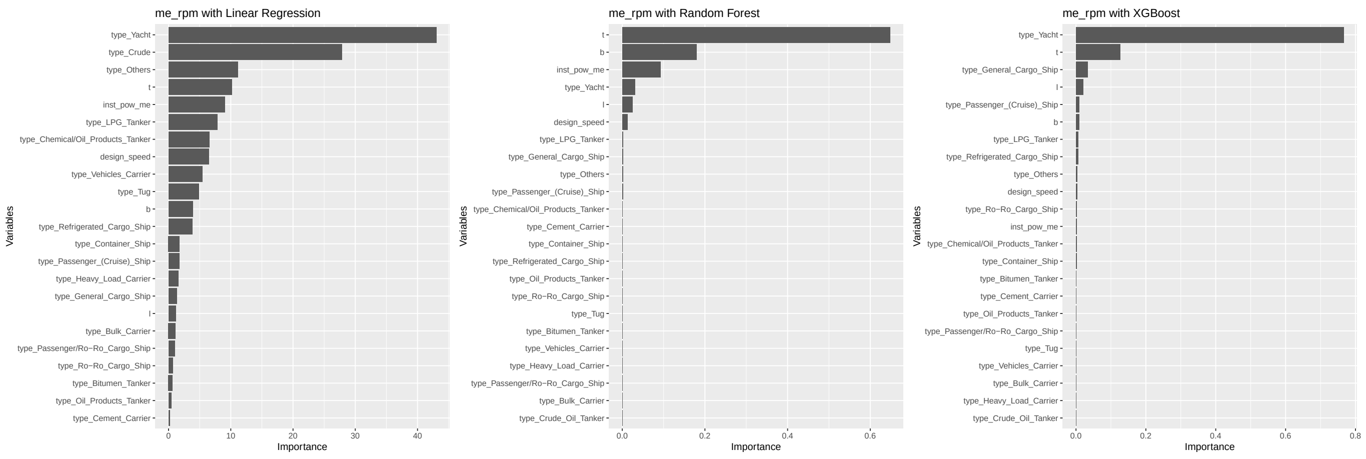
A.1.4 Main Engine Power



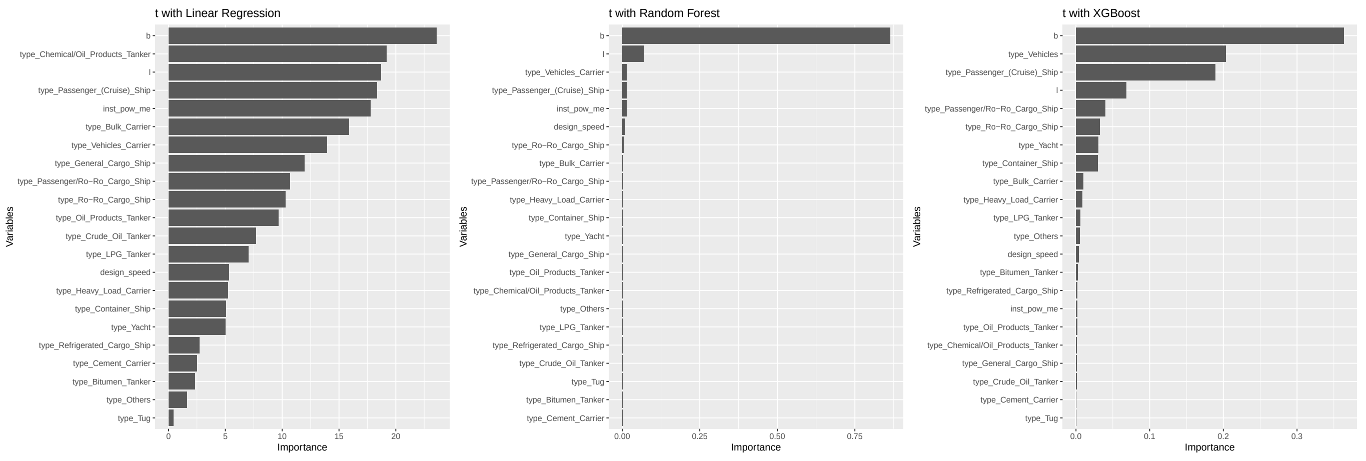
A.1.5 Length



A.1.6 Main Engine RPM



A.1.7 Draft



A.2 Modeling results

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam	XGBoost	1.0000	0.9921	0.0053	0.9476	0.0026	0.4590
	RF	0.9985	0.9918	0.4180	0.9644	0.2094	0.4951
	1_nnet	0.9752	0.9751	1.6797	1.6864	1.2317	1.2385
	2_nnet	0.9725	0.9727	1.7689	1.7650	1.3174	1.3040
	LM	0.9658	0.9664	1.9742	1.9574	1.4669	1.4528
	LinearSVR	0.9648	0.9651	2.0030	1.9955	1.4464	1.4540
	KNN	0.9617	0.9163	2.0872	3.0910	0.9394	1.5892
Design Speed	RF	0.9854	0.9504	0.4849	0.8881	0.2387	0.5133
	XGBoost	0.9728	0.9470	0.6611	0.9182	0.4487	0.6052
	1_nnet	0.9176	0.9144	1.1499	1.1666	0.7033	0.7617
	2_nnet	0.8971	0.9063	1.2851	1.2202	0.7645	0.7811
	KNN	0.9368	0.8878	1.0077	1.3354	0.5259	0.8142
	LM	0.8717	0.8800	1.4355	1.3809	0.9054	0.9016
	LinearSVR	0.8679	0.8788	1.4565	1.3877	0.8878	0.8937
AE Power	RF	0.9823	0.9097	477.5738	1109.8918	210.8580	496.8095
	XGBoost	0.9954	0.8969	242.0613	1185.4034	103.0322	530.4132
	1_nnet	0.8978	0.8700	1146.3312	1331.5715	584.3455	648.2852
	KNN	0.9166	0.8681	1035.5013	1341.1556	457.1699	671.5843
	2_nnet	0.8755	0.8575	1265.2180	1393.7973	641.4432	678.7502
	LM	0.7991	0.8099	1607.4082	1610.0121	870.5261	840.3555
	LinearSVR	0.7783	0.7882	1688.4671	1699.3521	803.0704	797.6508
ME Power	XGBoost	0.9999	0.9852	134.5394	2112.8145	22.7920	779.6078
	RF	0.9979	0.9850	768.2021	2130.9932	319.0958	844.8177
	KNN	0.9920	0.9808	1497.6143	2411.3010	622.7143	1002.6652
	1_nnet	0.9758	0.9740	2606.6319	2802.9684	1367.4226	1546.4405
	2_nnet	0.9759	0.9720	2604.7079	2910.0035	1386.7669	1614.4761
	LM	0.9181	0.9156	4798.3514	5050.3459	3540.9936	3728.3430
	LinearSVR	0.8806	0.8752	5791.9700	6142.0933	3354.1845	3607.7780
Length	RF	0.9984	0.9909	3.0077	7.2957	1.5252	3.7745
	XGBoost	0.9999	0.9900	0.7402	7.6552	0.1760	3.8023
	1_nnet	0.9844	0.9832	9.4250	9.9355	6.8689	7.4538
	2_nnet	0.9769	0.9769	11.4673	11.6457	8.9308	9.1718
	LM	0.9773	0.9764	11.3698	11.7616	8.5005	9.0830
	LinearSVR	0.9766	0.9756	11.5581	11.9503	8.4038	9.0489
	KNN	0.9663	0.9192	13.8470	21.7596	7.0771	12.1295
ME RPM	RF	0.9903	0.9233	45.2185	125.5659	18.6240	53.1499
	XGBoost	0.9995	0.9178	10.0946	130.0155	4.6482	52.2184
	1_nnet	0.9184	0.9036	131.3299	140.7847	75.8189	82.3809
	2_nnet	0.9086	0.8988	138.9945	144.2164	80.3988	81.6040
	LinearSVR	0.8625	0.8683	170.4456	164.5590	110.9466	114.1900
	LM	0.8766	0.8648	161.5082	166.7049	112.0303	117.6057
	KNN	0.8966	0.7500	147.8039	226.6873	54.2122	96.8462
Draft	XGBoost	0.9997	0.9882	0.0592	0.4033	0.0349	0.2209
	RF	0.9974	0.9870	0.1891	0.4238	0.1018	0.2420
	1_nnet	0.9783	0.9758	0.5470	0.5778	0.4032	0.4358
	2_nnet	0.9720	0.9708	0.6213	0.6344	0.4652	0.4899
	LM	0.9598	0.9581	0.7452	0.7592	0.5689	0.5919
	LinearSVR	0.9573	0.9565	0.7680	0.7736	0.5523	0.5792
	KNN	0.9315	0.8504	0.9725	1.4353	0.4640	0.7746

Table A.1. Error measures for training and test for the target variable predicted with all the other variables. Ordered by test R^2.

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam (Length, Design Speed)	RF	0.9975	0.9900	0.5377	1.0659	0.2618	0.5651
	XGBoost	0.9976	0.9893	0.5214	1.1075	0.3092	0.6100
	KNN	0.9904	0.9864	1.0459	1.2453	0.4745	0.6591
	1_nnet	0.9520	0.9513	2.3391	2.3567	1.7915	1.8127
	2_nnet	0.9517	0.9512	2.3454	2.3587	1.7839	1.8114
	LM	0.9390	0.9393	2.6352	2.6317	2.0339	2.0283
Design Speed (ME Power, Draft)	XGBoost	0.9563	0.9193	0.8381	1.1323	0.5812	0.7296
	RF	0.9739	0.9132	0.6471	1.1747	0.4019	0.6501
	1_nnet	0.8767	0.8826	1.4068	1.3661	0.9168	0.9582
	KNN	0.9205	0.8804	1.1298	1.3789	0.5910	0.8444
	2_nnet	0.8679	0.8791	1.4563	1.3863	0.9518	0.9756
	LM	0.6674	0.6830	2.3109	2.2446	1.7474	1.7296
AE Power (ME Power, Beam)	RF	0.9761	0.8981	554.7271	1178.8827	257.6787	550.5837
	XGBoost	0.9934	0.8821	290.2708	1268.1226	85.3203	511.4801
	KNN	0.9139	0.8633	1052.0568	1365.0408	472.4986	678.3114
	1_nnet	0.8179	0.8260	1530.1503	1540.4794	831.0015	817.2586
	2_nnet	0.8064	0.8164	1577.7856	1582.2966	835.1635	842.2111
	LM	0.7512	0.7680	1788.6386	1778.7299	969.6205	960.3790
ME Power (Length, Design Speed)	RF	0.9969	0.9802	933.4769	2443.5418	407.8315	967.2775
	XGBoost	0.9979	0.9781	776.8909	2571.8108	524.7569	1056.9163
	KNN	0.9871	0.9755	1901.8463	2721.2939	768.5421	1143.6786
	2_nnet	0.9599	0.9620	3356.9234	3388.0010	1697.9294	1780.7107
	1_nnet	0.9594	0.9610	3379.2634	3435.1680	1732.0385	1836.4477
	LM	0.8222	0.8208	7068.9221	7360.7513	5446.5547	5667.8594
Length (Beam, ME Power)	RF	0.9966	0.9854	4.3831	9.2533	2.3068	4.9576
	XGBoost	0.9907	0.9800	7.2951	10.8370	5.2606	7.2009
	1_nnet	0.9664	0.9664	13.8348	14.0375	10.1501	10.4048
	2_nnet	0.9622	0.9617	14.6715	14.9932	10.8966	11.2012
	LM	0.9583	0.9561	15.4130	16.0371	11.6735	12.1337
	KNN	0.9650	0.9197	14.1169	21.7004	7.4996	12.3789
ME RPM (Draft, ME Power)	RF	0.9853	0.8721	55.6414	162.1726	22.6584	64.8916
	XGBoost	0.9892	0.8675	47.8553	165.0145	22.8113	66.3813
	2_nnet	0.8676	0.8648	167.2442	166.6835	95.6990	93.7148
	1_nnet	0.8653	0.8609	168.6983	169.0908	107.4685	105.6136
	KNN	0.8608	0.6393	171.4968	272.2922	64.7574	121.2643
	LM	0.6239	0.6388	281.9134	272.4814	199.7586	191.6954
Draft (Beam, Length)	RF	0.9900	0.9622	0.3721	0.7214	0.1892	0.3871
	XGBoost	0.9893	0.9612	0.3849	0.7305	0.2378	0.4133
	KNN	0.9680	0.9429	0.6649	0.8864	0.3453	0.4915
	2_nnet	0.8993	0.8898	1.1788	1.2318	0.8410	0.8882
	1_nnet	0.8971	0.8867	1.1914	1.2489	0.8430	0.8906
	LM	0.8826	0.8726	1.2729	1.3243	0.9694	1.0162

TABLE A.2. Error measures for training and test for the target variable predicted only with the relevant variables. The variables used to predict are written below the variable to predict. Ordered by test R^2.

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam	RF	0.0005	0.0011	-0.0506	-0.0611	-0.0313	-0.0613
(Length, Design Speed)	XGBoost	0.0015	0.0013	-0.2103	-0.0700	-0.1832	-0.1047
	KNN	0.0006	0.0037	-0.0314	-0.1820	-0.0249	-0.0825
	1_nnet	0.0229	0.0230	-0.6475	-0.6464	-0.5835	-0.5942
	2_nnet	0.0146	0.0160	-0.3869	-0.4239	-0.3212	-0.3688
	LM	0.0265	0.0266	-0.6544	-0.6594	-0.5671	-0.5682
Design Speed	XGBoost	0.0277	0.0269	-0.3309	-0.2082	-0.2798	-0.1867
(ME Power, Draft)	RF	0.0220	0.0280	-0.3925	-0.2078	-0.2702	-0.1159
	1_nnet	0.0275	0.0188	-0.1666	-0.1141	-0.1400	-0.1628
	KNN	-0.0264	0.0180	0.1742	-0.1081	0.2326	0.0070
	2_nnet	0.0531	0.0028	-0.3302	-0.0164	-0.3661	-0.1452
	LM	0.2033	0.1944	-0.8699	-0.8490	-0.8367	-0.8152
AE Power	RF	0.0004	-0.0009	-4.9752	5.2422	-8.7550	-16.7539
(ME Power, Beam)	XGBoost	-0.0151	0.0051	237.2241	-27.9569	250.9958	43.4501
	KNN	0.0001	0.0001	-0.8868	-0.4871	-2.3689	-2.3310
	1_nnet	0.0412	0.0283	-184.2369	-130.9249	-146.6328	-108.9719
	2_nnet	-0.0136	-0.0079	54.4307	33.6192	82.1859	49.9145
	LM	0.0392	0.0386	-146.7625	-154.7597	-82.0153	-103.5795
ME Power	RF	-0.0011	-0.0012	150.1328	73.8664	248.2645	137.4452
(Length, Design Speed)	XGBoost	0.0016	-0.0003	-381.3468	18.2249	-276.7815	-62.3720
	KNN	0.0003	0.0003	-22.3113	-17.6821	-36.3525	-53.4362
	2_nnet	0.0143	0.0090	-666.4766	-427.7605	-303.5019	-159.7036
	1_nnet	0.0127	0.0099	-576.0113	-466.4431	-283.3240	-190.4030
	LM	0.0734	0.0722	-1651.9714	-1671.6840	-1331.5225	-1394.5369
Length	RF	0.0019	0.0062	-1.4683	-2.2352	-0.8114	-0.9614
(Beam, ME Power)	XGBoost	0.0069	0.0097	-3.5487	-3.0396	-3.3472	-3.1494
	1_nnet	0.0105	0.0099	-2.3676	-2.2438	-1.4518	-1.4262
	2_nnet	0.0135	0.0137	-2.9168	-2.9703	-1.9030	-2.0084
	LM	0.0136	0.0151	-2.7539	-3.0579	-2.0267	-2.1668
	KNN	0.0001	-0.0001	-0.0283	0.0171	-0.0590	0.0124
ME RPM	RF	0.0115	0.0253	-29.7687	-16.9290	-11.2778	-7.2151
(Draft, ME Power)	XGBoost	-0.0049	0.0297	9.7729	-19.6946	0.5686	-6.1110
	2_nnet	0.0409	0.0302	-28.2491	-19.7805	-16.2814	-10.8410
	1_nnet	0.0379	0.0263	-25.6869	-16.8007	-18.3823	-13.8981
	KNN	0.0107	0.2162	-6.7456	-99.9706	52.4929	1.3610
	LM	0.2386	0.0031	-111.4247	-1.1652	-135.4681	-71.9654
Draft	RF	0.0068	0.0259	-0.1616	-0.3174	-0.0728	-0.1559
(Beam, Length)	XGBoost	0.0092	0.0251	-0.2393	-0.2967	-0.1553	-0.1761
	KNN	0.0061	0.0299	-0.0668	-0.2749	0.0995	-0.0221
	2_nnet	0.0751	0.0827	-0.5846	-0.6162	-0.4035	-0.4243
	1_nnet	0.0785	0.0678	-0.6115	-0.4578	-0.5593	-0.4689
	LM	0.0692	0.0768	-0.4575	-0.4900	-0.3315	-0.3574

Table A.3. Difference of error when adding *type* variable as a predictor with respect to Table 5.4. The variables used to predict (excluding type which is used in every experiment) are written below the variable to predict. Increase in R^2 and decrease in RMSE and MAE means better fit. Ordered by test R^2.

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam (Length)	XGBoost	0.9868	0.9744	1.2264	1.7085	0.8013	1.0654
	RF	0.9805	0.9727	1.4892	1.7648	0.9385	1.1449
	KNN	0.9785	0.9721	1.5636	1.7839	0.8181	1.0666
	2_nnet	0.9166	0.9154	3.0818	3.1067	2.2648	2.2795
	1_nnet	0.9109	0.9120	3.1856	3.1693	2.2747	2.2939
	LM	0.9001	0.9006	3.3731	3.3675	2.4921	2.4923
Design Speed (ME Power)	XGBoost	0.8695	0.8574	1.4473	1.5057	0.9991	1.0932
	KNN	0.8922	0.8561	1.3157	1.5125	0.7975	0.9997
	RF	0.8853	0.8495	1.3572	1.5467	0.8984	1.0560
	1_nnet	0.7529	0.7831	1.9919	1.8567	1.4820	1.4357
	2_nnet	0.7359	0.7678	2.0593	1.9211	1.5473	1.4991
	LM	0.6616	0.6813	2.3309	2.2508	1.8127	1.7850
AE Power (ME Power)	XGBoost	0.9026	0.8624	1119.2508	1369.8897	651.3921	713.7344
	KNN	0.8767	0.8505	1259.3877	1427.8572	611.7369	707.7976
	RF	0.9158	0.8330	1040.4658	1508.7723	576.0408	728.9788
	1_nnet	0.7504	0.7645	1791.5748	1791.8935	978.9485	975.6295
	LM	0.7432	0.7585	1817.1219	1814.8075	1000.7697	1007.3105
	2_nnet	0.7431	0.7582	1817.5333	1815.7456	1006.2254	1013.1059
ME Power (Length)	KNN	0.9665	0.9537	3070.5436	3742.5165	1407.5615	1874.7986
	XGBoost	0.9693	0.9517	2937.0998	3819.8798	1682.1618	2032.9765
	RF	0.9745	0.9517	2675.5138	3820.3715	1473.3089	1941.0761
	1_nnet	0.8564	0.8730	6353.7081	6195.3436	3889.8189	3849.9655
	2_nnet	0.8566	0.8729	6348.0268	6198.7748	3786.8094	3753.9687
	LM	0.7103	0.7116	9023.7655	9336.1202	6694.5049	6891.3160
Length (Beam)	XGBoost	0.9559	0.9473	15.8575	17.5739	9.8878	10.9411
	RF	0.9553	0.9471	15.9653	17.6154	9.9571	11.0681
	KNN	0.9335	0.9288	19.4619	20.4260	10.2588	10.9091
	LM	0.9001	0.9006	23.8586	24.1373	17.5235	17.8596
	1_nnet	0.8995	0.9004	23.9278	24.1684	17.6645	17.9829
	2_nnet	0.9001	0.9002	23.8561	24.1844	17.5174	17.8817
ME RPM (Draft)	1_nnet	0.8671	0.8622	167.5842	168.3189	96.4330	95.5028
	RF	0.9073	0.8597	139.9690	169.8230	78.0129	89.9753
	2_nnet	0.8648	0.8576	169.0427	171.0586	99.3597	97.5212
	XGBoost	0.9206	0.8570	129.5553	171.4625	70.7456	89.7742
	KNN	0.9203	0.8434	129.8078	179.4019	61.9109	87.0557
	LM	0.6123	0.6262	286.2200	277.1728	205.7639	197.9765
Draft (Beam)	RF	0.9367	0.9216	0.9344	1.0391	0.6337	0.7072
	XGBoost	0.9345	0.9211	0.9510	1.0420	0.6555	0.7124
	KNN	0.9241	0.9123	1.0232	1.0991	0.6926	0.7264
	1_nnet	0.8933	0.8879	1.2133	1.2421	0.8778	0.9168
	2_nnet	0.8906	0.8840	1.2285	1.2636	0.9186	0.9569
	LM	0.8820	0.8727	1.2761	1.3237	0.9733	1.0181

TABLE A.4. Error measures for training and test for the target variable. The variable used to predict is written below the variable to predict. Predicted only with the most relevant variable. Ordered by test R^2.

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Beam	RF	0.9957	0.9842	0.6994	1.3413	0.3479	0.7555
	XGBoost	0.9966	0.9823	0.6188	1.4219	0.3228	0.8451
	3NN	0.9874	0.9823	1.1967	1.4226	0.5906	0.8152
	1NN	0.9979	0.9805	0.4920	1.4902	0.0512	0.6413
	MSV	0.9973	0.9730	0.5527	1.7562	0.0598	0.7836
	1_nnet	0.9505	0.9496	2.3753	2.3974	1.7842	1.8258
	2_nnet	0.9493	0.9476	2.4039	2.4455	1.8259	1.8725
	LM	0.9121	0.9124	3.1644	3.1621	2.4926	2.4942
AE Power	RF	0.9729	0.8890	590.2442	1230.4580	281.1902	580.1873
	XGBoost	0.9679	0.8847	642.5920	1253.9018	314.9681	597.1411
	1NN	0.9848	0.8825	441.6284	1265.9871	83.2525	541.7500
	3NN	0.9292	0.8789	953.9023	1285.2560	449.6223	632.9286
	MSV	0.9886	0.8669	382.5517	1347.0451	75.5772	554.3957
	1_nnet	0.8093	0.8194	1565.8429	1569.1679	838.8283	823.0631
	2_nnet	0.7985	0.8089	1609.7925	1614.1963	888.2305	853.7966
	LM	0.6904	0.7046	1995.3682	2006.9952	1293.5314	1261.6120
ME Power	RF	0.9963	0.9799	1025.3560	2465.9112	428.6885	978.8303
	3NN	0.9871	0.9755	1901.8463	2721.2939	768.5421	1143.6786
	XGBoost	0.9889	0.9749	1765.0277	2755.9600	1071.7574	1360.7880
	1NN	0.9995	0.9717	368.2261	2927.1558	77.7506	963.1471
	MSV	0.9995	0.9710	365.3108	2963.1109	71.6225	1041.1629
	1_nnet	0.9607	0.9612	3323.5682	3426.6575	1716.1289	1813.7950
	2_nnet	0.9601	0.9609	3350.3343	3439.1123	1756.7301	1868.6926
	LM	0.8222	0.8208	7068.9221	7360.7513	5446.5547	5667.8594
ME RPM	XGBoost	0.9742	0.8710	73.7837	162.8481	36.6714	82.0743
	RF	0.9759	0.8700	71.3819	163.4616	38.1288	83.3670
	3NN	0.9297	0.8557	121.8638	172.2444	59.0172	91.1690
	1_nnet	0.8302	0.8307	189.4050	186.5293	113.5908	110.3079
	2_nnet	0.8213	0.8289	194.3407	187.5346	113.9136	111.6540
	1NN	0.9963	0.8223	28.0424	191.1055	3.0852	75.2486
	MSV	0.9957	0.7816	30.0128	211.8606	3.5080	85.6100
	LM	0.5209	0.5222	318.1662	313.3820	241.4797	236.8057
Draft	RF	0.9890	0.9541	0.3890	0.7946	0.2411	0.4687
	1NN	0.9963	0.9486	0.2264	0.8413	0.0377	0.3893
	3NN	0.9670	0.9410	0.6747	0.9009	0.3497	0.5210
	XGBoost	0.9553	0.9331	0.7857	0.9594	0.5346	0.6618
	MSV	0.9970	0.9311	0.2029	0.9740	0.0325	0.4515
	1_nnet	0.9275	0.9192	1.0004	1.0547	0.6880	0.7527
	2_nnet	0.9251	0.9155	1.0168	1.0789	0.7119	0.7880
	LM	0.8658	0.8531	1.3607	1.4220	1.0250	1.0796

TABLE A.5. Comparison of the best algorithms found, KNN and MSV. Ordered by test R^2.

		R^2		RMSE		MAE	
Target	**Algorithm**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
ME Power (Length*Beam)	RF	0.9787	0.9307	2444.9302	4576.5184	951.1324	2006.6138
	XGBoost	0.9485	0.9257	3803.4448	4739.4473	2081.3133	2576.2111
	KNN	0.9485	0.9156	3805.4634	5052.3433	1482.7439	2236.7024
	1_nnet	0.7253	0.7497	8786.1284	8699.0627	5153.0976	5164.2718
	2_nnet	0.7251	0.7480	8789.1669	8728.6609	5179.9932	5202.7271
	poly2	0.7225	0.7473	8831.9435	8740.2747	5310.2186	5273.8748
	LM	0.7017	0.7180	9155.9720	9232.4103	6155.3972	6231.0306
AE Power (Length, Beam)	RF	0.9934	0.9699	1360.2618	3015.9177	627.2374	1302.5491
	XGBoost	0.9917	0.9648	1523.9397	3262.3938	942.4226	1515.6529
	KNN	0.9810	0.9630	2308.7373	3343.5428	1083.6797	1572.7905
	2_nnet	0.9197	0.9269	4749.8280	4700.3711	2961.2582	3018.5387
	1_nnet	0.9193	0.9210	4762.0252	4888.1198	2896.2361	3046.5558
	poly2	0.8995	0.9008	5315.7676	5474.9995	3124.0154	3245.4752
	LM	0.7924	0.7847	7637.8210	8067.4960	5432.4080	5712.9536

TABLE A.6. Installed power main engine predicted with Length*Beam and (Length and Beam). Ordered by test R^2.

A.3 Glossary

AE	Auxiliary Engine	24
AIS	Automatic Identification System	5
AQMS	Air Quality Modelling Systems	14
BSC	Barcelona Supercomputing Center	4
BS	Batch Size	48
CALIOPE	CALIdad del aire Operacional Para España	4
CNN	Convolutional Neural Network	28
CPU	Central Processing Unit	28
CRBM	Conditional Restricted Boltzmann Machine	9
CoG	Course over Ground	23
EIONET	European Environment Information and Observation NETwork	4
FCD	Floating Car Data	20
FL	Federated Learning	7
GB-RBM	Gaussian Bernoulli RBM	11
GPS	Global Positioning System	4
GPU	Graphics Processing Unit	8
GRU	Gated Recurrent Unit	8
GT	Gross Tonnage	5

HERMES	High-Elective Resolution Modelling Emission System	4
HPC	High Performance Computing	29
IID	Independent and Identically Distributed	42
IMO	International Maritime Organization	5
IoT	Internet of Things	3
KNN	*k* Nearest Neighbors	31
MAE	Mean Absolute Error	57
MCU	MicroController Unit	8
ME	Main Engine	24
MLP	Multi-Layer Perceptron	55
ML	Machine Learning	3
MMSI	Maritime Mobile Service Identity	15
MSE	Mean Squared Error	31
MSV	Most Similar Vessel	31
NAEI	National Atmospheric Emissions Inventory	31
NN	Neural Network	28
PM	Particulate Matter	4
RBM	Restricted Boltzmann Machine	11
RF	Random Forest	56

RMSE	Root Mean Squared Error	45
RNN	Recurrent Neural Network	13
RPM	Revolutions Per Minute	17
SFOC	Specific Fuel-Oil Consumption	16
STEAM	Ship Traffic Emission Assesment Model	15
STEMI	ST Elevation Myocardial Infarction	4
SoG	Speed over Ground	22
SWA	Stochastic Weight Averaging	42
VAR	Vector Auto-Regression	43
VHF	Very High Frecuency	15

Bibliography

[1] J.-P. Jalkanen, a. Brink, J. Kalli, H. Pettersson, J. Kukkonen, and T. Stipa. A modelling system for the exhaust emissions of marine traffic and its application in the Baltic Sea area. *Atmospheric Chemistry and Physics Discussions* **9** (4), 15339–15373 (2009). ISSN 1680-7375. doi: 10.5194/acpd-9-15339-2009. Cited on page/s xv, 5, 8, 15, 16, 17, 18, 20, 29, 30, 31, 51, 52, 57, 67, 68, 69, 71, 72, 73, 74, 84.

[2] Miyeon Jeon, Yoojeong Noh, Kyunghwan Jeon, Sangbong Lee, and Inwon Lee. Data gap analysis of ship and maritime data using meta learning. *Applied Soft Computing* **101**, 107048 (2021). Cited on page/s xvii, 8, 31, 53, 54.

[3] Xin Peng, Yuanqiao Wen, Lichuan Wu, Changshi Xiao, Chunhui Zhou, and Dong Han. A sampling method for calculating regional ship emission inventories. *Transportation Research Part D: Transport and Environment* **89**, 102617 (2020). Cited on page/s xix, 6, 30, 51, 52, 54.

[4] Liang Huang, Yuanqiao Wen, Yimeng Zhang, Chunhui Zhou, Fan Zhang, and Tiantian Yang. Dynamic calculation of ship exhaust emissions based on real-time ais data. *Transportation Research Part D: Transport and Environment* **80**, 102277 (2020). Cited on page/s xix, 30, 31, 51, 52, 54, 63, 64.

[5] Frank Van Lingen, *et al.* The Unavoidable Convergence of NFV, 5G, and Fog: A Model-Driven Approach to Bridge Cloud and Edge. *IEEE Communications Magazine* **55** (8), 28–35 (2017). ISSN 01636804. doi: 10.1109/MCOM.2017.1600907. Cited on page/s 3.

[6] Natalie Mueller, David Rojas-Rueda, Xavier Basagaña, Marta Cirach, Tom Cole-Hunter, Payam Dadvand, David Donaire-Gonzalez, Maria Foraster, Mireia Gascon, and David Martinez. Urban and transport planning related exposures and mortality: a health impact assessment for cities. *Environ Health Perspect* **125**, 89–96 (2017). Cited on page/s 3.

[7] Jordi Bañeras, Ignacio Ferreira-González, Josep Ramon Marsal, José A. Barrabés, Aida Ribera, Rosa Maria Lidón, Enric Domingo, Gerard Martí, and David García-Dorado. Short-term exposure to air pollutants increases the risk of ST elevation myocardial infarction and of infarct-related ventricular arrhythmias and mortality. *International Journal of Cardiology* **250**, 35–42 (2018). ISSN 18741754. doi: 10.1016/j.ijcard.2017.10.004. URL https://doi.org/10.1016/j.ijcard.2017.10.004. Cited on page/s 3.

[8] A. Soret, M. Guevara, and J.M. Baldasano. The potential impacts of electric vehicles on air quality in the urban areas of barcelona and madrid (spain). *Atmospheric Environment* **99**, 51 – 63 (2014). ISSN 1352-2310. doi: http://dx.doi.org/10.1016/j.atmosenv.2014.09.048. Cited on page/s 4.

[9] M. Guevara, F. Martínez, G. Arevalo, S. Gasso, and J. Baldasano. An improved system for modelling spanish emissions: Hermesv2.0. *Atmospheric environment* **81**, 209–221 (Dec 2013). doi: 10.1016/j.atmosenv.2013.08.053. Cited on page/s 4.

[10] Ajuntament de Barcelona. Pla de millora de la qualitat de l'aire de barcelona 2015-2018 (2015). URL http://hdl.handle.net/11703/83944. Cited on page/s 4.

[11] European Community Shipowners Associations (ECSA). The economic value of the eu shipping industry. update (February 2015). Cited on page/s 5, 51, 67.

[12] Tristan Smith et al. Third imo greenhouse gas study (2014). Cited on page/s 5, 67.

[13] Francesco Di Natale and Claudia Carotenuto. Particulate matter in marine diesel engines exhausts: Emissions and control strategies. *Transportation Research Part D: Transport and Environment* **40**, 166 – 191 (2015). ISSN 1361-9209. doi: http://dx.doi.org/10.1016/j.trd.2015.08.011. Cited on page/s 5, 67.

[14] Mar Viana, Pieter Hammingh, Augustin Colette, Xavier Querol, Bart Degraeuwe, Ina de Vlieger, and John van Aardenne. Impact of maritime transport emissions on coastal air quality in europe. *Atmospheric Environment* **90**, 96 – 105 (2014). ISSN 1352-2310. doi: http://dx.doi.org/10.1016/j.atmosenv.2014.03.046. Cited on page/s 5, 67.

[15] Daniel Rodriguez-Rey, Marc Guevara, Ma Paz Linares, Josep Casanovas, Juan Salmerón, Albert Soret, Oriol Jorba, Carles Tena, and Carlos Pérez García-Pando. A coupled macroscopic traffic and pollutant emission modelling system for barcelona. *Transportation Research Part D: Transport and Environment* **92**, 102725 (2021). Cited on page/s 5.

[16] J. P. Jalkanen, L. Johansson, J. Kukkonen, A. Brink, J. Kalli, and T. Stipa. Extension of an assessment model of ship traffic exhaust emissions for particulate matter and carbon monoxide. *Atmospheric Chemistry and Physics* **12** (5), 2641–2659 (2012). ISSN 16807316. doi: 10.5194/acp-12-2641-2012. Cited on page/s 5, 8, 15, 16, 17, 19, 23, 51, 52, 53, 65, 67, 68, 73, 74, 84.

[17] International Maritime Organization. Ais transponders (February 2017). Cited on page/s 5, 15.

[18] Abbas Harati-Mokhtari, Alan Wall, Philip Brooks, and Jin Wang. Automatic identification system (ais): data reliability and human error implications. *The Journal of Navigation* **60** (3), 373 (2007). Cited on page/s 5, 29, 51.

[19] Laurie Goldsworthy and Brett Goldsworthy. Modelling of ship engine exhaust emissions in ports and extensive coastal waters based on terrestrial ais data–an australian case study. *Environmental Modelling & Software* **63**, 45–60 (2015). Cited on page/s 5, 30, 31.

[20] A. Miola, B. Ciuffo, E. Giovine, and M. Marra. Regulating air emissions from ships. the state of the art on methodologies, technologies and policy options. *Joint Research Centre Reference Report, EUR24602EN, 978-92* (2010). Cited on page/s 6.

[21] Alberto Gutierrez-Torre, Kiyana Bahadori, Shuja-ur-Rehman Baig, Waheed Iqbal, Tullio Vardanega, Josep Lluís Berral, and David Carrera. Automatic distributed deep learning using resource-constrained edge devices. *IEEE Internet of Things Journal* (2021). doi: https://doi.org/10.1109/JIOT.2021.3098973. Cited on page/s 8, 36, 37, 39, 43, 44, 46, 47, 49, 82.

[22] Graham W. Taylor, Geoffrey E Hinton, and Sam T. Roweis. Modeling human motion using binary latent variables. In P. B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19* pages 1345–1352. MIT Press (2007). Cited on page/s 11.

[23] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation (2014). Cited on page/s 11.

[24] Graham W. Taylor, Geoffrey E. Hinton, and Sam Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems* page 2007. MIT Press (2006). Cited on page/s 12, 31.

[25] Marc Guevara, Carles Tena, Manuel Porquet, Oriol Jorba, and Carlos Pérez García-Pando. Hermesv3, a stand-alone multi-scale atmospheric emission modelling framework–part 1: global and regional module. *Geoscientific Model Development* **12** (5), 1885–1907 (2019). Cited on page/s 14, 51, 67, 84.

[26] Lasse Johansson, Jukka-Pekka Jalkanen, and Jaakko Kukkonen. Global assessment of shipping emissions in 2015 on a high spatial and temporal resolution. *Atmospheric Environment* **167**, 403–415 (2017). Cited on page/s 15, 31, 51, 55, 62, 63, 65.

[27] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Soeren Auer. Quality assessment for linked data: A survey. *Semantic Web* 7 (1), 63–93 (2016). Cited on page/s 25.

[28] Barna Saha and Divesh Srivastava. Data quality: The other face of big data. In *2014 IEEE 30th international conference on data engineering* pages 1294–1297. IEEE (2014). Cited on page/s 25.

[29] Swarnava Dey, Ankur Chakraborty, Soumitra Naskar, and Prateep Misra. Smart city surveillance: Leveraging benefits of cloud data stores. In *37th Annual IEEE Conference on Local Computer Networks-Workshops* pages 868–876. IEEE (2012). Cited on page/s 27.

[30] Kehua Su, Jie Li, and Hongbo Fu. Smart city and the applications. In *2011 international conference on electronics, communications and control (ICECC)* pages 1028–1031. IEEE (2011). Cited on page/s 27.

[31] Miguel Castro, Antonio J Jara, and Antonio FG Skarmeta. Smart lighting solutions for smart cities. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops* pages 1374–1379. IEEE (2013). Cited on page/s 27.

[32] A. Corsaro. Connected boulevard – it's what makes nice, france, a smart city (September 2014). URL https://goo.gl/sFjCUq. "on-line; accessed in: 15-Nov-2017". Cited on page/s 27.

[33] L. Sanchez, V. Gutierrez, J. A. Galache, P. Sotres, J. R. Santana, J. Casanueva, and L. Muñoz. Smartsantander: Experimentation and service provision in the smart city. In *16th International Symposium on Wireless Personal Multimedia Communications (WPMC)* pages 1–6 (June 2013). Cited on page/s 27.

[34] Technology Review. Cities get smarter (2015). URL https://goo.gl/fSd1mu. "on-line; accessed in: 15-Nov-2017". Cited on page/s 27.

[35] Yu Chao Hu, Milan Patel, Dario Sabella, Nurlt Sprecher, and Valerie Young. Mobile edge computing: A key technology towards 5g, white paper, etsi (2015). URL https://goo.gl/Qx9W4k. "on-line; accessed in: 15-Nov-2017". Cited on page/s 27.

[36] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM (2017). Cited on page/s 27.

[37] M. Yannuzzi, *et al.* A new era for cities with fog computing. *IEEE Internet Computing* **21** (2), 54–67 (Mar 2017). ISSN 1089-7801. doi: 10.1109/MIC.2017.25. Cited on page/s 27.

[38] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems* pages 802–810 (2015). Cited on page/s 27.

[39] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799* (2018). Cited on page/s 27.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* pages 1097–1105 (2012). Cited on page/s 27.

[41] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745* (2012). Cited on page/s 27.

[42] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* pages 215–223 (2011). Cited on page/s 27.

[43] C. Rudin, *et al.* Machine learning for the new york city power grid. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **34** (2), 328–345 (Feb 2012). ISSN 0162-8828. doi: 10.1109/TPAMI.2011.108. Cited on page/s 27.

[44] Harishchandra Dubey, Jing Yang, Nick Constant, Amir Mohammad Amiri, Qing Yang, and Kunal Makodiya. Fog data: Enhancing telehealth big data through fog computing. In *Proceedings of the ASE BigData &#38; SocialInformatics 2015* ASE BD&#38;SI '15 pages 14:1–14:6 New York, NY, USA (2015). ACM. ISBN 978-1-4503-3735-9. doi: 10.1145/2818869.2818889. Cited on page/s 27.

[45] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, and Quoc V Le. Large scale distributed deep networks. In *Advances in neural information processing systems* pages 1223–1231 (2012). Cited on page/s 28.

[46] Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1112.6209* (2011). Cited on page/s 28.

[47] Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association* pages 1488–1492 (2015). Cited on page/s 28.

[48] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *International conference on machine learning* pages 1337–1345 (2013). Cited on page/s 28.

[49] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint:1706.02677* (2017). Cited on page/s 28.

[50] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Opportunistic spatio-temporal event processing for mobile situation awareness. In *Proceedings of the 7th ACM international conference on Distributed event-based systems* pages 195–206. ACM (2013). Cited on page/s 28.

[51] L. Yu, Z. Li, J. Liu, and R. Zhou. Resources sharing in 5g networks: Learning-enabled incentives and coalitional games. *IEEE Systems Journal* pages 1–12 (2019). Cited on page/s 28.

[52] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *ArXiv* **abs/1602.05629** (2016). Cited on page/s 28, 37, 41, 42, 45.

[53] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, and H Brendan McMahan. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046* (2019). Cited on page/s 28, 37.

[54] B. Hu, Y. Gao, L. Liu, and H. Ma. Federated region-learning: An edge computing based framework for urban environment sensing. In *2018 IEEE Global Communications Conference (GLOBECOM)* pages 1–7 (Dec 2018). doi: 10.1109/GLOCOM.2018.8647649. Cited on page/s 28.

[55] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems (2018). Cited on page/s 28.

[56] Alberto Marchisio, Muhammad Abdullah Hanif, Faiq Khalid, George Plastiras, Christos Kyrkou, Theocharis Theocharides, and Muhammad Shafique. Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* pages 553–559. IEEE (2019). Cited on page/s 28.

[57] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. Rce-nn: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices. In *Proceedings of the 10th International Conference on the Internet of Things* pages 1–8 (2020). Cited on page/s 28.

[58] R. Sanchez-Iborra and A. F. Skarmeta. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine* **20** (3), 4–18 (2020). doi: 10.1109/MCAS.2020.3005467. Cited on page/s 28, 85.

[59] Seulki Lee and Shahriar Nirjon. Neuro.zero: A zero-energy neural network accelerator for embedded sensing and inference systems. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* SenSys '19 page 138–152 New York, NY, USA (2019). Association for Computing Machinery. ISBN 9781450369503. doi: 10.1145/3356250.3360030. URL https://doi-org.recursos.biblioteca.upc.edu/10.1145/3356250.3360030. Cited on page/s 28.

[60] Orazio Briante, Claudia Campolo, Antonio Iera, Antonella Molinaro, Stefano Yuri Paratore, and Giuseppe Ruggeri. Supporting augmented floating car data through smartphone-based crowd-sensing. *Vehicular Communications* **1** (4), 181 – 196 (2014). ISSN 2214-2096. doi: https://doi.org/10.1016/j.vehcom.2014.08.002. Cited on page/s 29.

[61] K. Ali, D. Al-Yaseen, A. Ejaz, T. Javed, and H. S. Hassanein. Crowdits: Crowdsourcing in intelligent transportation systems. In *2012 IEEE Wireless Communications and Networking Conference (WCNC)* pages 3307–3311 (April 2012). doi: 10.1109/WCNC.2012.6214379. Cited on page/s 29.

[62] S. Ancona, R. Stanica, and M. Fiore. Performance boundaries of massive floating car data offloading. In *2014 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS)* pages 89–96 (April 2014). doi: 10.1109/WONS.2014.6814727. Cited on page/s 29.

[63] Jianjun Yu, Fuchun Jiang, and Tongyu Zhu. RTIC-C: A big data system for massive traffic information mining. *Proceedings - 2013 International Conference on Cloud Computing and Big Data, CLOUDCOM-ASIA 2013* pages 395–402 (2013). ISSN 2378-3680. doi: 10.1109/CLOUDCOM-ASIA.2013.91. Cited on page/s 29.

[64] Fabio Moretti, Stefano Pizzuti, Stefano Panzieri, and Mauro Annunziato. Urban traffic flow forecasting through statistical and neural network bagging ensemble hybrid modeling. *Neurocomputing* **167**, 3–7 (2015). ISSN 18728286. doi: 10.1016/j.neucom.2014.08.100. Cited on page/s 29.

[65] Dawen Xia, Binfeng Wang, Huaqing Li, Yantao Li, and Zili Zhang. A distributed spatial-temporal weighted model on MapReduce for short-term traffic flow forecasting. *Neurocomputing* **179**, 246–26 (2016). ISSN 18728286. doi: 10.1016/j.neucom.2015.12.013. Cited on page/s 29.

[66] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei Yue Wang. Traffic Flow Prediction With Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems* **16** (2), 865–873 (2014). ISSN 15249050. doi: 10.1109/TITS.2014.2345663. Cited on page/s 29.

[67] Martin Svanberg, Vendela Santen, Axel Hörteborn, Henrik Holm, and Christian Finnsgård. Ais in maritime research. *Marine Policy* **106**, 103520 (2019). Cited on page/s 29.

[68] Miluše Tichavska, Francisco Cabrera, Beatriz Tovar, and Víctor Araña. Use of the automatic identification system in academic research. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2015* pages 33–40 Cham (2015). Springer International Publishing. ISBN 978-3-319-27340-2. Cited on page/s 29.

[69] Mikko Lensu and Floris Goerlandt. Big maritime data for the baltic sea with a focus on the winter navigation system. *Marine Policy* **104**, 53–65 (2019). Cited on page/s 29.

[70] Miluše Tichavska and Beatriz Tovar. Port-city exhaust emission model: An application to cruise and ferry operations in Las Palmas Port. *Transportation Research Part A: Policy and Practice* **78** (C), 347–360 (2015). doi: 10.1016/j.tra.2015.05.021. Cited on page/s 29.

[71] Xinjia Gao, Hidenari Makino, and Masao Furusho. Ship behavior analysis for real operating of container ships using ais data. *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation* **10** (2), 213–220 (2016). ISSN 2083-6473. doi: 10.12716/1001.10.02.04. Cited on page/s 29.

[72] Jukka-Pekka Jalkanen, Lasse Johansson, and Jaakko Kukkonen. A comprehensive inventory of the ship traffic exhaust emissions in the baltic sea from 2006 to 2009. *AMBIO* **43** (3), 311–324 (2014). ISSN 1654-7209. doi: 10.1007/s13280-013-0389-3. Cited on page/s 29.

[73] Ø. Buhaug, *et al.* Second imo ghg study 2009, prevention of air pollution from ships. IMO (2009). Cited on page/s 29.

[74] Apollonia Miola and Biagio Ciuffo. Estimating air emissions from ships: Meta-analysis of modelling approaches and available data sources. *Atmospheric Environment* **45** (13), 2242 – 2251 (2011). ISSN 1352-2310. doi: https://doi.org/10.1016/j.atmosenv.2011.01.046. URL http://www.sciencedirect.com/science/article/pii/S1352231011000872. Cited on page/s 30.

[75] T Fletcher, V Garaniya, S Chai, R Abbassi, H Yu, TC Van, RJ Brown, and F Khan. An application of machine learning to shipping emission inventory. *Royal Institution of Naval Architects. Transactions. Part A. International Journal of Maritime Engineering* **160** (A4), A381–A396 (2018). Cited on page/s 30.

[76] Minxing Si, Tyler J Tarnoczi, Brett M Wiens, and Ke Du. Development of predictive emissions monitoring system using open source machine learning library–keras: A case study on a cogeneration unit. *IEEE Access* 7, 113463–113475 (2019). Cited on page/s 30.

[77] TK Chan and CS Chin. Data analysis to predictive modeling of marine engine performance using machine learning. In *2016 IEEE Region 10 Conference (TENCON)* pages 2076–2080. IEEE (2016). Cited on page/s 30.

[78] W Mohd Noor, R Mamat, G Najafi, MH Yasin, CK Ihsan, and MM Noor. Prediction of marine diesel engine performance by using artificial neural network model. *Journal of Mechanical Engineering and Sciences* **10** (1), 1917–1930 (2016). Cited on page/s 30.

[79] Luan Thanh Le, Gunwoo Lee, Keun-Sik Park, and Hwayoung Kim. Neural network-based fuel consumption estimation for container ships in korea. *Maritime Policy & Management* pages 1–18 (2020). Cited on page/s 30.

[80] Tayfun Uyanık, Çağlar Karatuğ, and Yasin Arslanoğlu. Machine learning approach to ship fuel consumption: A case of container vessel. *Transportation Research Part D: Transport and Environment* **84**, 102389 (2020). Cited on page/s 30.

[81] Qin Liang, Hans Anton Tvete, and Hendrik W Brinks. Prediction of vessel propulsion power from machine learning models based on synchronized ais-, ship performance measurements and ecmwf weather data. In *IOP Conference Series: Materials Science and Engineering* volume 929 page 012012. IOP Publishing (2020). Cited on page/s 30.

[82] W. Qiu and A. Bandara. Gps trace mining for discovering behaviour patterns. In *2015 International Conference on Intelligent Environments* pages 65–72. IEEE IEEE (July 2015). doi: 10.1109/IE.2015.17. Cited on page/s 30.

[83] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.* **1** (3), 259–289 (January 1997). ISSN 1384-5810. doi: 10.1023/A:1009748302351. Cited on page/s 30.

[84] Vincent W. Zheng, Yu Zheng, Xing Xie, and Qiang Yang. Collaborative location and activity recommendations with gps history data. In *Proceedings of the 19th International Conference on World Wide Web* WWW '10 pages 1029–1038 New York, NY, USA (2010). ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772795. Cited on page/s 31.

[85] Zhenhui Li, Jiawei Han, Ming Ji, Lu-An Tang, Yintao Yu, Bolin Ding, Jae-Gil Lee, and Roland Kays. Movemine: Mining moving object data for discovery of animal movement patterns. *ACM Trans. Intell. Syst. Technol.* **2** (4), 37:1–37:32 (July 2011). ISSN 2157-6904. doi: 10.1145/1989734.1989741. Cited on page/s 31.

[86] Q. Lin, D. Zhang, X. Huang, H. Ni, and X. Zhou. Detecting wandering behavior based on gps traces for elders with dementia. In *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)* pages 672–677. IEEE (Dec 2012). doi: 10.1109/ICARCV.2012.6485238. Cited on page/s 31.

[87] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In Luis Álvarez, Marta Mejail, Luís Gómez Déniz, and Julio C. Jacobo, editors, *CIARP* volume 7441 of *Lecture Notes in Computer Science* pages 14–36. Springer (2012). ISBN 978-3-642-33274-6. Cited on page/s 31.

[88] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)* volume 7700 of *Lecture Notes in Computer Science* pages 599–619. Springer (2012). ISBN 978-3-642-35288-1. Cited on page/s 31.

[89] Volodymyr Mnih, Hugo Larochelle, and Geoffrey E. Hinton. Conditional restricted boltzmann machines for structured output prediction. In Fábio Gagliardi Cozman and Avi Pfeffer, editors, *UAI* pages 514–522. AUAI Press (2011). ISBN 978-0-9749039-7-2. Cited on page/s 31, 68.

[90] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning* ICML '07 pages 791–798 New York, NY, USA (2007). ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273596. Cited on page/s 31.

[91] Graham W. Taylor and Geoffrey E. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the 26th Annual International Conference on Machine Learning* ICML '09 pages 1025–1032 New York, NY, USA (2009). ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553505. Cited on page/s 31.

[92] Guy Lebanon and S. V. N. Vishwanathan, editors. Proceedings of the eighteenth international conference on artificial intelligence and statistics, AISTATS 2015, san diego, california, usa, may 9-12, 2015 volume 38 of *JMLR Workshop and Conference Proceedings* (2015). JMLR.org. Cited on page/s 31.

[93] Jaedong Lee, Heera Kim, Noo-ri Kim, and Jee-Hyong Lee. An approach for multi-label classification by directed acyclic graph with label correlation maximization. *Inf. Sci.* **351** (C), 101–114 (July 2016). ISSN 0020-0255. doi: 10.1016/j.ins.2016.02.037. Cited on page/s 31.

[94] Christian Velasco-Gallego and Iraklis Lazakis. Real-time data-driven missing data imputation for short-term sensor data of marine systems. a comparative study. *Ocean Engineering* **218**, 108261 (2020). Cited on page/s 31.

[95] NAEI. Ricardo energy. a review of the naei shipping emissions methodology. NAEI ed61406-issue edition (2017). Cited on page/s 31, 51, 53, 84.

[96] T. W. P. Smith, *et al.* Third imo greenhouse gas study 2014. International Maritime Organization United Kingdom (April 2015). Cited on page/s 31, 51.

[97] J Faber, S Hanayama, S Zhang, P Pereda, B Comer, E Hauerhof, and H Yuan. Fourth imo greenhouse gas study. *Retrieved from the International Maritime Organization website: https://docs. imo. org* (2020). Cited on page/s 31, 65, 84.

[98] Huan Liu, Mingliang Fu, Xinxin Jin, Yi Shang, Drew Shindell, Greg Faluvegi, Cary Shindell, and Kebin He. Health and climate impacts of ocean-going vessels in east asia. *Nature climate change* **6** (11), 1037–1041 (2016). Cited on page/s 31, 51, 52, 64.

[99] Tomasz Cepowski. Regression formulas for the estimation of engine total power for tankers, container ships and bulk carriers on the basis of cargo capacity and design speed. *Polish Maritime Research* (2019). Cited on page/s 32.

[100] Juan Luis Pérez, Alberto Gutierrez-Torre, Josep Ll. Berral, and David Carrera. A resilient and distributed near real-time traffic forecasting application for fog computing environments. *Future Generation Computer Systems* **87**, 198 – 212 (2018). ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.05.013. Cited on page/s 35, 36, 39, 42, 43, 82, 85.

[101] S. Ali, A. Ashraf, S. B. Qaisar, M. Kamran Afridi, H. Saeed, S. Rashid, E. A. Felemban, and A. A. Sheikh. Simplimote: A wireless sensor network monitoring platform for oil and gas pipelines. *IEEE Systems Journal* **12** (1), 778–789 (2018). Cited on page/s 35.

[102] M. Ghorbanian, S. H. Dolatabadi, and P. Siano. Big data issues in smart grids: A survey. *IEEE Systems Journal* **13** (4), 4158–4168 (2019). Cited on page/s 35.

[103] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. Edge computing for the internet of things: A case study. *IEEE Internet of Things Journal* **5** (2), 1275–1284 (2018). Cited on page/s 35.

[104] G. Plastiras, M. Terzi, C. Kyrkou, and T. Theocharidcs. Edge intelligence: Challenges and opportunities of near-sensor machine learning applications. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)* pages 1–7 (2018). Cited on page/s 35, 36.

[105] Claudio Savaglio and Giancarlo Fortino. A simulation-driven methodology for iot data mining based on edge computing. *ACM Transactions on Internet Technology (TOIT)* **21** (2), 1–22 (2021). Cited on page/s 35.

[106] Albert Bifet. Classifier concept drift detection and the illusion of progress. In *International Conference on Artificial Intelligence and Soft Computing* pages 715–725. Springer (2017). Cited on page/s 36, 85.

[107] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* pages 13–16. ACM (2012). Cited on page/s 38.

[108] Jonathan McChesney, Nan Wang, Ashish Tanwer, Eyal de Lara, and Blesson Varghese. Defog: Fog computing benchmarks. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* SEC '19 page 47–58 New York, NY, USA (2019). Association for Computing Machinery. ISBN 9781450367332. doi: 10.1145/3318216.3363299. URL https://doi.org/10.1145/3318216.3363299. Cited on page/s 39.

[109] Nvidia autonomous machines: Jetson nano (July 2019). URL https://www.nvidia.com/en-us/autonomous-machines. Cited on page/s 40.

[110] Yan Li, Changxin Bai, and Chandan K Reddy. A distributed ensemble approach for mining healthcare data under privacy constraints. *Information sciences* **330**, 245–259 (2016). Cited on page/s 41, 85.

[111] Alberto Gutierrez-Torre, Josep Ll. Berral, David Buchaca, Marc Guevara, Albert Soret, and David Carrera. Improving maritime traffic emission estimations on missing data with crbms. *Engineering Applications of Artificial Intelligence* **94**, 103793 (2020). ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2020.103793. URL http://www.sciencedirect.com/science/article/pii/S0952197620301822. Cited on page/s 42, 43.

[112] Akis Linardos, Kaisar Kushibar, Sean Walsh, Polyxeni Gkontra, and Karim Lekadir. Federated learning for multi-center imaging diagnostics: A study in cardiovascular disease. *Nature* (2021). doi: 10.48550/ARXIV.2107.03901. URL https://arxiv.org/abs/2107.03901. Cited on page/s 42, 49.

[113] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407* (2018). Cited on page/s 42, 49, 82.

[114] Bernard Rosner. Percentage points for a generalized esd many-outlier procedure. *Technometrics* **25** (2), 165–172 (1983). Cited on page/s 54.

[115] Christopher M Bishop. Pattern recognition and machine learning. springer (2006). Cited on page/s 55.

[116] Leo Breiman. Random forests. *Machine learning* **45** (1), 5–32 (2001). Cited on page/s 56.

[117] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '16 pages 785–794 New York, NY, USA (2016). ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785. Cited on page/s 56.

[118] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944* (2012). Cited on page/s 57.

[119] Sistema caliope, prediction of air quality (May 2019). URL http://www.bsc.es/caliope/es. Cited on page/s 67.

[120] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining* pages 443–448. SIAM SIAM (2007). Cited on page/s 68.

[121] Graham W. Taylor and Geoffrey E. Hinton. Factored conditional restricted Boltzmann Machines for modeling motion style. *Proceedings of the 26th International Conference on Machine Learning (ICML 09)* pages 1025–1032 (2009). ISSN 1520510X. doi: 10.1145/1553374.1553505. Cited on page/s 68.

[122] Zoubin Ghahramani. Hidden markov models. In *Hidden Markov Models: Applications in Computer Vision* chapter An Introduction to Hidden Markov Models and Bayesian Networks, pages 9–42. World Scientific Publishing Co., Inc. River Edge, NJ, USA (2002). ISBN 981-02-4564-5. Cited on page/s 68.

[123] D. B. Prats, J. L. Berral, and D. Carrera. Automatic generation of workload profiles using unsupervised learning pipelines. *IEEE Transactions on Network and Service Management* **15** (1), 142–155 (March 2018). ISSN 1932-4537. doi: 10.1109/TNSM.2017.2786047. Cited on page/s 68.

[124] VesselFinder. Free AIS Ship Tracking of Marine Traffic (May 2019). URL http://www.vesselfinder.com. Cited on page/s 69.

[125] Cepesca. El arrastre: El arte de la pesca sostenible (February 2017). Cited on page/s 69, 76.

[126] Ihs fairplay ship data base (May 2019). URL https://fairplay.ihs.com/. Cited on page/s 73.

[127] Stefanie Warnat-Herresthal, *et al.* Swarm learning for decentralized and confidential clinical machine learning. *Nature* **594** (7862), 265–270 (2021). Cited on page/s 85.