FACULDADE TRÊS DE MAIO
O CONHECIMENTO FAZ A DIFERENÇA!

Sociedade Educacional Três de Maio - SETREM
Bacharelado em Sistemas de Informação

RICARDO PIEPER

ANAEROBIC DIGESTER ANALYTICS: TOWARDS A SMART SOFTWARE AS A SERVICE

Três de Maio

2016

RICARDO PIEPER

ANAEROBIC DIGESTER ANALYTICS: TOWARDS A SMART SOFTWARE AS A SERVICE

Undergraduate Thesis
Sociedade Educacional Três de Maio
Faculdade Três de Maio
Information Systems

Advisor:
Prof. Dr. Dalvan Griebler
Coadvisor:
M. Sc. Adalberto Lovato

Três de Maio
2016

TERMO DE APROVAÇÃO

RICARDO PIEPER

ANAEROBIC DIGESTER ANALYTICS: TOWARDS A SMART SOFTWARE AS A SERVICE

Relatório aprovado como requisito parcial para obtenção do título de **Bacharel em Sistemas de Informação** concedido pela Faculdade de Sistemas de Informação da Sociedade Educacional Três de Maio, pela seguinte Banca examinadora:

Orientador: Prof. Dalvan Griebler, Dr.
Faculdade de Ciências da Computação da PUC RS

Orientador: Prof. Adalberto Lovato, M.Sc.
Faculdade de Sistemas de Informação da SETREM

Prof. Samuel Camargo de Souza, M.Sc.
Faculdade de Sistemas de Informação da SETREM

Prof. Tiago Luis Cesa Seibel, M.Sc.
Faculdade de Sistemas de Informação da SETREM

Profa. Vera Lúcia Lorenset Benedetti, M.Sc.
Coordenação do Curso Bacharelado em Sistemas de Informação
Faculdade de Sistemas de Informação da SETREM

Três de Maio, 08 de agosto de 2016.

AGRADECIMENTOS

Agradeço a minha família pelo apoio, paciência e carinho dados durante o desenvolvimento do trabalho.

Agradeço também o professor Adalberto Lovato pelas discussões frutíferas que tivemos.

Em especial, agradeço o professor Dalvan Griebler, que orientou com genuína dedicação mesmo durante momentos difíceis, e por ter me ajudado muito na escrita.

Agradeço também a todas as pessoas que de certa forma contribuiram, mesmo que indiretamente, para a realização do trabalho.

ABSTRACT

The machine learning field is becoming even more important in the last years. The ever-increasing amount of data challenges the current available technology. Meanwhile, anaerobic digesters represent a good alternative for renewable energy production in Brazil. However, performing efficient and accurate predictions/analytics while completely abstracting machine learning details from end-users might not be a simple task to achieve. Usually, such tools are made for a specific scenario and may not fit with particular and general needs in other projects. The thesis goal was to create a SaaS on biogas data analytics by using a neural network. Therefore, an open source, cloud-enabled SaaS (Software as a Service) was developed and deployed in LARCC (Laboratory of Advanced Researches for Cloud Computing) at SETREM. The results have shown the neural network's accuracy is not significantly worse than a state-of-the-art implementation, and its training speed is faster. However, the algorithm is yet to be tested using real world biogas data. The user interface demonstrates to be intuitive, and the predictions with synthetic data were accurate when the training algorithm is provided with good quality data. Also, the file processing and network training time were good enough under traditional workload conditions.

Keywords: Information Systems, Machine Learning, Anaerobic Digesters, SaaS.

RESUMO

A área de *machine learning* vem se tornando cada vez mais importante nos últimos anos, enquanto a quantidade de informação desafia a tecnologia atualmente disponível. Em outra área, digestores anaeróbicos representam uma boa alternativa para geração de energia renovável no Brasil. Porém, executar previsões de produção de biogás de forma eficiente, abstraindo detalhes de *machine learning* da interface de usuário pode não ser uma tarefa fácil de se executar. Normalmente, ferramentas similares são feitas para cenários específicos e podem não se encaixar em necessidades de outros projetos. O objetivo do trabalho é criar uma aplicação SaaS para análise de dados de biogás usando redes neurais. Uma aplicação *open source* SaaS foi criada e implantada no laboratório LARCC na SETREM. Os resultados mostraram que a performance da rede neural não é significantemente pior que uma implementação estado-da-arte, e o tempo de treinamento é menor. Porém, o algoritmo ainda precisa ser testado com dados reais de biogás. A interface de usuário se mostra intuitiva, e as previsões com dados sintéticos são precisas quando dados de boa qualidade são usados. Além disso, o tempo de processamento de arquivos e tempo de treinamento da rede neural são bons o suficiente durante cargas de trabalho normais.

Palavras-Chave: Sistemas de Informação, Aprendizado de Máquina, Biodigestores, SaaS.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AD	Anaerobic Digester
API	Application Programming Interface
CLR	Common Language Runtime
CQL	Cassandra Query Language
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DOM	Document Object Model
FLENS	Flexible Library for Efficient Numerical Solutions
GCC	GNU Compiler Collection
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastructure as a Service
IT	Information Technology
JIT	Just-In-Time compiler
JSON	Javascript Object Notation
JVM	Java Virtual Machine
I/O	Input/Output
LARCC	Laboratory of Advanced Researches for Cloud Computing
ML	Machine Learning
MSVC	Microsoft Visual C++

MVC	Model View Controller
MVVM	Model View View Model
NoSQL	Not Only SQL
OS	Operating System
PaaS	Platform as a Service
SaaS	Software as a Service
SPA	Single Page Application
SGD	Stochastic Gradient Descent
SQL	Structured Query Language
SETREM	Sociedade Educacional Três de Maio
UI	User Interface
VM	Virtual Machine
WEKA	Waikato Environment for Knowledge Analysis
XML	eXtensible Markup Language

CONTENTS

INTRODUCTION

Anaerobic Digester (AD) is a technology that produces methane gas. Inside an AD, the substrate (composed by organic waste, manure or other substances) undergo a process of degradation that breaks multi-molecular substances, producing biogas (Aslanzadeh et al. (2013)). According to Gutiérrez-Castro et al. (2015), the implementation of AD technology is less costly than comparable renewable technologies (*e.g.* diesel-like fuels) and has great potential of energy production in developing countries. This idea is also shared by Alves et al. (2015), stating that the production of electric energy through biomass shows great potential in Brazil due to the large agricultural and livestock activity. However, Labatut and Gooch (2012) shows inadequate management and lack of process control causes ADs to perform less efficiently.

The goal of this work is to provide a SaaS software for research on biogas data and machine learning to improve the gas and electricity production. The software should display relevant information about the biogas production process through a friendly user interface and accessible from anywhere. Also, the software will use the information to make predictions about the biogas process, using a Machine Learning (ML) algorithm called Neural Network. Some efforts have been done in the area of ML and Biogas, such as Kusiak and Wei (2014) and Qdais, Bani-Hani and Shatnawi (2010), which have shown the possibility of using ML to extract information about the biogas production process.

There are some general-purpose software for ML and Data Mining, such as Orange Canvas[1] and WEKA[2]. This work will compare its own implementation of a neural network against WEKA. Also, this work aims to abstract the ML details from the user. The main contributions are the following:

- A New SaaS for AD analytics and electricity production;
- A graphical and user friendly interface to displaying relevant information;
- An integration of ML algorithms to predict and improve AD efficiency;

This work is organized in 3 chapters. The first one explains the methodological aspects of the project, presents the research problems, hypotheses and methodology. The second chapter presents the literature review along with a study of the related work in the area of machine learning for biogas. The third chapter presents a discussion about the achieved results.

[1]http://orange.biolab.si/
[2]http://www.cs.waikato.ac.nz/ml/weka/

CHAPTER1: RESEARCH PLAN

1.1 THEME

Anaerobic Digesters Analytics: Towards a Smart Software as a Service

1.1.1 Theme Delimitation

At SETREM, several ADs were built in order to perform research on biogas production. This research is performed by the Bioagropec project [1]. SETREM is located at the northwest region of Rio Grande do Sul in Brazil, a region with a significant amount of agricultural and livestock activity. As explained by Alves et al. (2015), Brazil is a country with great potential for electric energy production through biomass due to this kind of economic activity, especially in the south region of the country.

This project represents a cooperation between LARCC and the Bioagropec project, in which the institution brings IT into biogas research. The main goal is to provide a SaaS application that brings data analysis in an easy to use interface for the user, accessible from anywhere. This application aims to display relevant information about the biogas processes, focusing on predictions of biogas and electricity production.

The application provides prediction services using Neural Networks. Due to

[1] http://bioagropec.setrem.com.br

the lack of real world biogas data, the neural network will be trained using synthetic, generated data to test its performance. However, when using real biogas data, the results of these predictions might be used to improve AD efficiency, as shown by Kusiak and Wei (2014) and Qdais, Bani-Hani and Shatnawi (2010). Although some general purpose tools for ML already exists, such as Orange Canvas and WEKA, this project aims to provide an initial implementation of a future SaaS application that provides a high-level interface to the user, abstracting low-level details of ML. In the future, more features and ML algorithms can be implemented and continually improved further, for instance, by implementing features such as parallelism and real time recommendations. Also, the application should provide ML services without requiring the user to have previous knowledge about ML algorithms and their implementations.

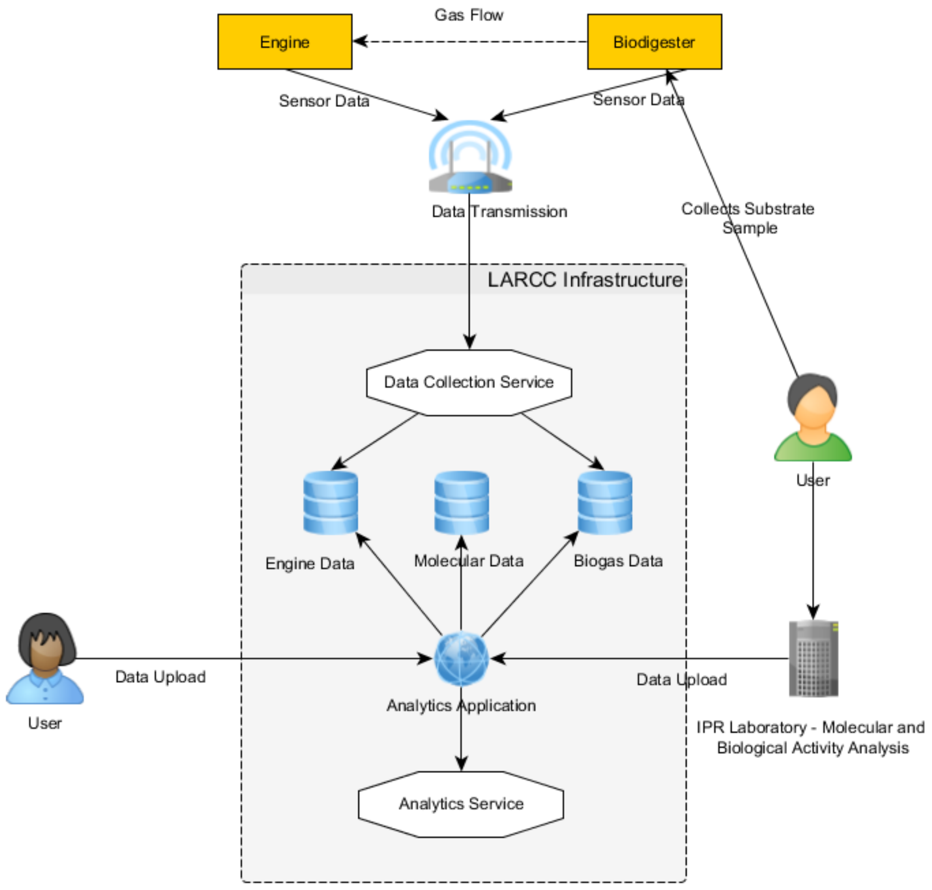


Figure 1.1: Project Workflow

As shown in Figure 1.1, the application is installed in the LARCC infrastructure.

The project aims to create an application that in the future will be able to receive data from several sources, such as AD sensor data (via data upload or real time sensor data), engine data and molecular activity data from laboratory analysis. However, in this initial implementation, the user will only be able to upload data to the application. The application consists of two main components: The web front-end implemented using Node.js, and the back-end data analysis service written in C++, responsible for doing most of the heavy workload (processing of data uploads, training the neural network and performing predictions). The Cassandra Database was also used. The project is open source, available to everyone for free to use and modify.

The project is a requirement for the final undergraduate thesis, and it will be developed by Ricardo Pieper, being advised by Prof. Dr. Dalvan Griebler and co-advised by M. Sc. Adalberto Lovato at SETREM during November 2015 until August 2016. The research will take place in the LARCC at SETREM.

1.2 PROBLEMS

The software will implement ML to provide prediction services. The implementation of ML brings two problems:

– Will the SaaS application be able to perform predictions?

– Will the implementation of a neural network be efficient in terms of performance and accuracy in comparison with WEKA?

1.3 HYPOTHESES

- The neural network is able to predict biogas production with more than 90% accuracy.

- The neural network prediction accuracy is not significantly worse (about 2%) in comparison with WEKA's $MultilayerPerceptron$ algorithm.

- The SaaS application gets results in a timely manner.

1.4 VARIABLES

- Neural network training time

- Correlation between predictions and actual data samples, where the correlation is given by the Pearson Correlation Coefficient, described by Osborne (2008):

$$r = \frac{\sum(XY) - \frac{\sum X \sum Y}{N}}{\sqrt{(\sum(X^2) - \frac{(\sum X)^2}{N})(\sum(Y^2) - \frac{(\sum Y)^2}{N})}} \tag{1.1}$$

  where X is a collection of predictions, Y is a collection of actual data samples and N is the number of data samples (the size of X and Y).

1.5 OBJECTIVES

To develop the project, some objectives must be set. This section describes the objectives of the current work in the chronological order they are expected to be achieved.

1.5.1 General Objective

The general objective is to create a SaaS application to provide an interface for data analysis on biogas data.

1.5.2 Specific Objectives

- Study related work on ADs, searching for applications of machine learning algorithms in AD data.
- Survey the related software that manages digesters, searching for SaaS applications.
- Define which machine learning algorithm will be used.
- Define the technology stack (languages and frameworks)
- Develop the system using the defined technology stack
- Document the research steps in the report

1.6 JUSTIFICATION

A survey was performed in order to determine the current state of the technology for AD and biogas. The main focus of the survey is in SaaS applications. The results of the survey suggests that several solutions are available for different needs such as monitoring the AD status (Gas Data (2015) and Green Lagoon (2015)), evaluating the financial gains from AD operations (BT IT (2015)) and simulating the digestion process in laboratory (Bioprocess Control (2015)). In addition, Sota Solutions (2016) designed a software that uses Neural Networks to predict and optimize biogas production. Therefore, the AD technology is well established, even though Labatut and Gooch (2012) states that lack of control is a major contributor for AD failures.

This work aims to provide information about the AD and biogas production, performing predictions with historical data from AD sensors, electric energy production engine and laboratory analysis. In order to improve biogas production, it is important to predict the behavior of the biogas plant, simulating how different conditions such as pH, temperature, pressure, retention time and others affect the production of biogas.

By performing predictions, the user can potentially find a set of conditions that improve biogas output, and might be able to control these conditions in a real world biogas plant. The use of ML algorithms by this project will not require the user to know the implementation details of such algorithms, as well as the interface itself should be easy to use and accessible from anywhere. The implementation is also open source, allowing other researches to continue improving the proposed implementation, adding more features and ML implementations.

1.7 METHODOLOGY

This section describes how the hypotheses were tested and how the research was conducted.

1.7.1 Methods

The project uses the quantitative method. The implemented algorithms provide results that need to be evaluated. The trained algorithms try to model the input data, and the learned models can be also used to make predictions. Those predictions can be compared against real data. Therefore, this is how the hypotheses will be answered:

- The neural network is able to predict biogas production with 90% accuracy: The correlation between the predictions and real data indicates the level of accuracy. The method used to calculate the correlation is described in Section 1.4.
- The neural network prediction accuracy is not significantly worse (about 2%) in comparison with WEKA's $MultilayerPerceptron$ algorithm: WEKA reports the correlation after the model is trained and tested. The correlation of this work's implementation will be compared against WEKA's correlation. If WEKA reports a correlation more than 2% higher than this work's implementation, the hypothesis can be denied.

- The SaaS application gets results in a timely manner: The application time performing file upload, file processing, training and predictions are measured in order to verify that the application delivers results in a timely manner using different workloads.

1.7.2 Procedures

The project uses exploratory research. The data was processed using a neural network, and the outputs of the network were tested in order to discover how to show it in the system, as well as information about the data set. Also, three ML algorithms Neural Network (Weka's Multilayer Perceptron), Linear Regression, K-Nearest-Neighbors were studied to decide which one would be used.

1.7.3 Research Techniques

The hypotheses are validated using experiments. The learned models can be used to predict values, and they can be validated using a training set and a test set (which is a data set that is not part of the original training set). The correlation between the predicted values and real data is used to suggest whether the neural network is behaving correctly. Also, the information discovered using ML can be tested in the real world if the variables can be controlled. Bibliographic research is also used to study applications of machine learning algorithms in the biogas field.

1.8 RESOURCES

This section will list the resources that will be using during the development of the project.

1.8.1 Human Resources

Advisors, college staff, professors.

1.8.2 Material Resources

Computers, books, paper.

1.8.3 Institutional Resources

The LARCC lab, library and IT labs.

1.9 SCHEDULE

Table 1.1 shows the schedule of this work. The gray cells represent the expected periods, and X represent when the activity was performed.

Table 1.1: Activities Schedule

Activity	2015		2016							
	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Ago
Write the research project	X	X								
Deliver the project		X								
Study the related work		X	X	X	X	▨				
Study ML algorithms	X	X	X	X	X	X	X	▨		
Develop the system						▨	X	X		
Write the final thesis report				▨	▨	X	X	X	X	
Present the work										X

1.10 ESTIMATED COSTS

Table 1.2 shows the financial resources needed to complete the project.

Table 1.2: Estimated Costs

Item	Amount	Unit Value	Total Value
Number of printouts	800	R$ 0,15	R$ 120,00
Spiral Bindings	3	R$ 5,00	R$ 15,00
Hardcover bindings	1	R$ 70,00	R$ 70,00
Work hours	500	R$ 40,00	R$ 20.000,00
Total	**R$ 20.205,00**		

CHAPTER2: LITERATURE REVIEW

2.1 MACHINE LEARNING IN THE AGRICULTURAL CONTEXT

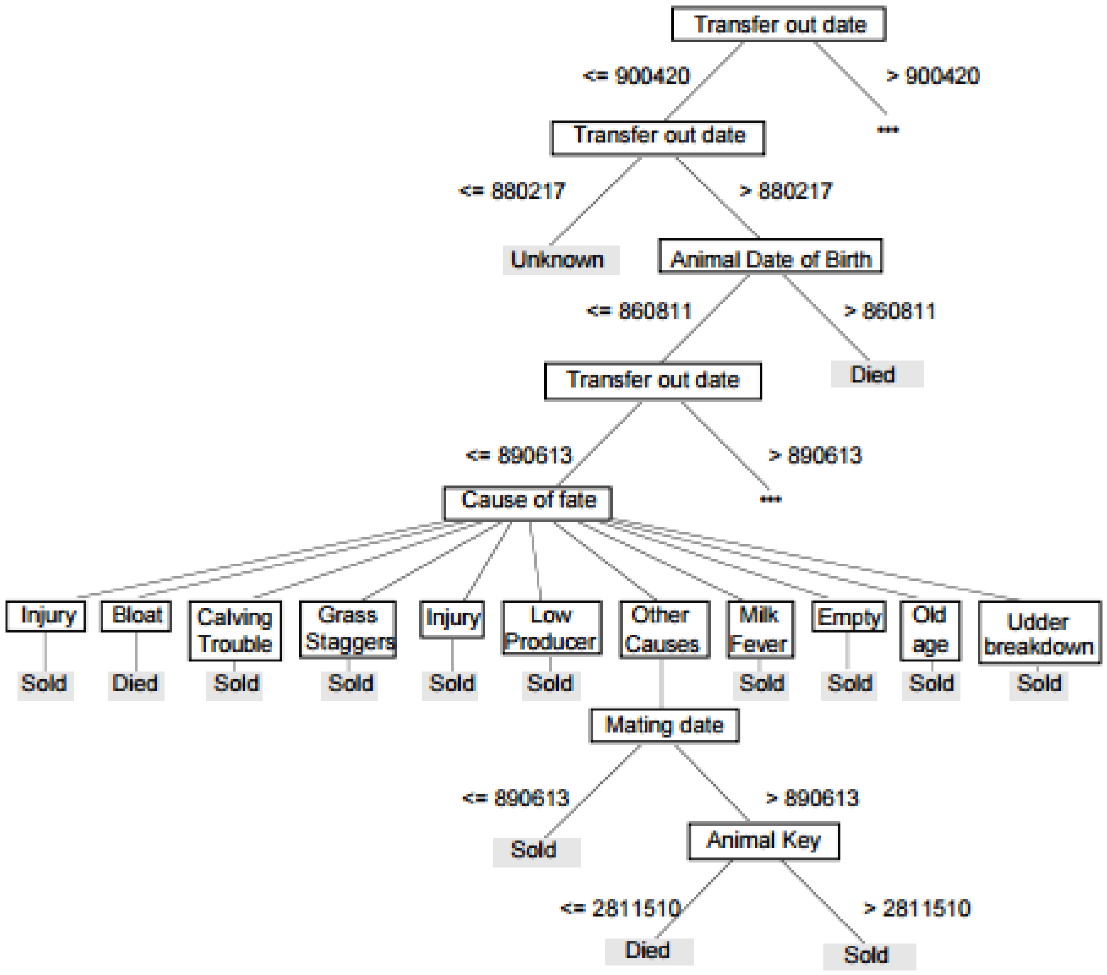
This section will present some scientific works where ML is applied to several areas of agriculture, including research on biogas and ADs.

2.1.1 Decision trees for herd culling

McQueen et al. (1995) discuss the application of machine learning methods applied to agricultural data, specifically on dairy herd culling data. Agriculture is the base economic activity in New Zealand and dairy is one of the largest parts of the agricultural sector, so it is important to improve the dairy productivity.

The authors worked in collaboration with The Livestock Improvement Corporation, a organization created to improve the genetics of dairy cows in New Zealand. A main problem for farmers is to decide whether an animal should be culled. This decision may involve several variables, such as the animal's age, health problems, undesirable behavior, or not being in calf for the following season.

The work used the C4.5 algorithm to analyze a data set of 19000 records, each record containing 705 attributes. The first attempt resulted in a very complex decision tree that takes into account irrelevant parameters for the problem such as the identification key of the animal, the mating date and others.

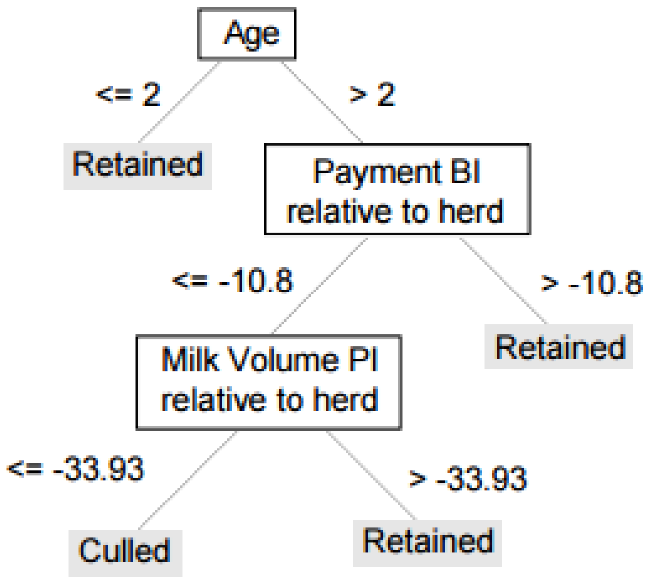


Source: McQueen et al. (1995)

Figure 2.1: Decision tree from raw data set

The data set also had another set of problems: missing data, irrelevant data, missing important attributes and others. One of the biggest problems is that some types of data were collected in different time periods: yearly, monthly or when available. The work focused on making adjustments to the data set, with the help from The Livestock Improvement Corporation. After doing all the necessary adjustments, the C4.5 algorithm was used again, and the resulting decision tree is much more compact, and more plausible from a farming perspective.

McQueen et al. (1995) shows that the quality of the data is a very important aspect of machine learning. By doing the necessary adjustments to improve the quality



Source: McQueen et al. (1995)

Figure 2.2: Decision tree from processed data set

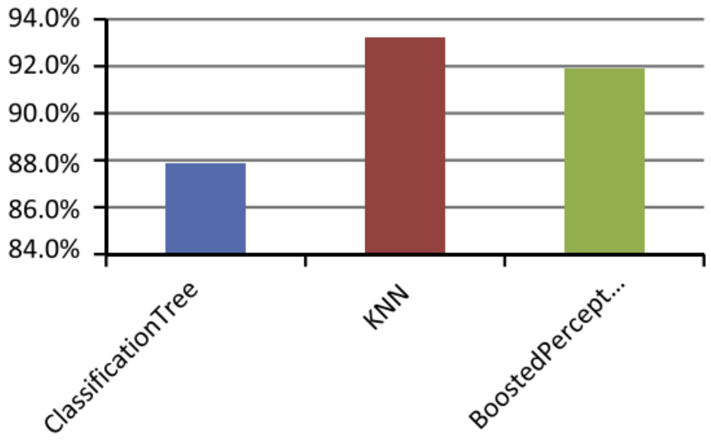
of the data, the C4.5 algorithm was able to extract a compact decision tree plausible to be used in farms.

2.1.2 Machine learning for soil drying prediction

Coopersmith et al. (2014) evaluated whether it is possible to predict the soil drying only from publicly accessible data in the United States. The authors found gaps in the literature about soil drying prediction. These gaps are related to information availability. For instance: land owners might not install soil sensors due to financial limitations, limited accessibility or technology complexity.

When the soil is wet, it is hard for farm workers to use their equipment, since they can become mired in off-road areas, causing expensive delays. For this work, the authors had help of experienced workers in the agricultural area in order to create the training set.

Also, the research made by Coopersmith et al. (2014) is important in order to improve information accessibility to many farmers throughout United States. Furthermore, the information used by the work is likely to become available globally from satellite sensors in near future.



Source: Coopersmith et al. (2014)

Figure 2.3: Prediction agreement on validation data

As shown in Figure 2.3, the work uses three algorithms: Classification trees, K-nearest-neighbors (KNN) and boosted perceptrons. While the classification trees had the worst performance, the boosted perceptrons and KNN algorithms performed with 92-93% accuracy on validation data, respectively.

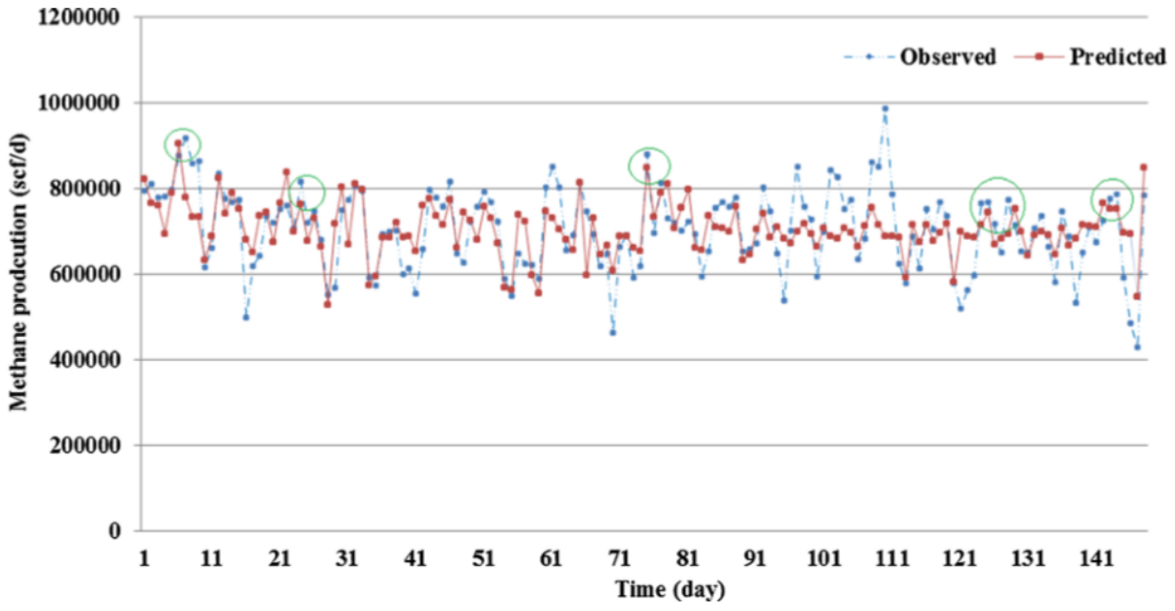
The work shows that even in a scenario where information is limited, machine learning algorithms can perform well, solving problems when no other alternative is feasible.

2.1.3 Prediction of methane production on wastewater treatment facilities

The work of Kusiak and Wei (2014) tried to predict the methane production in a wastewater treatment facility. The work was conducted on the Des Moines Wastewater Reclamation Facility in one of its complexes, which contains 2 digesters.

A data set of 577 records was used to train an ANFIS - Adaptive Neuro-Fuzzy

Inference System algorithm available in Matlab 10.0. Another data set with 148 records was used as a test data set. The data contained 10 features: Flow Rate, Total Solids, Volatile Solids, Volatile Loads, Organic Loads, Detention Time, Sludge Retention Time, Digester 1 temperature and Digester 2 temperature. The parameter Volatile Solids was removed because it is redundant with volatile loads, and Sludge retention time was removed because it is not important to the digestion process. The figure 2.4 shows the results achieved by the algorithm, when testing it with the test data set with 148 records:

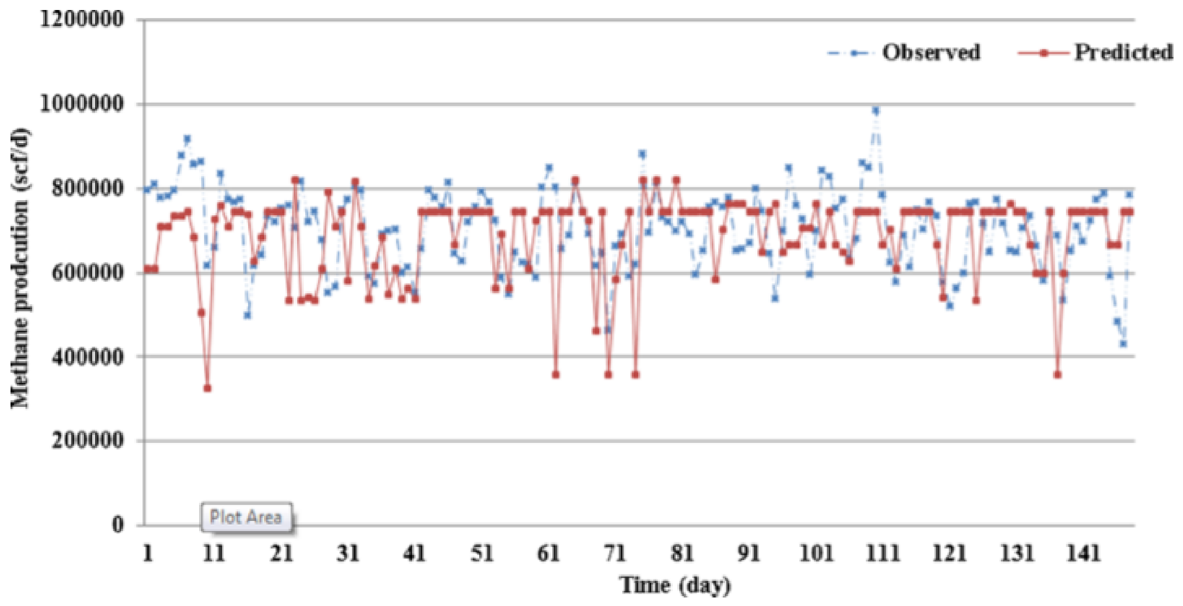


Source: Kusiak and Wei (2014)

Figure 2.4: Prediction of the ANFIS algorithm

Kusiak and Wei (2014) also tested other algorithms. One of them is the K-Nearest-Neighbors algorithm, which yielded different results for the same testing data set, as shown in Figure 2.5.

The authors conclude that the ANFIS algorithm yielded the best results, because the predictions are closer to the actual observations than the other algorithms, achieving a correlation coefficient of 0.99. The K-Nearest-Neighbors yielded the worst results, since the predictions do not fit the observed data very well.



Source: Kusiak and Wei (2014)

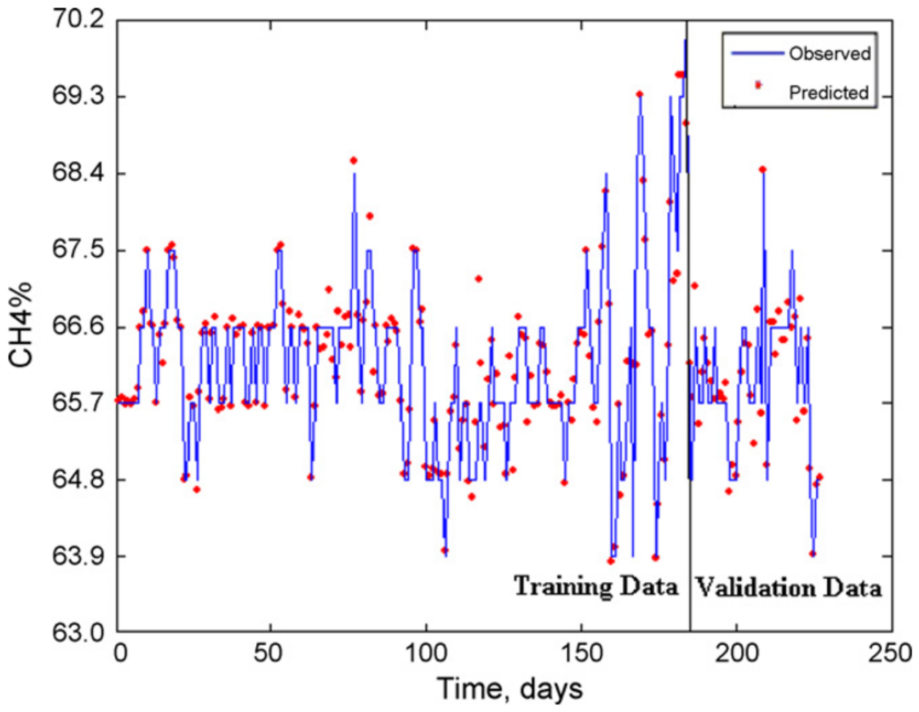
Figure 2.5: Prediction of the K-Nearest-Neighbors algorithm

2.1.4 Prediction and optimization of biogas production

The work of Qdais, Bani-Hani and Shatnawi (2010) tried to create a neural network to optimize the output of methane gas. The work was conducted on the Russeifah biogas plant that belongs to the Jordan Biogas Company. The data analyzed has 177 records and several features, such as Total Solids, Total Volatile Solids, pH and and temperature.

The resulting neural network was tested against another data set with 50 records. The training and validation results are shown in Figure 2.6.

Qdais, Bani-Hani and Shatnawi (2010) consider that the neural network was able to predict the methane output with a correlation coefficient of 0.87. To optimize the methane output, the authors used a genetic algorithm. The neural network receives as parameters four features of the data set described previously. The genetic algorithm tried to discover the parameters that makes the neural network yield the most methane production.



Source: Qdais, Bani-Hani and Shatnawi (2010)

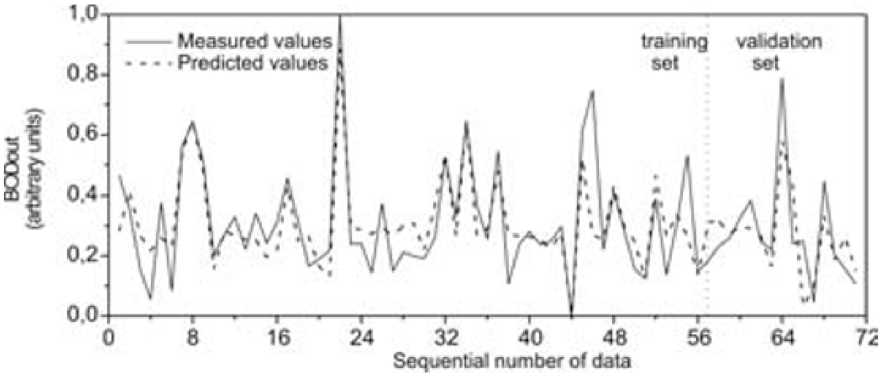
Figure 2.6: Prediction of the neural network algorithm

After several runs, the genetic algorithm discovered that the following parameters yield a methane production of 77% (relative to the total production of gas): temperature at 36°C, Total Solids 6.6%, Total Volatile Solids 52.8% and pH 6.4. The recorded data shows a peak of methane production at 70,1%. The results suggest that the power plant could improve the methane production by 6,9%.

2.1.5 Simulation of industrial wastewater facility using neural networks

Oliveira-Esquerre, Mori and Bruns (2002) studied the application of neural networks and principal component analysis to the problem of simulating a wastewater treatment plant. The goal was to predict the Biochemical Oxygen Demand (BOD) of the output stream of a wastewater treatment plant (RIPASA S/A Celulose e Papel). The research used a data set of 71 records and 8 parameters, but not all of these parameters were meaningful, since they did not contribute much to the variation of the output variable.

A technique called PCA (Principal Component Analysis) was used in order to improve the prediction accuracy. The research found out that a simple feed-forward neural network with a single hidden layer did not provide satisfactory results. Before using PCA, the neural network hit a correlation index of 0.60, while the neural network with PCA hit 0.77. Figure 2.7 shows the results of the neural network.



Source: Oliveira-Esquerre, Mori and Bruns (2002)

Figure 2.7: Prediction of the neural network with PCA

The research indicates that neural networks can represent highly nonlinear relationships, and also shows that pre-processing the data with techniques such as PCA can lead to improved neural network performance.

2.1.6 Remarks

The intent of this research on machine learning for agriculture is to learn how these algorithms are being used, not only for biogas but also for other agricultural areas. Each work brings knowledge about how to use several ML algorithms.

In general, the studied works do not use big quantities of data and still get good results out of several ML algorithms, including neural networks. Furthermore, some of these works had to deal with several problems, including lack of data and poor data set quality. Significant effort is done ensuring that the training data set has good quality in order to eliminate mispredictions. However, this work will focus on provide the tools to allow the user to perform predictions using a neural network.

2.2 DATA ANALYSIS

Data analysis, as explained by Runkler (2012), is the application of computer systems to the analysis of big data sets in order to reveal information to support decisions. Data analysis is a very interdisciplinary field, adopting concepts from areas such as statistics, pattern recognition, computational intelligence and machine learning.

Watson (2014) states that data analytics is an umbrella term for data analysis. The author explains that the term has been used in different ways, from decision support systems to business intelligence applications. Moreover, multiple interpretations of data analysis can be made, for instance, analyzing data in a BI system or using machine learning algorithms. The author points out that stored data generates no business value, however, this data can be analyzed to generate business value by extracting useful information. In the next sections, some terms of the data analysis field will be discussed.

2.2.1 Big Data

The term Big Data is often used as an umbrella term, as described by Dumbill (2012), referring to data that grows in size too fast or does not fit the structures of a database architecture, thus being necessary to process it with alternative methods. This data may contain useful patterns and information, therefore, it is important to analyze it. In organizations, this information can be used, for instance, to enable new products and to do analytical jobs.

Companies such as Google and Walmart have been able to analyze such amounts of data, but the cost was often too high for less resourceful organizations. However, the latest developments in cloud computing are enabling big data processing with a reasonable cost.

Magoulas and Lorica (2009) explains that big data is when the data management process has to take performance and data size requirements into consideration. For some companies, when the data is gigabytes in size, it may be sufficient to reconsider their data management options. For other companies, such problems might arise only when the data goes over terabytes in size. The aspects of Big Data can be summarized by the “three Vs”: Volume, Velocity and Variety.

Dumbill (2012) explains that the volume is the most immediate challenge of big data. In one hand, as the data grows, more storage and better distributed models are needed. In the other hand, having a large data set is often better than having better analysis models.

Velocity is the rate at which data flows through the organization or the system. This has become an increasingly bigger problem as new technologies such as smartphones and new analysis methods emerge. For instance, online retailers can analyze customer interaction at the level of clicks in the website, and not just sales.

The fast moving data is often called “streaming data”, which sometimes can be too fast to store it in its entirety. According to Dumbill (2012), some level of analysis might occur to determine which data is going to be stored. In more extreme cases, some data has to be discarded. For instance, the Large Hadron Collider at CERN generates data so fast that the majority of it needs to be discarded.

Another aspect of Big Data is its variety: data comes in all shapes and sizes, and rarely will have a structured format. Such data might be raw sensor data, social network stream of texts (such as Twitter), and other types of unstructured data. A principle of Big Data is to store everything when possible, because the act of processing it (transforming the unstructured data into something meaningful) often end up resulting in data loss. Some of the data thrown away might have useful pieces of information. There is no way to extract it if the source data has been thrown away, therefore, one of the principles of big data is to store everything when possible.

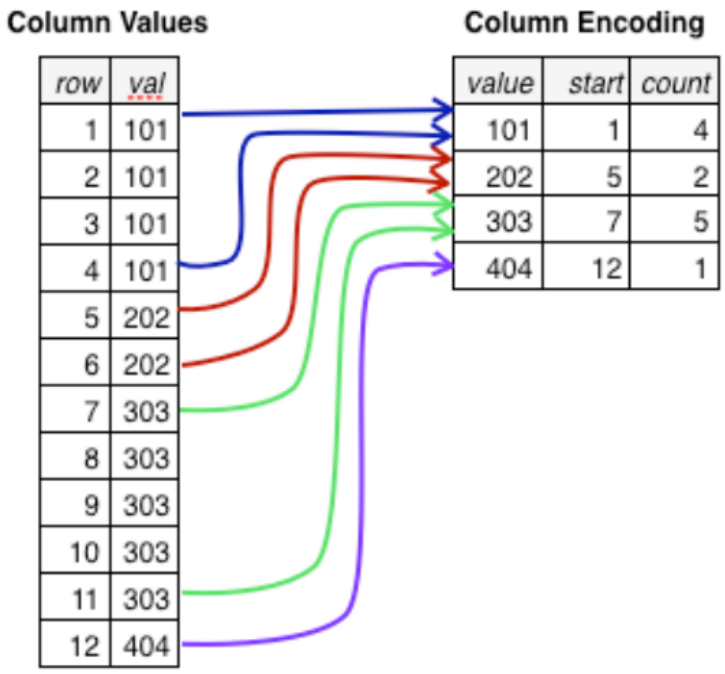
Magoulas and Lorica (2009) explains that two technologies are key factors to build big data systems: Massively Parallel Processing (MPP) and Column-Oriented Databases.

2.2.1.1 Column-Oriented Databases

According to Magoulas and Lorica (2009), relational databases usually store data as rows. Column-Oriented databases store them as columns instead, enabling the database to compress the data, reducing the storage needed to handle the full data set, and the I/O required to store and retrieve the data. It also increases the amount of data that can be stored in memory. The users can still use SQL-like languages to work with the database, while the column orientation is implemented in the database engine. The author illustrates in Figure 2.8 how the columns are encoded in column-oriented databases.

Instead of storing every row and column, the database can compress it in different ways. Figure 2.8 shows one strategy, which is to store the ranges of repeating values for each distinct value in the column. However, the compression may not perform well then the column has a large number of distinct values.

The authors conclude that column-oriented databases enables the use of commodity hardware, and the performance is still good, but it requires planning for large data sets. The compression can slow down write operations and compression might not perform well for data sets with high cardinality (columns with many distinct values).



row	val
1	101
2	101
3	101
4	101
5	202
6	202
7	303
8	303
9	303
10	303
11	303
12	404

value	start	count
101	1	4
202	5	2
303	7	5
404	12	1

Source: Magoulas and Lorica (2009)

Figure 2.8: Compressing of database columns

2.2.2 Data Mining

Data mining, according to Runkler (2012), is the extraction of knowledge from data, where knowledge is defined as interesting patterns that are useful and understandable to humans. Ratner (2012) argues that the definition of Data Mining is the use of statistics and exploratory data analysis for big and small data, using computers to learn the structures within the data.

The term first appeared in the database marketing community between 1970 and 1980, as explained by to Ratner (2012). Techniques used by data mining were already known by statisticians of the time, so the term caused confusion when it first appeared. The author also explains that data mining was already known for a long time, under different names.

O'Brien and Marakas (2007) explains data mining in an organizational context.

Data mining consists of analyzing data to discover hidden patterns in the organization history, where the history could be located in a Data Warehouse. These patterns are revealed by advanced algorithms that use mathematical and statistical techniques to process terabytes of data, in order to reveal strategical information. Côrtes (2008) similarly explains that data mining is a process executed over big data sets in order to reveal patterns and tendencies that could be used in the decision making process.

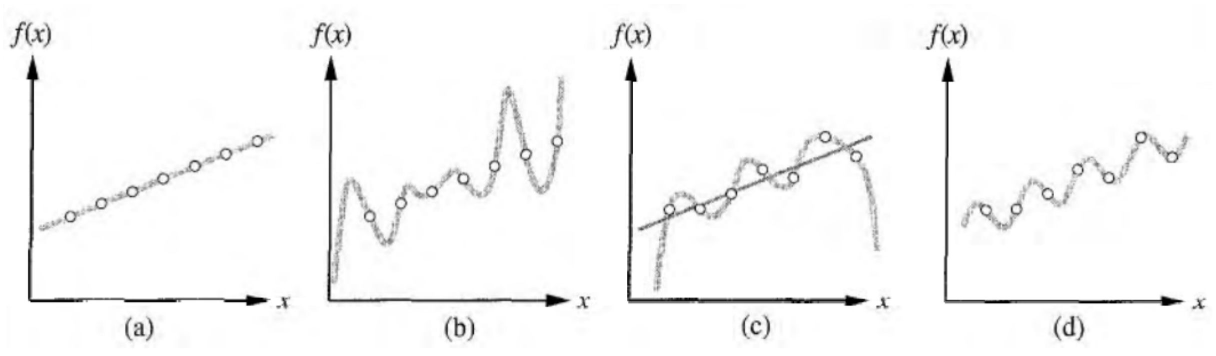
O'Brien and Marakas (2007) gives some examples about data mining. The author argues data mining can be achieved by using algorithms such as linear regression, decision trees, neural networks and clustering. These techniques could reveal, for instance, useful information such as shopping patterns, customer tendencies and profitable relationships not known before in a customer data set.

2.3 MACHINE LEARNING

The term Machine Learning was coined by Artur Lee Samuel in 1959, as explained by Ratner (2012), in which the author defines as a field of study that investigates how computers can learn without being explicitly programmed, acquiring knowledge from data and learning how to solve problems.

According to Bittencourt (2001), machine learning is a subarea of AI. It is also a type of formal reasoning called Inductive Inference, that observes a set of experimental facts and verifies the validity of a hypothesis. Likewise, Norvig and Russel (2004) explain the concept of Inductive Learning: Given a set of examples of a function f, the machine learning algorithm will try to return a function h that is close to the original function f through inductive learning. The function h is called *hypothesis*.

Norvig and Russel (2004) also explain that it is hard to know if a hypothesis is a good approximation of the original function. The authors also introduce the concept of *generalization*, that represents how well a hypothesis predicts the output of new examples. The data set and the hypotheses can be plotted, as shown in Figure 2.9.



Source: Norvig and Russel (2004)

Figure 2.9: Plots of different hypotheses

Figure 2.9 shows four plots for different functions. The hypotheses for plots A and B are consistent with the data, and so are C and D. However, the hypothesis A is simpler than the hypothesis B. The authors recommend that the simpler hypothesis should be the preferred one. Also, the hypothesis shows how well the machine learning algorithm is extracting a pattern from the data. The plot C and D are also consistent with the data, but Norvig and Russel (2004) explains that the hypothesis C is failing, due to its complexity. It is not expected that the hypothesis C will generalize well, meaning that new data samples might not fit in the line. Plot D is expected to perform better, because it is a simple sinusoidal function.

Norvig and Russel (2004) also explains that the true function might not be deterministic. A hypothesis might not be able to be consistent with all the data, but it might be useful due to the possibility of getting reasonable predictions from it. The authors also explain the Computational Learning Theory, that follows the principle that a wrong hypothesis is quickly discarded due to the incorrect predictions. A hypothesis that performs well over a large data set is unlikely to be wrong. A learning algorithm that returns a hypothesis that is Probably Approximately Correct is called a PAC-learning algorithm.

To summarize the concepts of Norvig and Russel (2004), the machine learning field studies algorithms that try to find a hypothesis that approximately predicts the data, based a data set of examples.

2.3.1 Supervised Learning

Norvig and Russel (2004) defines that supervised learning algorithms learn through examples of inputs and outputs. Likewise defines Hastie, Tibshirani and Friedman (2008), also explaining that the term *inputs* might also be called *features*. The authors also explain that this category is called “supervised” because the data set has an output variable that guides the learning process.

The supervised learning algorithms are able to predict quantitative values (for example, biogas output prediction) or qualitative values (for example, classifying whether an e-mail is spam or not). The problem of predicting a quantitative value is called a *regression* problem, while the problem of predicting a qualitative value is called *classification* problem (Hastie, Tibshirani and Friedman (2008) and Barber (2012)).

Barber (2012) defines that the task of a supervised learning algorithm is to learn the relationship between the inputs and outputs in a data set, so that when a new input (not in the original data set) is given, the the predicted output is accurate. The author gives an example: Considering a database of face images, each image labeled as *male* or *female*, the task of a supervised learning algorithm is to predict whether a new image in the data set is a male or female face. The use of machine learning in this case is necessary, because it is difficult to formally specify a rule that differentiates male and female faces.

2.3.2 Unsupervised Learning

According to Norvig and Russel (2004), unsupervised learning algorithms relies only on the input values, when there is no output values. Hastie, Tibshirani and

Friedman (2008) in an analogy, explains that unsupervised learning is like “learning without a teacher”, where the teacher is the output value that exists on supervised learning algorithms, but does not exist on unsupervised learning. In this case, an algorithm needs other ways to analyze the data. Several algorithms fit in this category, for instance: cluster analysis and self-organizing maps.

Barber (2012) defines that in unsupervised learning, the goal is to find a plausible compact description of the data. In this case, the output variable that guides the learning is not present. The author gives an example, where this kind of algorithm might be used to discover consumer behaviours in supermarkets. However, Hastie, Tibshirani and Friedman (2008) points out that it is difficult to validate the output of most unsupervised learning algorithms, and one has to rely on heuristics and judgement to verify the quality of the results.

2.3.3 Overfitting

Murphy (2012) explains that the prediction models should be carefully developed in order to avoid overfitting. Overfitting occurs when the model takes into account every single variation of the input. Some of the input is likely to be noise instead of true signal. If a model tries to predict an output taking into account every single detail from the training set, it is likely to result in mispredictions.

Overfitting also occurs if the data set is very complex, according to Norvig and Russel (2010). Overfitting becomes more likely to happen if there is a large number of attributes in each training example. But overfitting becomes also less likely to happen if there is a large number of training examples. The authors also explain that the input variables irrelevant for the problem should be eliminated, since they increase the probability of overfitting.

2.3.4 Machine Learning Algorithms

The next items will describe some machine learning algorithms. This work will try to predict output variables based on input variables, therefore supervised learning algorithms are presented.

2.3.4.1 Linear Regression

As explained by Hastie, Tibshirani and Friedman (2008), linear regression was developed in statistics even before computers, but is still used due to its simplicity and accuracy. Sometimes linear regression can perform better than more complicated algorithms when there is not a large number of examples and the data has a low signal-to-noise ratio.

Barber (2012) explains that the linear regression algorithm takes $X = x_1, x_2...x_n$ as input data and $Y = y_1, y_2...y_n$ as the output continuous value for each input in X. The goal is to find a function h (the hypothesis) which takes the following form:

$$h(x) = a + bx \tag{2.1}$$

where x is the input value, and a and b are the parameters for the hypothesis. The algorithm must find values for a and b that minimizes the cost function, which calculates the error between the prediction and the actual value. According to Norvig and Russel (2010), one way to find the errors is by using the following function:

$$Error(a, b) = \sum_{j=1}^{N} (y_j - (a + x_j b))^2 \tag{2.2}$$

It is possible to find the parameters a and b that provides a low error rate using techniques such as Gradient Descent, resulting in a hypothesis h that represents a straight line that better represents the data set.

An example of a problem that can be solved using linear regression is the prediction of the price of a house, based on its area in m^2. Linear regression algorithms can also take more attributes as input, such as: house size, number of rooms, number of bathrooms, and many others.

There is also another algorithm called Logistic Regression, in which the output value is a discrete value, instead of a continuous value. Norvig and Russel (2010) explains that logistic regression uses the sigmoid function to classify an input in 2 classes. The sigmoid function is:

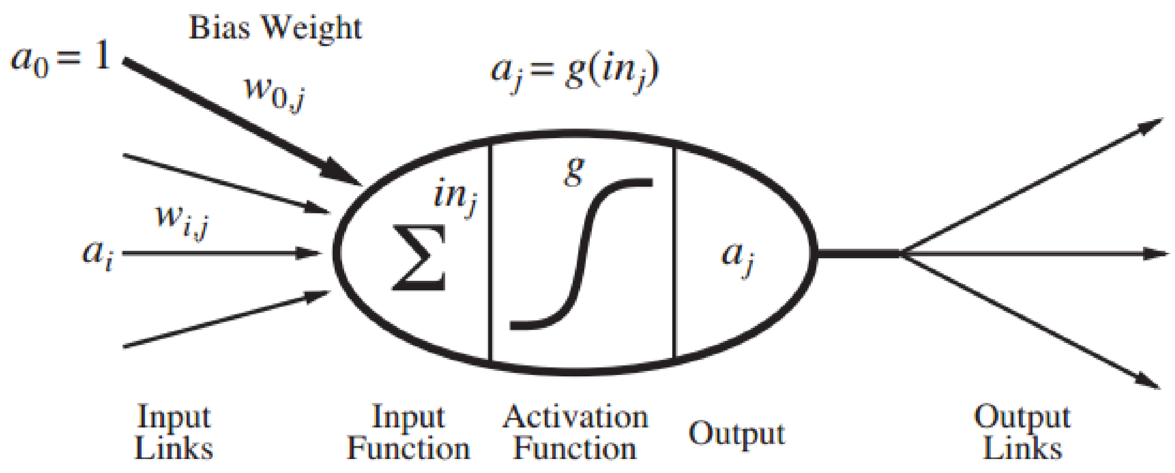
$$Sigmoid(x) = 1/(1 + e^{-x}) \tag{2.3}$$

By using the sigmoid function and a different cost function, it is possible to classify data in 2 classes. A simple example of logistic regression is to classify whether a tumor is malignant or not, based on its size and other attributes.

2.3.4.2 Neural Networks

Neural Network is one of the most popular and effective learning methods, as explained by Norvig and Russel (2010). A neural network is composed of several nodes (or neurons) and layers, linked to each other in order to form a network.

Each node is composed of a set of weight values for each node in the previous layer w_ij and has links associated to each of these nodes. The neuron computes a weighted sum in the form:



Source: Norvig and Russel (2010)

Figure 2.10: Mathematical model of an artificial neuron

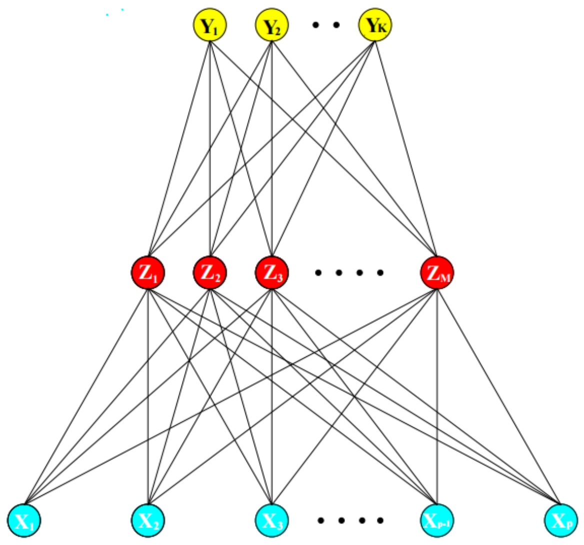
$$input = \sum_{n=1}^{N} a_n w_n \tag{2.4}$$

where N is the number of nodes in the previous layer, a_n is the activation value of each node in the previous layer and w_n is the weight value trained for that specific link.

The input value is used as input for the activation function. The result becomes the activation value for that node. By completing these steps for all nodes and layers, the feed-forward computation of a neural network is done. The end product is the prediction made by the network.

A neural network is composed by layers. The first layer is the input layer, a layer used to feed data into the network. The last layer is the output layer, which represents the result of the computation made by the network. A neural network can be composed of many layers in between the input and output layer, which are called hidden layers.

As shown in Figure 2.11, the neural network can contain any number of hidden



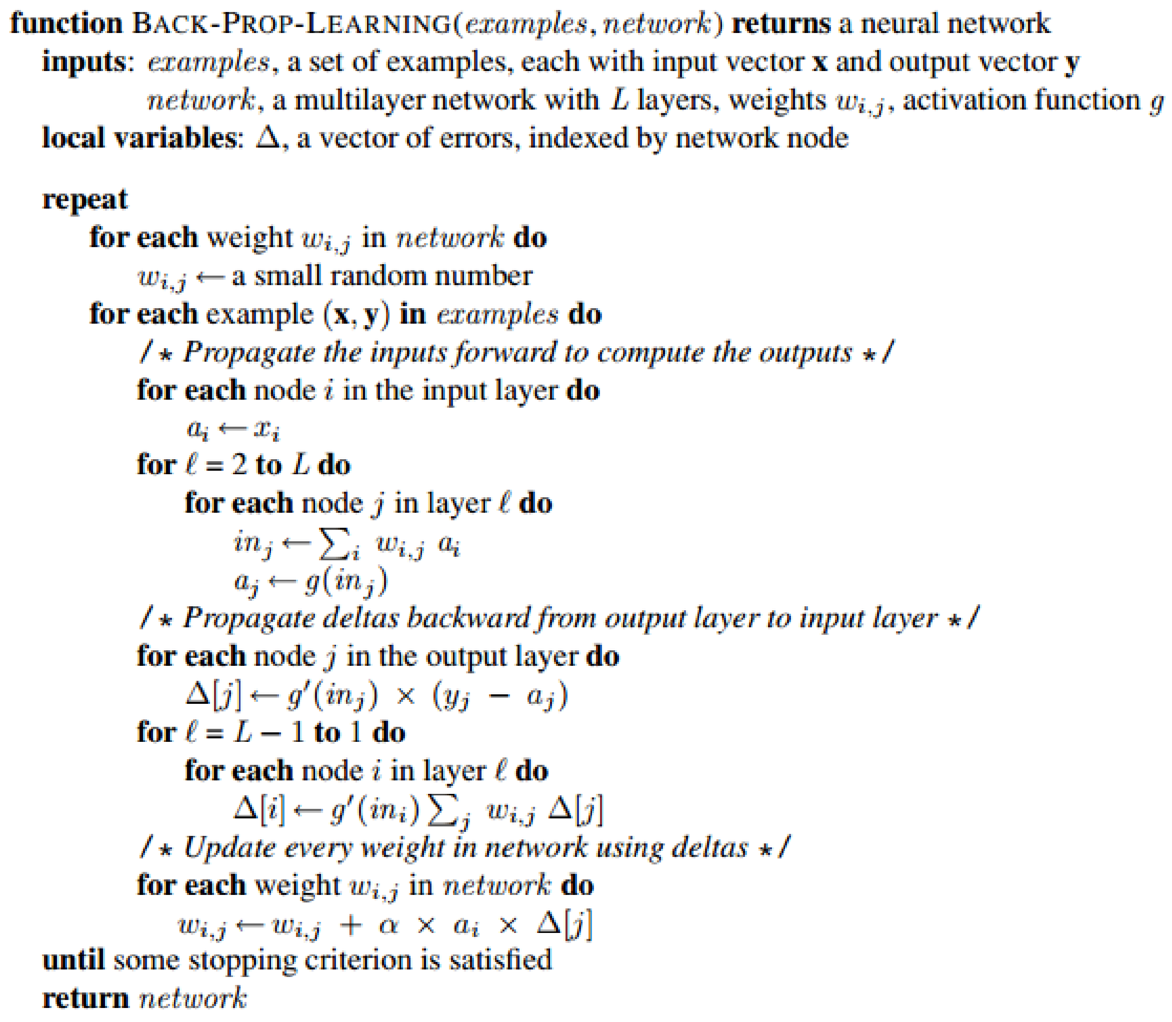
Source: Hastie, Tibshirani and Friedman (2008)

Figure 2.11: Neural Network representation

layers. According to Hastie, Tibshirani and Friedman (2008), it is better to have many layers and nodes than just a few. If there is not enough hidden layers and nodes, the neural network might not be able to capture the non-linearity of the data.

Training a network requires the use of a learning algorithm, such as the Back-propagation algorithm. This algorithm, as explained by Norvig and Russel (2010), calculates the error of each node and updates the network weights as needed.

Figure 2.12 shows the pseudocode for the back-propagation algorithm. It is important to notice that the first layer is not calculated, because it is the input layer, which has no weights associated with. Also, as shown in Figure 2.10, there is always an extra node called “Bias unit”, represented in that figure by the Bias Weight. In that case, the activation value for the bias unit is always 1, therefore, this node does not need to be calculated.



```
function BACK-PROP-LEARNING($examples$, $network$) returns a neural network
  inputs: $examples$, a set of examples, each with input vector x and output vector y
          $network$, a multilayer network with $L$ layers, weights $w_{i,j}$, activation function $g$
  local variables: $\Delta$, a vector of errors, indexed by network node

  repeat
      for each weight $w_{i,j}$ in $network$ do
          $w_{i,j} \leftarrow$ a small random number
      for each example $(\mathbf{x}, \mathbf{y})$ in $examples$ do
          /* Propagate the inputs forward to compute the outputs */
          for each node $i$ in the input layer do
              $a_i \leftarrow x_i$
          for $\ell = 2$ to $L$ do
              for each node $j$ in layer $\ell$ do
                  $in_j \leftarrow \sum_i w_{i,j}\, a_i$
                  $a_j \leftarrow g(in_j)$
          /* Propagate deltas backward from output layer to input layer */
          for each node $j$ in the output layer do
              $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
          for $\ell = L - 1$ to $1$ do
              for each node $i$ in layer $\ell$ do
                  $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
          /* Update every weight in network using deltas */
          for each weight $w_{i,j}$ in $network$ do
              $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return $network$
```

Source: Norvig and Russel (2010)

Figure 2.12: Back-propagation algorithm

2.3.4.3 K-Nearest Neighbors

According to Murphy (2012), the K-Nearest Neighbors (KNN) is a simple algorithm that works by searching the K nearest points to x, where $x$is the input value for which the output will be calculated.

First, all the distances are calculated using a formula. One of them is the Euclidean Distance. For regression problems, the output value of the prediction is an average of the K closest points. For classification problems, each point in the data set belongs to a class. The classes of the k-nearest points are counted, and the class with more data points in the k-nearest points is the final result of the prediction.

Hastie, Tibshirani and Friedman (2008) explains that the K-Nearest Neighbors is a simple algorithm, but has been used with successful in many problems, including classifying handwritten digits. The algorithm normally succeeds when the decision boundary is very irregular. However, Murphy (2012) notes that the algorithm suffers from the curse of dimensionality, an effect that happens when the data set has many features (many dimensions).

2.4 DISTRIBUTED SYSTEMS

Tanenbaum and Steen (2007) defines that a distributed system is a set of independent computers that is seen by its users as a unique and coherent system. The users always think that they are working with a single system, therefore the independent components need to collaborate. The core aspect of distributed systems is how to establish this collaboration. The author also considers that an important aspect of distributed systems is that the communication, organization and specification of the independent computers is often hidden to the users. A distributed system should offer easy access to its resources and should be extensible.

According to Coulouris, Dollimore and Kindberg (2005), distributed systems have computers communicating though a network by passing messages. Distributed systems have the following characteristics: concurrency, lack of a global clock and independent failure of components. One example of a distributed system is the Internet.

Tanenbaum and Steen (2007) considers that the advances in technology in the last 50 years made it possible to easily create distributed systems, with many computers communicating through high-speed network connections. Distributed systems are also a shift from the previous centralized systems, that consisted of a single computer and maybe several other remote terminals.

2.4.1 Access to resources

Tanenbaum and Steen (2007) explains that the main goal of a distributed system is to provide easy access to applications, remote resources, and sharing those resources efficiently. One of the reasons to connect users and resources is to provide mechanisms through which users could collaborate and share information, for instance, the Internet. However, security measures have to be taken, but unfortunately this is not the common practice. User privacy is a concern, because the communications made by a user can be monitored, which is an explicit violation of privacy if the user did not give permission to it.

Coulouris, Dollimore and Kindberg (2005) also explain that the motivation for constructing and using distributed systems comes from a desire to share resources, where “resource” can be anything that can be shared in a networked computer system, including printers, disks, files, databases, video and audio streams, and others. The author also uses the Internet as an example of a distributed system that shares resources to users.

2.4.2 Heterogeneity

Coulouris, Dollimore and Kindberg (2005) explain the term Heterogeneity in the context of the Internet. The term refers to the variety and difference of networks, hardware, operating systems, programming languages and implementations.

Despite all the differences among the different nodes of the Internet, there are protocols coordinating the communication between the different elements. If different programs need to communicate with each other, they must agree and adopt standards, in a similar way that the Internet protocols do.

Tanenbaum and Steen (2007) also explain the term, in the context of grid computing. In a cluster computing system, the computers generally have the same specification. In a grid computing system, the computers might have different operating systems, hardware, and other aspects.

2.4.3 Transparency

According to Tanenbaum and Steen (2007), an important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across several computers. There are several types of transparency defined by Tanenbaum and Steen (2007).

2.4.3.1 Access Transparency

Access Transparency, explained by Tanenbaum and Steen (2007), aims to provide a single way to represent data and operate on it, even if the different computers in the network have different operating systems and hardware. For instance, different operating systems have different ways to represent files. A system with access transparency should hide these differences and provide a consistent way to access the resources, regardless of the underlying operating system and machine.

Coulouris, Dollimore and Kindberg (2005) explains that, in the context of a file service, client programs should be unaware of the distribution of files, and programs that work with local files whould be able to access remote files without modification.

2.4.3.2 Location Transparency

Tanenbaum and Steen (2007) defines Location Transparency as the fact that the users do not know the physical location of a resource in the system. A way to get location transparency is to assign logical names to the resources, in which the physical location of the resource is not encoded in the name. An example of location transparency on the Internet is the URL. The URL does not tell where a resource is located.

The URL also provides Migration Transparency, which is the possibility to move a resource around without changing the way it can be accessed. Coulouris, Dollimore and Kindberg (2005) uses the term Mobility Transparency to explain Migration Transparency in the context of file services, stating that the client programs do not need to be changed when files are moved.

2.4.3.3 Replication Transparency

In order to improve availability and performance, the resources can be replicated, as Tanenbaum and Steen (2007) explains. Replication Transparency seeks to hide the fact that there are multiple copies of the same resources in the system. The author explains that a system that supports replication transparency should also be able to support location transparency, so the resource can be found in the same way without having to specify the location.

Coulouris, Dollimore and Kindberg (2005) also explains that clients should not be aware that multiple physical copies of the same data exist. The clients only request the object by asking for its logical name, and expect to receive only one set of values related to this name. The operations, however, can be performed upon several physical copies.

2.4.3.4 Concurrency Transparency

Tanenbaum and Steen (2007) explain that in some cases, several users share the same resource, for instance, a database table. Concurrency transparency means that a user should not be able to notice that another user is using the same resource.

It is important that the system ensures that the resources are always in a consistent state.

2.4.4 Distributed Databases

According to Özsu and Valduriez (2001), a distributed database can be defined as a collection of databases logically related, distributed over a computer network. Distributed Databases shares some of the concepts of Distributed Systems, for instance, Replication Transparency. The next items will describe some of the concepts used in distributed databases.

2.4.4.1 Data Independence

Data Independence, as defined by Özsu and Valduriez (2001), is also an important concept for centralized databases. Data Independence is about the immunity of applications and users to changes on the definition and organization of data, and vice-versa. The author describe that the data definition can happen in two levels. The first level is known as Schema Definition, and the second is the Physical Data Description. There are also two types of data independence: Logic Data Independence and Physical Data Independence.

Logic Data Independence is about the application being immune to schema changes. For example, if more columns are added to a database table, the application should not be affected by it, if the application is dealing with a specific set of columns.

Physical Data Independence hides the details about how the data is physically stored. The application should not need to know how the data is actually stored, and should not be modified if those details change, for example, for performance reasons.

2.4.4.2 Network Transparency

Özsu and Valduriez (2001) explains that in centralized databases, the data must be isolated from the users. But in distributed databases, the network should be isolated too: the user should not need to know the operational details of the network environment. For the user, there should be no difference between using a centralized database and a distributed one.

2.4.4.3 Replication Transparency

Özsu and Valduriez (2001) defines Replication Transparency similarly as Tanenbaum and Steen (2007). The data might be replicated in order to improve performance and availability. As described before, the user should not be involved in the process of replicating the data, and should not be aware of the data being replicated.

2.4.4.4 Fragmentation Transparency

According to Özsu and Valduriez (2001), it is desirable to split the data in smaller fragments, to increase performance, availability and reliability. The author also explains that when the data is fragmented, there’s a need to manage user queries, so the query is executed upon the fragments, and not a single database.

Database fragmentation has two alternatives: Horizontal fragmentation, when a relation (for example, a table) is partitioned in a set of sub-relations, in which those have a subset of the original records. Vertical fragmentation is when each sub-relation is defined as a subset of the attributes of the original relation.

2.4.5 Cloud Computing

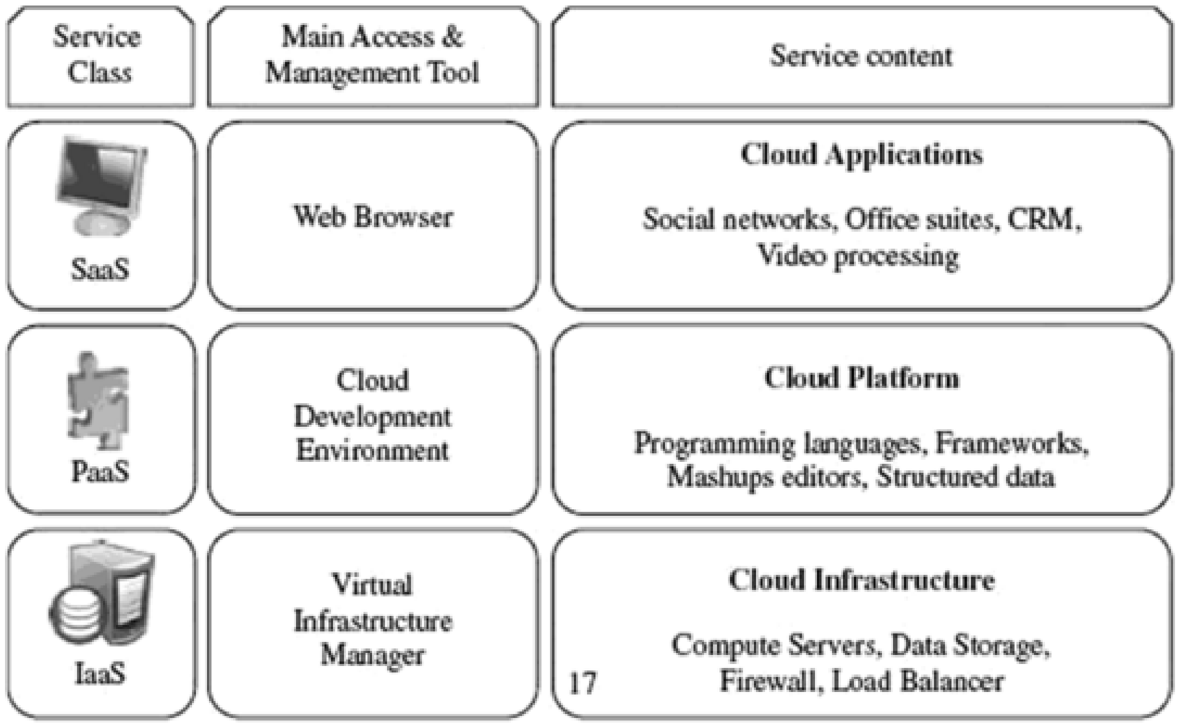
According to Rittinghouse and Ransome (2009), the term “cloud” has been used historically as a metaphor for the Internet. The idea of cloud computing dates back to 1960, but in the middle of the 70’s the idea faded away due to technological limits. But since the turn of the millennium, the term “cloud computing” began to appear in technology circles.

Buyya, Broberg and Goscinski (2010) explains that several attempts to define Cloud Computing have been done. Cloud Computing has been used as an umbrella term that describes computing services offered on-demand by companies such as Amazon, Google and Microsoft. Those services provide an infrastructure viewed by its users as a “cloud”, accessible from anywhere, on demand. The author compiles a list of common characteristics between several definitions made by other authors on the field. These characteristics are:

- Pay-per-use: No ongoing commitment, utility prices;
- Elastic capacity and the illusion of infinite resources;
- Self-service interface;
- Resources that are abstracted or virtualized;

Rittinghouse and Ransome (2009) also explains that cloud computing lowers the barriers for the offering of service solutions, because the customer does not have to purchase the infrastructure needed. Also, the user is not tied to a specific device, since they only need Internet access to use the service.

Cloud computing services are divided in 3 categories: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service).


Service Class
Main Access & Management Tool
Service content
SaaS
Web Browser
Cloud Applications
Social networks, Office suites, CRM, Video processing
PaaS
Cloud Development Environment
Cloud Platform
Programming languages, Frameworks, Mashups editors, Structured data
IaaS
Virtual Infrastructure Manager
Cloud Infrastructure
Compute Servers, Data Storage, Firewall, Load Balancer
17

Source: Buyya, Broberg and Goscinski (2010)

Figure 2.13: Cloud Computing Stack

2.4.5.1 Infrastructure as a Service

According to Buyya, Broberg and Goscinski (2010), the offering of virtualized resources such as computation, storage and communication is called Infrastructure as a Service (IaaS). It enables on-demand provisioning of services running several options of operating systems and a customized software stack. IaaS is considered to be the bottom layer of cloud computing systems, as shown in Figure 2.13.

Rittinghouse and Ransome (2009) in a similar way explain that IaaS is the delivery of computer infrastructure as a service, where the clients outsource all their infrastructure, rather than purchasing servers, software, network equipment, and others. The clients pay for what they have used, so the cost of such services usually reflects the level of activity.

2.4.5.2 Platform as a Service

According to Buyya, Broberg and Goscinski (2010), in addition to infrastructure-oriented clouds, another approach is to provide a higher level of abstraction in order to make the cloud easier to program. This concept is called Platform as a Service. A PaaS environment offers tools that allows developers to create applications without necessarily knowing the hardware details on which it will run. Such applications often use the service's own technology stack (programming languages, databases, and others) to create the application.

Rittinghouse and Ransome (2009) explains that SaaS is closely related to PaaS, but a PaaS environment offers a cloud service to build applications upon, instead of an application to work with. This approach has a main drawback: the services provided by a PaaS environment are often limited by the vendor's design and capabilities. The author also mentions that this means a compromise between development freedom and application predictability, performance and integration

Buyya, Broberg and Goscinski (2010) and Rittinghouse and Ransome (2009) uses the Google App Engine as an example of a PaaS environment. The applications running on Google App Engine should be written in Java or Python and use an specific data storage technology.

2.4.5.3 Software as a Service

As shown in the figure 2.13, the SaaS is considered to reside in the top of the cloud stack, as explained by Buyya, Broberg and Goscinski (2010). SaaS is a model of delivering applications to end users through web portals. Instead of using a locally installed application, users can access it using the Internet. This model of delivering makes it easier to maintain, test and develop software.

In a similar way, Rittinghouse and Ransome (2009) defines SaaS as a model of software deployment. The application is licensed to users on demand, which relieves the end user of the burden of installing all the software they need, since it is possible to access it using web browsers.

Martin and Cendrowski (2014) explains that many variations of SaaS are possible, but a simple explanation is that it is a software accessible via the Internet. The SaaS model has its origins in the 90's, and it predates the term “cloud computing”, which Rittinghouse and Ransome (2009) explains that it only appeared after the turn of the millennium. This deployment model was necessary due to the ever increasing demand for software solutions. The scalability of SaaS solutions make it possible to improve their level of service (such as storage and data capacity). It also allows for a “pay-as-you-need” use, so that the clients only pay for the features and level of service that they require. This is a major reason why the SaaS model is so common in the technology sector.

Buyya, Vecchiola and Selvi (2013) adds that SaaS applications are able to centralize the effort of managing infrastructure and maintain applications, transparent to the users. The resources are optimized by sharing the costs among the user base. This concept is called Multi-tenancy.

Therefore, the SaaS model is in the top of the cloud stack, which allows users to access applications mainly through web browsers, on demand, in a pay-as-you-need model. The application does not need to be installed in the client's computer, and the client does not need a specific operating system to use the application. This model also allows companies to centralize their efforts to maintain the application and reduce costs.

CHAPTER3: EXPERIMENTS AND RESULTS

In this chapter, the results are going to be shown and discussed. A survey of biogas software will be presented as well as an introduction to the LARCC laboratory. Finally, the results of this work will be presented, with conclusions and future works.

3.1 LARCC

LARCC (Laboratory of Advanced Researches for Cloud Computing)[1] is a laboratory located at SETREM (Sociedade Educacional Três de Maio) dedicated to researches on cloud computing technologies. The laboratory works in cooperation with other universities, such as PUC-RS and Unipampa, and is aided by the private sector in the region.

The laboratory published several works that evaluate the characteristics of cloud computing technologies. Vogel (2015) surveyed several open source cloud computing technologies, comparing their features related to their robustness. Other works also explored different features of private open source clouds, such as Roveda et al. (2015), Roveda, Vogel and Griebler (2015) and Vogel et al. (2016).

The laboratory performs research mainly on cloud computing, but has a list of research areas, such as machine learning for agriculture and distributed systems, and works with other areas inside the institution. Bioagropec is the institution project that

[1] http://larcc.setrem.com.br/

does research on ADs, and this work represents a cooperation between LARCC and Bioagropec.

3.2 STATE-OF-THE-ART IN BIOGAS SOFTWARE

As mentioned in Section 1.6 of the Chapter 1, the work performed a survey focused on SaaS biogas software. In this section, some software are going to be presented in order to evaluate their features.

The solutions presented here are focusing on monitoring the biogas production process. An example is Click! System from Gas Data (Gas Data (2015)). Click! focuses on monitoring an array of sensors installed in the biogas plant as well as offering a cloud-based software from which the user can control the system. The solution also includes data transmission via TCP/IP and implements standardized output formats.

Carbon Cloud (Green Lagoon (2015)) from Green Lagoon provides the same cloud-based software features as Click! System, allowing to control the equipment installed in the biogas plant and visualize data, monitoring and notifying via email and SMS. Carbon Cloud's website[2] provides a document including an image of the system, as shown in Figure 3.1

BOGIS (BT IT (2015)) is a SaaS solution that focuses on the economic approach of biogas plants[3]. BOGIS evaluates the profit obtained from biogas operations as well as energy and biogas production in a given time period. BOGIS is also independent of biogas plant manufacturers, and receives data directly from biogas plant interfaces.

Click! System and Carbon Cloud are able to get data from sensors and present them to the final users. The technical team working on the biogas plant can get more

[2]http://www.glt.my/downloads/glt-cloud-monitoring-and-reporting-system.pdf
[3]http://www.bt-it.de/download/flyer.bogis.en.pdf

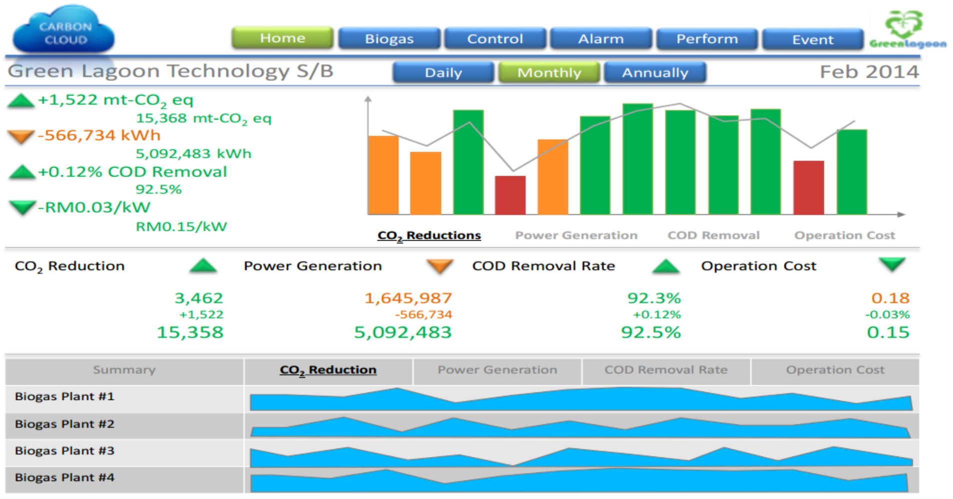


Figure 3.1: Green Lagoon Carbon Cloud

insight about the digestion process, helping on the diagnostic of problems and optimizations of the production process. Therefore, a biogas system should display general information about the biogas processes.

However, discovering methods to improve the biogas production can be done with alternative methods, as shown by Sota Solutions (2016). Sota Solutions provides software focused on simulation and optimization of processes. The company develops a software that predicts energy production from biogas, based on historical data[4] using neural networks. The company found out the neural network method yields predictions with 5% more quality than other comparable procedures such as linear regression.

The applications surveyed are focused on several aspects of the biogas context. Each application serves a different purpose. It is important to perform this survey in order to know what kind of information is important in the context of biogas. Differently than the surveyed applications, in this work a SaaS application was developed to perform biogas prediction in a more general way, in which the user provides the information and determines what information (which is not limited to biogas production, but also energy production, methane levels and others) is important to their scenario.

[4]http://sota-solutions.de/wordpress_en/#anwendungen

3.3 APPLICATION ARCHITECTURE

In this section, the architecture of this work's SaaS application is going to be described. The application is composed of two main parts: the web front-end and the back-end background service. Figure 3.2 shows the application architecture.

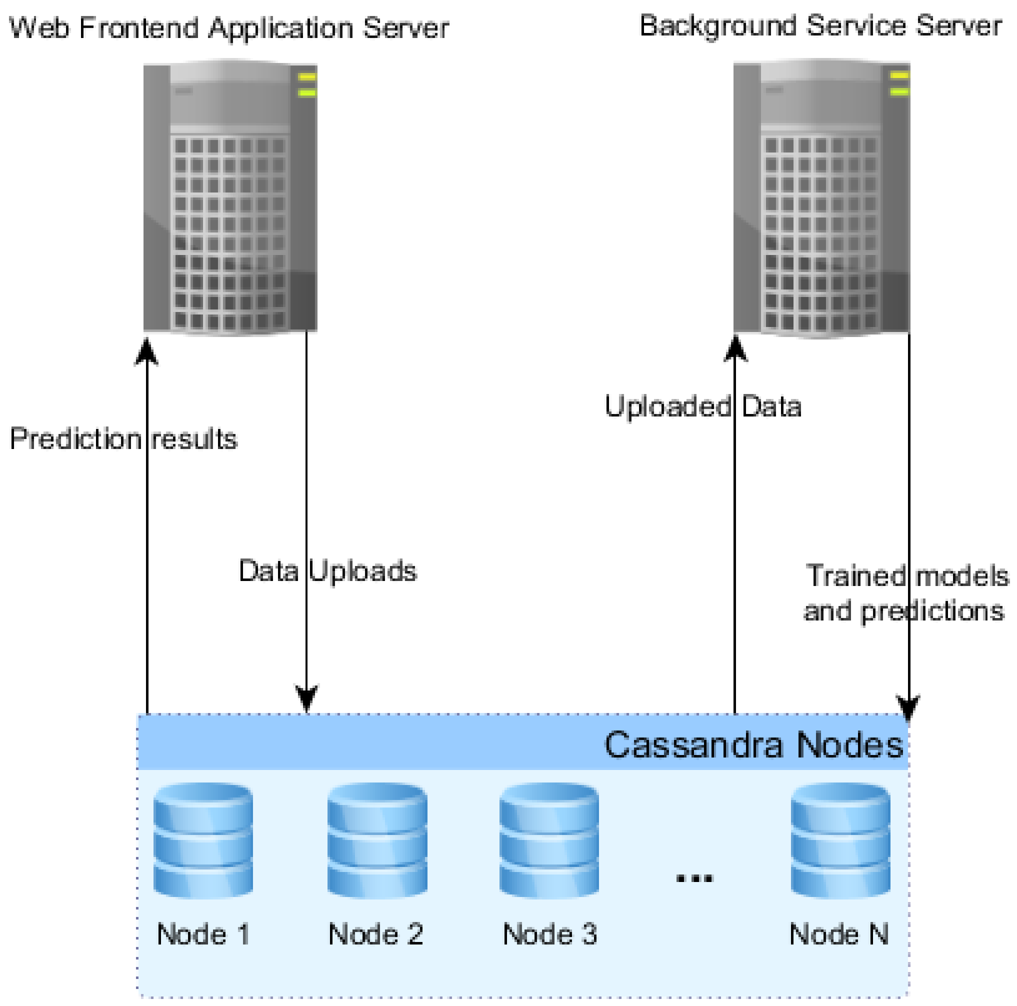

Web Frontend Application Server
Background Service Server
Prediction results
Data Uploads
Uploaded Data
Trained models and predictions
Cassandra Nodes
Node 1
Node 2
Node 3
...
Node N

Figure 3.2: Application Architecture

The front-end web application server does not communicate directly with the background service. Instead, both applications use the Cassandra database to store data and perform background tasks such as training models, perform predictions and process uploaded files. The web application receives the file and stores it in Cassandra. Afterwards, the background service takes the file and process it, doing any transformation necessary to get the data from the file and storing it in a structured database table. Furthermore, the background service uses the data to train models, where these models are also stored in the database to make predictions. The technology stack used in this work will be presented in the following sections.

3.3.1 Web front-end Application

In order to develop the front-end application, several technologies were used. The main component is Node.js[5], an event-driven Javascript server based on the V8 Javascript engine. Javascript has been normally used on the client-side of web applications to provide extra features such as input validation, DOM (Document Object Model) manipulation, and asyncronous requests to the server (Ajax). However, Node.js uses the V8 Javascript engine to enable Javascript on the server, focusing on high-performance and scalability using asyncronous I/O. Node.js has also been used for general Javascript development. For instance, the text editors Atom[6] and Visual Studio Code[7] were built on top of Chromium and Node.js. Due to its general purpose nature, Node is described as "an asynchronous event driven JavaScript runtime". This work uses Node.js due to the usage of the Javascript language. Besides being a very known and familiar language, it is easy to develop applications with, and since it was also used on the client-side, there was no need to use another language for the web application.

However, Node.js only provides basic features and APIs to create applications on top of it, for instance, a low-level HTTP server library[8]. Even though it is relatively simple to serve HTTP requests using only Node.js, there are web frameworks built for Node.js, providing more features suitable to create MVC applications. This work

[5]http://nodejs.org
[6]https://atom.io
[7]https://code.visualstudio.com
[8]https://nodejs.org/api/synopsis.html

uses Express.js [9], which provides common features to create complete web browser applications, namely the routing and view rendering systems.

For client-side development, other libraries were used. The layout aspects are inspired on Material Design[10], which defines a set of rules and guidelines for applications. Material Design was created by Google, and is present in the Android operating system. The framework Materialize [11] is a Javascript and CSS implementation of the Material Design for web applications. Materialize is focused on responsive web apps, thus the web application developed by this work should be able to run on phones and tablets as well. There are other alternatives, such as Angular Material[12], Material Design Lite[13], Material UI[14] and many others, but Materialize was used because it is framework independent, simple to use and already provides many features that the application needs.

To handle page transitions, form submissions and general management of the web interface, Angular.js[15] was used. Angular is a MVVM (Model-View View-Model) framework that lets the developer define dynamic pages and extend the functionality provided by HTML. These can be achieved by using Javascript to manipulate the DOM, but MVVM frameworks already implement several features that are usually needed by web applications. There are other MVVM frameworks such as Ember.js[16] and Knockout.js[17], but Angular.js was chosen because it is easy to integrate with (Ember.js recommends the use of command-line tools to build applications, while Angular.js does not) and feature-rich. However, Angular.js has some different concepts such as directives, scopes, states and services, making the learning curve steeper.

[9]http://expressjs.com/
[10]https://www.google.com/design/spec/material-design/introduction.html
[11]http://materializecss.com
[12]https://material.angularjs.org/
[13]https://getmdl.io
[14]http://www.material-ui.com
[15]https://angularjs.org
[16]http://emberjs.com/
[17]http://knockoutjs.com

DataStax Node.js[18] is used in order to connect with the Cassandra database. The driver provides an easy to use API that fits well the asynchronous programming model of Node.js.

3.3.2 back-end application

The web application is responsible for presenting the data and functionalities to the end-user. The file uploads processing, model trainings and predictions is done by a separate application written in C++, a background service that performs all the needed heavy tasks. This background application was developed using Visual Studio Community 2015 on Windows. Even though the development took place in a Windows environment, the code does not rely on any Windows-only libraries and APIs. The program runs in a Linux environment installed at LARCC, therefore, the program was compiled with GCC using CMake and Make.

The application uses the FLENS library to perform matrix calculations. FLENS is easy to use and allows to link the program against fast math libraries to improve calculation times. This work used Openblas for this purpose, and the results are shown in the section 3.5.3. The use of Openblas made the neural network perform much faster, where the training times went from almost 4 minutes down to around 10 seconds.

Figure 3.3 shows a small class diagram representing the most important classes in the C++ application. It has three main services: "TaskProcessUpload", "TaskTrainModel" and "TaskPerformPredictions".

The "TaskProcessUpload" class monitors the Cassandra database to act upon data upload as soon as possible. The task repeatedly queries the "modeluploads" table because the DataStax C++ driver-at the time of development-did not provide a streaming API as the Node.js driver does. The task validates the uploaded file to

[18]https://github.com/datastax/nodejs-driver

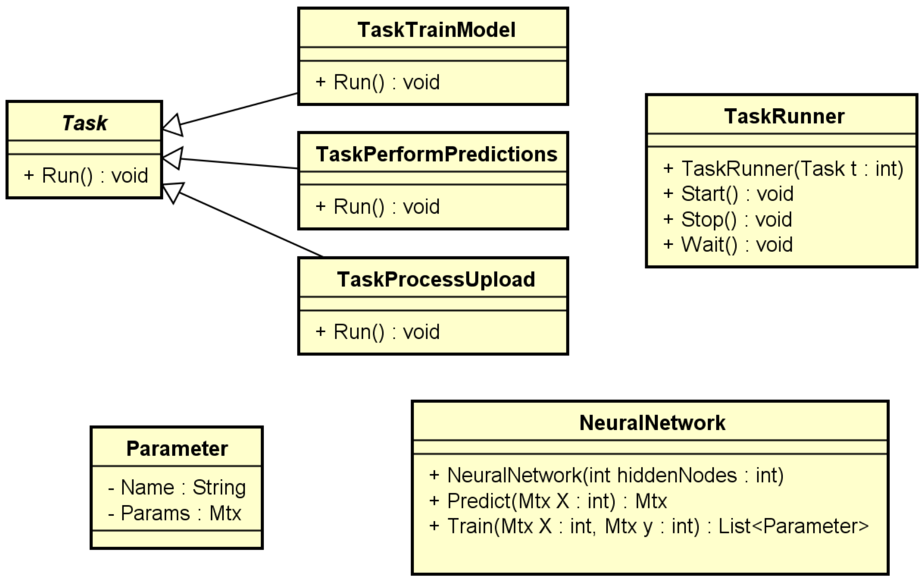


Figure 3.3: Class Diagram for the C++ background software

verify whether all the necessary columns (declared in the model) are presented in the uploaded file. Once validated, the data is stored in the “modeldatasets” table, which has individual columns for every type of information (such as biogas production in mililiters, temperature, pressure, and others).

The “TaskTrainModel” class is responsible for reading the data set and model configurations to train a neural network. The back-propagation algorithm runs several times to achieve a low network error. As described in Section 3.5.2, the values are scaled to a new set of values between 0 and 1, and the output value is rescaled back from 0 to 1 to a compatible, human readable value. After performing the training job, the values for the connections in the neural network are saved into the database in a binary format.

The “TaskPerformPredictions” task is responsible for performing the predictions using the trained models saved in the database. The user configures the predictions in the front-end application by setting the input values and a time variable with a given range. The job generates an array of predictions saved as a string in the database, so it is straightforward to parse with Javascript.

The three tasks explained inherit the class Task, which is used as an interface to execute the method “Task::Run”. The class “TaskRunner” spawns a new thread for each task, enabling the tasks to run independently. The class also exposes the methods Start and Stop to control the execution of the task. The method Wait is called to keep the threads running without closing the program by calling the method “std::thread::join()”.

The “NeuralNetwork” class provides the basic methods to train and predict values. The parameter $hiddenNodes$ is calculated by the “TaskTrainModels” class previously introduced. The method “NeuralNetwork::Train()” returns a list of parameters θ_0 and θ_1 that can be saved in the database afterwards.

3.3.3 Cassandra Database

The Cassandra Database is a column-oriented NoSQL database focused on scalability and write performance. According to Lakshman and Malik (2009), the database was created at Facebook due to need of a database system that is both resilient and scalable in order to support the continuous growth of the platform. The project is mantained by the Apache Foundation with the help of DataStax, which provides open source drivers for C++, Node.js and other languages. The database offers a language called CQL (Cassandra Query Language) that is familiar to the users of SQL syntax. The database is written in Java, raising a lot of questions on how well the database performs, since Java is not compiled to machine code and includes a garbage collector which adds overhead to reclaim unused memory. However, the database has been used in big data clusters, as shown in the Cassandra’s homepage [19], suggesting that the database is able to perform well regarding performance and scalability.

Kuhlenkamp, Klems and Ross (2014) tested Cassandra and HBase in order to evaluate their scalability and performance. The authors conclude that both databases

[19]http://cassandra.apache.org

scale linearly as more nodes are added to the cluster. In general, the write performance of HBase is better than Cassandra's write performance, while Cassandra's read performance is better than HBase's read performance. The current work uses Cassandra due to the good write performance, which would be very important in a scenario where an array of sensors would stream data into the database. Also, the use of Cassandra offers an opportunity to test other databases compatible with the CQL language, such as ScyllaDB[20], which is a database compatible with Cassadra written in C++. The developers claim that ScyllaDB is up to 10 times faster than Cassandra. At the time of writing this thesis, there was little information to back up that claim.

3.3.3.1 Database Tables

Cassandra's NoSQL model allows for a very dynamic schema and may resemble the relational model, often found in SQL databases like MySQL and Oracle. However, the concept of foreign keys and referential integrity is not present in Cassandra, and if necessary, referential integrity must be implemented in the application rather than in the database. However, Cassandra is not meant for the same purposes as relational databases, so the comparison must be made carefully. Cassandra is designed for data analysis services and scalable data storage, whereas relational databases are useful for applications that need strict rules on how the data should be stored and validated.

The developer must be careful when designing the schema. The tables must be properly indexed in order to achieve good performance by defining primary and clustering keys. When the keys are properly defined, Cassandra can fetch the requested data in a particularly efficient way [21]. If the query defines a filter with a field that is not part of the table's key, an error is thrown, alerting the developer that the query would not perform in a predictable way, forcing the user to explicitly allow Cassandra to do

[20]http://www.scylladb.com
[21]https://cassandra.apache.org/doc/cql3/CQL-3.0.html

the filtering by that field. Figure 3.4 shows the database tables, and afterwards they are explained in detail. This explaination will also give more information on how the system actually works. Some fields are particularly important and will be explained in more detail.

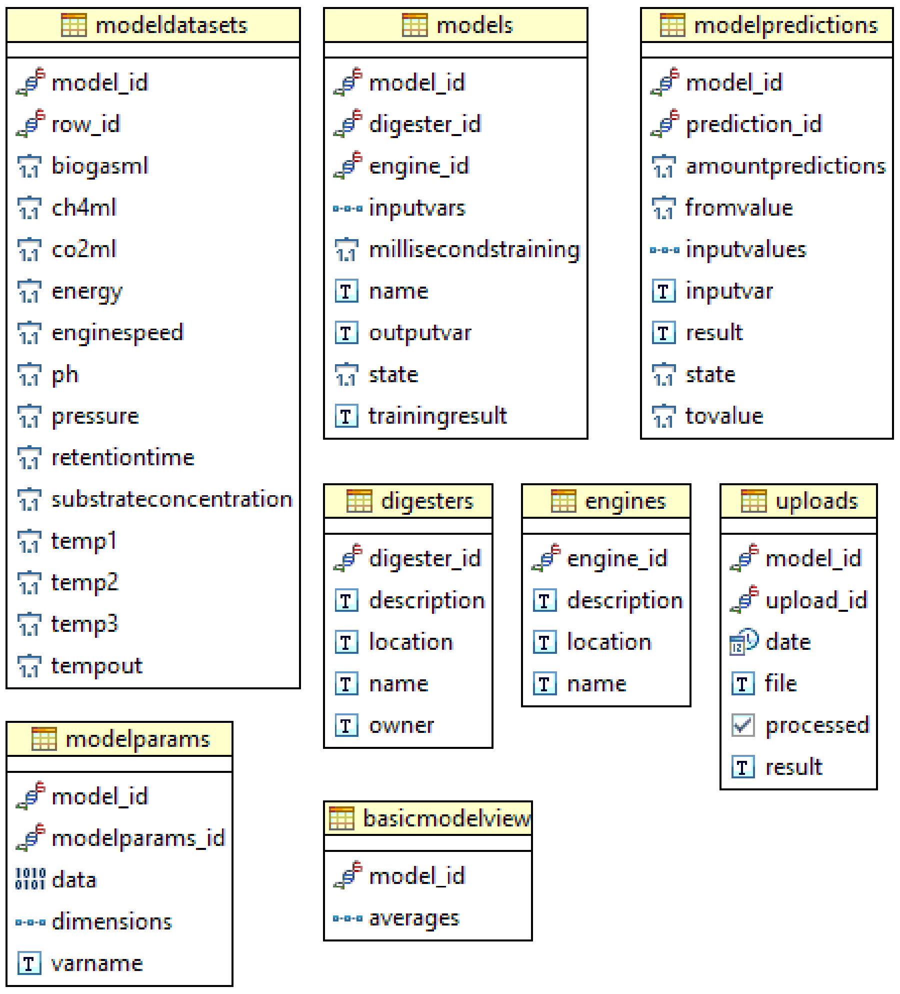


Figure 3.4: Cassandra Tables

As shown in Figure 3.4, the system has 8 tables. There are 2 tables called “engines” and “digesters” that allows the user to identify the engine/digester by its name, location and description. It is a simple record just to identify a particular engine or digester.

The “models” table, which will be described in more detail in Section 3.4.1, is a configuration of a given engine or digester, being particularly useful to differentiate digesters when they are operating under drastically different conditions. The user can configure the input and output variables of the model, and this configuration (together with the uploaded data) is used to train the model afterwards.

The field “inputvars” in the table “models” is a list of strings that represents the metadata of the input variables. For instance, the field could store values such as “temp1”,“temp2”,“temp3”, “pH”,“pressure” and so on, representing the information that will be used as input data for the neural network. In the case of a CSV file upload, these input variables (and the output variable) will be matched against the CSV header in order to check whether the uploaded file has all the CSV columns needed.

The field “outputvar” works the same way as the “inputvars”, but it is a single variable name, and not a list. It also needs to be present in the uploaded files (or any input data that the system could support in the future), therefore, it is also matched against the header of a CSV file.

The field “state” tells the background C++ application what should be done with the model. In case of the value 0, nothing should be done. When the value is 1, the model will actually be trained and the neural networks’ θ parameters representing the connections between the nodes will be saved into the table “modelparams”.

The table “modelparams” holds the matrices for the parameters of any model. This work implements a neural network, but it should work for any other algorithm that needs to save a vector or matrix of values in order to make the necessary predictions. The fields are very simple: “data” is a blob that stores matrices, “dimensions” is a list of integers representing the size of each dimension (number of rows and columns in a matrix, for example) and “varname” stores the names of the parameters.

In order to make predictions using the calculated model, the user inserts a new configuration of values in the table “modelpredictions”. The predictions are then saved into the field called “result”, which is a simple string field, allowing it to be easily parsed by any other language.

The table “basicmodelview” holds basic information about the model. It is currently updated after an upload is completed. The background application calculates the averages for the whole data set (in the table “modeldatasets”, indexed by the ID of the model), while the web application just shows that information. A possible improvement is to use web sockets and Cassandra’s streams (supported by the Node.js driver) to update the web page as soon as the model is updated, but this is not done in the current application.

3.3.3.2 Cassandra Limitations

DataStax provides documentation about the limits of CQL and Cassandra [22], as well as the Apache Cassandra’s page[23]. As documented in both pages, a single cell has a maximum size of 2GB. This does not affect the functionality of uploading files in the application because the files are being saved in smaller pieces of 5MB instead of saving the whole file in a single cell. Other limitations, such as 2 billion cells in a partition (a partition is a Cassandra node), maximum item size of 65535 for lists should not impact the application. There is also a limit of 65535 batch statements per session, but the application was prepared to execute a maximum of 200 batch statements per session.

[22]https://docs.datastax.com/en/cql/3.3/cql/cql_reference/refLimits.html
[23]https://wiki.apache.org/cassandra/CassandraLimitations

3.4 APPLICATION CONCEPTS

In the previous sections, some details of the application were explained. In this secion, specific concepts are going to be explained and clarified.

3.4.1 Models

ADs can digest many types of substrates, including vegetal and animal waste as well as rests of food and other organic substances. However, different types of substrates can possibly produce different amounts of biogas with different proportions of methane gas. If the same digester is used to digest two or more types of substances, collected data should be separated in the system to provide accurate predictions. If all data is used to train a single model and the data was collected during the digestion of two or more types of substances, the predictions might not be accurate, since the input variables could report drastically different outputs without changing its values. To solve this problem, the concept of “Models” was introduced in the application.

The concept should be applicable not only for different types of substrates, but also for any condition that could dramatically change the gas output. In that sense, a model is a temporary configuration of an AD, and the data collected represents the events that happened while that configuration was taking place. For instance, when the same digester is used to process two or more types of substrates at separate times, the user should create two or more models in the system, and each model should have the appropriate data to perform the model training. By creating models, any important change of configuration can be isolated from the others, and this model can be identified by its name or description. It is a simple method that solves the problem by letting the user isolate different situations as needed. A model consists of:

- A name and description to identify the model;

- The set of input variables that will be used to predict an output variable;
- An output variable that will be predicted based on the input variables;
- A data set that will be used to train the model, which must have all the input and output variables.

A model allows the user to upload data in the system. This data is added to the model data set after validating the file, where the file is checked whether it contains all the input and output variables. When the file gets processed, the system allows the user to perform two actions with the model: Training and performing predictions. “Training a model” means that the model will use its data set to train a neural network. The trained model is used afterwards to perform predictions.

3.4.2 Datasets

The application supports CSV file uploads to add data to a model data set. The uploaded file is read and validated in order to save it in the database. Validation consists of:

- Checking whether the header has same amount of columns than the rest of the lines;
- Checking whether the header columns have all the necessary columns as defined in the model;
- Checking whether all cells consists of valid numeric values (except for the header);

After the file is validated, the file is processed and the contents of the CSV file is copied into the database. The input file uploaded by the user should have all the variables selected in the model, otherwise the file gets rejected. If the file has more columns than necessary, any unnecessary columns are ignored, but the file is still processed. The current implementation matches the database columns with the CSV columns using a simple method described in Section 3.3.3.1. The meaning of the field names can be found in the Appendix 3.10.

Furthermore, the current architecture allows for further improvements, including implementations to read from several file formats (such as XML and JSON) without changing the neural network code.

When the data gets stored in the model data set, the training algorithms should be able to use it to perform the necessary tasks, such as training and performing predictions. The process of training a neural network is described in Section 3.5, while the next section explains what predictions are.

3.4.3 Predictions

When the model finishes training, the neural network is saved and is ready to use to perform predictions. A prediction must be configured by the user, and it consists of:

- A range variable. This variable is displayed as the horizontal axis in a chart;
- Minimum and Maximum values for the range variable;
- Number of predictions to be performed;
- Fixed values for the other variables in the model;

A prediction is configured based on a model. As explained before, the model's input variables are used to predict an output variable. Therefore, all the input variables must be present in the file header. An example of a prediction can be stated as follows:

The user wants to see the variation of biogas production in a given AD during a period of time. If the user has configured the model to use the AD temperature, substrate pH, pressure and retention time, all the variables must be set in order to perform a single prediction. To generate a range of predictions, a data set of these input variables is also generated. The latter contains a repetition of all the variables, except for the retention time, which varies linearly n times, where n is the number of predictions to be performed. Therefore, the size of the generated data set is also n. It is not necessary to choose the output value in the prediction configuration, it is already defined in the model.

The generated data set is passed to the neural network algorithm, and the predictions are saved into the database as a string value. Other applications can read the string and use it the way they need. In this work, the web front-end reads this value and generates a chart displaying the predictions.

3.4.4 Limitations

The current CSV file reader implementation expects the CSV columns to match the fields in the “modeldatasets” table. Further improvements should find other methods to match these columns, for instance, creating an intermediate representation of these columns, called aliases. Aliases would allow the user to create CSV files with column names not limited to the fields in the “modeldatasets” table, where a single field could have several supported names. For instance, the field “biogasml” could also be called “milliliters_biogas”, “mlbiogas”, “Milliliters of Biogas” and others. The same process should be applied on further implementations of other file readers.

Also, the application assumes no particular unit for its columns. For instance, the field “retentiontime” should be used as time spans (e.g., 1 second, 1 minute, 1 day) and not dates and times, but the system currently has no knowledge on how to consider these units and make appropriate conversions of units of temperature, time, volume and pressure. Even the column “biogasml”, which specifies the unit of volume in milliliters, has no validation on its value, and the name merely provides a hint to the user on what is the purpose of the column. This is also the reason why the front-end does not specify the units of the variables, i.e. whether the temperatures are in celsius, kelvin or fahrenheit, or whether the retention time is in seconds, minutes or any other period of time. The user is expected to know its data and what the values mean, and as long as the data is consistent, the user should not experience any problems related to incorrect units.

Another limitation is related to missing values in the CSV file: the system expects the file to be complete, without missing cells. This is a problem that the system does not solve, so the user should verify the file is correct and complete. However, future works should find ways to deal with this problem, especially if sensor data streams are implemented. The application will have to handle sensor failures and misreadings, while it stores the data in the “modeldatasets” table.

3.5 NEURAL NETWORK

As shown in the sections 2.1 and 3.2, there are use-cases of neural networks in the field of AD both in scientific researches and commercial uses. These cases (Sota Solutions (2016), Kusiak and Wei (2014), Qdais, Bani-Hani and Shatnawi (2010), Oliveira-Esquerre, Mori and Bruns (2002)) suggests neural network is a good ML algorithm to be implemented in this work.

There are libraries for neural networks implemented in C/C++ like FANN [24], which could be used in this work to provide neural network functionality. WEKA also provides a neural network algorithm called $MultilayerPerceptron$, implemented in Java. However, this work aims to explore the technical side of developing such algorithms, and study how they work and how they are implemented. As a project decision, it was

[24]http://leenissen.dk/fann/wp/

decided to use C++ to develop a neural network without using specialized libraries. Future works can improve this work's implementation, including features such as parallelism, online learning/predictions and other types of neural networks.

The neural network was developed in C++ using the FLENS [25] library for linear algebra. The reason for using this library is its ease of use. The library provides a Matlab-like syntax for matrix operations, and provides ways to link the program against BLAS libraries, like Openblas [26], which is also used in the neural network in order to improve calculation times for matrix operations. The following code shows a code snippet of FLENS.

```
typedef GeMatrix<FullStorage<double, ColMajor>> Mtx;
Mtx A(1,2) = 2, 3;
Mtx B(2,1) = 4, 5;
Mtx result = A * B;
std::cout << result << std::endl;
```

The neural network is composed of layers, nodes and connections. In this work, the input layer has as many nodes as there are variables in the input data, plus a bias unit. This bias unit, or bias node has a fixed value of 1. If the input data is, for example, a 20x20 pixels image, the input layer has 1 bias node and 400 nodes, one node for each pixel. If the input data is a CSV file, each column will result in a node in the input layer. All layers, except the output layer, has a bias unit.

As shown in Figure 3.5, the hidden layer has N nodes, and each node is connected to all nodes in the previous layer (the input layer). If the hidden layer has 10 nodes, and the input layer has 3 nodes, each one of the 10 nodes in the hidden layer will be connected to the 3 nodes in the input layer. The application implements the connections from a layer $N-1$ to N as a single matrix θ_N. This matrix has size $11X3$,

[25]http://www.mathematik.uni-ulm.de/ lehn/FLENS/
[26]http://www.openblas.net

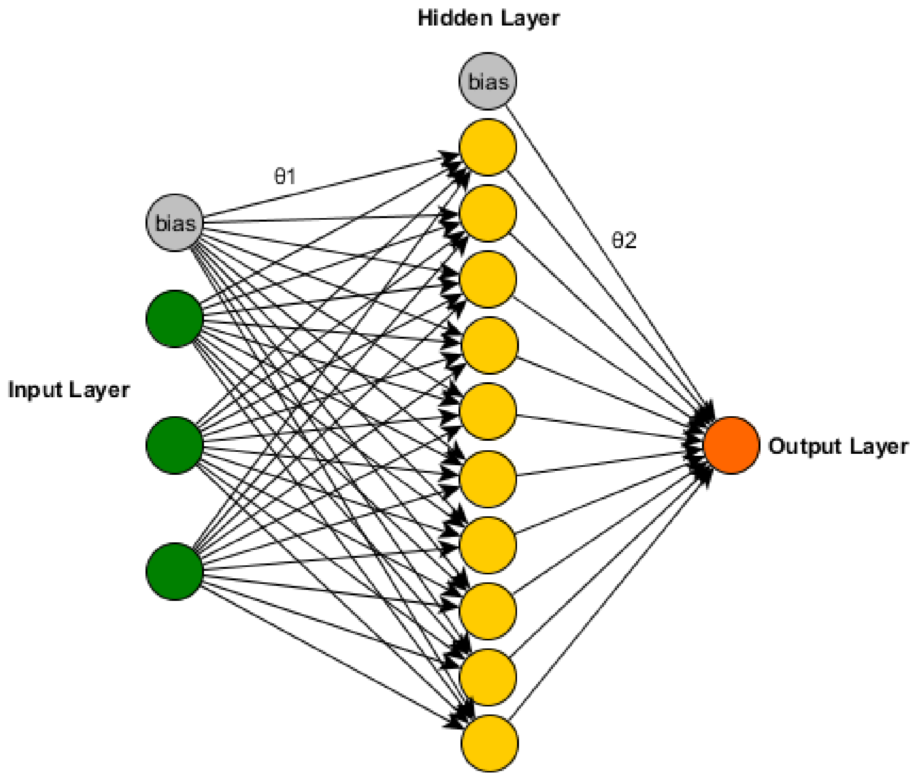


Figure 3.5: Neural network representation

resulting in a total of 33 values. It is important to notice that the number 11 comes from the extra bias node added in the input layer. The output layer works in the same way: all the nodes in the hidden layer are connected to all the nodes in the output layer, except that the output layer has no bias unit.

In order to perform predictions based on the current values of θ_0 and θ_1, a feed-forward algorithm was developed. The connections between the nodes are the weights of the neural network. In an analogy with biological brains, the weights represent a connection between neurons and how strong the connection is. Each node implements a weighted sum of the node values and the weights. After performing the summation, the value is used in the sigmoid function. The result of the sigmoid function becomes the node value.

Algorithm 1 Node activation algorithm

```
1: procedure ACTIVATION($nodes$,$weights$)
2:     for $nodeIndex \leftarrow 1\ \ to\ \ length(nodes)$ do
3:         $sum \leftarrow sum + (nodes(nodeIndex) * w(nodeIndex))$
4:     end for
5:     return $sigmoid(sum)$
6: end procedure
```

It is important to notice that this algorithm is implemented with matrix multiplication. The layer calculation is not performed explicitly for each node in the layer, but for all the nodes in the layer in a single matrix multiplication. After the multiplication, the sigmoid function is used:

$$Sigmoid(x) = 1/(1 + e^{-x}) \tag{3.1}$$

The actual feed-forward algorithm is fully vectorized and can perform many predictions at once. It is important to notice that the $addBias$ function just prepends a column of 1's to the matrix.

Algorithm 2 Feed Forward Algorithm

```
1: procedure FEEDFORWARD($\theta_1$, $\theta_2$, $input$)
2:     $layer1 \leftarrow addBias(input)$
3:     $layer2 \leftarrow addBias(sigmoid(layer1 * \theta_1^T))$
4:     $output \leftarrow sigmoid(layer2 * \theta_2^T)$
5:     return $output$
6: end procedure
```

3.5.1 Training the Neural Network

The process of training the neural network involves the use of the Backpropagation Algorithm. It is an algorithm that needs to calculate the gradient of the neural network using the derivative of the sigmoid function, defined as:

$$Sigmoid'(x) = x * (1 - x) \tag{3.2}$$

The algorithm tries to find values for θ_1 and θ_2 that produces a neural network with the lowest possible error rate. The derivative of the sigmoid function plays an important role at the calculation, helping the algorithm to discover how to change the θ parameters correctly. Moreover, the back-propagation works by “going backwards” through the layers (hence the name), calculating how much a given node affects the error of the network.

Algorithm 3 Backpropagation Algorithm

```
1: procedure BACKPROPAGATION($\theta_1, \theta_2, \lambda, layer1, actualData$)
2:     $bpLayer1 \leftarrow sigmoid((\theta_1 * layer1^T)^T)$
3:     $bpLayer1Bias \leftarrow addBias(bpLayer1)$
4:     $bpLayer2 \leftarrow sigmoid((\theta_2 * bpLayer1Bias^T)^T)$
5:     $outputError \leftarrow bpLayer2 - actualData$
6:     $d2 \leftarrow (\theta_2^T * outputError^T)^T$
7:     $d2sigmoid \leftarrow addBias(sigmoid'(bpLayer1))$
8:     $d2(i,j) \leftarrow d2(i,j) * d2sigmoid(i,j)$
9:     $G\theta_1 \leftarrow G\theta_1 + (d2^T * layer1)$
10:    $G\theta_2 \leftarrow G\theta_2 + (outputError^T * bpLayer1Bias)$
11:    $G\theta_1 \leftarrow (1/m) * G\theta_1 + (\lambda/m) * \theta_1$
12:    $G\theta_2 \leftarrow (1/m) * G\theta_2 + (\lambda/m) * \theta_2$
13:    return $G\theta_1, \theta_2$
14: end procedure
```

The algorithm uses the current weights and a λ term to perform the training. The λ term is called a regularization term. It helps the training process by reducing or increasing the overall training cost and reducing the importance of some input variables that could be just noise in the data, and also affects the gradient values for the network. This value should be tested multiple times, but if the λ value is too high, there is a chance that the neural network would perform poorly, because it would consider that everything is noise. The current implementation of the neural network uses a constant value of 0.1.

Finally, a Matlab/Octave function called $fmincg$ was ported to C++ [27]. The function minimizes the neural network's cost by taking the current cost and the gradient for the neural network and provides a new set of θ_1 and θ_2 values. The new θ_1 and θ_2 becomes the current parameters for the neural network, which are used again to calculate the cost of the neural network using back-propagation. By repeatedly providing new θ values and recalculating the cost, $fmincg$ keeps improving on the current solution. If the changes made to the θ values actually made the neural network perform worse, the algorithm discards the result and try again with new parameters, and might even fail due to not being able to find a better solution. Although there are other algorithms such as SGD (Stochastic Gradient Descent), $fmincg$ is used because it does not require extra parameters such as learning rate or momentum.

3.5.2 Testing

In order to test the neural network, two problems were created: Test #1: a simple test with two input variables and 350 records, and Test #2: a test with 5,000 20x20 images of handwritten digits from the MNIST data set[28], used in various examples of machine learning, including the example explained and implemented by Nielsen (2016), resulting in 400 input variables. Nielsen (2016) explains that neural networks are capable of computing any function, and act as universal approximators. By "approximation", it means that the neural network should not be expected to compute a function with 100% accuracy. The tests are being executed in a VirtualBox guest virtual machine running Ubuntu 15.10, 3.5GB of RAM, Windows 10 host operating system and a Core i7-3610QM 2.3GHz processor.

The data set of the test #1 includes the two input variables and an output variable designed to have a simple linear relation to the input variables. The output is

[27]http://www.mathworks.com/matlabcentral/fileexchange/42770-logistic-regression-with-regularization-used-to-classify-hand-written-digits/content/Logistic%20Regression%20with%20regularisation/fmincg.m

[28]The data set is used on the Machine Learning online classes on Coursera, by Andrew Ng. The MNIST website provides a new data set with larger images.

calculated by $(var1/var2) * 1000$. The values of $var1$ varies between 25 and 55, and the values of $var2$ varies between 15 and 25. This test is designed to verify that the network is capable of making simple predictions based on linear data.

Therefore, the first neural network was developed using 2 input nodes, 1 hidden node and 1 output node. The network does not behave well when raw data is fed directly into the input layer. After performing feature scaling, in which the range of the input data was changed to an interval between 0 and 1, the network output behaved as expected, modeling the linearity of the data. After training the neural network, a new test data set was created, containing 100 records. This test resulted in a correlation between the predictions and the data set of 0.98. The correlation is calculated according to the formula shown in the section 1.6. The predictions are not perfect, as expected and explained by Nielsen (2016), but are approximately correct. This experiment shows that the neural network is able to perform simple prediction. Performance-wise, the training takes around 40 milliseconds to conclude. Figure 3.6 shows a chart representing the predictions and the actual data.

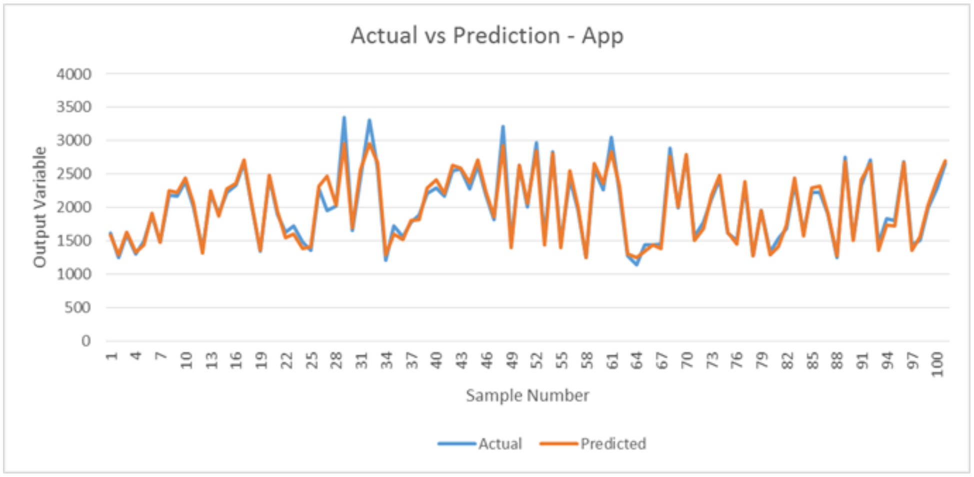


Figure 3.6: Simple Prediction

The same neural network algorithm was used to perform pattern detection on images of handwritten digits using the MNIST data set. The network was configured to have 25 hidden nodes and 10 output nodes. Each output node represents a digit from 0 to 9. The index of the node with the higher output tells what digit is written in the image. Each input record has 400 features, each one representing a pixel on a 20x20 image. Each pixel has a value between -0.1 and 1.2 (representing a grayscale value), which are suitable for the activation function. Therefore, no feature scaling was performed.

The network behaved well, predicting correctly around 4920 out of 5000 images on the same training data set. The results of the neural network are slightly different each time the back-propagation algorithm runs, since the nodes in the neural network are initialized randomly at every run. The results show the same algorithm solving two different problems: a simple linear relation and a handwritten digit classification problem. Therefore, based on these results, the neural network is a good algorithm to be used in the application in order to provide prediction services. The algorithm is precise enough and also has a feature of being very general, solving very different problems using the same code.

For better evaluation of the predictive capabilities, this work recommends that future works should use a new data set of handwritten digits to verify whether the network can predict digits based on images not used during the training process.

3.5.3 Performance comparison

The original algorithm was developed in Octave[29], an interpreted high level language intended for numerical computations. The algorithm was ported to C++ afterwards using the FLENS library. Both implementations achieve similar accuracy rates, but it is interesting to evaluate the runtime of several implementations.

Therefore, the performance of the algorithm was evaluated. Both algorithms (Octave and C++) were tested in the same computer, described in Section 3.5.2. After setting up the test and running both implementations 5 times (C++ and Octave), the

[29]https://www.gnu.org/software/octave/

completion times were measured and compared. Octave takes an average of 12.8 seconds to complete, while the implementation in C++ takes an average of 10.22 seconds. Both implementations achieve similar accuracy rates, since the C++ implementation is a port of the Octave version. Figure 3.7 provide more detail on the completion times over 5 runs.

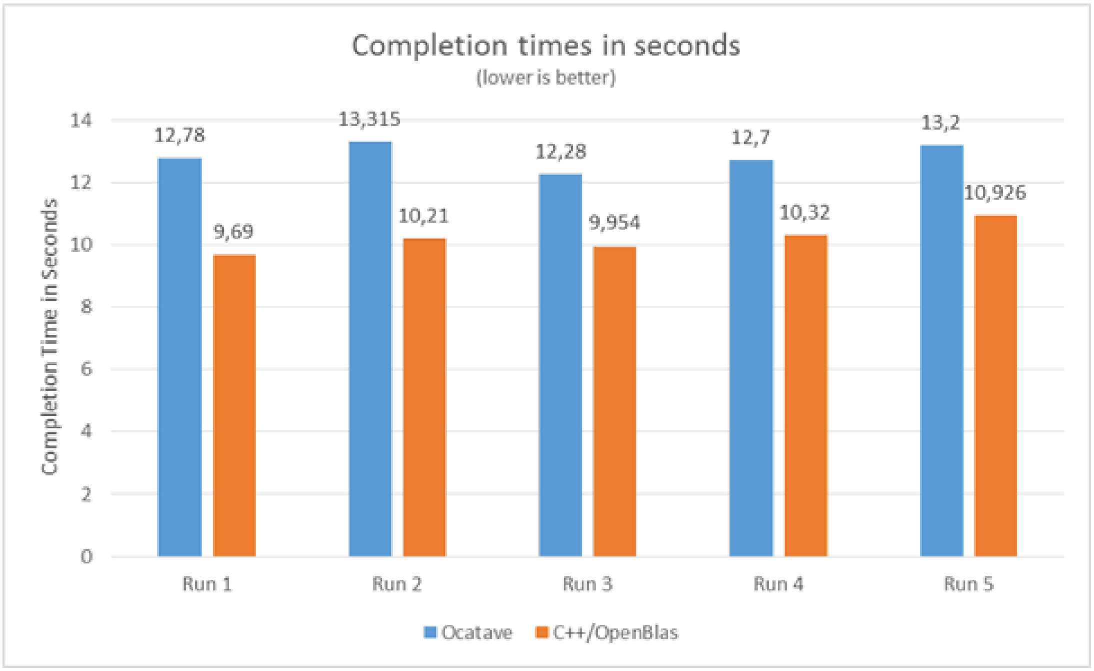


Figure 3.7: Completion times of C++ vs Octave Implementations

It is possible to conclude that the C++ implementation is faster than the Octave implementation, but some remarks should be added. Both implementations are using Openblas in order to speedup calculations. The C++ implementation is performing 2 seconds faster considering the averages, but the biggest performance factor is the use of the Openblas library. Octave spends most of its time performing matrix operations, which are handled by Openblas. The same applies to the C++ implementation. In order to measure the performance impact of Openblas, the library was removed from the C++ implementation temporarily. In this case, FLENS uses a fallback implementation that is less performant.

The results of this test were not added to the completion times comparison in

Figure 3.7 because it takes up to 4 minutes to run, much slower than the 12.8 seconds of Octave. This shows that the use of a library for fast matrix operations is very useful in order to achieve better performance for machine learning algorithms.

3.5.4 WEKA Data Mining

WEKA[30](Waikato Environment for Knowledge Analysis) is an open source project maintained by the Waikato University in New Zealand. The project aims to develop and provide a general machine learning tool, while using it to solve practical problems of the New Zealand industry.

WEKA allows to use existing data in order to train and execute ML algorithms. One of those algorithms is called MultilayerPerceptron. The same simple linear test executed before (Section 3.5.2) was also executed with WEKA in order to compare the system's C++ algorithm against WEKA version 3.8. WEKA's algorithm was set up to have only one hidden node, and the rest of parameters were configured as their default values. As expected, WEKA performs better than the system's implementation, with a correlation of 0.99 in the simple test, taking around 20 milliseconds to train. As stated before, the system's implementation of the neural network achieves a correlation of 0.98. Figure 3.8 shows the relation between the data and the predictions generated in WEKA.

There are some differences between WEKA's algorithm and the system implementation. WEKA uses a different strategy in order to make predictions, instead of using the sigmoid activation function, it uses a simpler linear function[31] when the output values are numeric. However, WEKA asks for several other parameters that this work's current implementation does not. For instance, WEKA asks for "learning rate", "momentum", "normalize attributes" and many others that can be seen on Figure 3.9.

[30]http://www.cs.waikato.ac.nz/ml/weka/index.html
[31]http://grepcode.com/file/repo1.maven.org/maven2/nz.ac.waikato.cms.weka/weka-stable/3.6.7/weka/classifiers/functions/neural/LinearUnit.java

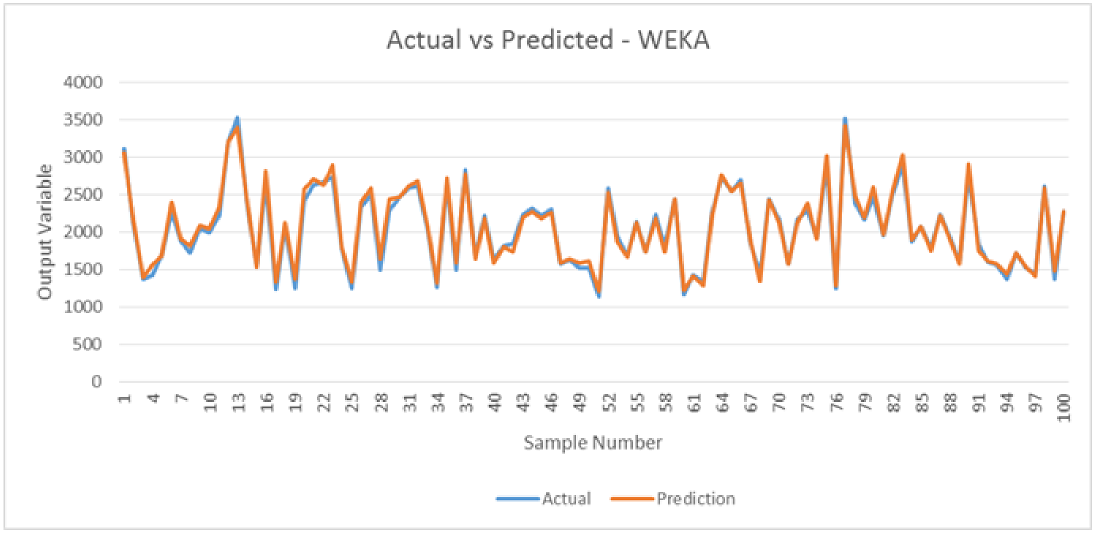
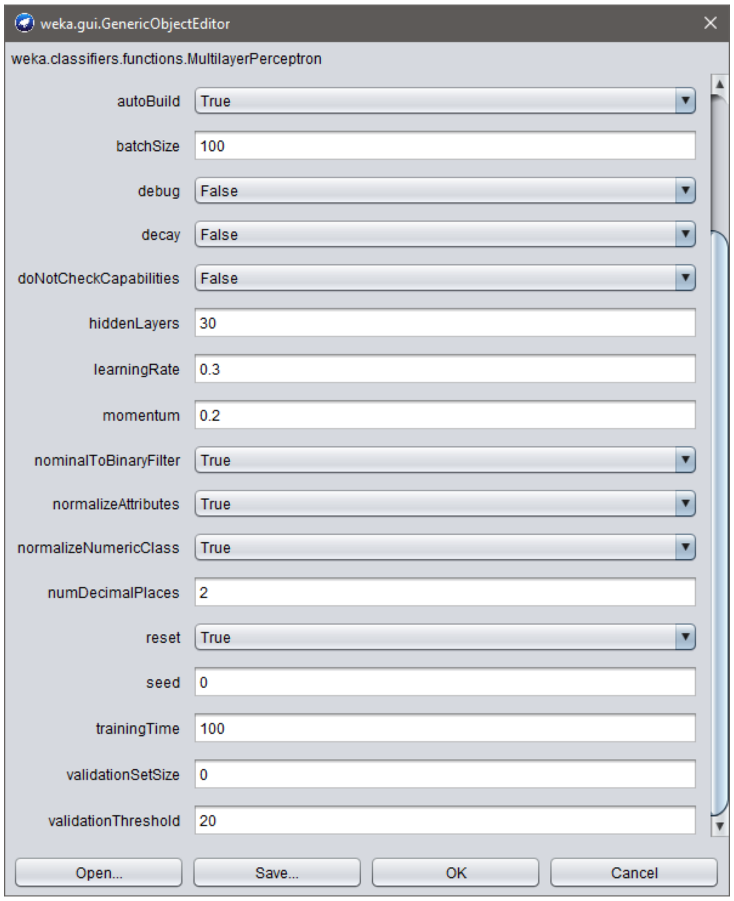


Figure 3.8: Weka Prediction

The amount of parameters shows the flexibility of WEKA. It can be fine-tuned to better fit the user's needs. The current implementation of the system does not provide tunable parameters for neural networks. It is a tradeoff between control and ease of use: while the system is not as fine-tunable as WEKA, it should result in a system that is easier to use.

This simple test is not enough to compare the system implementation and WEKA's implementation due to the simplicity. This work also tested the prediction of handwritten digits in WEKA. In order to test WEKA's raw performance, the same test was performed using Multilayer Perceptron. The only settings changed were "Batch Size" to 5000 (which is the total size of the data set) and number of hidden nodes to 25. Figure 3.10 shows the output of Weka.

In this case, the work could not verify if the settings on Weka were set up correctly, since Weka reports a correlation efficient of 0.833 (the C++ implementation varies around 0.98). This is still a strong correlation, suggesting that the neural network is working correctly. Regardless of the correlation efficient, the training process takes up to 75 seconds. WEKA implements a very different code architecture in Java,



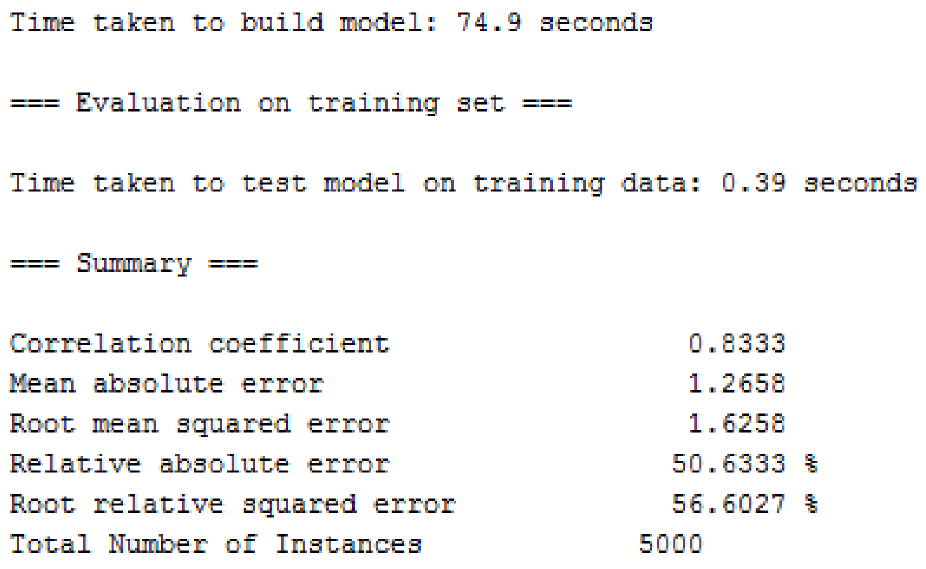
weka.classifiers.functions.MultilayerPerceptron

autoBuild	True
batchSize	100
debug	False
decay	False
doNotCheckCapabilities	False
hiddenLayers	30
learningRate	0.3
momentum	0.2
nominalToBinaryFilter	True
normalizeAttributes	True
normalizeNumericClass	True
numDecimalPlaces	2
reset	True
seed	0
trainingTime	100
validationSetSize	0
validationThreshold	20

Figure 3.9: Weka Parameters for Multilayer Perceptron

if compared to this work's algorithm, which can be verified in its source code [32]. The system's implementation relies on matrix operations which are calculated using Openblas, while WEKA uses simple math operations with arrays and object-oriented code in order to perform the necessary calculations. Moreover, the differences of correlation between this work's implementation and WEKA are due to WEKA using a linear activation function instead of the sigmoid function. However, changes in configurations and in the data set could make WEKA use the sigmoid function as well, probably causing WEKA to perform better in this test.

[32]http://grepcode.com/file/repo1.maven.org/maven2/nz.ac.waikato.cms.weka/weka-stable/3.6.7/weka/classifiers/functions/neural/LinearUnit.java#LinearUnit



```
Time taken to build model: 74.9 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.39 seconds

=== Summary ===

Correlation coefficient                  0.8333
Mean absolute error                      1.2658
Root mean squared error                  1.6258
Relative absolute error                 50.6333 %
Root relative squared error             56.6027 %
Total Number of Instances             5000
```

Figure 3.10: Weka Output of Multilayer Perceptron

3.5.5 Background Service Limitations

The final implementation of the neural network is located at the background service application. The implementation has some limitations on performance that should be solved in future works. In this secion, the limitations will be discussed and alternatives will be proposed.

The first problem is related to memory usage. The back-propagation algorithm is run 100 times using the function “fmincg” to minimize the neural network error. In order to speed up calculation times, the training data set is kept in memory during the process, which might become a problem with larger data sets. Futhermore, the algorithm has to allocate more memory to perform the necessary calculation steps. Therefore, the algorithm might use large amounts of memory depending on the amount of data stored in the databse. One alternative is to stream data from Cassandra and perform the training using smaller slices of the data set, instead of performing the training process using the whole data set at once. However, it introduces another problem: the algorithm currently uses the same data set repeatedly, so it would be necessary to re-stream the data repeatedly from the database, resulting in increased network I/O. This work decided to keep the data in memory during the training process, but future works might evaluate other alternatives to solve this problem.

This work also decided to use a fixed method to calculate the number of hidden nodes. The handwritten digit recognition problem explained in Section 3.5.2 uses 400 input nodes and 25 hidden nodes, as this is the number of hidden nodes in the original Octave implementation. The following function calculates the number of hidden nodes:

$$h(i) = \lfloor \sqrt{i} * 2 \rfloor \tag{3.3}$$

where i is the number of input nodes.

When using 400 input nodes, the function returns 40 hidden nodes, almost twice the previous number of 25. For a problem with 100 input nodes, the function calculates 20 hidden nodes, and when having 15 variables, the number returned from the function is 7. Future improvements could implement a configuration in the system that chooses the function to use in order to determine the number of nodes, but it would still be abstracted from the user interface.

Another fixed setting is the number of iterations for the back-propagation algorithm, which is 100. This is a number that its expected to bring the values of the network to a low value. The handwritten recognition problem also used this setting, and since the results were good enough (more than 90% accuracy), this work decided to use this number of iterations.

3.6 THE PROPHET APPLICATION: AN OVERVIEW OF THE SAAS

“Prophet” is the name of the application. The background C++ application is called “Prophet Service”, while the SaaS application is called “Prophet Web”. In this section, Prophet will finally be shown and explained. A simple activity diagram was developed in order to understand the system.

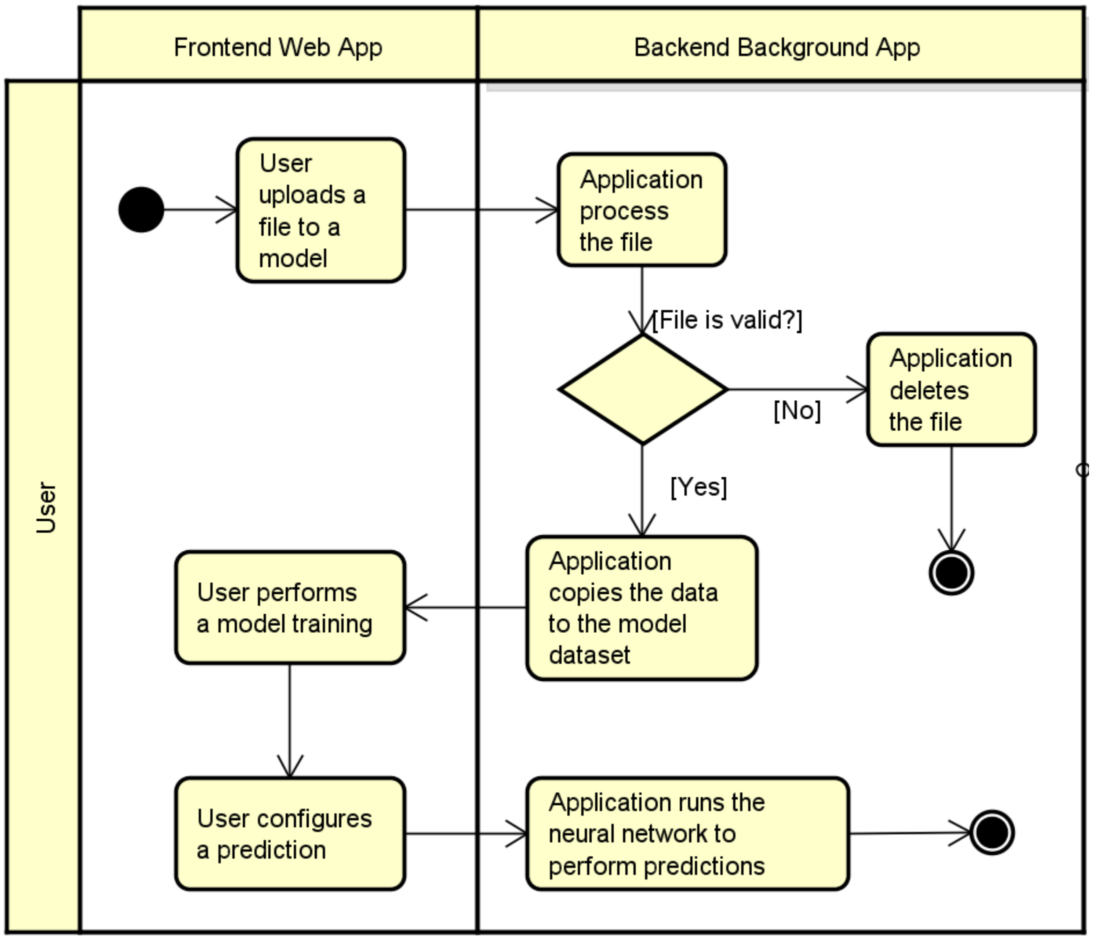
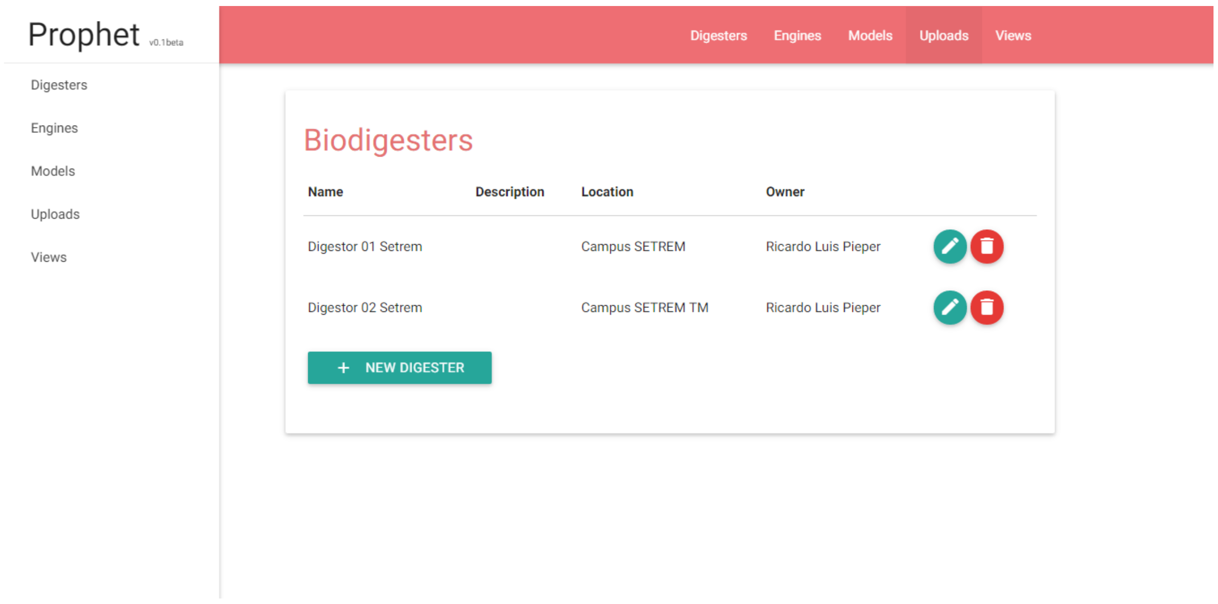

Frontend Web App
Backend Background App
User
User uploads a file to a model
Application process the file
[File is valid?]
[No]
Application deletes the file
[Yes]
Application copies the data to the model dataset
User performs a model training
User configures a prediction
Application runs the neural network to perform predictions

Figure 3.11: Activity Diagram

Figure 3.11 shows how the system should be used. The user needs to upload data into the system. When the upload is done, the background application will verify the file (this process has been already described in Section 3.4.2) and check for errors. If the file contains one or more errors, the process is interrupted and the file is deleted. Otherwise, the user should be able to perform a model training and make predictions.

3.6.1 Layout Overview

The system layout consists of a navigation menu at the left side, containing 5 links that lead the user to the listings of digesters, engines, models, uploads and views. The same links are also located at the top navigation bar, which is hidden when using the system in a mobile device. The layout is responsive, the same code is used for both desktop and mobile browsers. This functionality is provided by Materialize, as explained in Section 3.3.1. Figure 3.12 shows the overall look of the system.



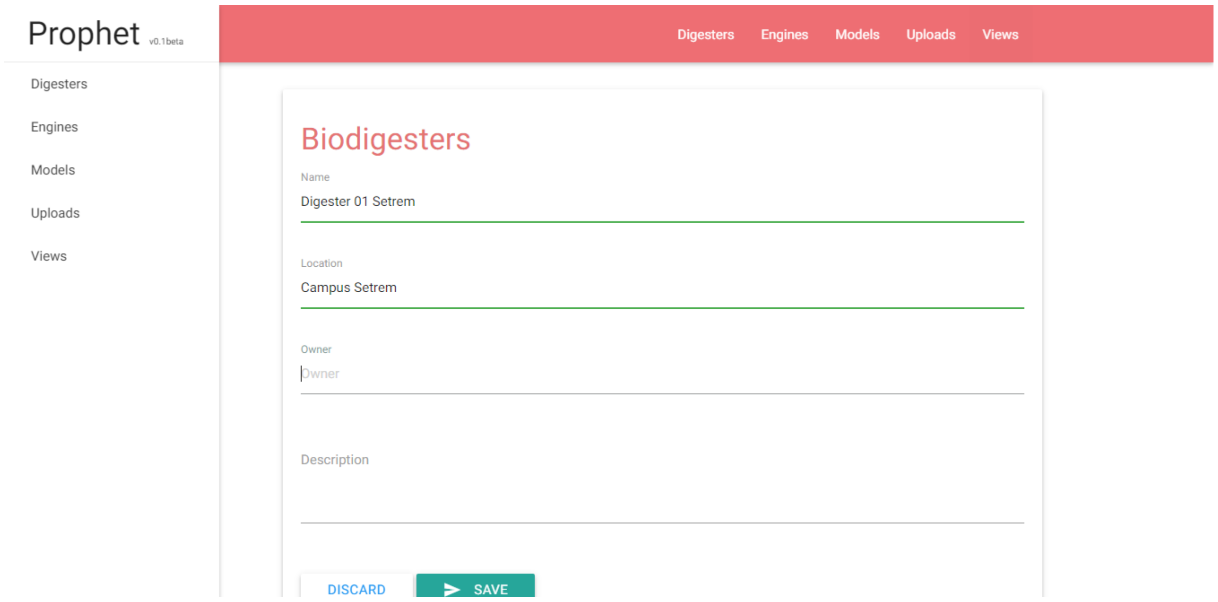
Name	Description	Location	Owner
Digestor 01 Setrem		Campus SETREM	Ricardo Luis Pieper
Digestor 02 Setrem		Campus SETREM TM	Ricardo Luis Pieper

Figure 3.12: List of Digesters

Another important aspect of the system is that it is a Single Page Application (SPA). It means there are no page reloads when navigating through the system: the page is loaded only once, and Angular.js manages the process of rendering and changing pages. Therefore, most of the Prophet Web code is located at the client, while the server only responds to data requests. For instance: when the system needs to display a list of digesters, the Node.js server does not generate HTML to produce it. Instead, it just provides the data (using JSON) so that the client code (using Angular) can render it.

3.6.2 Digesters and Engines

In order to identify a digester or engine in the system, the user can perform CRUD (Create Read Update Delete) operations with digesters and engines. Figure 3.12 shows these concepts in action. Each line in the grid has a green “Edit” button and a red “Delete” button, and at the bottom of the grid, a button “New Digester” is presented. Each different listing screen changes this button description to match the title. When clicked, the “New Digester” button leads to a new page responsible for creating and editing the digester, shown in Figure 3.13.



Biodigesters

Name
Digester 01 Setrem

Location
Campus Setrem

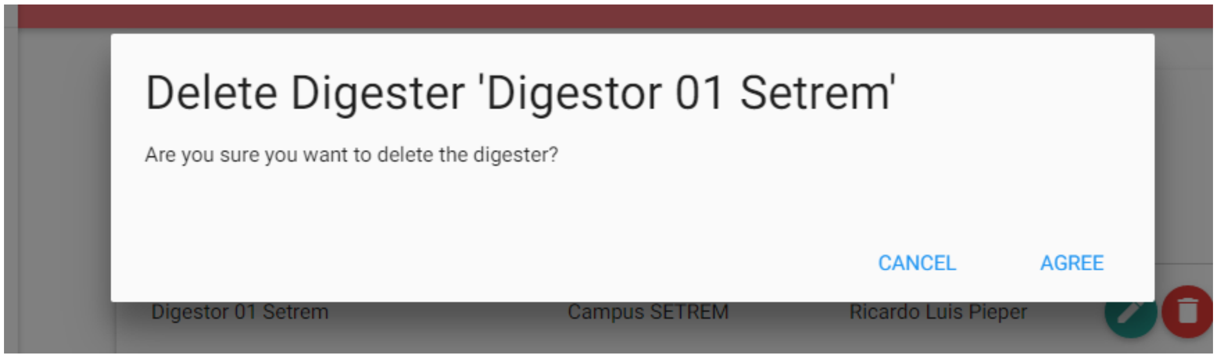
Owner
Owner

Description

Figure 3.13: New Digesters

There are two buttons at the bottom of the page: a “Discard” button and a “Save” button. When creating a new item, the “Save” button inserts a new item in the database, and when editing an item, the button updates this item in the database. The button “Discard” just returns to the listing page without performing any actions.

Deleting an item requires confirmation, as shown in Figure 3.14. This is done in order to prevent accidental clicks from performing destructive actions. The user should then agree to delete the item.

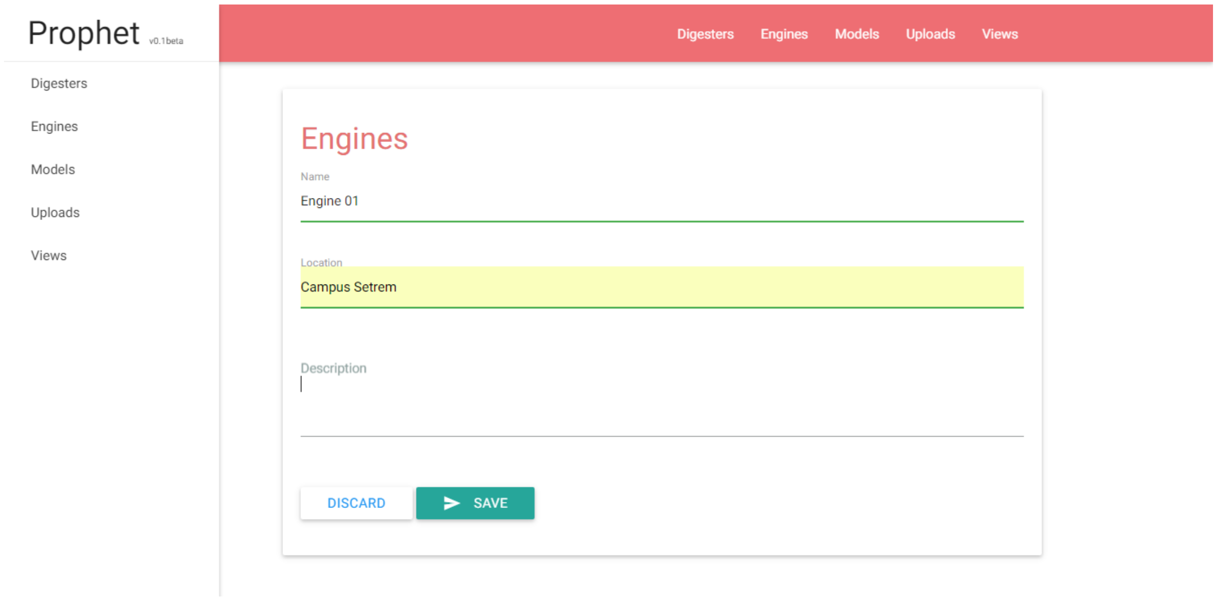


Delete Digester 'Digestor 01 Setrem'

Are you sure you want to delete the digester?

Figure 3.14: Delete Digester

The Engines section works exactly the same way as the Digesters. Figure 3.15 shows the creation of an engine, and the engines listing page is very similar to the digesters listing.



Engines

Name: Engine 01

Location: Campus Setrem

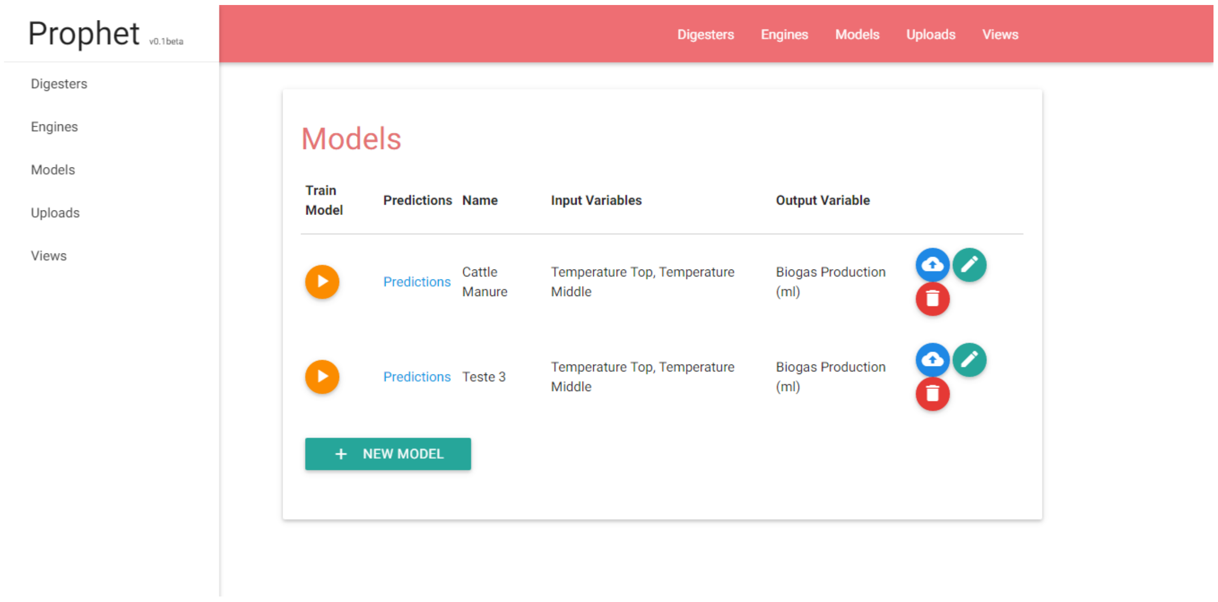
Description:

Figure 3.15: New Engine

3.6.3 Models

Prophet centralizes its features in the Models page. Although the CRUD process is still the same, the listing of models provides more options to the user. Each line in the grid includes the usual delete and edit buttons, and includes three more items: Upload files, Train Model and go to the Predictions page by clicking on the link. Figure

3.16 shows the listing of models.

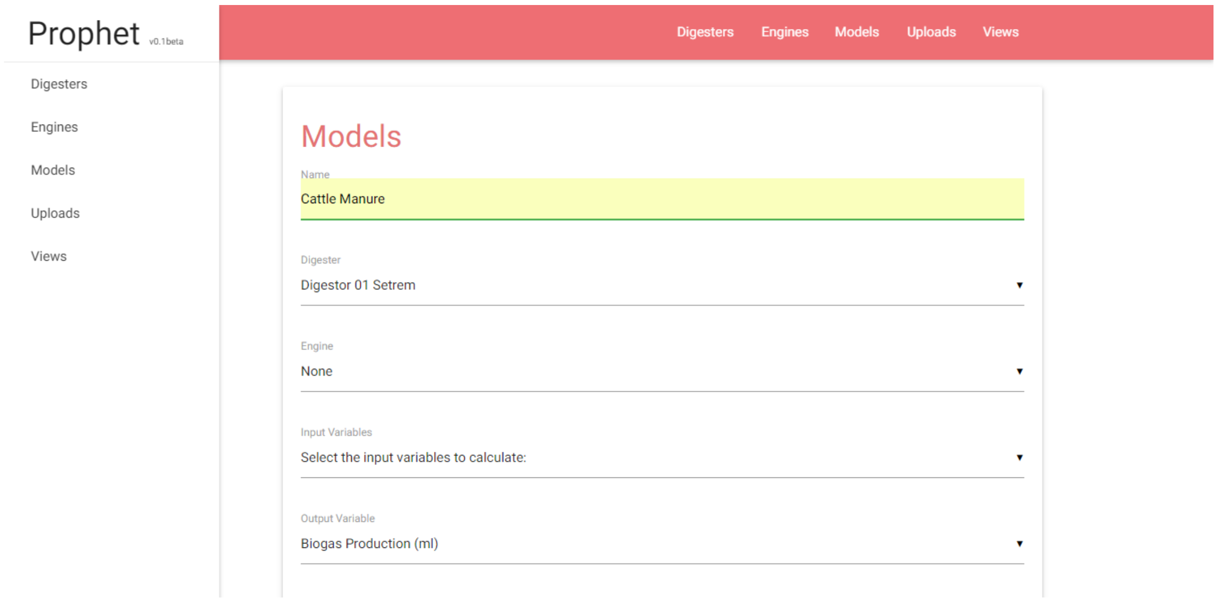


Models

Train Model	Predictions	Name	Input Variables	Output Variable
	Predictions	Cattle Manure	Temperature Top, Temperature Middle	Biogas Production (ml)
	Predictions	Teste 3	Temperature Top, Temperature Middle	Biogas Production (ml)

Figure 3.16: List of Models

As described in Section 3.4.1, the model needs a name, description and its variables. Figure 3.17 shows the page responsible for creating and editing models. The user should choose a name for the model (e.g. Cattle Manure, Mixed Substances, Pig Manure), an optional description, and choose the digester or engine it refers to.



Models

Name: Cattle Manure

Digester: Digestor 01 Setrem

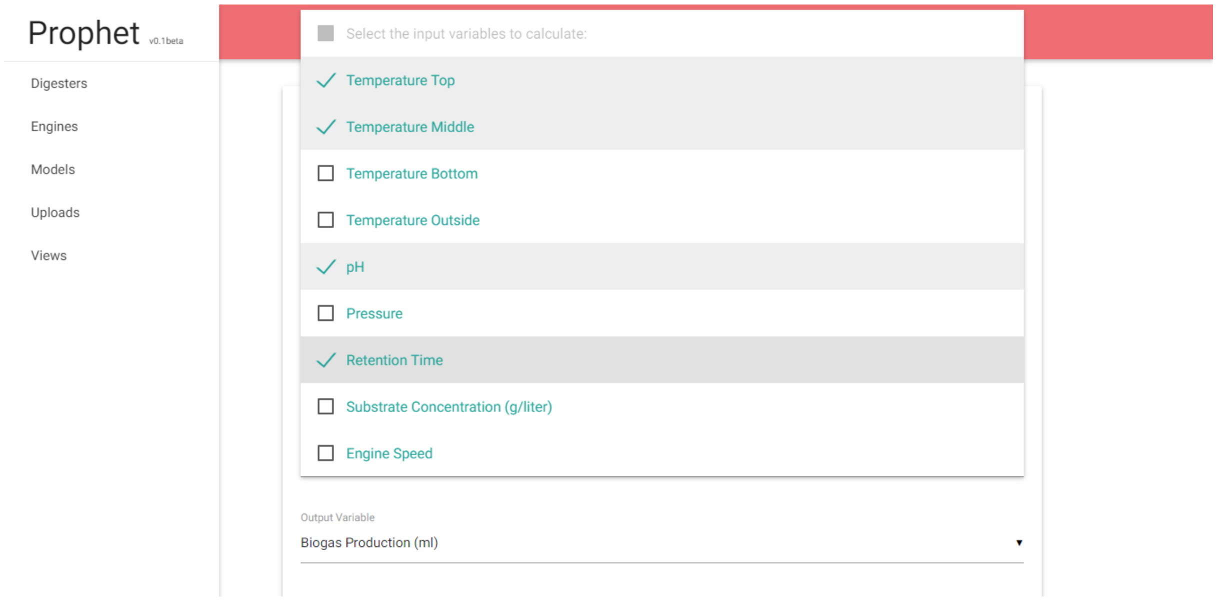
Engine: None

Input Variables: Select the input variables to calculate:

Output Variable: Biogas Production (ml)

Figure 3.17: New Model

To select the model variables, the interface allows the users to choose the necessary variables by clicking in the dropdown element. When clicking the dropdown, the page renders a list where the user can select the variables, as shown in Figure 3.18. The user should choose the input variables and a single output variable.



Select the input variables to calculate:
✓ Temperature Top
✓ Temperature Middle
☐ Temperature Bottom
☐ Temperature Outside
✓ pH
☐ Pressure
✓ Retention Time
☐ Substrate Concentration (g/liter)
☐ Engine Speed

Output Variable
Biogas Production (ml)

Figure 3.18: Selecting Variables

After inserting a new model, the user can upload a CSV file. To upload a file, the user should click the blue button in the model list. A modal will be displayed, and the user can choose a file in the computer to upload. Figure 3.19 shows the upload file modal.

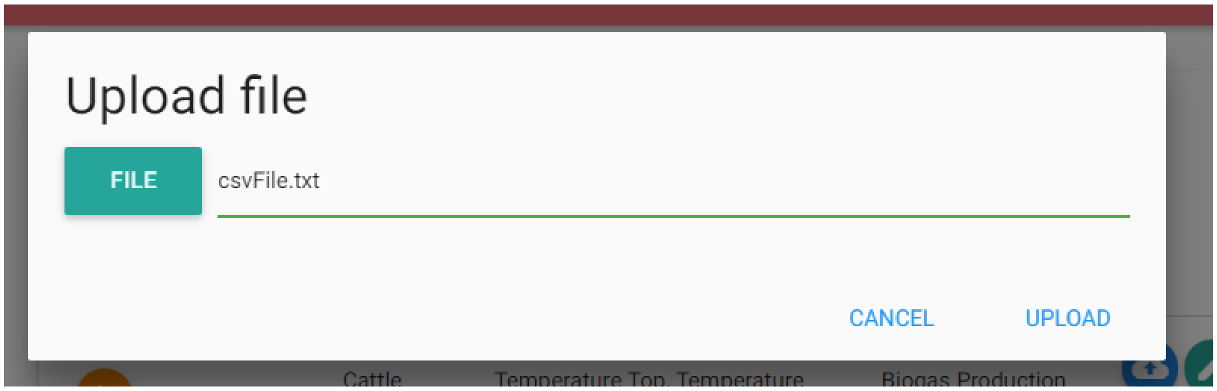
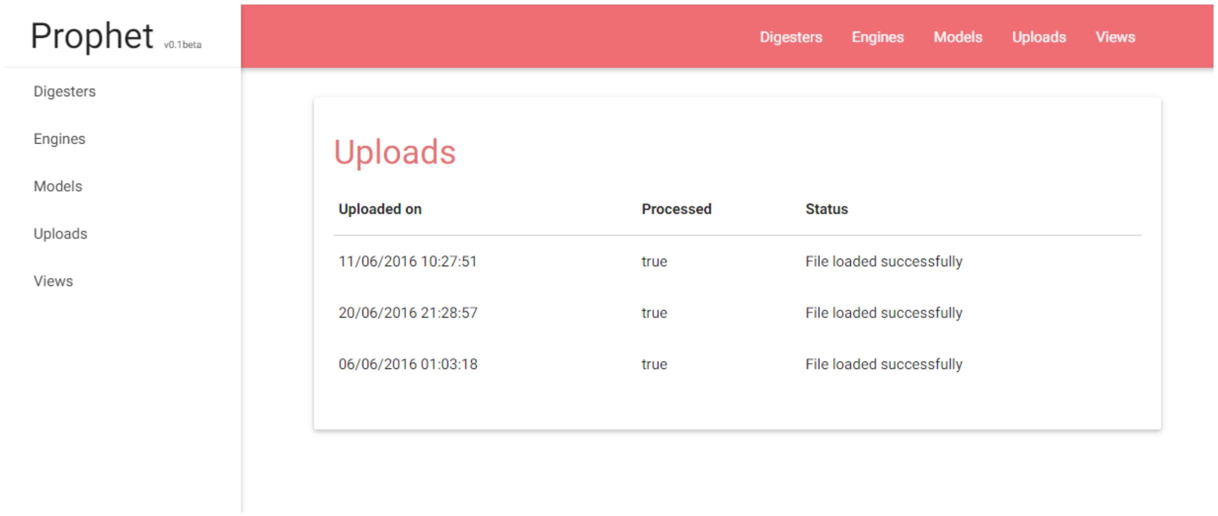


Figure 3.19: File Upload

After uploading the file, the background task (Prophet Service) will start processing the file as soon as possible. Prophet Service checks every 2 seconds if a new task should be performed. The status of the upload can be checked in the Uploads page, shown in Figure 3.20. The grid displays all the upload tasks and their statuses. The columns Processed and Status shows whether the files have been processed yet, and should display “File loaded successfully” when completed. If the process of reading and saving the file results in one or more errors, the line will contain a description of what went wrong during the process.



Uploaded on	Processed	Status
11/06/2016 10:27:51	true	File loaded successfully
20/06/2016 21:28:57	true	File loaded successfully
06/06/2016 01:03:18	true	File loaded successfully

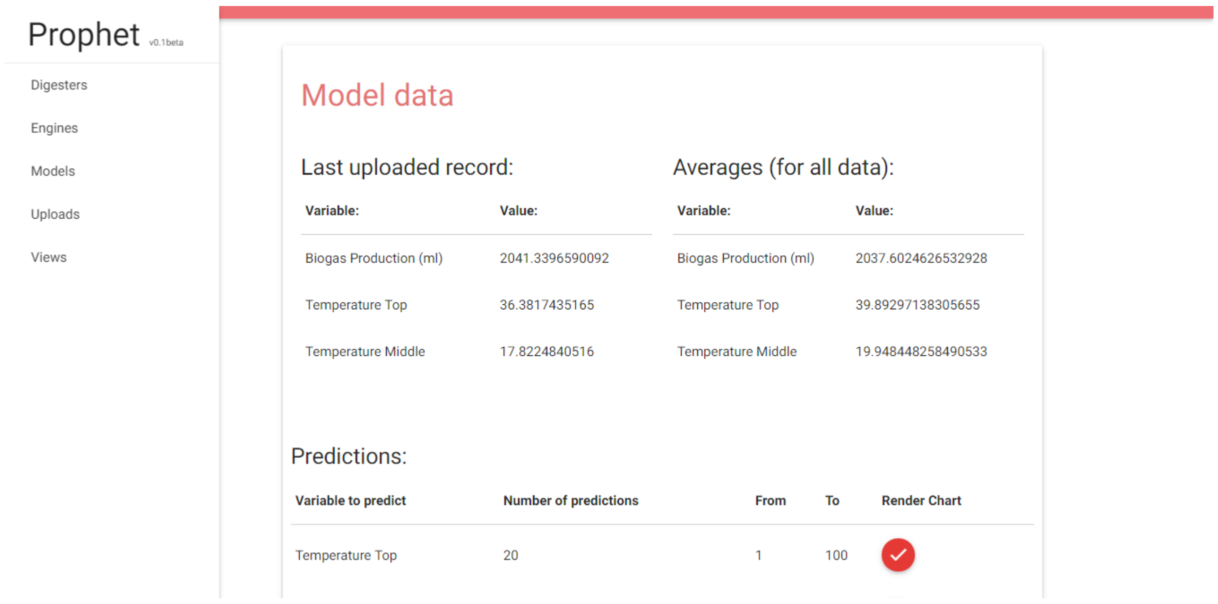
Figure 3.20: Upload Report

As shown in Figure 3.16, the model listing provides a link to the predictions page. The page displays the variables in the last uploaded record and an average of the whole data set, which is calculated during the processing of a file upload. Figure 3.21 shows the predictions page.

The predictions are displayed in a grid. Each prediction has an option to render a chart, and the user can create a new prediction by clicking in the “New Prediction” button. Figure 3.22 displays the list of predictions in the model.

When clicking the “Render Chart” button, the page will render a chart using Chart.js[33]. There are three buttons below the chart: “Download chart as image”, “Add

[33]http://www.chartjs.org


Prophet v0.1beta

Digesters
Engines
Models
Uploads
Views

Model data

Last uploaded record:

Variable:	Value:
Biogas Production (ml)	2041.3396590092
Temperature Top	36.3817435165
Temperature Middle	17.8224840516

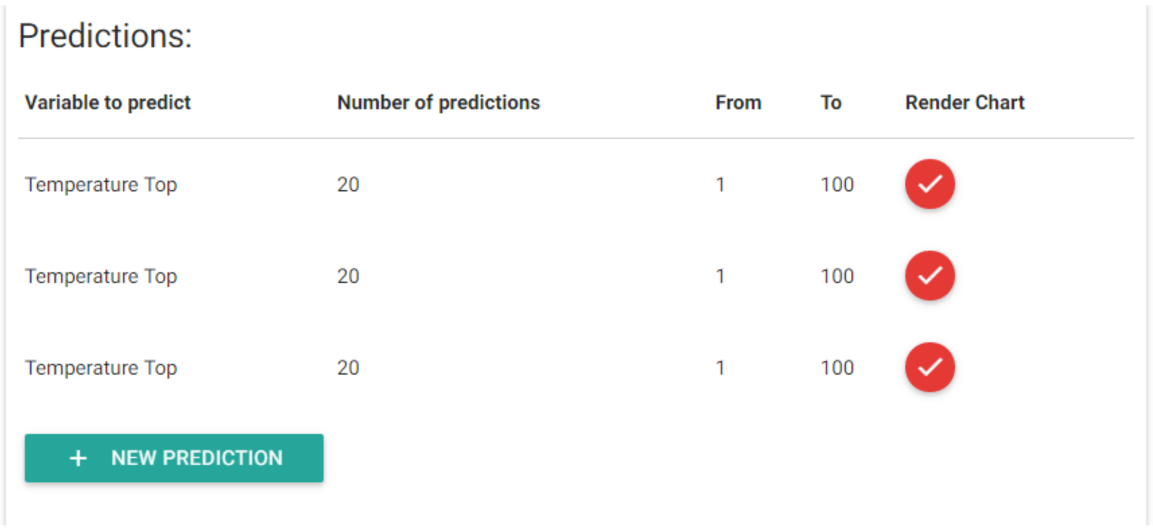
Averages (for all data):

Variable:	Value:
Biogas Production (ml)	2037.6024626532928
Temperature Top	39.89297138305655
Temperature Middle	19.948448258490533

Predictions:

Variable to predict	Number of predictions	From	To	Render Chart
Temperature Top	20	1	100	✓

Figure 3.21: Model Data


Predictions:

Variable to predict	Number of predictions	From	To	Render Chart
Temperature Top	20	1	100	✓
Temperature Top	20	1	100	✓
Temperature Top	20	1	100	✓

Figure 3.22: List of Predictions

to View” and “Create new view and add”. The concept of views is going to be explained later in the section 3.6.4. Figure 3.23 shows the rendered chart and the options. The button “Download chart as image” uses the rendered chart in the page and downloads it as a file. The server is not involved in this process.

To add the chart to a view, the user can either add to an existing view or create a new view and add the chart automatically. Figure 3.24 shows the modal

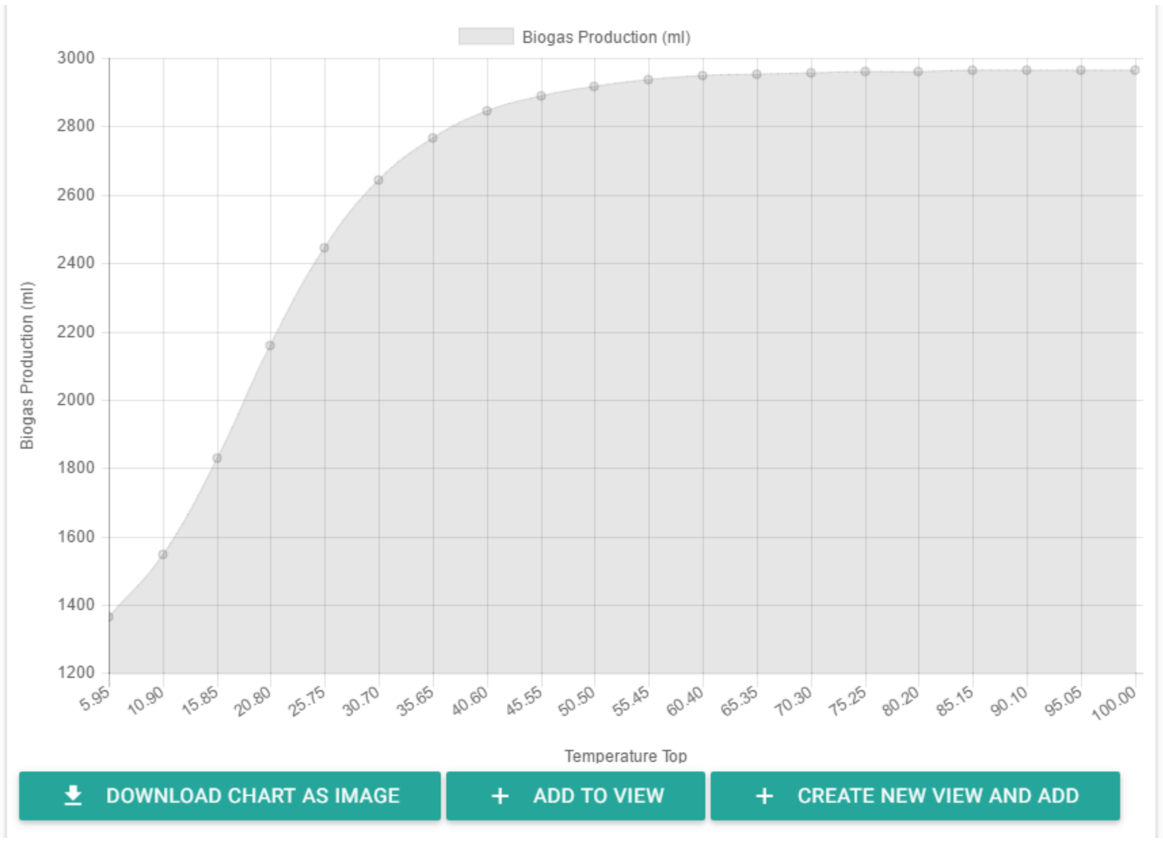


Figure 3.23: Prediction Chart

that is displayed when the button "Add to View". The user can select the view in the dropdown.

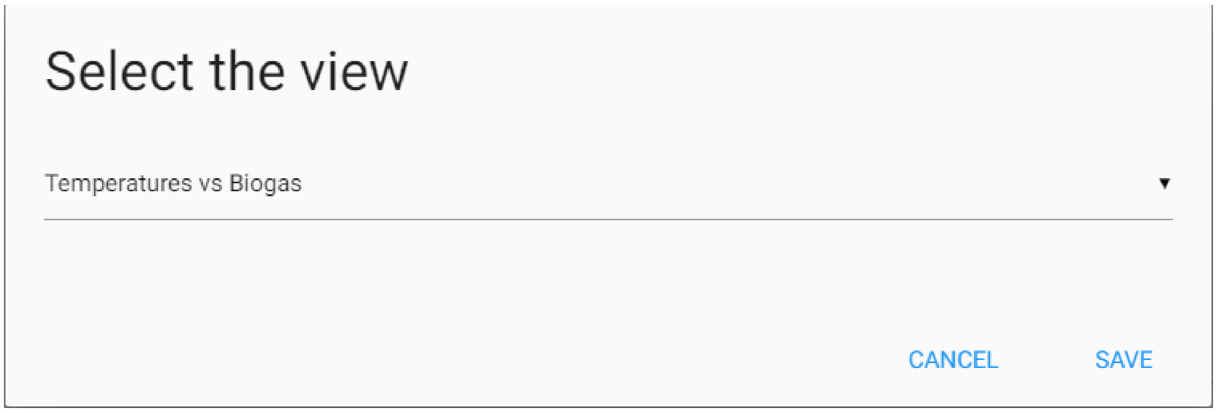
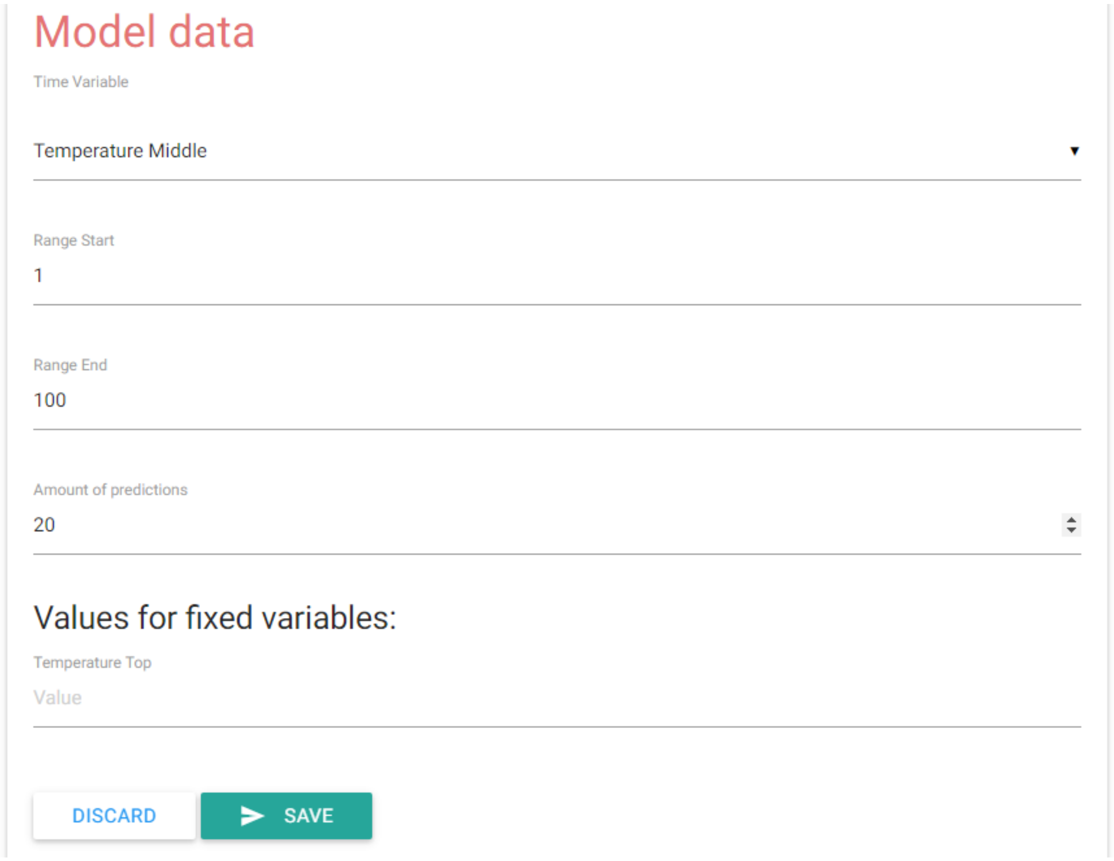

Select the view

Figure 3.24: Add to View

To create a new prediction, the user can click in the button "New Prediction". The page allows the user to configure the prediction by choosing the "Range variable" (the variable that will become the horizontal axis in the chart), the minimum and maximum values for the range, the amount of predictions (number of points in the chart) and the values for the fixed variables (the remaning variables in the model). Prophet Service will perform the prediction as soon as possible and display in the predictions page. Figure 3.25 shows the prediction configuration page;



Model data

Time Variable: Temperature Middle

Range Start: 1

Range End: 100

Amount of predictions: 20

Values for fixed variables:

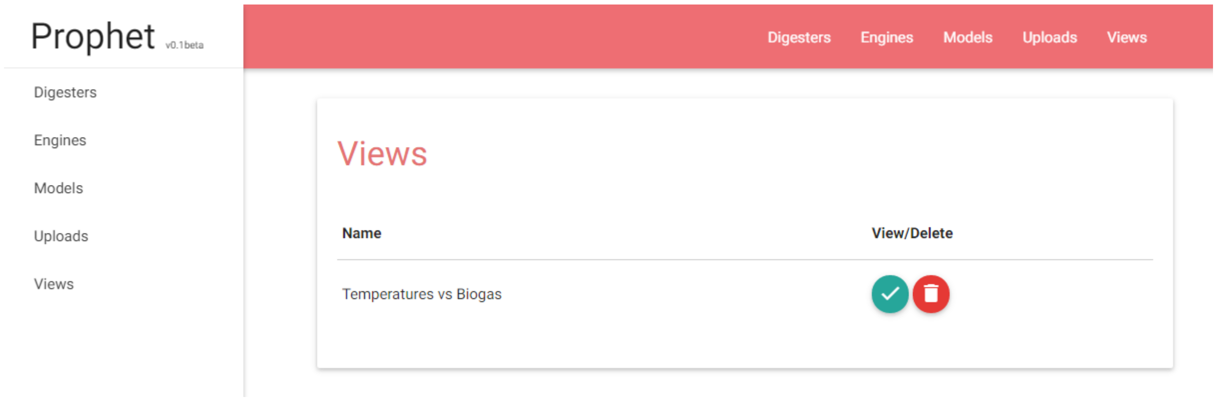
Temperature Top: Value

Figure 3.25: Configuring a prediction

3.6.4 View

Prophet Web includes views, which allow the user to add several charts together in order to compare multiple results at once. A view is identified by a name. The listing of views allows the user to delete and display a view. Creating a view is only possible in the predictions page. Figure 3.26 shows the listing of views.

The user can display a view. The page allows the user to edit the view's name.



Views

Name	View/Delete
Temperatures vs Biogas	✓ 🗑

Figure 3.26: List of Views

When the user clicks to edit the name, the title is replaced by a textbox, allowing the user to write a new name for the view and save it. Furthermore, the page allows the user to download a chart. The page is shown in Figure 3.27.

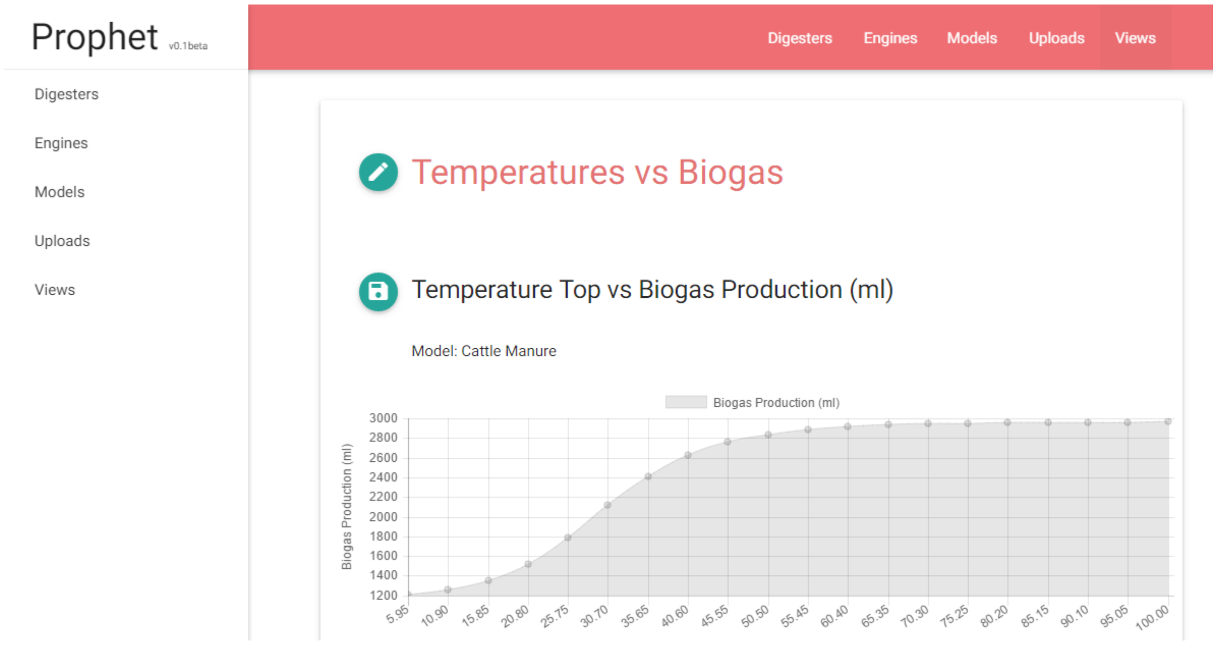


Figure 3.27: View Chart

Moreover, the page can display more than one chart, allowing the user to customize a view and compare multiple charts at the same time, as shown in Figure 3.28. For instance, the user could create a view that compares the biogas production of two or more ADs, or even compare the biogas production and energy production.

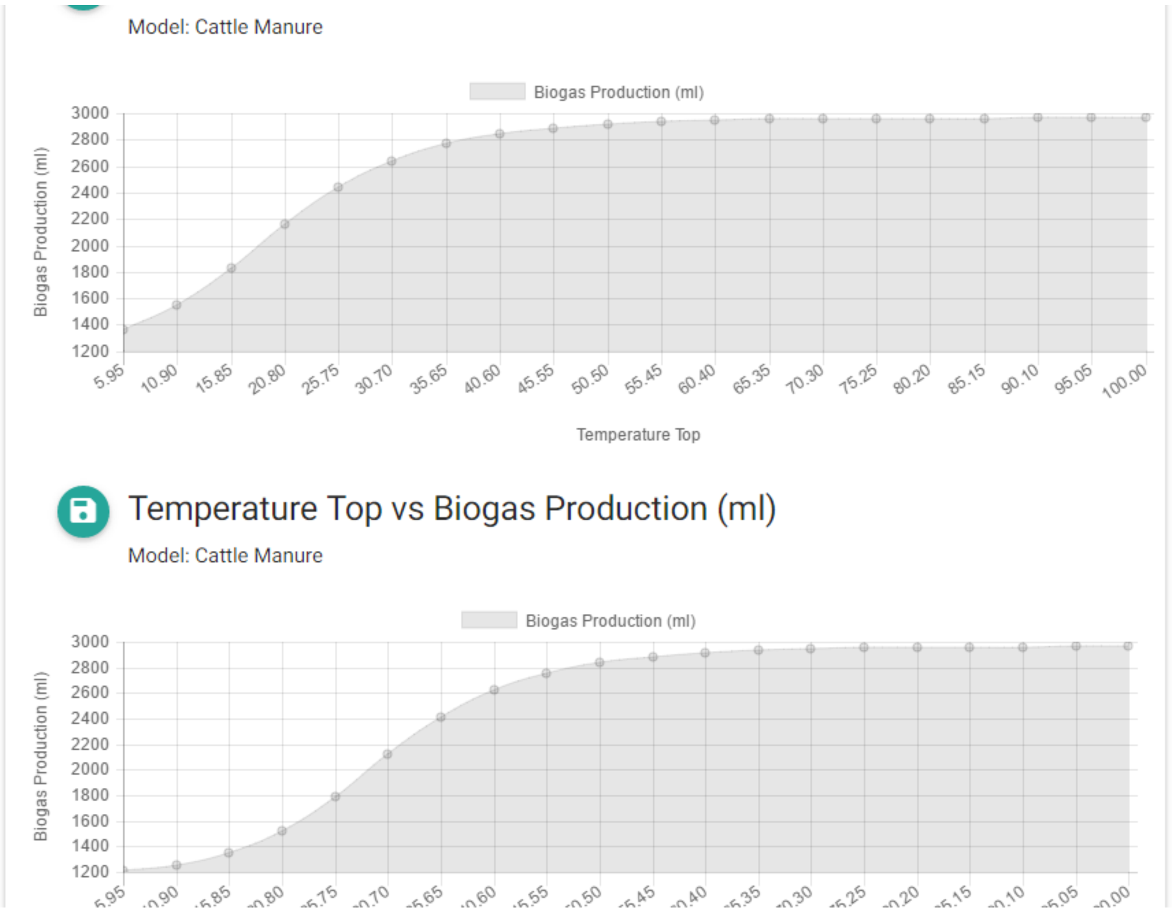


Figure 3.28: View Multiple Charts

3.7 DEPLOYMENT

Prophet Web and Prophet Service are running in the LARCC infrastructure at SETREM, which provides the necessary hardware and environment to deploy both applications. In this section, the process of deployment will be explained.

3.7.1 LARCC and Apache CloudStack

Being a laboratory focused on research for cloud computing, LARCC implements a private cloud IaaS solution using Apache CloudStack. The software allows users to create VMs (or instances) with custom computing power on demand. The LARCC team provided this work an account on Apache CloudStack, allowing to create

VMs as needed. Three Ubuntu 14.04 instances were created as described in Table 3.1.

Table 3.1: Virtual Machines at LARCC using Apache CloudStack

Name	Cores	RAM	Storage	Purpose
ProphetWeb	2	2GB	40GB	Prophet Web
ProphetService	4	8GB	20GB	Prophet Service
ProphetDB	4	12GB	100GB	Cassandra Database

To access the instances, the LARCC team exposed an SSH connection to the ProphetWeb server. Having access to ProphetWeb allows the user to SSH into the other instances as well. This is a necessary step in order to access all instances, because only ProphetWeb can be accessed remotely. Also, the LARCC team redirected all the network traffic from an external IP address to the port 80 of ProphetWeb, allowing to bind Node.js to the port 80 and serve web requests.

Overall, Apache CloudStack allows the user to easily create VMs to perform any necessary work. There is also no need to install an operating system manually, the user should just select the correct template using CloudStack's instance creation wizard. CloudStack takes care of the rest, and seconds after finishing the wizard, the instance is ready to be used.

Installing the applications on the instances was more complicated. In the next sections, the process of installing Prophet Web and Prophet Service will be described, as well as the installation of the Cassandra database.

3.7.2 Deploying the Cassandra Database

Cassandra Database requires Oracle JDK, therefore, Java needed to be installed first. After downloading Java and adding it to the OS' "PATH", Cassandra was installed. However, by default, Cassandra only listens to requests incoming from "localhost", thus not allowing external connections. In order to allow other applications outside ProphetDB to reach Cassandra, the file “conf/cassandra.yaml” was edited, changing the configurations “seed”, “listen_address” and “rpc_broadcast_address” from “localhost” to the server’s network interface IP address. The read request timeout was also increased from 10 to 60 seconds to avoid timeouts during the loading of data sets to perform model trainings. These configurations are documented in the DataStax website[34].

Even though several configurations needed to be changed, deploying Cassandra is relatively simple and can be done quickly. After setting up Cassandra, it started listening to external requests and was ready to be used by Prophet Service and Prophet Web.

3.7.3 Deploying Prophet Web

As stated before, Prophet Web is a Node.js application. Node.js was downloaded using “wget” and added to the OS’s “PATH”. By running the command “npm start” in Prophet Web’s root directory, Node.js starts the application and loads it in memory.

The file “bin/www” inside Prophet Web’s root directory contains some Javascript code that binds the application to a given port. This port was changed from 3000 to 80, allowing the application to serve web requests. Also, Cassandra’s endpoint was configured in the file “data/cassandra/config.js”, pointing to ProphetDB’s IP address.

Prophet Web is able to detect whether the database is prepared to be used or not. During the startup process, Prophet Web detects whether the Cassandra tables are created. If the keyspace “prophet” is not found, Prophet Web executes a CQL file containing the definition of all the necessary tables.

[34]https://docs.datastax.com/en/cassandra/2.1/cassandra/configuration/configCassandra_yaml_r.html

Overall, the installation of Prophet Web was very simple, and required no additional software to successfully run.

3.7.4 Deploying Prophet Service

Prophet Service was developed on Windows using Visual Studio, but the code uses only standard C++11 libraries available on Windows and Linux. The MSVC compiler allows some syntax that GCC 4.8 does not, therefore, some code had to be changed in order to support compilation with GCC 4.8.

The application depends on DataStax Cassandra C++ Driver and Openblas. Both libraries were compiled in another machine, powered by a Core i7-3610QM CPU. Openblas was compiled and deployed on LARCC with Prophet Service. Prophet Service was compiled on LARCC using CMake and Make.

The first attempt to run Prophet Service failed when the application had to invoke Openblas functions, throwing the error “Illegal Instruction”. Openblas has a long and complicated compilation process that is dependent on the system architecture. The Core i7-3610QM CPU is probably not compatible with the processor architecture installed on LARCC, therefore, Openblas had to be compiled again. The source code of Openblas was downloaded again and recompiled, setting the option “DYNAMIC_ARCH=1” to build a shared library suitable for multiple architectures, effectively solving the problem.

The second attempt to run also failed: while Prophet Service’s process started successfully, every attempt to train a model would result in a segmentation fault. It was discovered that Cassandra’s driver was behaving differently from the Windows’s version. When loading a binary field from the database, the pointer returned by the driver would get invalidated when used outside the caller. The problem was fixed by moving the contents of the blob to another memory location. In the future, better solutions could be implemented, since the researcher has little experience with the C++ language and manual memory handling.

Another important bugfix is relatd to Cassandra's maximum number of batch statements per section, which is 65,535. A 14MB file was uploaded, containing 400,000 rows. Each row in the file becomes a row in "modeldatasets" data, therefore, 400,000 insert operations need to be executed, which is more than 65,535. The upload processing algorithm creates batches of 100 operations, so the limit of 65,535 is not hit at any time. This value can possibily be increased from 100 to another value in the future.

Finally after doing more tests and fixing minor bugs, ProphetService was finally done and started running.

3.8 SOFTWARE EXPERIMENTS

After deploying Prophet Web and Prophet Service, the experience of using the system was evaluated. This section will describe how the system behaves with 5 different workloads while running on LARCC infrastructure. Furthermore, the process of using Prophet Web is compared with Octave, Matlab and WEKA. The potential users of Prophet will be identified in a later discussion.

Ideally, the user should get results in a timely manner. This means that not only the system should have good performance and allow improvements with further implementations, but also the time spent by the user on setting the system up (creating models, uploading files and making predictions) should also not take too long.

3.8.1 Software Performance

Five workloads were tested, using data sets with different numbers of rows: 400, 4000, 40,000, 400,000 and 5 million. The files are CSV files with 3 columns, which is the same format tested in Section 3.5.2. Table 3.2 shows how much time Prophet spent performing its tasks. The “Prediction time” column shows how much time it takes to perform 20 predictions. The times are measured in the following way:

- DB load: Prophet Web measures the time taken to create 5MB chunks of the original file and finishing upload all the chunks to Cassandra.
- File load: Prophet Service measures the time taken to process the file and load the data into the “modeldatasets” table.
- Training: Prophet Service measures the time taken to train a model, including the time taken to load the data from Cassandra to memory.
- Prediction: Prophet Service measures the time taken to load the model from cassandra, generate the prediction data set and executing the feed-forward algorithm.

Table 3.2: File uploads

#	**Rows**	**Size**	**DB load**	**File load**	**Training**	**Prediction**	**Total**
1	400	7.3KB	109 ms	154 ms	361 ms	89 ms	713 ms
2	4,000	78.1KB	10 ms	409 ms	881 ms	91 ms	1631 ms
3	40,000	781KB	32 ms	3163 ms	5291 ms	86 ms	9936 ms
4	400,000	7.62MB	390 ms	30 sec	50 sec	88 ms	81 sec
5	5,000,000	95.3MB	3851 ms	381 sec	489 sec	99 ms	874 sec

As can be seen on Table 3.2, the training times increase according to the size of the data set. Test #4 takes around 50 seconds to train, while test #5 takes 489 seconds. Test #5 is 12.5 times larger than Test #4, and takes 9.78 times more than Test #4 to complete. Prediction tasks execute almost instantly due to the following factors:

- The prediction data set is generated in memory, not fetched from Cassandra. Therefore, there are no network I/O operations or network latency involved in the process of creating the prediction data set.

- The algorithm uses simple matrix operations with Openblas, and the sizes of the matrices will never get too big. The training of this test resulted in 2 matrices of size 2x3 and 1x3, storing the neural network parameters.

- The size of the data set (number of predictions) will usually not be too big. In this test, 20 predictions are done, resulting in a small matrix of size 20x3.

Column “DB load” shows how much time it takes to store the file in Cassandra. Prophet Web creates smaller pieces of 5MB if the file is larger than that. To transfer the file used in test #5, Prophet took 3851 milliseconds (almost 4 seconds). For smaller workloads such as #3 and #4, Prophet should be able to transfer the file in less than a second. Before using the data, Prophet has to process the file first. Test #4 takes 30 seconds, while test #5 takes 381 seconds. Test #5 takes 12.7x more time to process than test #4 while being 12.5x larger.

The conclusion is that the application is performing well, except for data sets as large as 5,000,000 rows and 3 columns. When used with data sets as large as 100,000 rows and 3 columns, the training process should not take a long time to run (around 10 seconds). However, the test shows that some optimizations could be applied. For instance: if a model data set grows in an unexpected way (i.e. uploading several gigabytes of data), the algorithm might not be able to train the neural network or perform any prediction due to memory constraints: test #4 used 350MB of RAM during the training process, while test #5 used 2GB of RAM.

A possible solution is to use a subset of the data set (a smaller sample of the data set) in order to keep training times short and memory usage under reasonable levels that do not compromise the operating system and other tasks being performed concurrently. As explained before in Section 3.5.5, the memory usage problem exists because the algorithm uses the entire data set 100 times, using a large amount of RAM to store the entire data set in memory during the process.

Even though there are limitations in Prophet regarding sizes of data sets for training, some scientific works often use less than 1,000 records and achieve good results. Holubar et al. (2002) developed a back-propagation neural network with 9 input nodes, 3 hidden nodes and 2 ouput nodes to predict biogas production, achieving a regression coeficient of 90% using 500 records. Qdais, Bani-Hani and Shatnawi (2010) also created a back-propagation neural network with 4 input nodes, 3 hidden nodes and 3 output nodes using 177 records, achieving a correlation coefficient of 0.87. Kusiak and Wei (2014) used the Adaptative Neuro-Fuzzy Inference System (ANFIS) toolbox of Matlab 10.0 to predict methane production a with correlation coefficient of 0.99 using 725 records. Therefore, Prophet should perform well when used with similar or heavier workloads.

3.8.2 Software Usability

Creating models and predictions should be a very simple process. Prophet Web is already shown in Section 3.6, but the activity diagram in Figure 3.29 shows the complete process of creating a model, uploading the file and performing a prediction.

Figure 3.29 shows that there are a number of steps needed to be performed to finally make a prediction. However, the process is still very simple: the user essentially selects the variables to consider for predictions, and then configure how to make a prediction. The whole process does not require the user to have knowledge about ML or programming. Although there is some necessary effort to create the CSV file, other tools such as WEKA and Matlab would also require the same data in some format.

3.8.3 Comparison with Matlab, Octave and WEKA

This work's approach to Machine Learning is to implement the algorithm using C++, separating the application in two main parts. This approach is not comparable to the traditional use of Matlab and Octave, where the user has to run some code in order

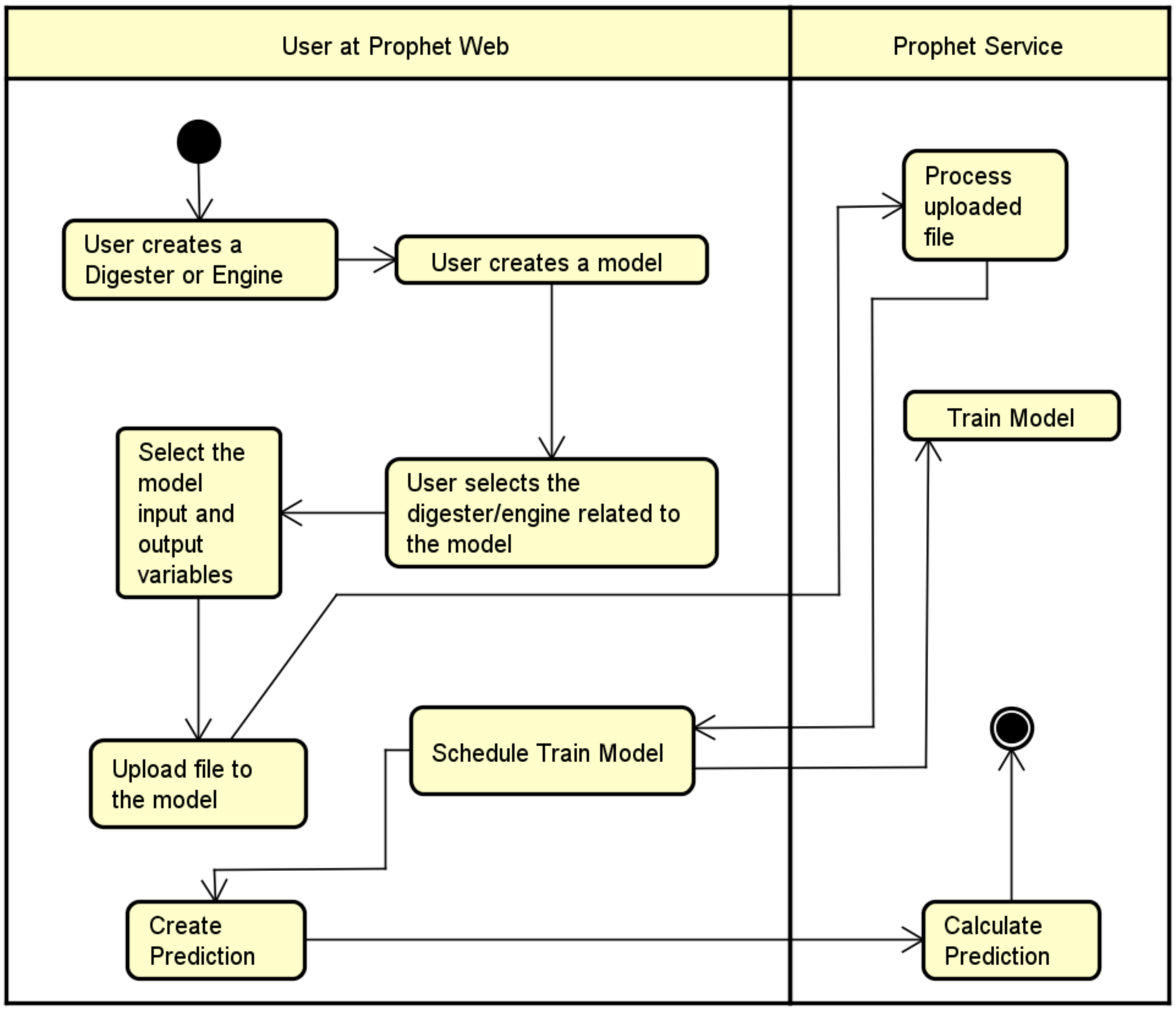


Figure 3.29: Activity diagram of Prophet Web

to perform the necessary work. However, Matlab allows the user to create desktop and web applications[35], which could enable developers to provide ML services the same way that this work's application does. This work decided to use C++ in order to achive good performance and enable future optimizations regarding better algorithms, libraries and parallelism.

Octave provides some features to create desktop user interfaces, but it is mainly a command-line language [36]. As a commmand-line language, it is possible to

[35]http://www.mathworks.com/solutions/desktop-web-deployment/
[36]https://www.gnu.org/software/octave/doc/v4.0.0/GUI-Development.html

create predefined functions that performs model trainings and predictions, as well as enabling the user to leverage the Octave language to perform other necessary tasks. This approach would also enable the developer to abstract the ML details from the user. This same approach can be achieved with Matlab, since Octave is a open source alternative to Matlab and the languages used by Matlab and Octave are very similar. However, this approach is not comparable to Prophet, as this would require the user to have some knowledge about Octave programming.

WEKA is also not comparable with Prophet. WEKA is a general-purpose tool for ML that requires the user to have knowledge about ML, as well as configuring the algorithms as necessary. In the next section, a final discussion about Prophet is presented.

3.9 DISCUSSION

The goal of the current work was to provide a SaaS software for analysis on biogas data. This version was focused on implementing machine learning algorithms to extract information from collected data, and many features can be implemented to improve it further. With this system, the work expects to bring data analysis functionality to the user with an easy-to-use interface.

Using machine learning algorithms, the user can perform predictions, for instance, on biogas production or energy production, possibly identifying input variables that results in better AD performance. In the conclusion of this work, an alternative method to optimize AD performance is presented, using optimization algorithms.

The potential users of the application are professionals from other areas not directly related to IT and data science interested on processing data and extract information from it. This work does not expect in any way to replace well established tools such as Matlab, Octave, WEKA and others. Moreover, data scientists and IT professionals that are comfortable using these tools should continue to use them, as they will provide the necessary tools for their job more than Prophet, which has a different purpose.

Even though data scientists are not the demographic of this project, Prophet can be developed and improved further by anyone who has the necessary ML and programming skills. Many features and performance improvements could be added to Prophet Service and Prophet Web, and both applications cover a wide range of subjects that can be worked upon. For instance, Prophet Web can be improved in terms of user experience, layout, responsiveness and interface feedback (give the user more clues on what the system is performing at a given moment such as training, upload and prediction progress). Additionally, Prophet Service would greatly benefit from more efforts done in areas related to memory management and parallelism.

Finally, even though this work brought the thematic of biogas and ADs to the area of Machine Learning, the implementation should be able to work for several other types of problems. Two different problems were tested using the neural network: a simple linear relation and a more complex handwritten digit recognition.

CONCLUSION

The field of machine learning has been getting increasingly more attention in the last years. The amount of data and the complexity of computational problems imposes a challenge for the current technology, requiring better ways to deal with large amounts of data and also requiring improvements on the current state-of-the-art regarding algorithms and infrastructure. Machine learning algorithms are being applied in a wide range of fields which includes agriculture and anaerobic digesters.

During the development of this work, a survey of the current state-of-the-art in biogas technology was performed. The list of evaluated technologies is not exhaustive, but it was considered that the current technology for the biogas is becoming even more important. These technologies cover several use-case scenarios for biogas and ADs, such as monitoring the AD state, controlling the biogas plant and using machine learning to predict biogas production, and all these solutions can be used as a SaaS software. However, it was found that it is difficult to collect data from the AD located in SETREM. Instead of collecting data, the efforts were focused on developing the neural network and the system architecture.

This work also found out that the scientific field has already made efforts in the field of biogas. Several works used different algorithms in order to simulate and optimize ADs using machine learning, including the use of neural networks. It was decided to use neural networks due to this factor: the scientific field and commercial technology includes neural networks in their solutions, suggesting that the algorithm is a good choice for the scenario of this work.

This work ended up created “Prophet”, an application for analysis on biogas data using neural networks. The application focuses on abstracting the ML details from the user, which is not required to have knowledge on machine learning in order to use the system. A base application was built using C++, Node.js and Cassandra, and in the future it is expected that other works implement more features in Prophet, as well as optimizing the current code in order to increase its speed, resource usage, stability and reliability.

After creating the application on a Windows environment, the C++ application (“Prophet Service”) was compiled on Linux. Since it was developed using only standard libraries available on both operating systems, only small changes needed to be done due to some non-standard syntax supported by MSVC and not on GCC. Prophet Service is able to run on Windows and Linux. Prophet Web, the web application made using Node.js, is cross-platform as well, and should be able to run in any Node.js environment.

HYPOTHESES VALIDATION

The first hypothesis predicts that the neural network will perform biogas predictions with 90% accuracy. This hypothesis was not confirmed, since the work was not able to get real world biogas data. However, in Section 3.5.2, some tests were performed in order to evaluate the prediction performance of the neural network. Two tests were performed, one with a simple data set with 2 input variables and 1 output variable, and another test with 5,000 20x20 images of handwritten digits. The first test resulted in a correlation of 0.98, while the second test classified correctly 4920 images, resulting in 98% of accuracy. Therefore, even though the hypothesis was not confirmed, it is expected that the neural network will output good predictions depending on the quality of the data set.

The second hypothesis compares the performance of Prophet against WEKA's "MultilayerPerceptron" algorithm, predicting that the neural network implemented by this work would not perform significantly (about 2%) worse than WEKA. After running tests in Section 3.5.4, WEKA reported a correlation of 0.99, while this work achieves 0.98. In another test using the handwritten digits data set, WEKA reported 0.83, while this work achieves 0.98 again. However, the comparisons between Prophet and WEKA should be made carefully, and it is advised that more tests should be performed. Therefore, while the conclusion is that this work's neural network is not significantly worse than WEKA, more tests are needed to verify which one actually performs better.

The third and last hypothesis predicts that the application developed by this work will give results in a timely manner. This work measured the performance of Prophet Web and Prophet Service. Five workloads were tested in the application, ranging from a 7KB file to a 95MB file, as can be seen on the Section 3.8.1. Some workloads can considerably stress the training process of Prophet Service, also using a considerable amount of RAM memory (2GB when processing a 95MB file). However, some scientific works often use data sets with less than 1000 records, while Prophet application was able to process 40.000 records in a total time of 10 seconds (precisely 9936 milliseconds). It is expected that Prophet will perform well under regular workloads of several hundred rows (or around 10MB of raw data), therefore, the hypothesis can be confirmed. Future works might improve the performance of Prophet Service during the training process.

FUTURE WORKS

This work used several technologies such as C++, Node.js and the Cassandra database. C++ was used because it is a medium-level language, where the developer can use high-level concepts such as classes and objects while being able to use low-level features such as manual memory management and pointers. C++ does not include a garbage collector like JVM (Java Virtual Machine) and CLR (.NET Common Language Runtime), thus the overhead of such features results in predictable performance. Even though this work found out that the C++ neural network developed in Prophet Service performs better than the Octave implementation, the difference was only 2 seconds (around 10 seconds in Prophet and 12 seconds in Octave). Future works could test whether the same difference is found on JIT-compiled languages such as Java, Scala and C#, while using the same algorithm and also using Openblas or other libraries to improve calculation times.

The application architecture shown in Section 3.3 shows that there is no direct communication between the web front-end application and the back-end service. Due to that limitation, the web front-end does not receive any feedback when a task gets completed by the back-end service. To provide more UI feedback to the user, technologies such as Redis Pub/Sub[37] can be used to implement an asynchronous eventing system that keeps both applications synchronized. Redis can also be used to improve the UI performance by providing a caching layer between the database and the application.

Future works that compare language implementations could also compare Cassandra with ScyllaDB. ScyllaDB is an implementation of Cassandra in C++, which claims to be 10 times faster than Cassandra, however, this work found little data to back up that claim. However, ScyllaDB is a promising drop-in replacement to Cassandra, and this work could not test because ScyllaDB requires a XFS filesystem[38]. Future works that test ScylllaDB and Cassandra should also consider that the algorithms used in Cassandra and ScyllaDB might be significantly different.

During the theme delimitation in Section 1.1.1, it was mentioned the possibility

[37]http://redis.io/topics/pubsub

[38]http://www.scylladb.com/kb/kb-fs-not-qualified-aio/

of using biological data from laboratorial analysis. This work did not went as far as collecting samples of the substrate and biogas and analyze them in a laboratorial environment. However, the neural network implemented in Prophet Service is expected to work if the biological input values are continuous and not discrete. For binary categorical values, this work suggests to create new input variables, mapping the binary value (e.g existence of a bacteria, substance exists or not in the substrate, and other binary categories) to 0 and 1 or -1 and 1, where -1 or 0 means false and 1 means true. For other categorical values, the suggestion is to create as many input values as there are categories. For example, if a given variable has 3 possible values, 3 input nodes are created, and each node might receive the value 0 or 1 indicating whether the value belongs to that category. However, these techniques were not tested by this work, and are left here as suggestions.

The current implementation shows that Prophet is not prepared for Big Data processes, as shown during tests in Section 3.5.2. A 100mb file takes around 500 seconds to train, and databases might be gigabytes or terabytes in size. These limitations exists because all the data is loaded in memory before training, and the training process might use up to 2GB of RAM for the same data set. Future works could use data streaming from Cassandra (by using streaming APIs or fetching the data in batches) in order to reduce memory usage, or use smaller samples of the complete data set. Performance comparisons could take in consideration memory usage, network and disk I/O, CPU usage and training time, exploring the tradeoffs of each approach.

Online training could also be implemented in the future. At SETREM there are some anaerobic digesters currently in operation, but sensor data streams are not implemented in Prophet yet. Prophet would need several modifications in order to support sensor data streams. This work could not implement a data collector service that reads sensor data streams, as this work focused on developing the neural network instead. To implement online training, a different type of model would need to be created in the system, allowing the user to change whether the model would train an existing data set or a sensor data stream, while still keeping the user interface easy to use. Online learning techniques would also need to be studied and implemented. If there is a need to perform online predictions, Prophet already has good performance, being able to perform many predictions under a second as shown in Section 3.8.1. The implementation of online training, predictions and data collection would allow even further improvements on Prophet. For instance, it could be possible to implement a real-time recommendation system that helps the user to diagnose problems before they happen and continually improve or keep biogas production at desired levels.

Another useful feature for Prophet is to implement optimization algorithms in order to automatically search for a set of input variables that produce the highest level of biogas production. A future work could use the same techniques as Qdais, Bani-Hani and Shatnawi (2010), where a genetic algorithm was used. This feature would greatly improve the current application, where the user has to set up the input variables manually. In the case of Prophet, the prediction algorithm could be used as the fitness function for a genetic algorithm. As explained before, the performance of the prediction algorithm is already good enough, performing 20 predictions in around 100 milliseconds (5ms per prediction). However, improvements on Prophet regarding parallelism could help the genetic algorithm to find a solution faster.

REFERENCES

ALVES, P. et al. Potential Assessment Tool of Biomass Electricity Generation for the State of Paraná. In: SUSTAINABLE ENERGY AND ENVIRONMENT PROTECTION - SEEP 2015, Paisley, Scotland. **Anais. . .** [S.l.: s.n.], 2015. p.148–153.

ASLANZADEH, S. et al. **The Effect of Effluent Recirculation in a Semi-Continuous Two-Stage Anaerobic Digestion System**. 2013. 2966-2981p. Tese (Doutorado em Ciência da Computação) — University of Borås, Borås, Sweden.

BARBER, D. **Bayesian Reasoning and Machine Learning**. [S.l.]: Cambridge University Press, 2012.

Bioprocess Control. **Biogas Endeavour**. 2015.

BITTENCOURT, G. **Inteligência Artificial**: ferramentas e teorias. 2.ed. Florianópolis - SC, Brazil: Editora da UFSC, 2001.

BT IT. **BOGIS**. 2015.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. **Cloud Computing**: principles and paradigms. 1.ed. [S.l.]: Wiley, 2010.

BUYYA, R.; VECCHIOLA, C.; SELVI, S. **Mastering Cloud Computing**. 1.ed. New Delhi, India: McGraw Hill, 2013.

COOPERSMITH, E. J. et al. Machine learning assessments of soil drying for agricultural planning. **Computers and Electronics in Agriculture**, Urbana, IL, United States, v.104, p.93–104, June 2014.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems**: concepts and design. 4.ed. [S.l.]: Pearson Education, 2005.

CôRTES, P. L. **Administração de Sistemas de Informação - Uma Introdução**. 1.ed. São Paulo - SP: Editora Saraiva, 2008.

DUMBILL, E. **Big Data Now**: 2012 edition. 1.ed. Sebastopol, CA: O'Reilly Media, 2012.

Gas Data. **Click! System**. 2015.

Green Lagoon. **Carbon Cloud**. 2015.

GUTIéRREZ-CASTRO, L. et al. Development of the Large Scale Biogas Technology for Energy Generation in Mexico City. In: INTERNATIONAL CONFERENCE ON EFFICIENCY, COST, OPTIMIZATION, SIMULATION AND ENVIRONMENTAL IMPACT ON ENERGY SYSTEMS, 28., Pau, France. **Anais...** [S.l.: s.n.], 2015. p.7–8.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The elements of Statistical Learning**: data mining, inference and prediction. 2.ed. Stanford, California: Springer, 2008.

HOLUBAR, P. et al. Advanced controlling of anaerobic digestion by means of hierarchical neural networks. **Water Research**, Vienna, Austria, v.36, p.2582–2588, May 2002.

KUHLENKAMP, J.; KLEMS, M.; ROSS, O. Benchmarking Scalability and Elasticity of Distributed Database Systems. **Proceedings of the VLDB Endowment**, Hangzhou, China, v.7, n.13, p.1219 – 1230, September 2014.

KUSIAK, A.; WEI, X. Prediction of Methane Production in Wastewater Treatment Facility: a data-mining approach. **Annals of Operations Research**, Iowa, v.216, n.1, p.71–81, April 2014.

LABATUT, R. A.; GOOCH, C. A. Monitoring of Anaerobic Digestion Process to Optimize Performance And Prevent System Failure. , Ithaca, NY, 2012.

LAKSHMAN, A.; MALIK, P. Cassandra - A Decentralized Structured Storage System. , Facebook, v.104, 2009.

MAGOULAS, R.; LORICA, B. **Big Data**: technologies and techniques for large-scale data. 1.ed. Sebastopol, CA: O'Reilly Media, 2009.

MARTIN, J. P.; CENDROWSKI, H. **Cloud Computing and Electronic Discovery**. 1.ed. Hoboken, New Jersey: Wiley, 2014.

MCQUEEN, R. J. et al. Applying Machine Learning to Agricultural Data. **Computers and Electronics in Agriculture**, [S.l.], v.12, p.275–293, June 1995.

MURPHY, K. P. **Machine Learning**: a probabilistic perspective. London, England: MIT Press, 2012.

NIELSEN, M. **Neural Networks and Deep Learning**. 1.ed. [S.l.]: , 2016.

NORVIG, P.; RUSSEL, S. **Inteligência Artificial**. 2.ed. Rio de Janeiro - RJ, Brazil: Elsevier, 2004.

NORVIG, P.; RUSSEL, S. **Artificial Intelligence - A modern approach**. 3.ed. Upper Saddle River, New Jersey: Pearson Education, 2010.

O'BRIEN, J.; MARAKAS, G. **Administração de Sistemas de Informação - Uma Introdução**. 13.ed. São Paulo - SP: McGraw Hill, 2007.

OLIVEIRA-ESQUERRE, K.; MORI, M.; BRUNS, R. Simulation of an industrial wastewater plant using Artificial Neural Networks and Principal Component Analysis. **Brazillian Journal of Chemical Engineering**, Unicamp, Brazil, v.19, n.04, p.365–370, December 2002.

OSBORNE, J. W. **Best Practices in Quantitative Methods**. 1.ed. North Carolina: SAGE Publications, 2008.

QDAIS, H. A. A.; BANI-HANI, K.; SHATNAWI, N. Modeling and optimization of biogas production from a waste digester using artificial neural network and genetic algorithm. **Resources Conservation and Recycling**, [S.l.], March 2010.

RATNER, B. **Statistical and Machine-Learning Data Mining**: techniques for better predictive modeling and analysis of big data. 2.ed. [S.l.]: CRC Press, 2012.

RITTINGHOUSE, J. W.; RANSOME, J. F. **Cloud Computing**: implementation, management, and security. 1.ed. [S.l.]: CRC Press, 2009.

ROVEDA, D. et al. Uma Avaliação Comparativa dos Mecanismos de Segurança nas Ferramentas OpenStack, OpenNebula e CloudStack. **Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação**, Três de Maio, Brazil, v.4, n.1, p.15, March 2015.

ROVEDA, D.; VOGEL, A.; GRIEBLER, D. Understanding, Discussing and Analyzing the OpenNebula's and OpenStack's IaaS Management Layers. **Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação**, Três de Maio, Brazil, v.3, n.1, p.15, August 2015.

RUNKLER, T. A. **Data Analytics**: models and algorithms for intelligent data analysis. 1.ed. [S.l.]: Springer, 2012.

Sota Solutions. **Sota Solutions**. 2016.

TANENBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos - Princípios e Paradigmas**. 2.ed. São Paulo - SP: Pearson Education, 2007.

VOGEL, A. **Surveying the Robustness and Analyzing the Performance Impact of Open Source Infrastructure as a Service Management Tools**. Faculdade Três de Maio - SETREM. Três de Maio, RS, Brazil: [s.n.], 2015.

VOGEL, A. et al. Private IaaS Clouds: a comparative analysis of opennebula, cloudstack and openstack. In: EUROMICRO INTERNATIONAL CONFERENCE ON PAR-

ALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING (PDP), 24., Heraklion Crete, Greece. **Anais. . .** IEEE, 2016.

WATSON, H. J. Tutorial: big data analytics: concepts, technologies, and applications. **Communications of the Association for Information Systems**, University of Georgia, v.34, p.1247–1268, 2014.

ÖZSU, M. T.; VALDURIEZ, P. **Princípios de Sistemas de Bancos de Dados Distribuídos**. 2.ed. Rio de Janeiro - RJ, Brazil: Editora Campus, 2001.

APPENDIX

3.10 COLUMN NAMES

These are the meanings of each field in the table “modelpredictions”:

Table 3.3: “modelpredictions” table fields

Field	Description
temp1	Temperature (Top)
temp2	Temperature (Middle)
temp3	Temperature (Bottom)
tempout	Temperature (Outside)
ph	pH
pressure	Pressure
retentiontime	Retention Time
substrateconcentration	Substrate Concentration (g/liter)
enginespeed	Engine Speed
biogasml	Biogas Production
ch4ml	CH4 (Methane) Production
co2ml	CO2 (Carbon Dioxide) Production
energy	Energy Production