8-25-2016 12:00 AM

Agora: A Knowledge Marketplace for Machine Learning

Mauro Ribeiro
The University of Western Ontario

Supervisor
Dr. Miriam A. M. Capretz
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Engineering Science

Abstract

More and more data are becoming part of people's lives. With the popularization of technologies like sensors, and the Internet of Things, data gathering is becoming possible and accessible for users. With these data in hand, users should be able to extract insights from them, and they want results as soon as possible. Average users have little or no experience in data analytics and machine learning and are not great observers who can collect enough data to build their own machine learning models. With large quantities of similar data being generated around the world and many machine learning models being used, it should be possible to use additional data and existing models to create accurate machine learning models for these users.

This thesis proposes Agora, a Web-based marketplace where users can share their data and machine learning models with other users with small datasets and little experience. This thesis includes an overview of all the components that make up Agora, as well as details of two of its main components: Hephaestus and Sibyl.

Hephaestus is a domain adaptation method for multi-feature regression models with seasonal adjustment, which can improve predictions for small datasets using information from additional datasets. Hephaestus works in the pre- and post- processing phases, making it possible to work with any standard machine learning algorithm. As a case study, we built predictive models using the proposed method to predict school energy consumption with only one month of data, improving accuracy to the same level as if 12 months of data were being used.

Sibyl is a flexible, scalable and non-blocking machine learning as a service, which facilitates the creation of multiple predictive models and running them at the same time. As a case study, we implemented Sibyl equipped with three machine learning algorithms to show the flexibility of adding new algorithms. We also executed three models at the same time to demonstrate that they can run without interference from another model.

The results obtained in this research demonstrates the concept of Agora. Users can share the same platform to provide or consume knowledge and create multiple concurrent machine learning models.

Keywords: marketplace, domain adaptation, seasonal adjustment, energy consumption, machine learning as a service, energy consumption

Acknowledgements

The writing of this thesis involved a mix of things: knowledge, patience, will power, healthy mind and loss of mind as well. This means that this thesis would not have been possible without the support of many people that during these last two years.

First, I would like to thank my supervisor Dr. Miriam Capretz for all the effort she made to make this thesis become a reality. Without her guidance and support, I would never have reached this point. Thank you for welcoming me to Western and for giving me this huge opportunity to accomplish this new milestone in my life.

Infinite thanks to my mother, Helena Nana Ribeiro, and my father, Itamar Bittencourt Ribeiro, for bringing me into this world and supporting all my decisions, even if they meant moving too far away from home. Thank you for allowing me always to pursue my dreams and accept new challenges.

Marjorie Rodrigues, my dear wife, you are the only one on this planet that really knows how hard it was to change our whole life to get to where we are. Our teamwork can make our score much higher in this game called Life. Thank you Marjorie, for guiding me during these years. Simply love you.

Thank you my old friends Dennis Bachmann and Roberto Barboza Jr. (a.k.a. Juju) for being nearby and having a beer together to chill out sometimes. Thanks to all my laboratory colleagues and friends Wilson Higashino, Daniel Berhane Araya, Wander Queiroz, and Alex L'Heureux, for laughing together and supporting each other. My special thanks to Dr. Katarina Grolinger and Dr. Hany ElYamany for believing in my potential, for all your patience reviewing my research papers, and for all the time you spent helping me.

Thanks to all my three brothers, my family, my friends, and my colleagues that accompanied me during my life. Thank you Misty, my dog, for listening to all my complaints without understanding anything of what they meant. Thank you, my video games, for de-stressing me after exhausting days. Thank you uncountable sweet treats and greasy snacks that were eaten during the writing of this thesis. Finally, thanks for all of those that will never pardon me because I forgot to mention them here.

Contents

Abstract i

Acknowledgements ii

List of Figures v

List of Tables vi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Contribution . . . 4
- 1.3 Organization of the Thesis . . . 5

2 Background and Literature Review 7
- 2.1 Background . . . 7
  - 2.1.1 Machine Learning . . . 7
  - 2.1.2 Normalization . . . 9
  - 2.1.3 Time Series Regression Model and Seasonal Adjustment . . . 10
  - 2.1.4 Transfer Learning . . . 12
  - 2.1.5 Domain Adaptation . . . 15
  - 2.1.6 Service Component Architecture . . . 15
- 2.2 Related Work . . . 17
  - 2.2.1 Transfer Learning . . . 17
  - 2.2.2 Machine Learning as a Service . . . 23
  - 2.2.3 Knowledge Sharing and Marketplaces . . . 24
- 2.3 Summary . . . 26

3 Agora: a Knowledge Marketplace for Machine Learning 28
- 3.1 Overview . . . 28
- 3.2 Architecture . . . 30
  - 3.2.1 Actors . . . 32
    - Providers . . . 32
    - Consumers . . . 32
  - 3.2.2 Providers' Database . . . 33
    - Source Domain Data . . . 33
    - Metadata . . . 33

Model Specification . . . 34
Global Data . . . 34
3.2.3 Consumers' Database . . . 34
Target Domain Data . . . 34
Metadata . . . 35
Predictive Set . . . 35
Prediction . . . 35
3.2.4 Processing Components . . . 35
Cerberus: Data Gatherer . . . 35
Argus: Similarity Search . . . 36
Hephaestus: Domain Adaptation . . . 36
Sibyl: Machine Learning as a Service . . . 37
3.3 Summary . . . 38

4 Hephaestus: a Domain Adaptation Method for Regression 39
4.1 Method . . . 39
4.1.1 Time Series Domain Adaptation . . . 41
4.1.2 Atemporal Domain Adaptation . . . 43
4.1.3 Standard Machine Learning . . . 45
4.1.4 Adjustment . . . 45
4.2 Case Study . . . 46
4.2.1 Evaluation . . . 47
4.2.2 Implementation . . . 49
4.2.3 Preliminary Analysis . . . 51
4.2.4 Results . . . 52
4.3 Summary . . . 55

5 Sibyl: a Machine Learning as a Service 58
5.1 Architectural Design . . . 58
5.1.1 Modeler Composite . . . 61
5.1.2 Model-μ Composite . . . 62
5.2 Process . . . 63
5.3 Case Study . . . 67
5.3.1 Algorithms . . . 68
5.3.2 Results . . . 69
5.4 Summary . . . 72

6 Conclusions and Future Work 75
6.1 Conclusions . . . 75
6.2 Future Work . . . 76

Bibliography 81

Curriculum Vitae 85

List of Figures

2.1 Machine learning tasks categorization. 8
2.2 Correlation between z-score of a normal curve and its probability distribution. . 11
2.3 Types of machine learning approaches. 14
2.4 SCA artifacts. 16

3.1 High-level diagram showing how Agora should work. 30
3.2 Overview of Agora's architecture. 31
3.3 SCA composite of Cerberus. 36

4.1 Overview of Hephaestus method. 40
4.2 Time series domain adaptation phase. 41
4.3 Local and global z-score normalizations. 43
4.4 Rolling base forecasting. 48
4.5 Sample of an energy consumption timeline. 49
4.6 Seasonal index for all four schools. 52
4.7 Polynomial interpolation of the correlations between mean temperature and energy consumption. 53
4.8 Errors for the schools' energy consumption prediction (MLP). 54
4.9 Timeline comparison between the predictions for MLP models. 55
4.10 Errors for the schools' energy consumption prediction (SVR). 56

5.1 High-level diagram showing how Sibyl should work. 59
5.2 Sibyl's simplified architecture using SCA notation. 60
5.3 Sibyl's detailed architecture using SCA notation. 61
5.4 Process flow of build phase. 64
5.5 Process flow of train phase. 65
5.6 Process flow of validate phase. 65
5.7 Process flow of test phase. 66
5.8 Process flow of predict phase. 66
5.9 Sibyl's models screen. 70
5.10 Sibyl's validate screen. 71
5.11 Sibyl's test screen. 73
5.12 Sibyl's predict screen. 74

List of Tables

4.1 Errors for the schools' energy consumption prediction (MLP) 54
4.2 Errors for the schools' energy consumption prediction (SVR) 56

Chapter 1

Introduction

1.1 Motivation

The world today relies on data as they play an important role in our civilization when it comes to decision making. Data enable understanding of past behaviors, patterns and trends, as well as the measurement of current performances. This information is necessary for organizations to properly design efficient strategic plans and maintain or accelerate their progress. For example, a governmental budget can be approved depending on social-economic indexes, or private investments can be made depending on sales performance of a company. Data come from many sources in addition to society and economy, including global data sources like Web sites, social media, mobile applications, news networks, weather and government. The amount of data from all these sources has been continuously growing. The technology enables the collection and sharing of various types of data during each millisecond. Sensors are becoming cheap and popular, enabling gathering of new data. Now that all these technologies can be connected to a network (*i.e.*, the Internet of Things), it is becoming viable to collect more data from specific contexts at higher level of detail. This is leading to an explosive growth of data in every dimension, including: (a) the number of attributes, (b) the number of data points and (c) the number of datasets.

The increase in the number of datasets is pushed by the interest of companies and individuals in monitoring things for various reasons. For example, monitoring energy consumption can reduce expenses, monitoring sales can increase revenue, and monitoring physical activities

can lead to personal achievements. This demand, allied with the supply of new technologies, creates a perfect scenario for creating new software applications combining data analytics.

However, no matter how big the data get, they may be useless without proper preparation and processing. The value is actually realized when meaningful information is extracted from all this data. Machine learning algorithms enable analysis and extraction of valuable information such as patterns and associations. Nonetheless, sometimes data are dirty or noisy and must be pre-processed to improve learning.

One important application of machine learning is predictive modeling, which can infer future values based on measured data from the past. With precise predictions, individuals, scientists and companies can better visualize the trends in the current situation and take early actions to avoid unwanted outcomes or to adapt to a specific condition. More specifically, a predictive regression model is used when the label (*i.e.*, the value to be predicted) is a numeric value. Some examples of predictive regression model applications are: (a) building managers can predict energy consumption to manage energy costs; (b) retailers can estimate the number of customers in the next few days so they can better manage their resources; (c) Web site builders can understand how visits to their sites are affected by external factors and recommend appropriate content; and (d) passengers can predict how crowded the next bus will be and decide whether to take the bus or wait for a less packed one.

The number and complexity of data samples required by regression models to provide precise predictions depend on data randomness and number of features [14, 21]. The randomness of data is its lack of pattern, whereas features are attributes that are correlated with the label to justify the randomness. The greater amount of data available, the more precise will be the correlation with features and labels. The problem is that final users and data analysts are in a hurry to see accurate predictions as soon as possible. They do not want to wait for several months or years of data collection to obtain accurate predictions. Nonetheless, it is hard to create an accurate prediction without enough data. However, it is possible to create more accurate predictions for small or new datasets using additional datasets. For example, building energy consumption depends on many factors such as human activities, machinery schedules and weather. Small datasets do not have enough data samples to represent all possible combinations of these factors. Hence, new buildings cannot generate accurate energy consumption

forecasts due to the short time they have been collecting data, at least until enough samples are collected, which can take a long time. However, new buildings can use data from other buildings to increase data variance, filling the gaps left by missing samples in the original dataset. In this way, it becomes possible to make predictions for new buildings under conditions they have not faced so far, shortening the time required for collecting enough data to make accurate predictions.

The solution would be straightforward if all datasets were in the same domain: just consider all datasets as one and run a standard machine learning algorithm. However, in the real world, data are gathered in diverse contexts, and any small deviation in context can cause an unwanted divergence in the domain represented. Hence, it is likely that two distinct datasets are not in the same domain (*i.e.* they have different distributions). A domain can vary based on an infinite number of factors. For example, if one building uses incandescent lights and another fluorescent lights, their energy consumptions will be different and cannot be used together directly in a predictive model without making adjustments.

Time can also affect domains. Almost everything in the real world experiences the effects of time, be they the inevitable aging of things, the seasons of the year, or — in the case of data depending on human activities — business hours, business days, and holidays. Each domain can have different seasonal patterns (e.g., one building can have peaks during Mondays while another on Fridays, depending on the routine each building follows). In order to analyze different domains with different seasonality together, the seasonality must be removed.

Analysts have been using data from different domains for a long time to better understand phenomena and patterns. For example, using social-economic data from diverse regions can help to understand the situation of a specific place. However, these analyses were usually done by human intuitions. Recently, many approaches for machine learning address the issue of transferring knowledge from one domain to another. Collectively, they are known as transfer learning. Most of them are difficult to understand by people with less experience in statistics and computer science. In addition, seasonal adjustment approaches are able to remove the seasonality from time series.

Another problem that makes it difficult to popularize machine learning is the complexity of understanding and implementing it, as well as the related costs. Large companies have enough

resources to hire data scientists and invest in their own machine learning solutions. On the other side, small companies, developers and researchers in general have difficulties climbing the steep learning curve of how machine learning works and then building their own solutions or integrating with third-party ones. Furthermore, machine learning may require computational resources with high costs.

Users that lack data and skills in machine learning are discouraged by the difficulty of creating accurate predictive models. If they could have access to more data and to pre-set machine learning specifications, they might be encouraged to generate models to create accurate predictions. However, gaining access to third-party data and machine learning specifications can be difficult. In addition, third-party data must probably be pre-processed before use and machine learning specifications are useless unless users know how to implement them, which is not the case in this scenario. Considering all these challenges, how could these users gain access to efficient machine learning services with accurate predictions?

1.2 Contribution

The main contribution of this thesis is Agora, a Web-based software as a service (SaaS) marketplace where users can share knowledge to build accurate machine learning models. Agora enables experienced users (providers) to share their knowledge, including data and machine learning specifications, with less experienced users (consumers) to create accurate machine learning models. Agora also makes it easier for consumers to find the best providers and knowledge. This thesis provides a modularized architecture for Agora, dividing the platform between various components with specific roles and functions. All the components as well as their roles, functions and relationships are explained. Particularly, two of its main components, Hephaestus and Sybil, are described in high level of detail as they are two important additional research contributions.

Hephaestus is a novel transfer learning method that enables creation of accurate machine learning models for a target domain using knowledge transferred from another domain. Hephaestus is a domain adaptation method for time series regression models with multi-feature datasets that are readable to understand and implement. By working as pre- and post-processing

stages, Hephaestus works with any standard machine learning algorithm. To provide a better understanding of Hephaestus, this thesis presents some background and related studies in transfer learning. This thesis also provides the details of Hephaestus design and operation, as well as a case study involving a real world regression problem. The results show that Hephaestus is a feasible and efficient method for transferring knowledge between domains and can work for regression tasks considering time series and multiple features.

Sibyl is a scalable, flexible, and non-blocking platform as a service (PaaS) for machine learning. Based on service component architecture (SCA), Sibyl builds on the advantages from service oriented architecture (SOA) and is scalable and practical to adapt by adding, removing, changing or linking any component. This approach also makes the system more flexible in handling multiple data sources and different machine learning algorithms at the same time. Because multiple users will be using the same platform, computational resources can be shared or allocated on demand, reducing overall costs. By specifying a well defined interface, users can have access to machine learning processing efficiently from anywhere, at any time. Users do not need to be concerned with implementation and computing resources, but are left free to focus on their data and their outcome. To provide a better understanding of Sibyl, this thesis presents some background and related works in machine learning, machine learning as a service and service oriented architecture (SOA). Details of Sibyl design and operation are also described and a case study on predictive modeling is presented. The results demonstrate that Sibyl can handle predictive models at the same time.

The results obtained in this research demonstrates that knowledge can be transferred between domains and that a machine learning as a service can be developed. Agora integrates both Hephaestus and Sibyl‘s outcomes to create various machine learning models using small datasets and knowledge from larger datasets to and run them at the same time. This thesis also contributes by starting some discussion about possible future research opportunities and applications related to Hephaestus, Sibyl and Agora.

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows:

- Chapter 2 provides a literature review, giving background information that is useful in understanding this work and describing related studies. First, an introduction to the technical terms and concepts used in the research described in this thesis is presented. Second, recent and popular techniques for transfer learning, machine learning as a service and marketplaces are reviewed. Finally, the contribution of this thesis is contrasted with existing work.

- Chapter 3 gives an overview of the contribution of this thesis by presenting the architecture for Agora and giving a brief description of each of its main components, including how they are related with each other.

- Chapter 4 delves into the details of Hephaestus, a novel domain adaptation method for multi-feature regression tasks with seasonal adjustment. This chapter is divided into two main sections: *Method*, where the concepts and design of Hephaestus are described; and *Case Study*, where Hephaestus is implemented and tested on energy consumption forecast using different schools. Finally, a summary concludes this chapter.

- Chapter 5 investigates Sibyl, a novel platform as a service platform for machine learning. This chapter is divided into three main sections: *Architectural Design*, which presents an overview of the architecture and describes each of the components and their interactions; *Process*, which describes the work-flow to provide a better understanding of how Sibyl works; and *Case Study*, which presents implementation details and application of this approach to energy consumption forecasting using multiple algorithms at the same time. Finally, a summary concludes this chapter.

- Chapter 6 concludes this thesis and discusses possible future research involving Agora.

Chapter 2

Background and Literature Review

The goal of this chapter is two-fold: first, it introduces the terms and concepts related to the topics to understand Agora; second, it gives an overview of the related works for marketplaces, machine learning as a service and transfer learning, as well as identifies the research gaps and explains briefly how Agora addresses them.

2.1 Background

This section defines and discuss the concepts of machine learning, transfer learning and service component architecture, which are foundation to understand Agora.

2.1.1 Machine Learning

Machine learning is one of the fastest growing fields in computer science [4]. It is a collection of statistical techniques for building mathematical models that can make inferences from data samples (known as a training set). Machine learning is a part of artificial intelligence: it must adapt itself to a changing environment.

Figure 2.1 roughly lists the main categories of machine learning tasks and how to choose between them. There are three main types of learning [4]: (a) *supervised learning*, when the training set is labeled (*i.e.*, it contains the attribute that the model is trying to estimate); (b) *unsupervised learning*, when the training set is not labeled, and (c) *reinforced learning*, when

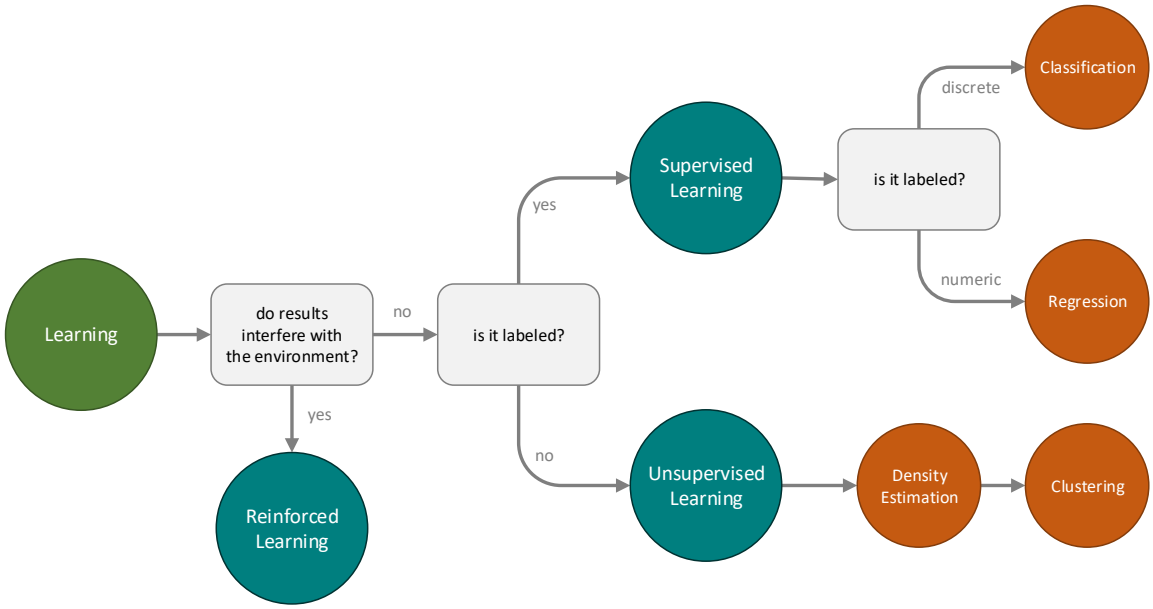


Figure 2.1: Machine learning tasks categorization.

the learned results lead to actions that change the environment.

The labels in supervised learning can be discrete or continuous, which are handled by *classification* and *regression* tasks respectively. Classification tasks are used mostly for prediction, pattern recognition and outlier detection, whereas *regression* tasks are used for prediction and ranking. Unsupervised learning is known as *density estimation* in statistics and is represented mainly by *clustering* algorithms. Classification, regression and clustering tasks are widely used in data mining (applications of machine learning to large databases), whereas reinforced learning is mostly used in decision-making problems (*e.g.*, a computer playing chess).

Independently of the applications just described, machine learning techniques work in a similar way: the model learns from a training set and then becomes able to make inferences for a new data set. This abstraction inspires the creation of a generic architecture to support any machine learning algorithm. This thesis will focus on regression predictive modeling, although the approach can be adapted for other algorithms.

In predictive modeling, once rules have been extracted from past data (the training set), the model can make accurate prediction for new instances of data (the predictor set) if the future is similar to the past. Spam filtering, investment risk and energy consumption forecasting

are some examples of predictive modeling. Predictive modeling approaches include: Artificial Neural Networks for energy consumption [3], Support Vector Machines for energy consumption [3] and K-Nearest Neighbors for wind power [48].

Validation for predictive models has a twofold importance: (a) choosing the most accurate algorithm and parameters; and (b) estimating the expected error for new predictions [4]. Accuracy can be related with errors, which can be calculated by comparing the estimated results from the model with the real measured results. A popular validation technique for predictive models is the K-Fold Cross-Validation. The data set is split randomly into K parts of the same size. One of the K folds is used to calculate the errors using the other K-1 folds to train the algorithm. The same process is repeated K times each time using different fold for validation. This method guarantees that the entire data set is validated with statistical significance.

Different models can perform better or worse, depending on the used algorithms, parameters and data set. However, there is no such a thing as the "best" learning algorithm [4]. For any algorithm, there are data sets that perform very accurately and others that perform very poorly. For the same data set, different algorithms can perform differently because of their own nature. Sibyl helps the user to run multiple algorithms and compare their performance, so the most suitable algorithm can be chosen.

2.1.2 Normalization

Normalization is defined in this thesis as follows:

Definition (Normalization). *Given two different sets of values S_1 and S_2, normalization Ψ is a linear transformation where $\Psi(S_1)$ and $\Psi(S_2)$ are in the same domain and have approximated distributions.*

It is the process of aligning values measured and stored at different scales or proportions to a common scale so that they can be compared and operated together. In machine learning, normalization is an essential step in pre-processing and, if well executed, can significantly improve model performance.

Min-max is one of the most popular normalization methods in data mining [23, 38]. It is commonly referred to simply as "normalization" or sometimes as "feature scaling" and is

represented by the equation:

$$\text{min-max} = \frac{x - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{2.1}$$

where x is the current value, X contains all values, X_{min} is the minimum value, and X_{max} is the maximum value. The main notion of min-max is to rescale and confine samples to an interval between 0 and 1.

The z-score, also known as the standard score or standardization, is another normalization method. It is represented by the equation:

$$z = \frac{x - \mu}{\sigma} \tag{2.2}$$

which returns the distance of x from the mean μ, measured in multiples of the standard deviation σ [6]. Figure 2.2 illustrates how a z-score is distributed in a Euclidean space.

Unlike min-max, the z-score does not limit values to lie between specific minimum and maximum. Instead, it maintains the normal distribution, in which the majority of values are statistically concentrated within a range of z-scores (*e.g.*, in Figure 2.2, more than 95.4% of the data are located between -2 and 2). Because a z-score can be any real number, outliers can still have higher values that are not limited by the minimum and maximum.

The use of z-score normalization is recommended for attributes that follow a normal distribution (a.k.a. the Gaussian or bell curve). From the central limit theorem [12], when an attribute is determined by independent random variables, its distribution approximates a normal distribution. Assuming that the observations of natural phenomena (such as outdoor temperature) and human activities are determined by randomly distributed variables, they can be considered to be normally distributed and, therefore, can be normalized using the z-score. Hephaestus uses the z-score not only to normalize the feature values within the target dataset, but also among the different datasets.

2.1.3 Time Series Regression Model and Seasonal Adjustment

A time series regression model is used to predict new values based on time series data, which contain successive measurements at different points in time. Its popularity came from eco-

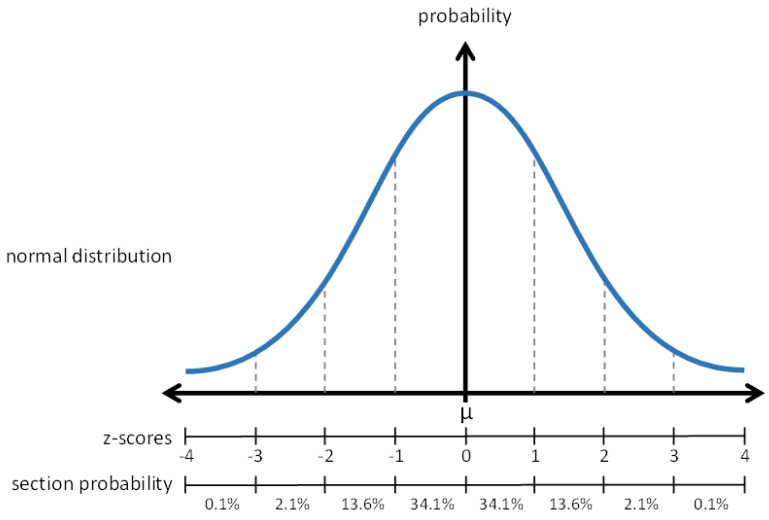


Figure 2.2: Correlation between z-score of a normal curve and its probability distribution.

nomics in the beginning of the 20th century with the growth and systematization of collection and publication of economic information [19]. Despite its age, it is still being used in many areas such as revenue prediction [40] and energy consumption forecasting [15, 46].

A time series regression model can be either additive [19]

$$y_t = T_t + C_t + S_t + I_t \tag{2.3}$$

or multiplicative

$$y_t = T_t \times C_t \times S_t \times I_t \tag{2.4}$$

The trend component (T_t) is a smooth, regular, and long-term statistical series and represents a general growth or decline; the cyclical (C_t) component is a pattern that occurs repeatedly several times during an irregular period; the seasonal component (S_t) is similar to the cyclical component, but the pattern occurs in a well-defined period (*e.g.*, daily, weekly, or monthly); and the irregular component (I_t) is the remainder, which can be related to atemporal factors or simply considered an error.

The cyclical component (C_t) can be analyzed together with the trend component (T_t), also known as the trend-cycle component (TC_t) [19, 20].

Choosing the most appropriate model depends on the data to be analyzed. The additive model is most suitable when the magnitude (the distance between highs and lows) remains relatively constant over time; the multiplicative model is most useful when the magnitude varies with the local average values (the higher the average, the higher the magnitude) [20]. For an economic time series, multiplicative models are commonly used [20].

Seasonal adjustment is a procedure to improve the properties of the parameter estimates for time series regressions [19]. In seasonal adjustment, the trend factor is an estimation of the trend-cycle component (TC_t) and the seasonal index is an estimation of the seasonal component (S_t) based on past observations. The trend factor and seasonal index are used to remove the trend-cycle and seasonal components from a time series, allowing the analysis of data without these components and later adjust the prediction.

The method proposed in this thesis can work with both additive and multiplicative models. It uses seasonal adjustment to remove the different seasonality within domains to approximate their distributions.

2.1.4 Transfer Learning

Transfer learning aims to improve the learning task in a target domain using the knowledge from other domains and learning tasks [34]. It is defined as follows:

Definition (Transfer Learning). *Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and a learning task $\mathcal{T}_T$, transfer learning aims to improve the learning of the target predictive function $r_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$ [34].*

From the definition above, a domain is defined as a pair $\mathcal{D} = \{\mathcal{F}, P(X)\}$, where $\mathcal{F} = \{f_1, .., f_n\}$ is a feature space with n dimensions, f_k is a feature, X is a learning sample such that $X = \{x_1, ..., x_n\} \in \mathcal{F}$ and $P(X)$ is the marginal probability distribution of X. A task is a pair $\mathcal{T} = \{\mathcal{Y}, r(\cdot)\}$, where $\mathcal{Y}$ is the label space and $r(\cdot)$ is the predictive function. From a probabilistic viewpoint, $r(X)$ can also be written as the conditional probability distribution $P(Y|X)$ [34].

Figure 2.3 compares different types of machine learning approaches regarding domains. Figure 2.3a shows how a standard machine learning works, using only a single domain for

each model to learn from separately. Figure 2.3b shows how transfer learning differs from the standard machine learning, using the knowledge extracted from multiple source domains to improve the prediction for a target domain.

Three questions arise when describing transfer learning approaches: "what to transfer?" means what knowledge can be transfered across domains or tasks; "how to transfer?" describes the approach itself; and "when to transfer?" refers to the choice of situations in which transfer learning should be used. Transfer learning approaches can also be grouped into four "what" categories [34]:

(a) Instance-based, where labeled data are selected and reweighted from the source domain to be used in the target domain;

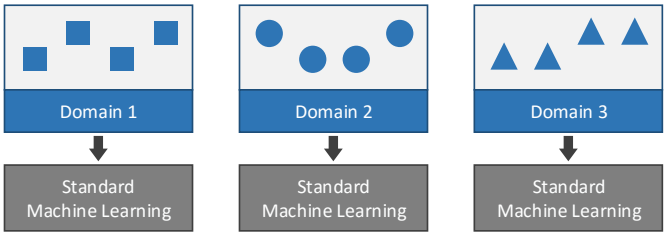
(b) Feature representation-based, where a new feature space is composed to satisfy all the different domains;

(c) Parameter-based, where the parameters used to train the source are used to train the target; and

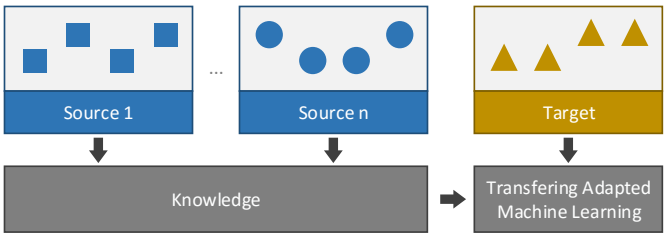
(d) Relational knowledge-based, where a mapping of relational knowledge is built between the source and target domains.

The "how" and "when" strongly depend on the transfer learning approach itself. The method proposed in this thesis is instance-based to add external information to the target and parameter-based for the domain adaptation when the target is not statistically sufficient to adjust itself.

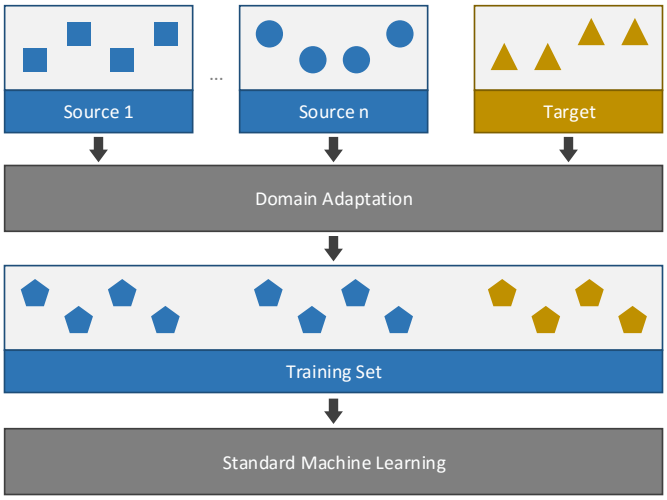
Transfer learning has three main settings [34, 50]: (a) inductive transfer learning, where the target task is different from the source task ($\mathcal{T}_S \neq \mathcal{T}_T$); (b) transductive learning, where the tasks are the same ($\mathcal{T}_S = \mathcal{T}_T$), but the domains are different ($\mathcal{D}_S \neq \mathcal{D}_T$); and (c) unsupervised transfer learning, which is similar to inductive transfer learning for unsupervised learning tasks, where no labeled data are available. Hephaestus works for the transductive learning setting.



(a) Traditional machine learning.



(b) Transfer learning.



(c) Domain adaptation learning.

Figure 2.3: Types of machine learning approaches.

2.1.5 Domain Adaptation

Domain adaptation addresses transductive learning by transforming domains or creating a latent domain that is common to all to reduce the difference between the distributions of source and target domain data [34]. Domain adaptation can be done either before or during the execution of a machine learning algorithm. Figure 2.3c shows how this works by adapting *n* different *source* domains and the *target* domain into a common domain and fusing the datasets into a single *training set*, which then becomes a standard machine learning problem.

Moreover, domain adaptation problems can be divided into two sub-categories according to the difference between source and target domains: (a) the feature spaces between domains are the same ($\mathcal{F}_S = \mathcal{F}_T$) (*e.g.*, transferring energy consumption knowledge from one building to another); and (b) the feature spaces between domains are different ($\mathcal{F}_S \neq \mathcal{F}_T$) (*e.g.*, transferring the knowledge from a Web page in English to another in Portuguese). Most existing studies on transfer learning fall into the first category [50].

Covariate shift and sample selection bias are also important to consider in domain adaptation. Covariate shift is the difference between two domains in which the conditional probabilities from source and target are the same ($P_S(Y|X) = P_T(Y|X)$), but their marginal probabilities are different ($P_S(X) \neq P_T(X)$). Although the interest is in the conditional probabilities, not in the marginal probabilities, this difference is important for misspecified models [43]. Sample selection bias follows the same requirements as covariate shift, but is caused by the exclusion of part of the entire population [49].

This thesis proposes a semi-supervised domain adaptation method in which the feature spaces between domains are the same ($\mathcal{F}_S = \mathcal{F}_T$), but the marginal distributions and conditional distributions between domains are different ($P_S(X) \neq P_T(X)$ and $P_S(Y|X) \neq P_T(Y|X)$). This method works in both pre- and post-processing stages (*i.e.*, before and after the execution of a machine learning algorithm) and can be used with any standard machine learning algorithm.

2.1.6 Service Component Architecture

A service component architecture (SCA) [1] is a modeling specification for composing systems according to the principles of Service-Oriented Architecture (SOA).

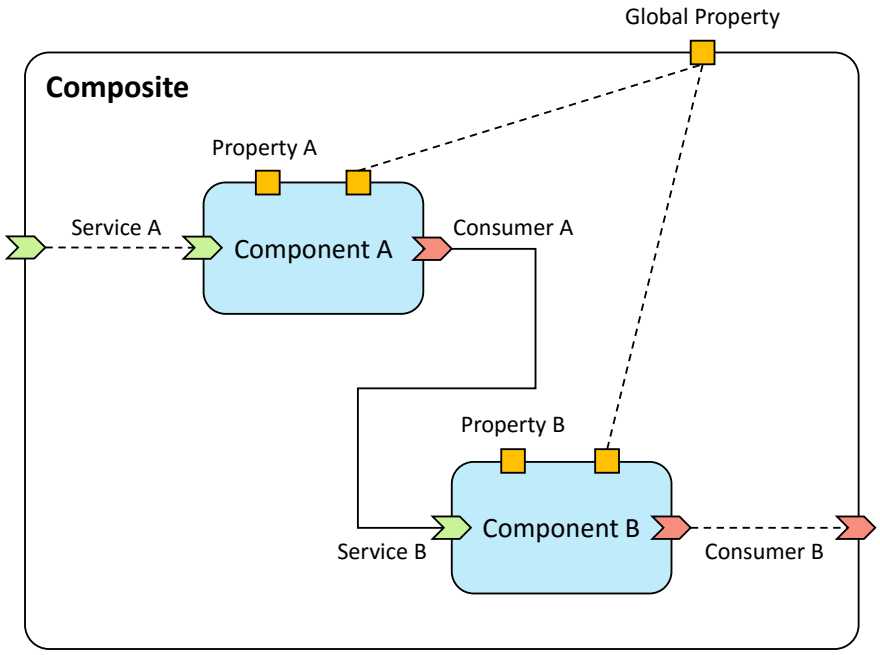


Figure 2.4: SCA artifacts.

SCA separates implementation concerns into four artifacts: (a) *components* implement its business function; (b) *composites* assemble various components together to create business solutions; (c) *services* create an interface for remote access to component and composite functions; and (d) *properties* contain global values for composites and local values for components. Figure 2.4 illustrates the SCA artifacts and their relationship. In a system, composites, services, and their relations with components are defined in a dynamic XML descriptor file.

Because SCA is built on top of SOA, it inherits all SOA's advantages — for example, intrinsic interoperability, inherent reuse, simplified architecture and solutions, and organizational agility [11]. In addition, whereas SOA focuses on building an architecture to design individual components, SCA focuses on assembling multiple components into a composite and facilitating design, implementation, and deployment. SCA systems have been successfully used, for example, in geographic information systems [25] and smart home systems [7] [26].

This research aims to build a platform which is capable of providing various machine learning algorithms to build different predictive models which will run at the same time. Adding a new algorithm must be simple. The system must provide well-defined APIs which can be

remotely accessed over the Web by any external system. SCA provides enough artifacts to meet these requirements.

2.2 Related Work

This section discusses previous work from the academy and industry for transfer learning, machine learning as a service and knowledge marketplaces. In addition, the research gaps are identified as well as it is explained how Agora addresses them.

2.2.1 Transfer Learning

In this section, existing methods for transfer learning and domain adaptation are discussed. Transfer learning can also be found in the literature under the term *cross-domain* plus other terms such as learning [51], prediction [27], data [29] and data fusion [50]. Sometimes the term *domain* is replaced by the domain category (*e.g.*, cross-company [29]). Covariate shift [43] and sample selection bias [49] are also related to domain adaptation.

Transfer learning has been recently used in many real-world problems from different areas: in software engineering, it has addressed cross-company software defect classification [27, 31] and cross-company software effort estimation [29]; in voice processing, it has improved mispronunciation detection [17]; in image processing, it has dealt with visual recognition [13, 16, 24, 30, 37, 47, 51]; and in natural language processing (NLP), it has addressed sentiment analysis [9, 10, 18, 24, 33].

Although most transfer learning studies in the literature deal with classification tasks [13, 16–18, 24, 27, 30, 31, 37, 51], regression tasks are highly restricted to specific areas [9, 29, 33, 47].

Many domain adaptation methods are gaining attention because of their ability to work with standard machine learning [10, 29, 33], whereas others work only for a small set of algorithms [9, 18, 24, 27].

In general, instance-based methods address the problem by importance reweighting [34] — where the general idea is to increase the weights of data in the source domain that are close to data in the target domain — especially for covariate shift [43] and sample bias selection

problems [49]. The definition of reweighting varies for each method, but a common definition consists of weighting a data point x of the training set as an estimate of the ratio $\omega(x) = P(x)/Q(x)$ where P is the target distribution and Q is the source distribution [9].

As for the feature representation-based methods, in general, they deal with the problem by extracting underlying features that are common to all domains [10, 18, 33].

The procedure to reduce the distance between domains can be explicit or implicit. An implicit procedure processes the data in such way that it consequently reduces the distance. On the other hand, an explicit procedure transforms distance reduction into an optimization problem by directly attempting to minimize the distance. In the next two subsections, various methods from the literature are grouped into implicit and explicit distance reduction methods, discussed and compared with Hephaestus, the method proposed in this thesis.

Explicit distance minimization

Hu *et al.* [18] proposed a feature representation-based method called multi-bridge transfer learning (MBTL). MBTL is formulated as an optimization problem based on nonnegative matrix tri-factorization (NMTF), which has already been widely used for text classification. MBTL constructs multiple different latent feature spaces using a clustering algorithm with different parameters. Simultaneously, it learns the marginal and conditional distributions in the latent spaces to construct bridges across domains. MBTL uses the latent factors to reduce the distribution divergences in the various latent feature spaces. However, MBTL relies strictly on NMTF and cannot work with other standard machine learning algorithms. Moreover, MBTL has been used only for text classification, and not for regression tasks.

Along the same line, nonetheless promising to work with regression tasks, Pan *et al.* [33] proposed a new feature-based domain adaptation method called transfer component analysis (TCA). TCA attempts to find a suitable feature representation across domains by learning a set of common transfer components (or latent variables) that underlie domains. This is done by explicitly minimizing the maximum mean discrepancy (MMD), which compares the distributions based on the corresponding reproducing kernel Hilbert space (RKHS) distance, while preserving data variance. The latent space projected by TCA can be used with standard machine learning algorithms for classification, regression, or clustering.

Similarly, Cortes and Mohri [9] introduced the notion of discrepancy — a distance calculation between distributions generalized with arbitrary loss functions — and introduced the discrepancy minimization (DM) algorithm, which attempts to minimize the discrepancy explicitly for kernel-based regularization algorithms (*e.g.*, support vector machines, support vector regression, and kernel ridge regression). Despite previous studies that claimed that their solution worked for regression tasks and demonstrated this with case studies (such as TCA), Cortes and Mohri adapted a classification problem into regression for their case study on real-world multi-domain sentiment analysis. The restriction of the limited number of algorithms with which DM is compatible makes it disadvantageous compared to others that are compatible with any standard marchine learning algorithm.

However, all the domain adaptation methods based on explicit distance minimization that were reviewed here (MBTL, TCA, and DM) require extensive knowledge of statistics and machine learning, which can make them complex. Moreover, all of the three rely on optimization algorithms, meaning that they are computationally expensive. Furthermore, they are not available as code libraries or components for any popular machine learning tool or programming language, which makes it harder for MBTL, TCA, and DM to be implemented and used.

Unlike MBTL and DM, we designed Hephaestus to work with any standard machine learning algorithm. Its ease of understanding and its compatibility with standard machine learning algorithms enables Hephaestus to be applied to a variety of problems, without the need for expert skills to implement it.

Implicit Distance Reduction

Following the tradition of instance reweighting for instance-based methods, Ma *et al.* [27] proposed the Transfer Naive Bayes (TNB) method. TNB first calculates the degree of similarity between each training sample and the test set by checking whether the attribute values are within the target domain boundaries (between the minimum and maximum values of each attribute). The weight for each training sample is calculated using data gravitation between the training sample and the test set using the degree of similarity. A Naive Bayes classifier is then run over the weighted training samples. However, this method, along with the most domain adaptation reweighting methods, tends to diminish the importance of data that are outside the

boundaries of the target domain, which could be used to predict a new situation. For example, if a model tries to predict a situation that the target has not faced so far (*i.e.*, that lies outside the target boundaries), obtaining this situation from a source domain could be necessary. Hephaestus does not weight data samples by their distance between target samples and therefore does not ignore samples outside the target boundaries. Instead, Hephaestus addresses domain adaptation by rescaling the feature values to achieve a better representation of the source data in the target domain.

Minku and Yao [29] created a relational knowledge-based algorithm called dynamic cross-company mapped model learning (Dycom) to estimate software effort (SEE) within a company (WC) using cross-company (CC) data. Dycon is able to work in an online scenario, where there is no need to retrain the whole model after a new training data has arrived. Minku and Yao assumed that the relationship between the two companies follows the equation:

$$f_A(x) = g_{BA}(f_B(x)) \tag{2.5}$$

where f_A is the true estimate for company A, f_B is the true estimate for company B and g_{BA} is the function that maps the effort from context B to context A. First, the CC data are split into M different clusters (*i.e.*, distinct sets of data points grouped by similarity), which are used to build M different models. Dycom assumes that

$$g_{B_iA}(f_{Bi}(x)) = f_{Bi}(x) \times b_i \tag{2.6}$$

where each B_i is a CC model and b_i is the factor calculated depending on the amount of training data received. For each new WC labeled sample that arrives, M estimations are performed for the CC models. Once Dycom gets each of the M estimates and the WC measurement, it learns the mapping functions between the WC measurement and the M CC models and reweights the $M + 1$ estimates (including WC) to return the right estimate. Dycom can be implemented using any standard machine learning algorithm. Although Dycom does not reweight the instance itself, it reweights instances indirectly by reweighting models that are built using clusters. This can lead to the same issue mentioned earlier, that of excluding samples that are out of bounds from the target domain.

Daumé [10] proposed a domain adaptation method called Frustratingly Easy Domain Adaptation (FE), which is a feature representation-based method. FE creates an augmented latent space using two simple kernel-mapping functions:

$$\Phi^S(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{x}, \boldsymbol{0} \rangle \qquad \Phi^T(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{0}, \boldsymbol{x} \rangle \tag{2.7}$$

to map the feature space into three spaces, representing the general, source and target spaces. The source mapping Φ^S keeps only the general and source spaces, whereas the target mapping Φ^T keeps only the general and target spaces. This method is suitable for natural language programming (NLP). To make it easier to use, Daumé also provided the method as a segment of Pearl code. To work properly, the augmented features must be weighted. For example, for text classification, if the meaning obtained for a single word were the same for all domains, the feature-augmented weight vector would be represented as $\langle \boldsymbol{1}, \boldsymbol{0}, \boldsymbol{0} \rangle$; otherwise it would be $\langle \boldsymbol{0}, \boldsymbol{1}, \boldsymbol{0} \rangle$ if the meaning were specifically from the source domain or $\langle \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{1} \rangle$ if it were from the target domain.

The simplicity of Daumé's FE inspired other works in different areas to use, adapt, improve or compare with it. For example, Yamada *et al.* [22] used FE for sentiment classification between restaurants and laptop domains. Kiritchenko *et al.* [47] used FE to estimate 3D full-body and head poses (a regression task).

However, Daumé's approach would not work properly for cross-domain regression problems where zero does not necessarily represent the absence of a feature. For example, consider a cross-building energy consumption problem in which energy consumption depends on external temperature. In this situation, 0 does not represent the absence of temperature, but a temperature of 0°C. Therefore, it would not be able to use the kernel-mapping functions $\Phi^S(\boldsymbol{x})$ and $\Phi^T(\boldsymbol{x})$. In addition, the training-set size increases quadratically because each new domain not only adds more data points but also creates new features, which makes weighting less accurate and training slower. Hephaestus can deal with any continuous features (*e.g.* temperature) where zero does not necessarily mean the absence of that feature. In addition, Hephaestus works directly on top of the original features, without needing to create new ones.

Furthermore, none of the methods reviewed above work when the conditional probabilities

are not the same ($P_S(Y|X) \neq P_T(Y|X)$). Hephaestus can rescale the label Y to reduce the distance between the conditional probabilities ($P_S(\Phi(Y)|X) = P_T(\Phi(Y)|X)$).

As a solution between implicit and explicit distance minimization, Li *et al.* [24] proposed heterogeneous domain adaptation (HDA), which adapts Daumé's FE by introducing two projection matrices P and Q for the general subspace of the mapping functions to improve the alignment between domains, as shown in the following equation:

$$\Phi^S(\boldsymbol{x}) = \langle \boldsymbol{P}\boldsymbol{x}, \boldsymbol{x}, \boldsymbol{0} \rangle \qquad \Phi^T(\boldsymbol{x}) = \langle \boldsymbol{Q}\boldsymbol{x}, \boldsymbol{0}, \boldsymbol{x} \rangle \tag{2.8}$$

HDA learns P and Q and the weight vector by minimizing the structural risk functional of support vector machines (SVM). Li *et al.* also claim that HDA can work with support vector regression (SVR). HDA works for classification tasks such as object recognition, multilingual text categorization and cross-lingual sentiment classification. However, it relies on SVM and SVR and also suffers from the same issues as FE.

Because Hephaestus acts directly on top of feature and label values, and not on the distance itself, it is considered to be a method based on implicit distance reduction.

Transfer Learning for Seasonal Adjustment

Although some studies mentioned at the beginning of this section naturally deal with the time component (*e.g.*, voice processing), none of them considered trends and seasonality (*e.g.*, cross-building energy consumption). Microsoft SQL Server (a relational database management system) can use more than one source to create time series predictions, an ability that Microsoft calls *cross prediction* [2]. SQL Server embeds two algorithms: an autoregressive tree model with cross-predictions (ARTXP) [28] for short term predictions and an autoregressive integrated moving average (ARIMA) for long term predictions. Cross-predictions are possible only for ARTXP models, and no information is publicly available about how it operates. However, it does not support features other than time.

Hephaestus deals with both time series and multi-feature regression together. It separates out the time component from the various domains, adapts the remainder component for all domains into a same domain and uses any standard machine learning algorithm to create a

predictive model.

2.2.2 Machine Learning as a Service

The increasing demand for machine learning is leveraging the emergence of new solutions. In this section, various machine learning platforms are reviewed.

PredictionIO [8] was launched in 2013. It is an open-source platform with an architecture that integrates multiple machine learning processes into a distributed and horizontally scalable system based on Hadoop. In addition, PredictionIO provides access through web APIs and graphical user interface (GUI).

Baldominos *et al.* [5] also proposed a platform built on top of Hadoop. Its implementation was capable of handling up to 30 requests at one time while maintaining a response time of less than one second.

OpenCPU [32] is another open-source platform, launched in 2014, that creates a Web API for R [41], a popular statistical analysis software environment. However, because it is practically a middleware for accessing R functions, it does not take into account many non-functional requirements like scalability and performance.

In the industry context, the giants Google, Microsoft, and Amazon have been releasing their own proprietary platforms. Google released its Prediction API[1] in 2014, a platform as a service (PaaS) for developers that need a third party machine learning service. Google Prediction API claims it chooses the best algorithm depending on the data provided by the user, without being able to be selected or customized, nor letting the user know which algorithm is being used. Also in 2014, Microsoft launched Azure Machine Learning[2], a software as a service (SaaS) that provides a web user interface to facilitate basic machine learning tasks for the user. In 2015, Amazon released AWS Machine Learning[3], another PaaS for machine learning, however restricted to the logistic regression only. BigML[4], a startup company founded in 2011, is another (SaaS) similar to Microsoft's. Their sales can prove that the demand exists. Unfortunately, the designs and implementation specifications of these products are not publicly

[1] `http://cloud.google.com/prediction`
[2] `http://azure.microsoft.com/en-us/services/machine-learning`
[3] `http://aws.amazon.com/pt/machine-learning`
[4] `http://bigml.com`

available.

PredictionIO, OpenCPU, and Baldominos' platforms are built on top of a specific analytical tools and suffer from their restrictions. This means less flexibility for adding new machine learning algorithms, for data storage, and for deployment. Although Hadoop and R are open-source projects, it is not a trivial challenge to adapt them to a new approach. The same happens with the industry players and their proprietary solutions when external developers cannot have access to the code to add new algorithms.

The platform proposed in this thesis focuses on predictive modeling. As an architecture based on SCA specifications, the architecture facilitates the addition of new algorithms, its improvement, and its adaptation to other machine learning applications. Even the revised platforms mentioned above can be attached to the proposed architecture to build prediction models.

2.2.3 Knowledge Sharing and Marketplaces

Stewart and Ruckdeschel [44] introduced the idea of intellectual capital: knowledge that brings wealth to the organization. For traditional accounting, intellectual capital does not fit the definition of an asset in general, which is required to be tangible and to have a solid known cost. Despite this, markets can establish the value of a knowledge asset. Knowledge assets can be divided into three categories: human capital, made up of the competencies of people; customer capital, including brand, reputation and relationships with customers; and structural (or organizational) capital such as patents, methods and models. Raw data cannot be considered a intellectual capital by themselves because they cannot bring wealth to the organization without being processed and understood. However, when considering that useful insights can be extracted from the data, they become an important knowledge asset. Although structural capital has more strategic value than market value for the organization, it is shareable and can be sold in the market. For example, the "lessons learned" of a company are important for its future projects, but can be of great value to other companies as well.

The Internet response to the demand for knowledge is the variety of marketplaces for knowledge in general. For example, on crowd-sourced question and answer (Q&A) Web-

sites such as Yahoo! Answers[5] (2005), Stackoverflow[6] (2008) and Quora[7] (2009), users post questions that others then answer.

Sometimes, data are already shared, but in the raw state and having almost no value. Governments, for example, are huge data generators. The Gov 2.0 movement has created a new level of data transparency to attract developers to make a civic contribution to society, creating a great opportunity to build a knowledge marketplace with government data. However, governments are unable to feed developers with well-defined structured data. Qanbari *et al.* [39] proposed an architecture for Open Government Data as a Service (GoDaaS), in which developers could have easy access to government data and share their applications with the entire population. GoDaaS is composed of three layers: the data infrastructure, to provide public and structured data; the development platform, where authorized developers can deploy services to retrieve information from the data infrastructure; and the app store, where citizens have access to government data thought mobile applications. Despite its focus on government data, GoDaaS has a very wide scope of potential applications, but does not consider transfer learning. In contrast, Agora focus much more on private data and predictive models (*e.g.*, retail sales prediction, energy consumption forecasts).

As a specific example of a marketplace for data analytics, Park *et al.* [35] created an architecture for a Web-based collaborative Big Data analytics where users can share data, algorithms, and services. This platform consists of two different Web portals. The Web service portal facilitates collaboration between users by means of a multi-tenancy architecture. Users can communicate and share information using the platform for efficient and rapid service development, by exploring the algorithms, data and services available in its catalog. The analytics portal focuses on the productivity of data analytics, enabling users to visualize and explore data as well as to monitor process and cluster resources. What differentiate this platform from Agora are the knowledge assets being shared. Although the users in this platform share data and algorithms to further development of new services, users in Agora can share data and models to facilitate creation of predictive models for other users with small datasets.

Parreiras *et al.* [36] presented a framework for a marketplace to connect software artifacts

[5] http://answers.yahoo.com
[6] http://stackoverflow.com
[7] http://www.quora.com

within the same project and across different projects as well. This marketplace enables suppliers (*i.e.*, software producers) to store and share artifacts such as bug reports, versions and source code. Built upon linked open data (LOD) techniques and semantic technologies, it offers easy access to related software data in the marketplace. In addition, this framework facilitates development of services for analytics and visualization of software data, enabling consumers to find, understand and reuse various pieces of open-source software. However, the focus of this framework is to describe how artifacts are related with each other and how users interact with them. It is not clear what type of statistics and data analytics services this framework can handle.

From an industry perspective, Microsoft offers the Azure Marketplace[8], where users can find a variety of free or paid applications and components to attach to projects created on the Azure Platform. For example, for projects in Azure Machine Learning, users can find everything from a single anomaly detection component to attach to their models to an entire predictive model already built and just waiting for data input. However, Azure Marketplace and Azure Machine Learning do not provide any method related with transfer learning to users with small datasets.

2.3 Summary

This chapter has introduced terms and concepts related to the various topics to assist in understanding Agora and to provide a background in machine learning, transfer learning and service component architecture.

Various related studies from the literature on transfer learning, machine learning as a service and knowledge sharing and marketplaces were discussed. The research gaps were identified and can be summarized as the following three topics:

- Lack of transfer learning methods for regression tasks considering time series.
- No studies in transfer learning considering both domains and learning tasks different ($\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S \neq \mathcal{T}_T$).

[8] `http://azure.microsoft.com/en-us/marketplace`

- No studies about flexible and scalable machine learning as a service.
- No studies about knowledge platform for transfer learning.

Moreover, the related work and research gaps were compared to the methods from Agora, the platform presented in this thesis. These methods are discussed in more details in the following chapters.

Chapter 3

Agora: a Knowledge Marketplace for Machine Learning

The Agora was the central square located in the city-states of Ancient Greece, in which people came together and athletics events, art performances and political gatherings took place. Merchants took advantage of the crowded space to sell their goods, turning it into a marketplace. Famous philosophers such as Socrates and Plato were frequently present in the Agora of Athens, where they could question the other visitors and spread their knowledge publicly.

Many years have passed and Agora is the name of the knowledge marketplace for machine learning and transfer learning proposed in this thesis. Knowledge can be shared between users to facilitate learning. Knowledge, in Agora, is the understanding about how to extract information from observations (*i.e.*, datasets) using certain learning technique (*i.e.*, machine learning algorithms). When users share their knowledge in the marketplace, it means both their data and model specifications are being provided to help other users.

3.1 Overview

Agora visitors (or users) can be either providers or consumers. Providers play the mixed role of philosophers and merchants: they control the knowledge that can be "sold" as a product in the marketplace. Providers are more observant and have more data samples collected. In addition, providers have more experience in data analytics and understand how to extract valuable

information from data. They can determine what features should be considered, what is the best algorithm for the task and what are the best parameter settings.

Consumers are the visitors looking for knowledge to gain insights. They are not as experienced as providers with regarding to data analytics. In addition, consumers are not necessarily observant or attentive to details, meaning that they do not have enough data samples. In summary, customers have enough information to ask questions and know what they want, but they do not have enough samples and knowledge to create an accurate model. They want insights immediately, without wasting time understanding complex algorithms or collecting more data. This is where they can use the knowledge learned by the providers.

Figure 3.1 outlines how this process works. Each tent represents a provider selling knowledge: data samples (represented by geometric shapes) and model specifications (represented by gears). Consumers must find providers that are selling the knowledge that they need. For example, consumer A should ask for knowledge from provider A because both have square samples, and consumer B should ask provider B because both have circular samples. Once consumers have enough data and models from providers, they can extract insights from their original data.

The main idea of Agora is to connect providers and consumers to facilitate knowledge sharing. Agora stores the datasets and the machine learning model specifications provided by providers. Consumers can efficiently create accurate models using knowledge shared by providers, within the context of the built-in domain adaptation method and the machine learning platform provided by Agora.

As an example, a consumer who is a school building manager needs to predict the school's energy consumption. However, this consumer does not understand what machine learning is and has available only a small dataset from the last month containing only the day and the associated energy consumption. This consumer has no idea what other factors (*e.g.*, weather attributes) could be related to energy consumption. In Agora, this consumer could efficiently create a machine learning model using knowledge from a specific provider. This provider is also a school building manager, but one with more experience who has many years of energy consumption data, knows which features can be related to energy consumption and has already created a machine learning model to predict energy consumption. All the consumer needs to

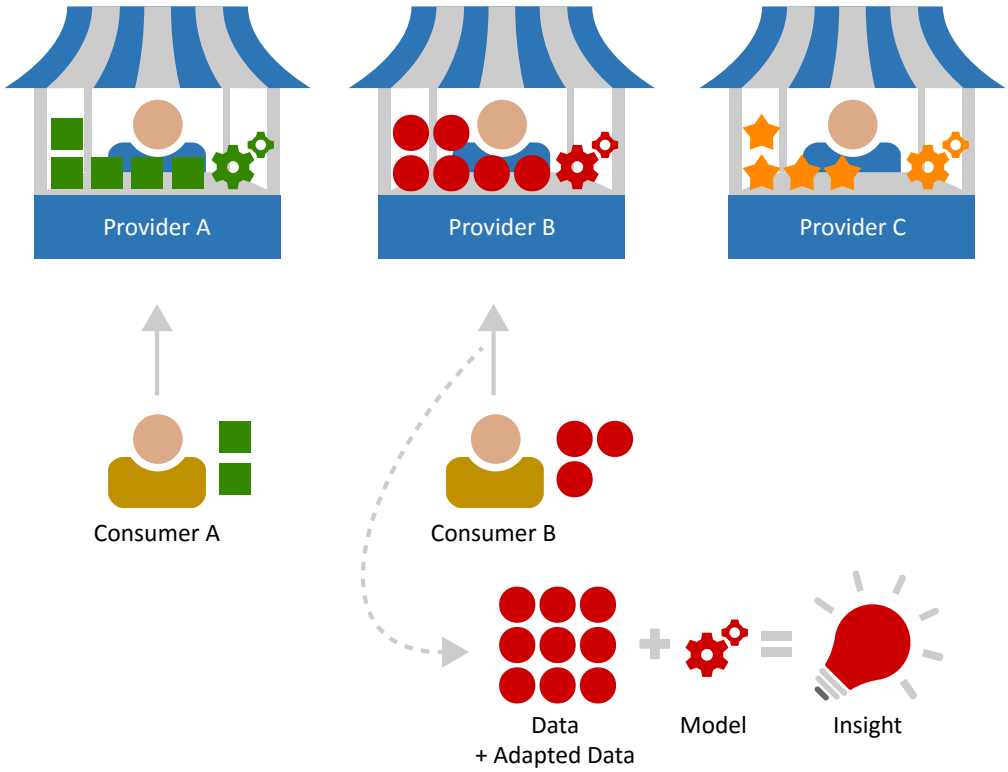


Figure 3.1: High-level diagram showing how Agora should work.

do is to upload his dataset to Agora, specify what exactly the dataset is and Agora will do the rest. At the end, the consumer will have the energy consumption prediction.

3.2 Architecture

The general architecture of Agora is illustrated in Figure 3.2. First of all, *providers* are the participants that control the available knowledge, which is a combination of information (*source domain data* and *global data*) and learning methods (*model specifications*). The *source domain data* contain data samples of a specific domain and are described by *metadata*. The *model specifications* contain details about how to extract insights from the *source domain data*. The *global data* can be shared between different models and domains. All this information is stored in the *database*.

On the other side are the *consumers*, who want to make use of the *providers*' knowledge, but do not own it. However, to retrieve knowledge, they must provide a description of their

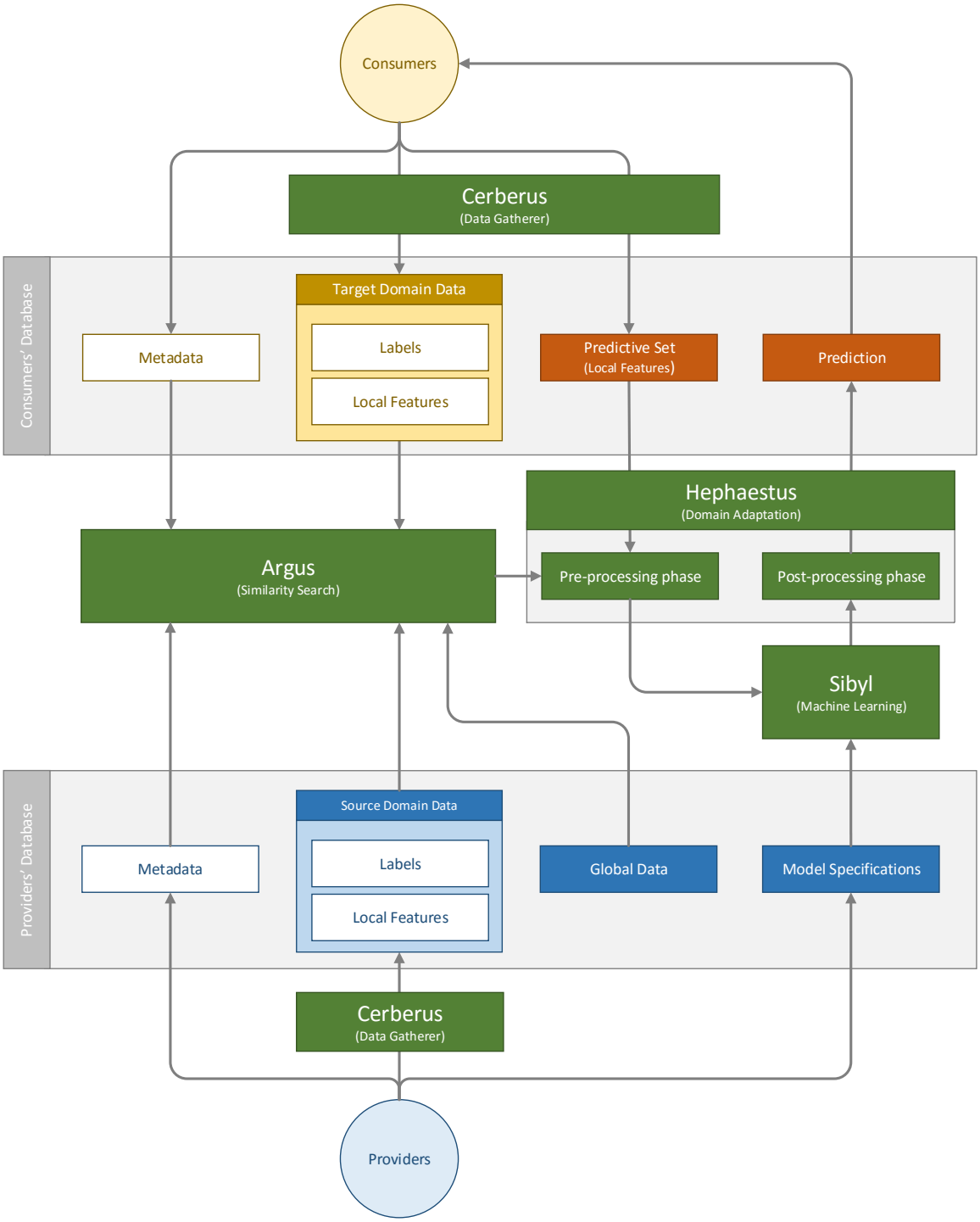


Figure 3.2: Overview of Agora's architecture.

context by providing their *target domain data*, the corresponding *metadata*, and information about the future in the form of a *predictive set*. Once the *prediction* is made, it is returned to the *consumer*.

To keep Agora clean of dirty or noisy data (e.g., zeros, incomplete data, empty fields), all data must be checked by the *Cerberus* processing component, which filters the data before they enter Agora.

In between relies the other processing components *Argus*, *Hephaestus* and *Sibyl*, which enable Agora to work properly. Once *Argus* receives *target domain data*, it can find the most similar *source domain data* in the *database* and forward them to *Hephaestus*. *Hephaestus* adapts all its input domains to become one, enabling them to be analyzed together by *Sibyl*. *Sibyl* can create predictions using the knowledge shared by the providers and returns the results back to *Hephaestus*, which makes the final adjustments to the original domain and return the results to the *consumers*.

3.2.1 Actors

Providers

Providers possess large and almost complete datasets and have knowledge of how to create efficient machine learning models. Agora enables *providers* to upload and store their data (*i.e.*, *source domain data*) in Agora's *database*. In addition, *providers* determine which features may be considered relevant to build the model. Furthermore, *providers* store *model specifications* in the *database* to specify the best way to create a machine learning model (to be built by others) using their data.

Consumers

Consumers have insufficient, incomplete, or small datasets (*i.e.*, the *target domain data*), which are sent to Agora in an attempt to create an accurate prediction. In addition, *consumers* are not required to understand which features should be considered to build a model as well as the best algorithms and parameters. This is the job of the *providers*.

Agora also facilitates communication between actors to help *consumers* find the right *provider* and knowledge they need.

3.2.2 Providers' Database

The *database* contains all the knowledge shared by the *providers*. Here, knowledge is determined by the combination of observations of events that happened locally (*source domain data*), observations of global events (*global data*) and details of how to learn from the data (*model specifications*).

Providers' database and *consumers' database* are only the names given to group the data provided by both actors. They do not necessarily mean they use different databases or tables. This thesis leaves the design of the database out of the scope due to the time limit and focuses on the machine learning aspects of Agora. For evaluating the rest of Agora, the data was fed directly to each component evaluated, without requiring access to any database.

Source Domain Data

The *source domain data* contains observations of events that happened locally, which means that they are very domain-specific. The *source domain data* are made up of *labels*, which is the target factor to be analyzed (*e.g.* predicted, or classified), and *local features*, which consist of local measurements that support analysis of the *labels*.

The *source domain data* must contain enough data samples to provide accurate predictions. If even the *provider* cannot predict with enough accuracy, the *consumer* will not be able to predict any better.

Metadata

The *metadata* describes the properties of the *source domain data*. They identify the *labels* and *local features*, as well as their data types (*e.g.*, integer, floating-point, character, and string), their domain in the case of numbers (*e.g.*, natural numbers, integers, and real numbers), their range (*e.g.* minimum and maximum), and other properties. These are useful for Agora to understand the nature of the datasets when comparing them.

In addition, *metadata* contain additional information to help Agora identify similar datasets. For example, because the *metadata* retain the location where the data were collected, Agora can analyze two datasets for the same location because the weather and cultural values within the same location are almost the same. The same can be stated when the *metadata* retain the type of entity where the data were collected (*e.g.*, from a school, house or office).

Model Specification

The *model specification* contains information about how to create a machine learning model for a specific task. When a *consumer* needs to create a specific model but has no idea how to do it, a *provider* with a similar task provides the details of which algorithms, normalization methods, and additional attributes should be used to build the model. With the *model specification*, Agora combines the *consumer*'s and *provider*'s data with the *model specification* to build and run the model.

Global Data

Similarly to *local features*, *global data* act like features, but can be shared between different models and domains. If a *provider* knows that a specific machine learning model depends on a specific feature from *global data*, *consumers*, even if they have not measured these features, can still retrieve a prediction because Agora has the data available. In this way, *consumers* can focus only on their specific measurements (*i.e.*, *labels* and *local features*).

3.2.3 Consumers' Database

Target Domain Data

The *target domain data* have the same structure as the *source domain data*. The difference is that the *target domain data* do not contain enough samples to create high-accuracy models and need the support of additional data to improve machine learning models. Moreover, the *source domain data* and the *target domain data* are in different domains.

Metadata

The *consumers*' *metadata* are similar to the *providers*' *metadata*, the only difference being that they describe the *target domain data*. By having *metadata* from both *providers* and *consumers*, Agora can analyze and identify similar datasets between domains.

Predictive Set

A *predictive set* contains information about what can happen in the future. It contains predictable values for *local features*, but not for *labels*, because these are the goal of the machine learning model.

Prediction

A *prediction* is the result of a machine learning model built by Agora. It estimates *label* values depending on the values of the *predictive set*.

3.2.4 Processing Components

The processing components are responsible for processing the data from *providers* and *consumers* to return the prediction to the *consumers*.

Cerberus: Data Gatherer

Cerberus was a dog with many heads from Greek Mythology. He was responsible for guarding the gates of Hades and not letting the dead return to the living world.

In Agora, *Cerberus* is the gate keeper responsible for receiving data and pre-processing them before letting them enter Agora. Figure 3.3 shows how *Cerberus* is designed using SCA notation. The two components are arranged in a pipeline and can be described as follows:

- The *merger* component merges all received data (single data points or batches) from different data sources (*e.g.*, sensors or databases) for a single *consumer*. Datasets with different schema are joined into a single multicolumn schema by related attributes (*e.g.*,

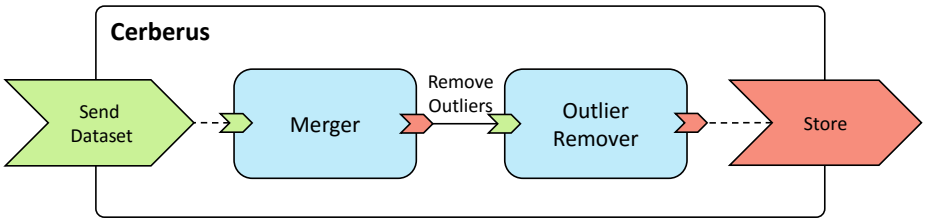


Figure 3.3: SCA composite of Cerberus.

time-stamp for time-series data, categories, identifiers, etc). When finished, the *merger* forwards the data to the *outliers remover* component.

- The *outliers remover* component removes outliers (*e.g.*, missing values, zeros, extremely high values, etc.). Outliers depends on the predictive model and must be specified by *providers* inside *model specifications*.

The *send dataset* service enables users to submit their datasets to Agora via *Cerberus*, which requests databases to store the new submitted data.

Argus: Similarity Search

Argus, in ancient Greek mythology, was a 100-eyed giant, capable of observing everything that was happening around him. This name has invaded modern pop culture, as in Harry Potter, for example, where Argus Filch was the Hogwarts School vigilante who knew all the school corridors to catch students breaking rules.

As the state of the art, *Argus* should be able to perform data similarity search by comparing the distributions of the source and target datasets to find the most similar source domain datasets to create accurate machine learning models. However, the details of *Argus* are omitted from this work due to the time limit of this thesis, and selection can be simply done by manually choosing the best dataset candidates.

Hephaestus: Domain Adaptation

Hephaestus comes from ancient Greek mythology as well and is the name of the god of craftsmen, sculptors, artisans, and blacksmiths. *Hephaestus* also has his popularity in pop culture

with some appearances such as in the Disney film Fantasia in 1940 and the video game God of War III[1]in 2010. Today, his crafting abilities make him a perfect candidate to work on transforming data.

Here, *Hephaestus* is a novel domain adaptation method for multi-feature regression with seasonal adjustment. This method reduces the distance between different domains to improve prediction of a target domain. For example, for a new building, it can make accurate energy consumption predictions using knowledge from other building energy consumption measurements over a much longer time.

Hephaestus is designed to work in the pre- and post-processing phases of standard machine learning, acting directly on top of features and label values, without the need to modify the machine learning algorithm itself, making it possible to use any standard regression algorithm. The pre-processing phase performs domain adaptation on top of the various datasets, whereas the post-processing phase adjusts the predicted values to the target domain.

Chapter 4 explains in substantial detail how *Hephaestus* works and presents a demonstration using a case study involving cross-building energy consumption prediction.

Sibyl: Machine Learning as a Service

In ancient Greek mythology, *Sibyls* were women that were believed to have oracle powers and the ability to predict the future. In the Japanese animation Psycho Pass[2], *Sibyl* was a complex system capable of analyzing the criminal tendencies of Japan's entire population using data from the whole country.

Sibyl is a platform as a service for machine learning, capable of running multiple machine learning models at the same time. *Sibyl* makes it possible to gather data from multiple sources and build multiple machine learning models using different algorithms.

The details of *Sibyl* are presented in Chapter 5, as well as a case study in energy consumption prediction.

[1]More information about God of War III in: `http://en.wikipedia.org/wiki/God_of_War_III`
[2]More information about Psycho Pass in: `http://en.wikipedia.org/wiki/Psycho-Pass`

3.3 Summary

This chapter has provided an introduction of Agora by giving an overview of its architecture, presenting its main components, and explaining the relationship between them. Two of the processing components, Hephaestus and Sybil, are detailed in Chapters 4 and 5 respectively.

Chapter 4

Hephaestus: a Domain Adaptation Method for Regression

This chapter gives the details of Hephaestus[1], a novel domain adaptation method for multi-feature regression with seasonal adjustment. It can extract knowledge from additional datasets in different domains to improve prediction for a target dataset. For example, for a new building, it can make accurate energy consumption predictions using knowledge from measurements of energy consumption in other buildings over a much longer time frame. Hephaestus is designed to work in the pre- and post-processing phases of standard machine learning, acting directly on top of feature and labels values, without the need to modify the machine learning algorithm itself, making it possible to use any standard regression algorithm.

4.1 Method

Figure 4.1 provides an overview of Hephaestus. The inputs are: (a) the *target*, which represents the target dataset and which contains past information from the target subject to be predicted; (b) *sources 1 .. n*, which represents additional datasets that will be used to improve the target prediction; and (c) the *predictive set*, which contains unlabeled data to be predicted.

[1]This research has been submitted as a journal paper to Elsevier Applied Soft Computing as "*Hephaestus: Domain Adaptation for Multi-Feature Regression with Seasonal Adjustment*" in July 2016, co-authored by Katarina Grolinger, Hany F. ElYamany, and Miriam A.M. Capretz

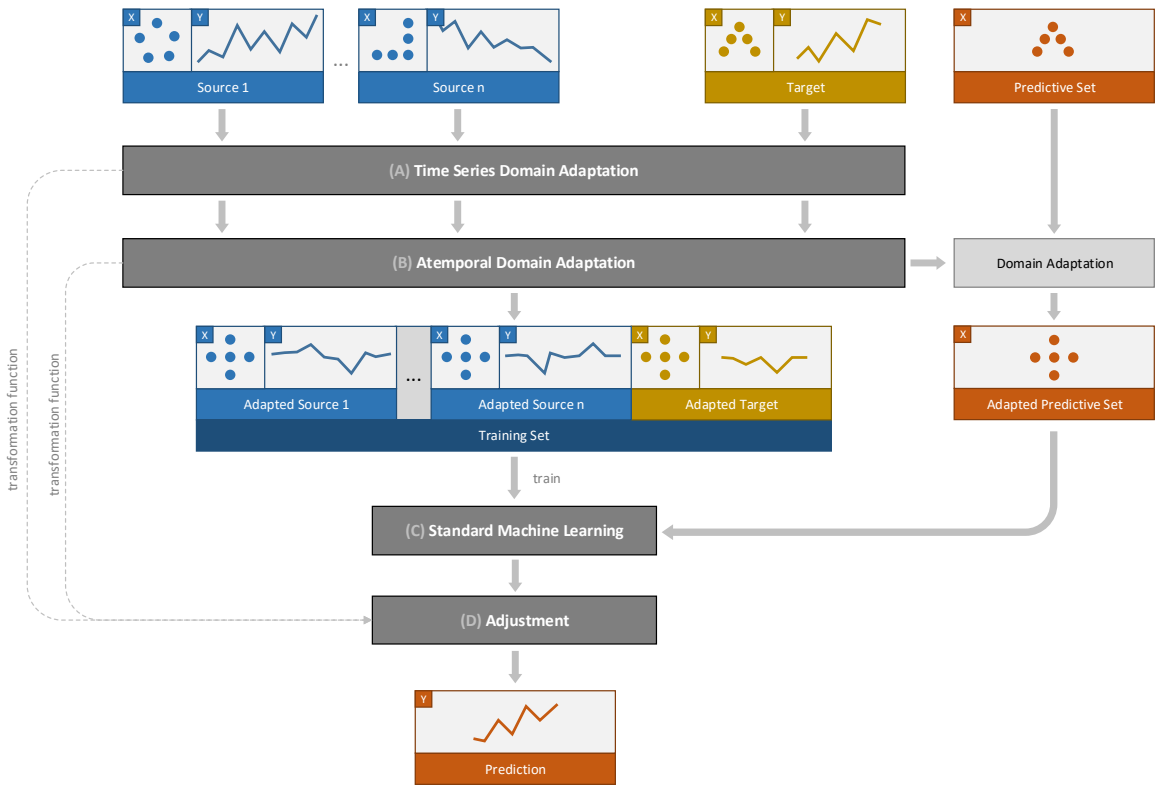


Figure 4.1: Overview of Hephaestus method.

Hephaestus consists of four main phases: (A) *time series domain adaptation*, in which the time effects (*e.g.* seasonality) for all *source* and *target* domains are analyzed, transferred to the target (if needed) and have the trend and seasonality removed; (B) *atemporal domain adaptation*, in which the domains of time-independent labels and features (those that do not define time that were removed by the first phase and do not depend on it) are adapted; (C) *standard machine learning*, which uses any standard algorithm to train the predictive model and generate a prediction using the *predictive set*; and (D) the *adjustment*, in which the prediction is adjusted using the factors calculated in the *time series* and *atemporal* domain adaptation phases.

Hephaestus can be considered as both a parameter and an instance transfer method. It transfers the time factors and normalization parameters (if the target dataset does not have enough data to calculate them) and transfers instances containing atemporal features only to train the predictive model.

In the next subsections, these four phases are discussed.

4.1.1 Time Series Domain Adaptation

Different datasets can have different time profiles, with similar behavior for atemporal features. Hence, it is important to remove trend and seasonal effects before analyzing and transferring atemporal feature correlations. For example, a building can have an energy consumption profile that follows a weekly pattern with peaks and depressions defined by the day of the week. Each building can have a different profile, making it difficult to transfer knowledge of atemporal features from one to another, unless the time effects (*e.g.*, seasonality and trend) are removed.

This phase has two objectives: (a) removing the effects of time from the dataset, and (b) transferring (if needed) time series knowledge from the sources to the target. Figure 4.2 illustrates this phase.

As input, the *time series domain adaptation* phase receives the n raw source datasets and the target dataset. As output, the labels Y for each dataset contain the residuals after trend and seasonality removal. If each input dataset has a unique time profile, the conditional probability $P(y|x_t)$, where x_t is a temporal feature (*i.e.*, contains a value in time space), is not equal for all the sources and the target. Hence, $\mathcal{T}_S \neq \mathcal{T}_T$, confirming that this is a transfer learning problem. The goal of *time series domain adaptation* $\Phi()$ here is to approximate the conditional probability $P(\Phi(Y)|X)$ of all domains by removing the effects of all temporal features (*e.g.* day

Figure 4.2: Time series domain adaptation phase.

of the week) from the label.

Trend Removal

Each dataset may have different trends. To reduce this difference, the label has the trend removed, losing its long-term variation, as can be seen in Figure 4.2. The *trend removal* calculates the trend factors to first remove the trend from the label using a removal function, based on the additive or multiplicative models, and later to adjust the prediction with the inverse of the removal function.

Small amounts of data in the target dataset can be statistically insufficient to determine the trend properly. This issue can be solved by using the trend factors calculated from one of the source datasets or a composite of all of them, assuming that they have similar trends. For example, two approaches can be used: (a) calculating the average of all the source datasets or (b) choosing the dataset that is most similar to the target's (*e.g.*, using K-Nearest Neighbors).

Seasonal Removal

Like trends, datasets can have different seasonal profiles (for example, weekly seasonality) and labels can have the seasonality removed to reduce seasonal impact, as shown in Figure 4.2. The *seasonality removal* calculates the seasonal indexes to first remove the seasonality from the label using a removal function, based on additive or multiplicative models, and later to adjust the prediction with the inverse of the removal function.

Similarly as in trend removal, the target dataset may not be large enough to calculate seasonal indexes with statistical relevance. If this is the case, the target's seasonal removal can be calculated using the seasonal indexes from one of the source domains or a composite of all of them, assuming that they have similar seasonal profiles. For example, this can be done by (a) calculating the average of all the source datasets or (b) choosing the source dataset most similar to the target's (*e.g.*, using K-Nearest Neighbors).

After trend and seasonal removal, the outcome of this phase is the residual component of the target variable with time effects removed.

4.1.2 Atemporal Domain Adaptation

In this thesis, a domain adaptation method is proposed in the *atemporal domain adaptation* phase. However, Hephaestus is flexible enough to work with any other domain adaptation technique. The goal here is to align the domains of all atemporal features and labels to enable them to be handled together. Figure 4.3 illustrates how this works.

For a single feature f, the values X_f from each source domain and the target domain $\mathcal{D}_k$, are subjected to a *global normalization* (if the relationship between Y and X_f are absolute) or *local normalization* (if the relationship between the label Y and the feature X_f are relative). These two normalization methods are described next.

Local Normalization

A relationship between Y and X_f is relative if the value of Y relies on a proportional value of X_f. In other words, the conditional distributions $P(Y|\Psi(X_f))$ of each source or target are the same, where $\Psi()$ is a normalization function. *Local normalization* is illustrated in Figure 4.3, which shows all the resulting similar marginal distributions centered in the same position for

Figure 4.3: Local and global z-score normalizations.

all source and target domains.

To illustrate a relative relationship, assume that energy consumption and external temperature are correlated and that buildings have different structures adapted to the climate where they were built. Therefore the air conditioner will not be set to an exact outdoor temperature, but turned on when the indoor temperature becomes warm, a decision which depends on the building's heat-exchange characteristics with the outside environment. In this situation, the relationship between the outside temperature and energy consumption is relative. The idea of normalizing the temperature for each building is to align the concept of warm temperature onto the same scale for all buildings.

Because most features in real-world regression problems follow a normal distribution, the linear transformation proposed for this method is the z-score normalization. The z-score normalization maintains all the feature distributions by aligning their means on the center and keeping outliers out of bounds. Figure 4.3 uses z-score normalization as an example.

Z-score normalization can be calculated using Eq. 2.2. For *local normalization*, only the feature values $X_f^{\mathcal{D}_k}$ from the current domain $\mathcal{D}_k$ are used.

Similarly in time series domain adaptation, depending on dataset size, it is difficult or statistically irrelevant to achieve proper normalization using only the local dataset. This can be solved by using the normalization parameters from one of the source domains or a composite of all of them, assuming that they have similar distributions.

Labels from source and target datasets can be in different domains, therefore they must also be normalized. For example, energy consumption depends on building size: the energy consumption peak in a bigger building can be expected to be higher than in a smaller building. In this case, *local normalization* should be used for the label.

Global Normalization

A relationship between Y and X_f is absolute if the value of Y relies directly on the absolute value of X_f. Therefore the X_f of each dataset is a subset of a superset F that contains all the subsets from all the datasets. In other words, the conditional distributions $P(Y|\Psi(X_f, F))$ for all source or target datasets are the same in a global context. In this case, the feature X_f is considered to be already in the same domain for all the sources and the target. *Global normalization*

is illustrated in Figure 4.3, where the dashed lines represent the assumed marginal distribution of the superset F and the continuous lines represents the marginal distribution of X_f from the source and target domains in relation to F.

For example, suppose that all the buildings are outfitted with the same type of equipment (*e.g.* computers, lighting) with equal consumption characteristics and that the number of pieces of equipment is available in the datasets. Because each piece of equipment consumes the same amount of energy for every building, its relationship with energy consumption would be considered absolute because it depends on the absolute number of pieces of equipment that are turned on.

However, even if features with a global relationship are already in the same global domain, they should also be normalized in the same way the features with a local relationship to maintain all the features at the same scale and consequently ensure prediction quality.

When using z-score normalization (Eq. 2.2) for *global normalization*, all the feature values X_f from all datasets are used at once.

4.1.3 Standard Machine Learning

In Hephaestus, how a machine learning algorithm works does not change because the input is still one single dataset. The pre- and post-processing phases do not affect the execution of the algorithm. Hence, Hephaestus can work with any standard regression algorithm (*e.g.* support vector regression and neural networks) to build the predictive model, feeding the composed *training set* directly into the algorithm without the need to adapt it.

In addition, any pre- or post-processing procedures, such as instance selection/weighting and feature selection/weighting, can be executed during the standard machine learning phase over the *Training Set*, either before training or after prediction.

4.1.4 Adjustment

It is important to note that the labels have been modified during *time series* and *atemporal domain adaptations* phases. To retrieve the correct predicted values, the adjustments for each phase must be done by applying the inverse functions from the *time series domain adaptation*

$\Phi^{-1}(\cdot)$ and the *atemporal domain adaptation* $\Psi^{-1}(\cdot)$.

4.2 Case Study

This case study aims to improve energy consumption prediction for a building with only one month of data available, with the help of data from additional buildings using Hephaestus.

There is no definition of what is an accurate model, because it depends on the predictive model itself. For this case study, we considered that it is necessary at least one year of data from a single domain to build accurate machine learning models, because it includes all the seasons of the year and contains a variety of data to justify the irregular component.

The goal is to reach similar accuracy as if the model had been used with one entire year of data for the target building. We assume that once Hephaestus is successfully evaluated for energy consumption forecast, it works for a variety of other tasks regarding the same nature (i.e., regression tasks considering time series and multiple features) such as retail sales prediction and Web site visitors prediction.

Around 40% of energy consumption from an average building is related to heating, ventilation and air conditioning (HVAC) and 15% to lighting [45]. The objective of HVAC and lighting is to maintain a comfortable and healthy environment for building occupants. In addition, HVAC can also be related to operational draws such as equipments (*e.g.*, to cool down computers). Therefore, we concluded that the energy consumption of a building is strongly related to human factors and machinery, which can be analyzed considering time and weather.

This case study makes use of data collected from four different buildings using Powersmiths meters. Powersmiths is a company for which one of the main product lines is meters and sensors to manage building resources. Their clients are located in several locations and have different profiles. Four schools have been chosen to increase the likelihood that the buildings would exhibit similar behavior, even though they were different in sizes and in the number of staff and students. The schools are located across Newfoundland, Canada, which slightly increases the diversity of weather conditions without making the schools too distinct from one another. Moreover, the schools share similar seasonal patterns, including hours of operations, holidays, and seasons of the year. Although they are all schools located in the same province,

their weekly profile is different from one another.

The datasets for the four schools contain three years of daily data, from January 1, 2013, to December 31, 2015. Each dataset contains 17 attributes, including four temporal attributes: (1) year, (2) month, (3) day of the year, (4) day of the week; 12 atemporal attributes, all related to external weather: (5) minimum temperature, (6) maximum temperature, (7) mean temperature, (8) difference of mean temperature in one day, (9) difference of mean temperature in two days, (10) difference of mean temperature in three days, (11) dew point, (12) mean dew point, (13) minimum dew point, (14) minimum humidity, (15) maximum humidity and (16) mean humidity; and finally, (17) energy consumption, measured by Powersmiths meters.

In the following subsections, the *Evaluation* subsection discusses how the performance of Hephaestus was evaluated; the *Implementation* subsection describes certain details of the implementation; the *Preliminary Analysis* subsection discusses how Hephaestus performs during its process; and the *Results* subsection discusses how well Hephaestus performed.

4.2.1 Evaluation

To evaluate this case study, four different models were implemented for each school:

- **T.1**: using one month of data from the target school only, to simulate the scenario where only recent data from the target domain are available.
- **T.12**: using 12 months of data from the target school, used only as a benchmark to compare with Hephaestus performance.
- **H.1-12**: using one month of data from the target school plus 12 months from the additional schools, using Hephaestus.
- **H.12-12**: using 12 months of data from the target school plus 12 months of data from the additional schools, using Hephaestus, used only as a benchmark.
- **N.1-12**: using one month of data from the target school plus 12 months of data from the additional schools, without using domain adaptation; only min-max normalization was used for each feature, to demonstrate the impact of domain adaptation compared to not using it.

The additional schools used to train the models were the other three schools besides the target school itself.

We made the prediction for one entire year using rolling base forecasting, which uses a certain number of months to train the model and one month to test. Figure 4.4 illustrates how this process works, using as an example one month of data from the target school plus 12 months from the additional schools.

The errors between the predicted values and the measured values were used to measure the accuracy of a model and to compare with the others. Two different errors were calculated: the mean absolute percentage error (MAPE) given by the equation

$$\text{MAPE} = \frac{1}{n}\sum_{i=0}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{4.1}$$

and mean square error (MSE) given by the equation

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2 \tag{4.2}$$

where y is the actual value and $\hat{y}$ is the predicted value.

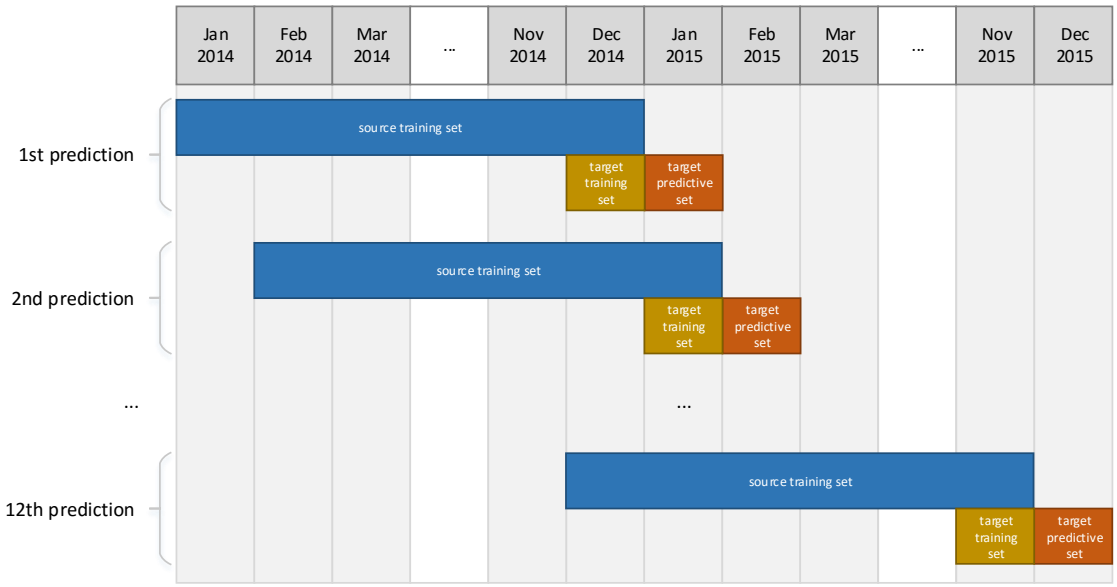


Figure 4.4: Rolling base forecasting.

Hephaestus models H.1-12 and H.12-12 should be better than model T.1 and similar to model T.12. It is expected that Model N.1-12 should not perform well because it is using different domains directly to train the predictive model.

4.2.2 Implementation

From the data for the analyzed schools, if the energy consumption is high, the seasonal magnitude will also be high. For example, Figure 4.5 shows a sample of energy consumption for School A from September 1, 2013 to December 31, 2013. It is evident how the average of energy consumption is lower during the first weeks and increases over time; the same effect is evident in the magnitude, where the differences between highs and lows in the beginning are lower and then increase over time. Because of this pattern, we implemented this experiment was implemented using the multiplicative model.

For trend removal, trend smoothing was performed using moving averages:

$$ts_i = \frac{1}{m}\sum_{j=0}^{m-1} y_{i-j} \tag{4.3}$$

where m represents the last m values in the time series. In this experiment, if a small number of days were chosen, this could remove the effects of weather attributes from the residuals. However, 365 days were used because this is enough to remove only long-term trends, and keeps the cyclical component related to seasonal weather features (such as temperature) over

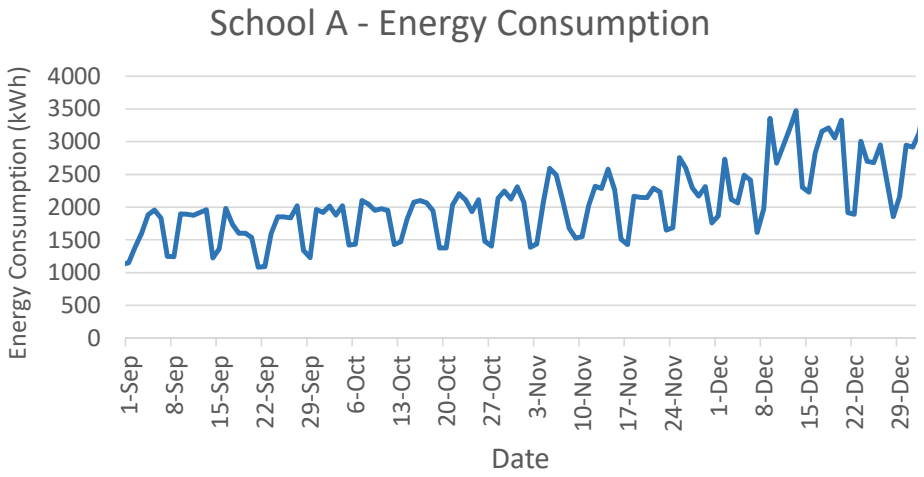


Figure 4.5: Sample of an energy consumption timeline.

the energy consumption. Otherwise, using a smaller number of days would remove, for example, the correlation between temperature and energy consumption, because the seasons of the year would be removed with the trend.

The trend factor, which measures the proportional increase of the trend at a specific point, was calculated by the equation

$$tf_i = ts_i / ts_r \tag{4.4}$$

where r is the index of the reference value, and therefore tf_i represents the proportion of the trend in i in relation to r. In this case, r was set to the last day of each source dataset. The trend factor was used to remove the long term trend from the additional datasets, by dividing the energy consumption by the trend factor.

From the three years of data, the first year's was used exclusively to calculate the long-term trend only for the additional schools. The remaining two years were used to run the evaluation. Because all the models predicted only one month, it was decided that calculating the target trend was not necessary.

For seasonal removal, only weekly seasonality was considered. Let P be the set of all points from a specific seasonality (*e.g.* days of the week), where p is a specific point in P, $y_{p,j}$ the jth of m observations that happened on p, and $\overline{y}$ the average of all the observations that happened in every $p \in P$. The seasonal index was calculated by:

$$s_p = \frac{1}{\overline{y}m} \sum_{j=1}^{m} y_{p,j} \qquad (p \in P) \tag{4.5}$$

to remove the seasonality by dividing the current value by its respective seasonal index.

During the *atemporal domain adaptation*, all the atemporal attributes were normalized locally using z-score (Eq. 2.2).

For *standard machine learning*, two experiments were constructed using two algorithms: (a) multilayer perceptron (MLP), and (b) support vector regression (SVR) because these are commonly used for energy consumption prediction [3]. This step was performed to demonstrate that Hephaestus works with any standard machine learning algorithm.

The *adjustment* was calculated by applying the inverse of the z-score function on the re-

sults from the *standard machine learning* and multiplying with the respective trend factor and seasonal index, which can be written as:

$$y = (z \times \sigma + \mu) \times tf \times s_p \tag{4.6}$$

We programmed Hephaestus using Node.js, using the Synaptic package[2] for multilayer perceptron and the Node-SVM package[3] for support vector regression. This experiment was made in a virtual machine with 4 Cores Intel Xeon E5-2630 2.3GHz and 16GB RAM DDR3 1600MHz, using VirtualBox[4] running Ubuntu 14.04[5].

4.2.3 Preliminary Analysis

The preliminary analysis helps explain the details of the data used in this case study and how Hephaestus works. For illustration reasons only, all the following analyses used the entire dataset from all the schools, instead of using the evaluation model previously described.

The weekly seasonal profiles for each school were calculated during *seasonal removal* in the *time series domain adaptation* phase. Figure 4.6 shows that each school's weekly profiles are different. Because each school presents different seasonal profiles, one school data cannot be used to predict the energy consumption of another school because the prediction's seasonality will not fit the target's seasonality.

Furthermore, the correlation between the atemporal features (*e.g.* external mean temperature) and the label (energy consumption) is not the same for all schools. For example, Figure 4.7 shows the difference between the correlations, using a polynomial interpolation of degree 3, between mean temperature and energy consumption. Figure 4.7a shows the correlation before *atemporal domain adaptation*. In this scenario, to determine the energy consumption for one school using the curve from another school may not be accurate because the curves do not overlay one another. This demonstrates the need for domain adaptation, which approximates these curves to improve machine learning accuracy.

[2] `http://synaptic.juancazala.com`
[3] `http://github.com/nicolaspanel/node-svm`
[4] `http://www.virtualbox.org`
[5] `http://www.ubuntu.com`

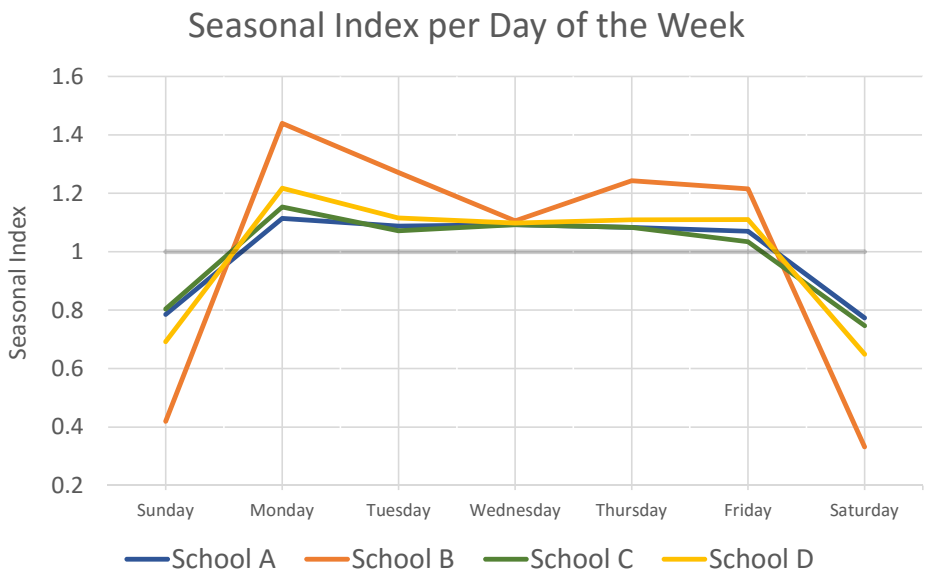


Figure 4.6: Seasonal index for all four schools.

Atemporal domain adaptation attempts to reduce the distance between the data distribution for each school, and the new correlation between an atemporal feature (*e.g.* mean temperature) and the school's energy consumption should be similar. Figure 4.7b shows the correlation between external mean temperature and energy consumption after atemporal domain adaptation.

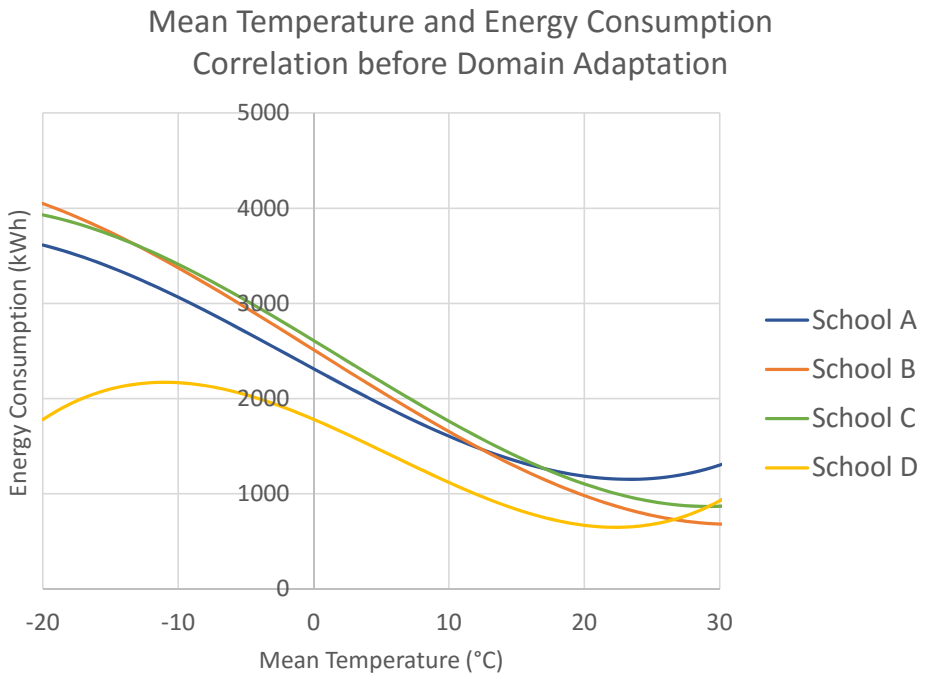
Once correlations for the atemporal features are approximated with one another, they are ready to feed the predictive model.

4.2.4 Results

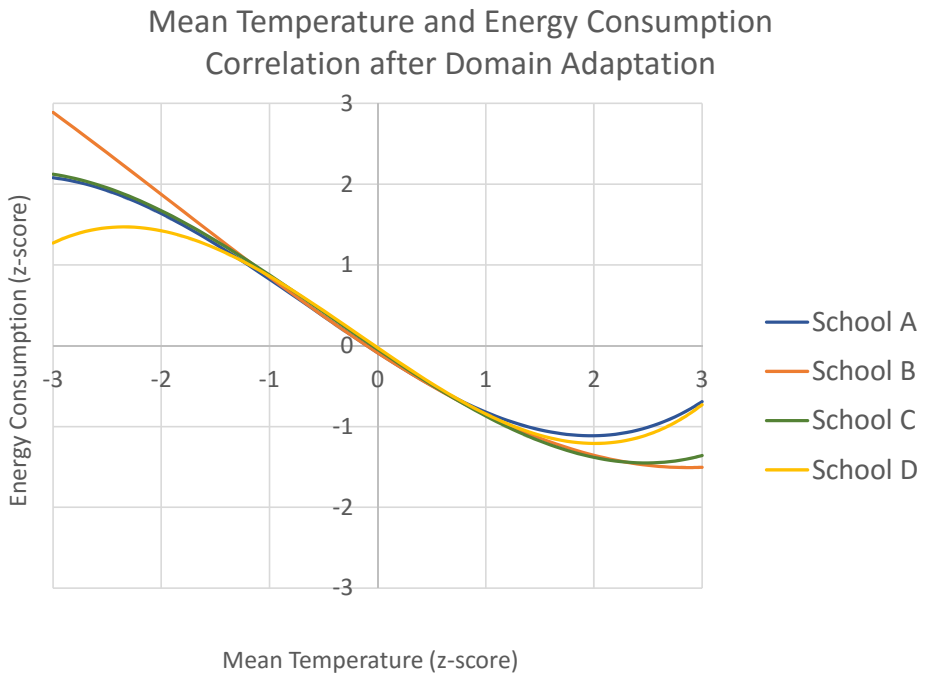
Table 4.1 shows the results of the experiment using multilayer perceptron (MLP). Each column represents the Mean Absolute Percentage Error (MAPE) and the Mean Square Error (MSE) for each target school. Those same values are plotted in Figure 4.8. Figure 4.9 shows a sample of a timeline chart comparing the predictions made by the various models using MLP.

Model N.1-12 proved to be the worst model for all schools. This was expected because the model used measurements that were outside the target domain, making wrong inferences and thus increasing the error. This proves the importance of domain adaptation.

Model T.1 had the second worst results for every school. By adding data from additional schools using Hephaestus in model H.1-12, all schools improved their results compared with



(a)

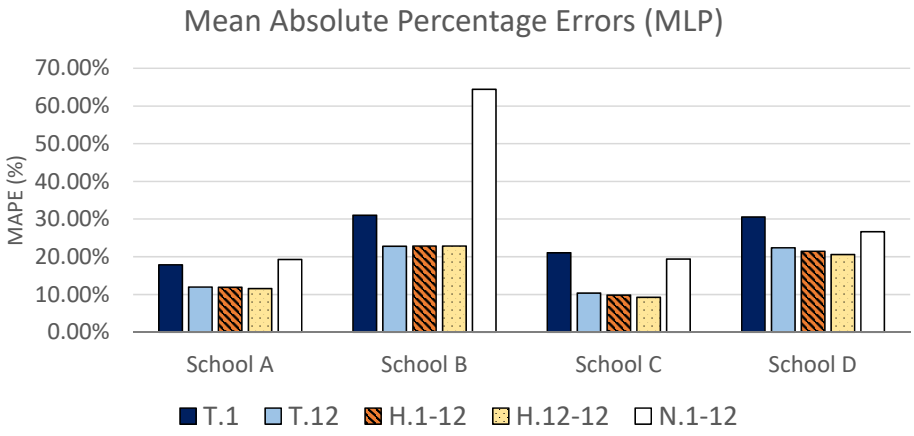


(b)

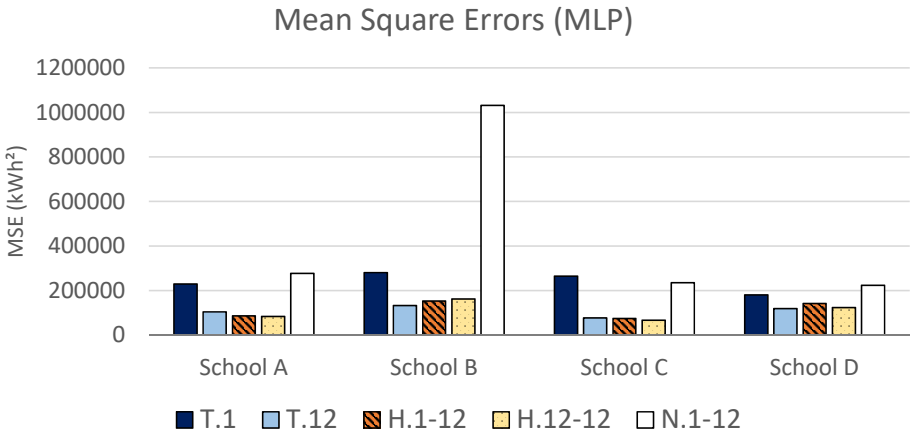
Figure 4.7: Polynomial interpolation of the correlations between mean temperature and energy consumption.

Table 4.1: Errors for the schools' energy consumption prediction (MLP)

Model	**School A**		**School B**		**School C**		**School D**	
	MAPE	MSE	MAPE	MSE	MAPE	MSE	MAPE	MSE
T.1	0.2088	229548	0.3405	320401	0.2105	264078	0.3055	180463
T.12	0.1196	104444	0.2281	132927	0.1040	76155	0.2237	118421
H.1-12	0.1191	86240	0.2288	152466	0.0986	74601	0.2149	141240
H.12-12	0.1156	83337	0.2284	162093	0.0928	66509	0.2060	123051
N.1-12	0.1928	277025	0.6439	1031830	0.1941	234497	0.2662	222852



(a)



(b)

Figure 4.8: Errors for the schools' energy consumption prediction (MLP).

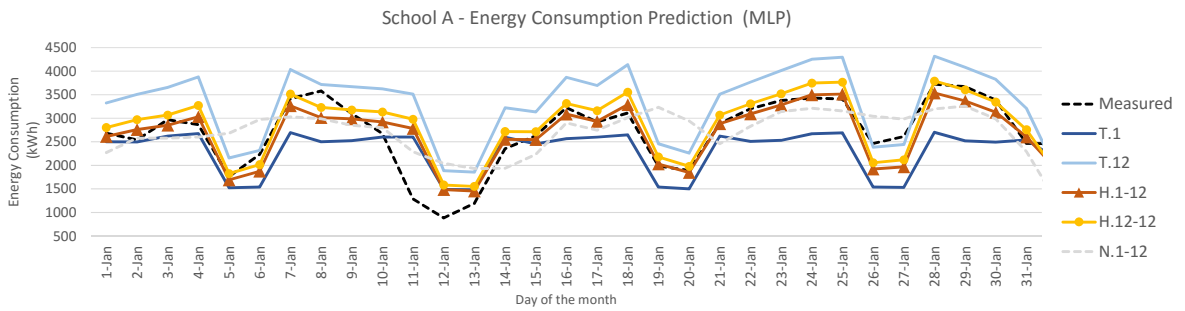


Figure 4.9: Timeline comparison between the predictions for MLP models.

model T.1. When calculating T.1 minus H.1-12 from Table 4.1, school C had the best improvement, reducing the MAPE by 11.19% and MSE by 189477.

In addition, the prediction accuracy for model H.1-12 was almost as good as for model T.12, which used only 12 months from the target school. Moreover, model H.1-12 performed slightly better than T.12 for schools A and C (when considering both MAPE and MSE), and D (when considering MAPE only).

It should be noted, however, that model T.12 gave better results than models H.1-12 and H.12-12 for school B (considering MAPE and MSE) and D (considering only MSE). The primary goal of Hephaestus was to improve prediction for small or new datasets, which was simulated by model T.1. T.12 was added only as a benchmark to verify whether H.1-12 was able to achieve a performance close to it where 12 months of the target dataset were available, which is not possible in the problem that Hephaestus was designed to solve.

The use of additional datasets without proper domain adaptation is not efficient, as demonstrated by model N.1-12. Hence, model H.1-12 was compared to T.1 and had a 100% success rate in this experiment, while still managing to achieve results compatible to T.12.

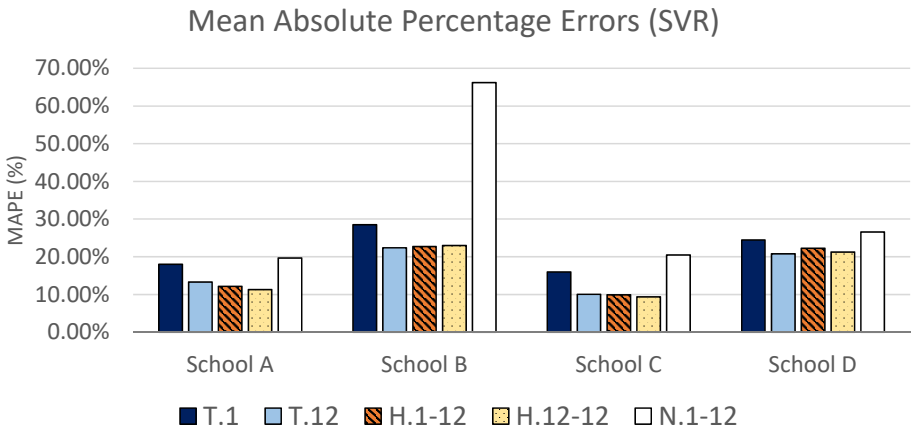
Table 4.2 shows the results for support vector regression (SVR), which are plotted in Figure 4.10. The results are similar to the MLP experiment. This demonstrates that Hephaestus can work with any standard machine learning algorithm.

4.3 Summary

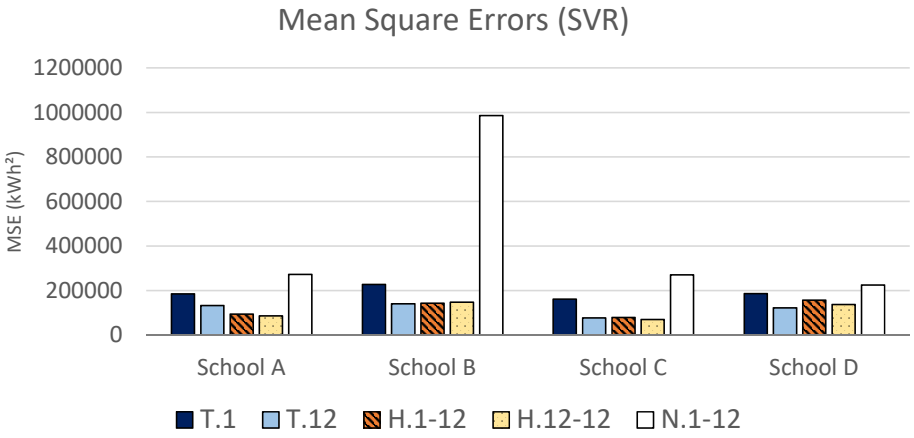
This chapter has proposed Hephaestus, a novel transfer learning method for predictive regression models for time series and multi-feature datasets. Hephaestus works in the pre- and post-

Table 4.2: Errors for the schools' energy consumption prediction (SVR)

Model	School A		School B		School C		School D	
	MAPE	MSE	MAPE	MSE	MAPE	MSE	MAPE	MSE
T.1	0.1798	185051	0.2853	226727	0.1594	160584	0.2447	185406
T.12	0.1327	132285	0.2240	140457	0.1006	76340	0.2081	121938
H.1-12	0.1218	94187	0.2272	142427	0.0991	79346	0.2227	156124
H.12-12	0.1133	85407	0.2296	147068	0.0940	70158	0.2124	136681
N.1-12	0.1964	272301	0.6623	986109	0.2048	269871	0.2656	224486



(a)



(b)

Figure 4.10: Errors for the schools' energy consumption prediction (SVR).

processing phases, enabling the use of standard machine learning algorithms. This method can adapt the target variable domain of multiple datasets by removing the effects of time, and adapts features by a simple z-score domain adaptation technique.

To validate Hephaestus, a case study on energy consumption prediction for multiple schools was successfully presented. Predictive performance increased by up to 11.2% when using data from additional schools compared with using only one month of data from the target school. The results were found to be similar or even better than using 12 months of data from the target school.

Chapter 5

Sibyl: a Machine Learning as a Service

This chapter describes Sibyl, a machine learning as a service architecture adapted from a previous study[1][42]. Sibyl's architecture allows multiple users to use the same platform to build and run multiple machine learning models at the same time.

5.1 Architectural Design

This section describes the proposed Sibyl architecture, which is designed to support machine learning by gathering data from multiple sources and building multiple machine learning models using different algorithms. The approach focuses on predictive modeling, but it is adaptable to other applications.

The scope of this architecture deals with the machine learning itself, ignoring the front-end aspects such as the user interface. In a model-view-controller (MVC) perspective, this architecture focus on the model layer while the controller and view layers are only implemented as part of the case study. The term model in the MVC context should not be confused with the same term used in machine learning and predictive models that is used in the rest of this thesis.

Figure 5.1 illustrates a high-level description of Sibyl, where many users share the same platform and each one creates as many machine learning models as they needed. The *modeler* is responsible to handle all the user requests to build a model or redirect the request to the

[1]This research has been published as a conference paper in 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA) as "*MLaaS: Machine Learning as a Service*" in December 2016, co-authored by Katarina Grolinger and Miriam A.M. Capretz

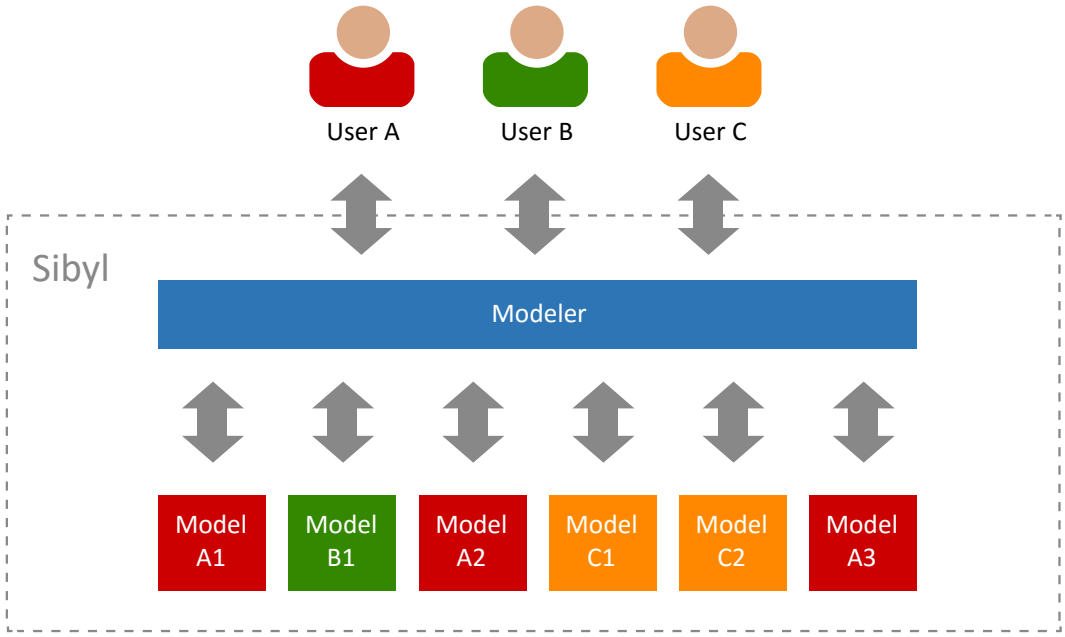


Figure 5.1: High-level diagram showing how Sibyl should work.

proper user's owned model.

The simplified SCA diagram in Figure 5.2 transforms the above concept into a technical overview of Sibyl architecture, hiding the details of the composites' implicit components. Sibyl provides a total of seven services, including five for managing *models*: *build*, *train*, *validate*, *test* and *predict*; and three for retrieving results: *get report*, *get prediction* and *get test*. These services are all linked to the *modeler* composite. The *modeler* composite works as an interface between users and *models*. The *modeler* composite is responsible for managing the creation of new *models* and addressing the users requests to the correct *model*. The *modeler* composite can handle as many *models* as requested by the users.

The five services for managing *models* represent the main machine learning tasks: *build* creates a new instance of a model; *train* trains the model; *validate* validates the model; *test* tests the model using a test set and *predict* creates a new prediction. All these services have corresponding consumers that are linked to a specific model instance.

The specified services provide well defined interfaces that increase the architecture's flexibility to new inputs and outputs. The five Sibyl's services for managing models allow any user to access the same platform to create their own models. The corresponding five *modeler*'s well

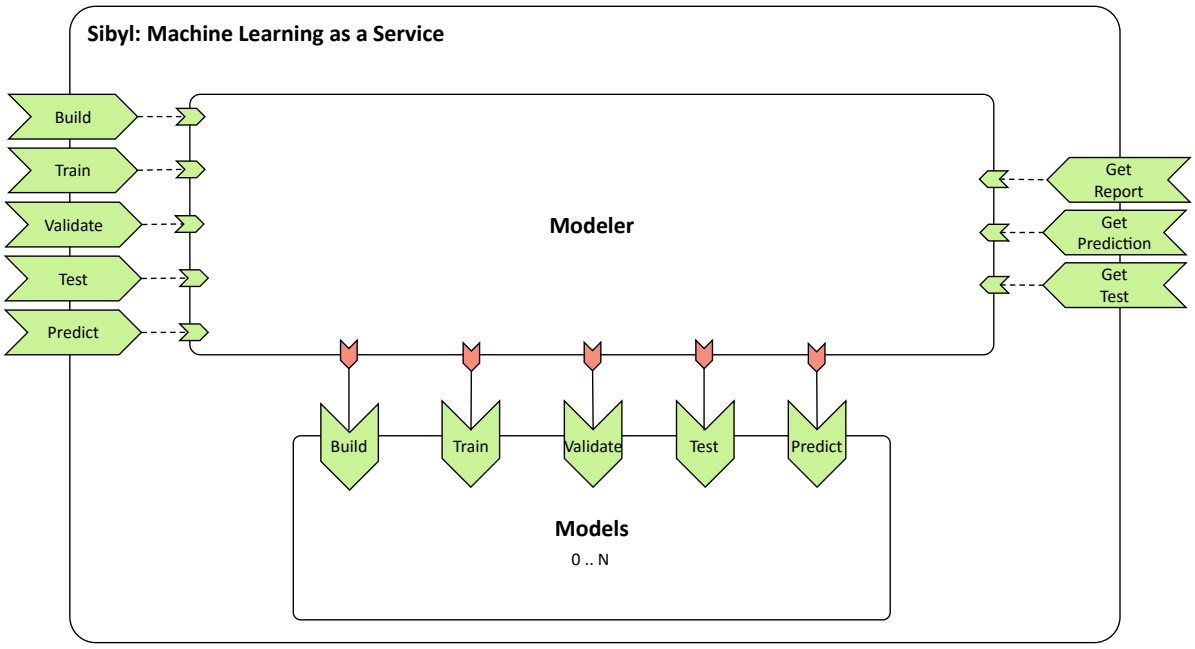


Figure 5.2: Sibyl's simplified architecture using SCA notation.

defined consumers enable the architecture to be pluggable with different *model-μ* instances. The three Sibyl's services *get report*, *get prediction* and *get test* enable different user interfaces and external systems to consume the resulting data.

The architecture works as follows: first, a model must be created by using the *build* service. Once the model is built, it must be trained by sending the training set through the *train* service. Once the model is trained, users can request it to be validated, to be tested or to make a prediction.

The SCA diagram in Figure 5.3 depicts the same architecture with higher level of details, displaying all the implicit components of the composites.

A model is an instance of *model-μ* composite, running a specific machine learning algorithm with specific parameters. The cardinality 0..N shows that Sibyl can create and run multiples instances of *model-μ* at the same time. The *model* property shows that each instance can run with different settings.

The following subsections describe each of Sibyl's artifacts.

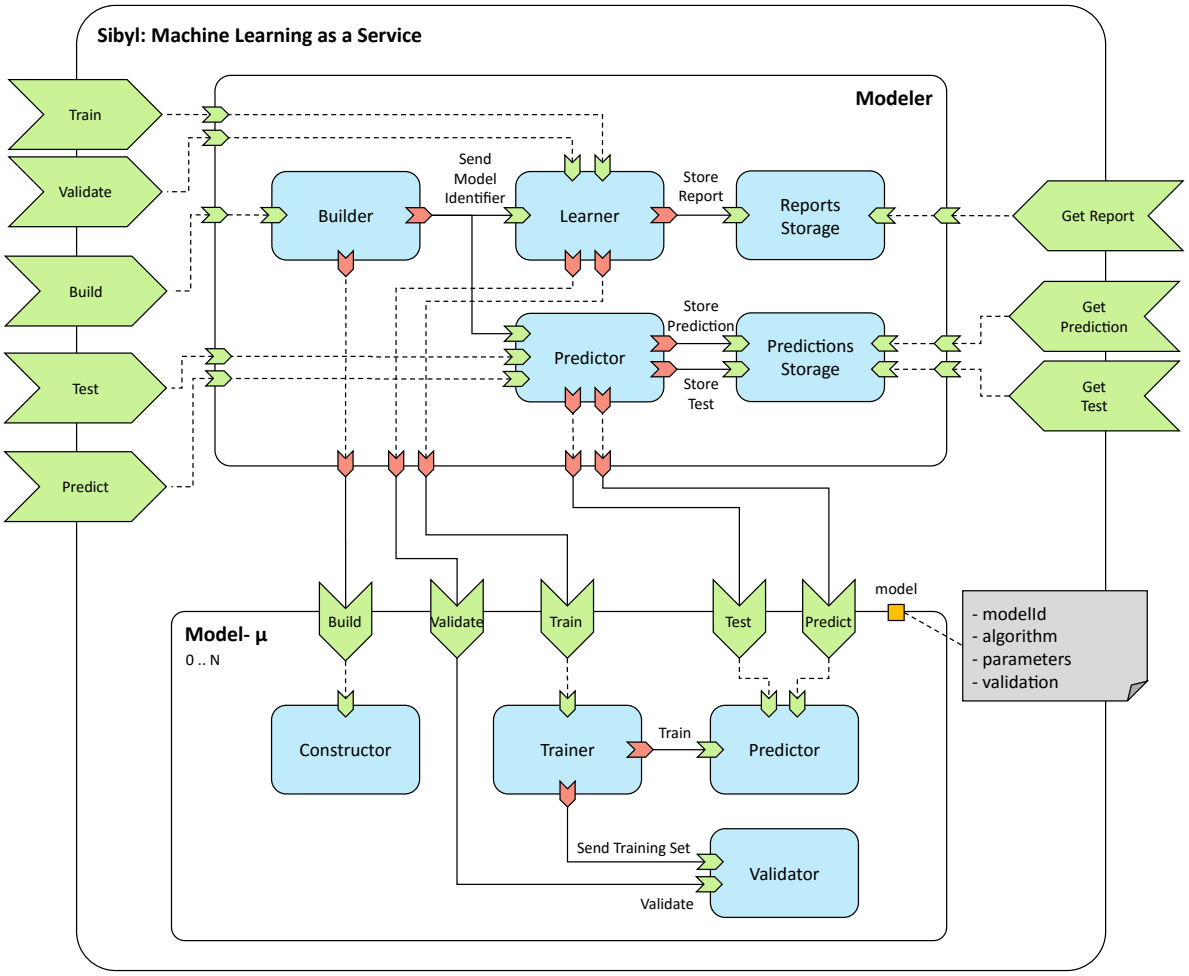


Figure 5.3: Sibyl's detailed architecture using SCA notation.

5.1.1 Modeler Composite

This is the core composite in the architecture, because it is an interface between the user and *model-μ* instances. It is responsible for building, training, validating, testing, and running the *model-μ* instances. It is made up of five components as illustrated in Figure 5.3, which can be described as follows:

- The *builder* component receives from *build* service the parameters (*e.g.*, algorithm and property values) to build and deploy a new model (a *model-μ* instance) for the *build* consumer. When the instance is created, *builder* sends the model identifier back to the consumer and forwards it to the *learner* and *predictor* components.

- The *learner* component receives the data from the *train* service and forwards them to

the destined *model-μ* instance. When it receives the training report from the *model-μ* instance through the *train* consumer callback, it forwards the training report to *reports storage*.

- The *reports storage* component receives the training reports from the *learner* component through the *store report* service. Reports can be from training or validating a model. The *reports storage* component serves the training reports to external consumers through the *get report* service.

- The *predictor* component receives the predictive set from the *predict* service and forwards it to the *model* through the *predict* consumer, which will return the prediction through a callback. The prediction will be returned to the *predict* requester and also forwarded to *predictions storage*. *predictor* is also responsible for forwarding the testing set.

- The *predictions storage* component receives and stores the predictions and tests from the *store prediction* and *store test* services and provides them to external consumers through the *get prediction* and *get test* services.

5.1.2 Model-μ Composite

The *model-μ* composite is an architecture for building different models. It holds all the implemented algorithms source codes (*e.g.*, multilayer perceptron or support vector regression), but only one must be loaded. The algorithm to be loaded and its parameters should be specified when calling the *build* service. In other words, for each *build Model* service request, a new instance of a *model-μ* composite is created.

The *model* property describes how the model needs to be built and executed. It is composed of four sub-properties: *modelId* is the model unique identifier; *algorithm* specifies which algorithm is going to be used by the model; *parameters* adjust the algorithm behavior; and *validation* specifies the validation method do be used.

The *train*, *validate*, *test*, and *predict* service specifications enable the *modeler* composite to interact with any *model-μ* instance, independently of the chosen algorithm or parameters.

The *model-μ* composite is made up of four components, which can be described as follows:

- The *constructor* component is responsible for loading the right algorithm and setting the properties of the model instance using the *build* service request parameters. When the instance is set up and running, it is ready to provide *train*, *validate*, *test* and *predict* services.
- The *trainer* component receives the training set from the *train* service and forwards it to *predictor* components through the *train* services. It also sends the training set to the *validator* component through the *send training set* service.
- The *validator* component receives the training set from the *trainer* component through the *send training set* service, feeds it to the algorithm and validates the algorithm with a validation method (*e.g.*, k-fold cross-validation or rolling base forecasting). Once validation is finished, the validation report is sent back through the service callback.
- The *predictor* component receives the training set from the *train* service to feed the model for future prediction requests. When receiving predictive sets through the *predict* service, it calculates and returns the predictions. When receiving test sets through the *test* service, it calculates and returns a comparison between predictions and the real measured values.

The implemented algorithms source code must be responsible only for training and predicting. Testing and validating do not depend on the algorithm itself, but on the results, which can be found by using the algorithm's training and predicting functions. Therefore, testing and validating functions are responsibilities of *validator* and *predictor* components, increasing standardization and reducing the effort when adding a new algorithm.

5.2 Process

An average user should follow the interactions with Sibyl in the following order: build, train, validate, test and predict. Figures 5.4, 5.5, 5.6, 5.7 and 5.8 illustrate these interaction phases between a *consumer*, the *modeler* composite and a *model-μ* composite. The term *consumer* in

the following discussion refers to a generic consumer from SCA notation using the *modeler* component and it must not be confused with Agora's actor consumer.

- *build*: it starts with the *consumer* requesting the *builder* component to build a new model through the *build Model* service. The *builder* component will then create and configure a new *model-μ* instance. When the building operation is complete, the *builder* component sends the new model identifier to the *learner* and *predictor* components and to the *consumer*.

- *train*: the *consumer* is now able to train the instantiated model. It sends the training set to the *learner* component through the *train* service, which will forward the training set to the *trainer* component of the *model-μ* instance. The *trainer* component will make two requests at the same time: one to the *validator* component to validate the model and another to the *predictor* component to be trained for future prediction requests. When *predictor* training is complete, it sends back a training report of the training process to the *learner* component, which requests the *reports storage* component to store the training report.

- *validate*: once the model is already fed with the training set, validation of the model is now possible. The *consumer* sends the validation settings to the *learner* compo-

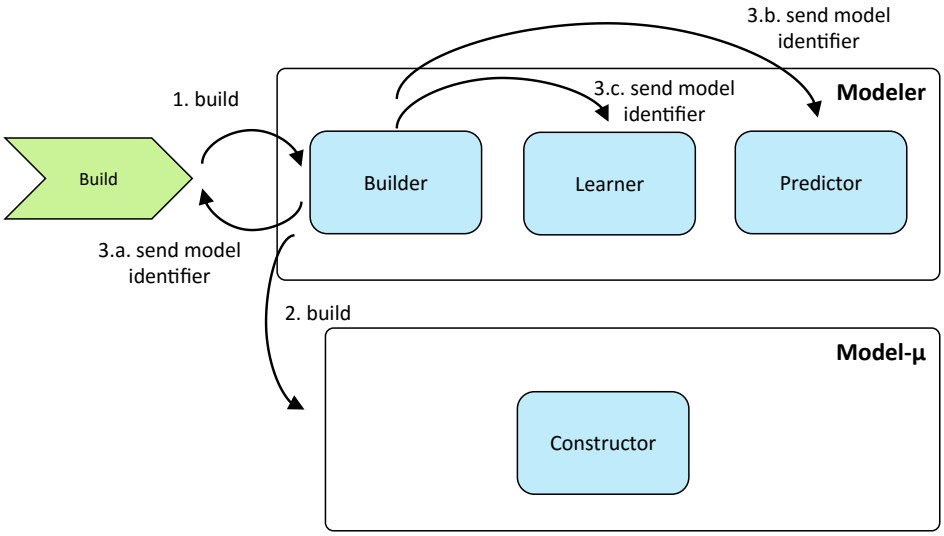


Figure 5.4: Process flow of build phase.

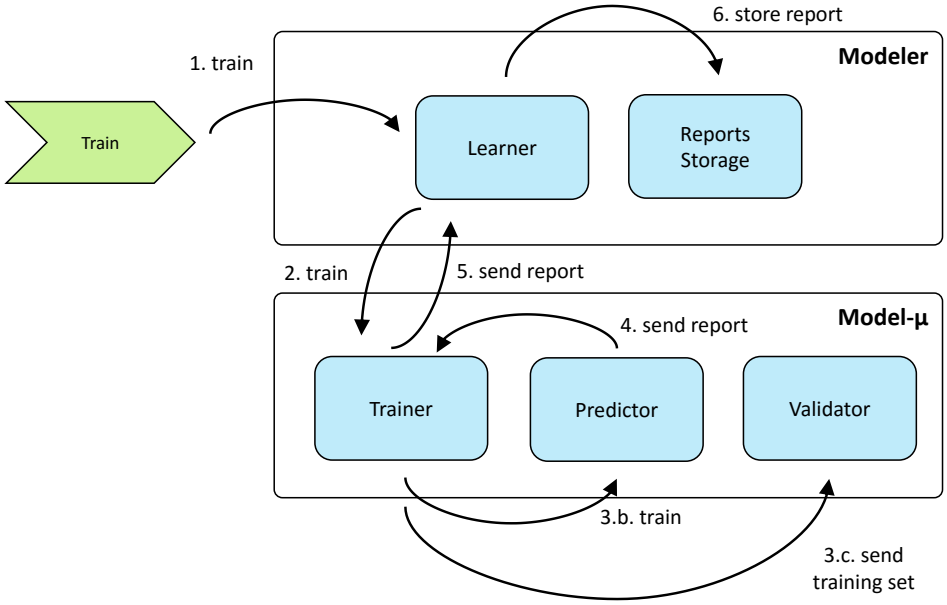


Figure 5.5: Process flow of train phase.

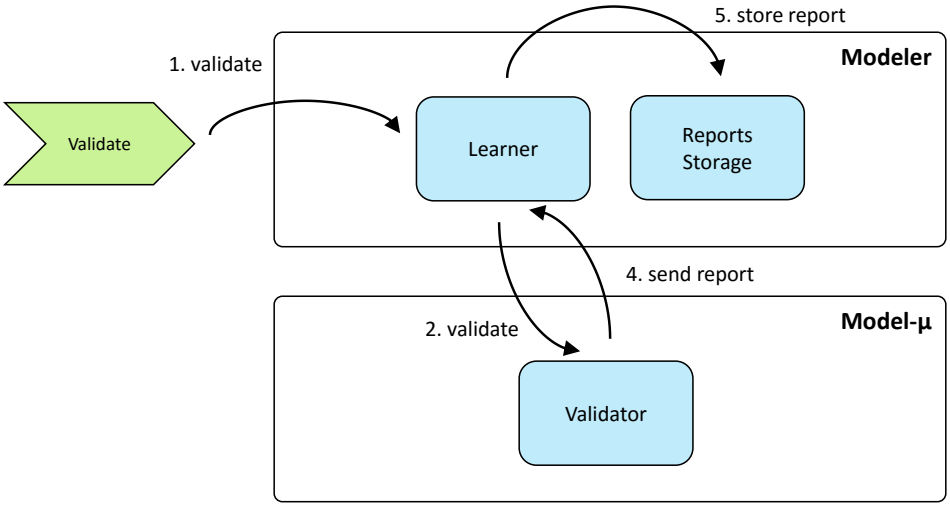


Figure 5.6: Process flow of validate phase.

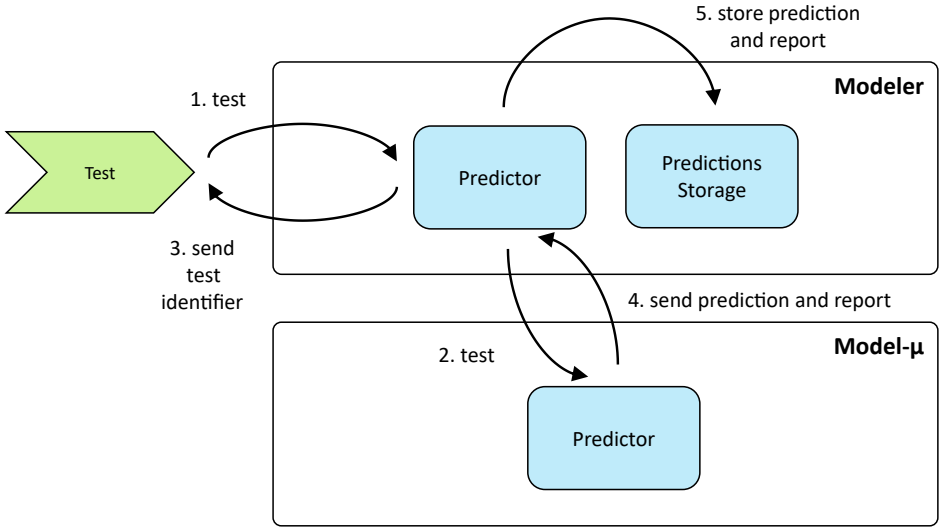


Figure 5.7: Process flow of test phase.

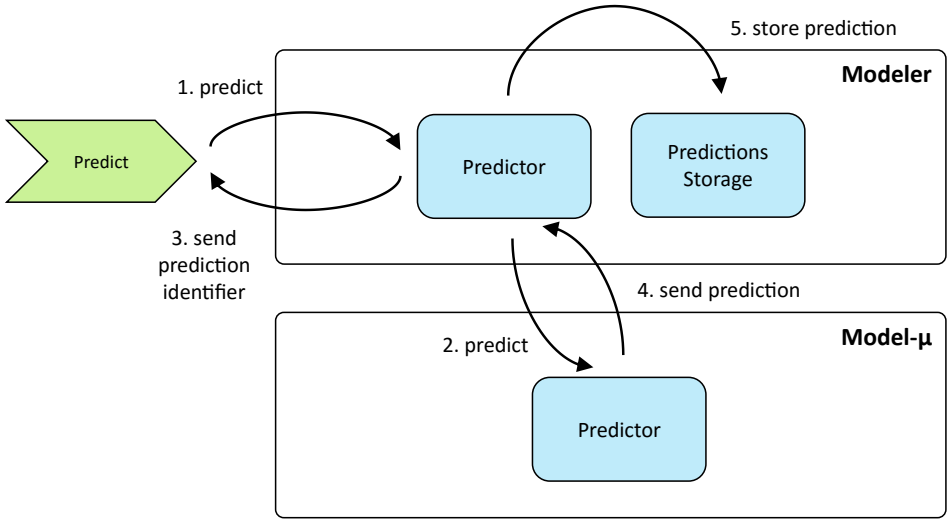


Figure 5.8: Process flow of predict phase.

nent through the *validate* service, which will forward to the *validator* component of the *model-μ* instance. When validation is complete, the *validator* component sends back a validation report to the *learner* component, which requests the *reports storage* component to store the validation report.

- *test*: the model is ready to be tested. The *consumer* sends the test set to the *modeler* composite's *predictor* component, which will forward to the *model-μ* instance's *predictor* component, where the prediction is calculated and compared to the measured labels. The prediction and a test report are returned to the *modeler*. The prediction and the test report are sent to the *predictions storage* to be stored and served.

- *predict*: works very similarly to the *test* phase. The *consumer* sends the predictive set to the *modeler* composite's *predictor* component, which will forward to the *model-μ* instance's *predictor* component, where the prediction is calculated and returned to the *modeler*. The predictions are sent to the *predictions storage* to be stored and served.

In *training*, *test* and *predicting* phases, the *consumer* receives the report and prediction identifiers as soon as the *learner* and *predictor* components receive the request, so it is not necessary to keep the connection while the entire request is be processed. When the report or prediction is ready, it can be accessed from *reports storage* and *predictions storage* components, using the specific identifier.

5.3 Case Study

The goal of this case study is to forecast energy consumption based on past data for a building, using different machine learning algorithms and finding the best-performing one.

We implemented Sibyl using energy data from Powersmiths' office building, in Brampton, ON, Canada. The dataset was pre-processed before feeding them to the system. The training set contains two years of daily data and the test set one month. The training set goes from January 1, 2014, to December 31, 2015 and contains 15 attributes, including two temporal attributes: (1) day of the year, (2) day of the week; and twelve atemporal attributes, all related

to external weather: (3) minimum temperature, (4) maximum temperature, (5) mean temperature, (6) difference of mean temperature in one day, (7) difference of mean temperature in two days, (8) difference of mean temperature in three days, (9) dew point, (10) mean dew point, (11) minimum dew point, (12) minimum humidity, (13) maximum humidity and (14) mean humidity; and finally, (15) energy consumption, measured by Powersmiths meters.

We built Sibyl using Node.js because of its ease and agility for coding and deploying Web services and handling JSON. Because there are currently no SCA frameworks for Node.js, we had to implement one. We used JSON for Web service communication and data storage. Also, instead of using the XML format for the SCA artifact descriptor file as the original design adopted, here we are using JSON because of its easy integration with Node.js. Finally, we also developed a simple user interface to generate effective illustrations of the results obtained.

This experiment was made in a virtual machine with 4 Cores Intel Xeon E5-2630 2.3GHz and 16GB RAM DDR3 1600MHz, using VirtualBox[2] running Ubuntu 14.04[3].

The source code is available in a public repository[4].

5.3.1 Algorithms

To evaluate the architectural flexibility of running different machine learning models at the same time, we implemented *model-μ* composite supporting the following algorithms:

- *Multilayer perceptron (MLP)*: one of the most used techniques when evaluating machine learning models, and one of the most used for electrical consumption problems [3]. For this case study, we used the Synaptic package[5].

- *Support vector regression (SVR)*: also one of the most used techniques for electrical consumption problems [3]. For this case study, we used the Node-SVM package[6].

- *K-nearest neighbors (KNN)*: simple to understand, to code, and to debug. We coded this algorithm for this experiment.

[2] `http://www.virtualbox.org`
[3] `http://www.ubuntu.com`
[4] `http://github.com/mauro0x52/mlaas`
[5] `http://synaptic.juancazala.com`
[6] `http://github.com/nicolaspanel/node-svm`

We developed a generic *Algorithm* class under an object-oriented programming structure, defining the standard interface for train and predict function calls. A new algorithm can be implemented simply by inheriting the *Algorithm* class and making minor adaptations. In this case study, we implemented the *KNN Algorithm* first to test and validate the *model-μ* composite. Later, using the same code structure, we coded *MLP Algorithm* and *SVR Algorithm* classes and imported into *model-μ* composite.

When a *model-μ* instance is built, the algorithm with the parameters (both specified in the *model* property) is loaded.

The test and validate functions are performed by *predictor* and *validation* components respectively, not by the *Algorithm* class. Both functions use the results from *Algorithm*'s train and predict calls.

The *validator* component implements the rolling base forecasting method (see Section 4.2.1) to validate models and compare their performance by calculating the mean absolute percentage errors (MAPE) given by the equation

$$\text{MAPE} = \frac{1}{n}\sum_{i=0}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{5.1}$$

and the mean square errors (MSE) given by the equation

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2 \tag{5.2}$$

where y is the actual value and $\hat{y}$ is the predicted value.

The architectural design and the dynamic artifacts descriptor file make it possible to create new *model-μ* instances dynamically. After the new *model-μ* instance is deployed and the artifacts descriptor file is updated, the new *model-μ* instance will be available without the need to recompile or restart the system.

5.3.2 Results

We created three different models by instantiating the *model-μ* composite. The models were asked to predict using a test set, which contains all 15 attributes including the real measured

daily energy consumption peaks. We also asked asked the models to run a prediction using a different predictive set. Figures 5.9, 5.10, 5.11 and 5.12 are screenshots of the Sibyl graphical user interface (GUI). They represent the main features of Sybil, which correspond to the five phases described in the process section: build and train are represented in Figure 5.9, validate in Figure 5.10, test in Figure 5.11, and predict in Figure 5.12.

Figure 5.9 shows the models screen. Here the user can create and build any number of predictive models. On the top, a navigation bar is replicated on all screens and enables the user to access models, validate, test and predict screens. In the models screen, the user can access a list of models as well as create new models or remove existing ones. In the table in Figure 5.9, each row represents a single model with its unique identifier number, its name created by the user, the chosen algorithm, the chosen training set file, the model status (*i.e.*, building, ready, training, or trained) and the actions available to train or delete the model.

Figure 5.10 shows the validate screen. In this screen, users can create validation tests using the training set to measure the performance of their models. The table in Figure 5.10 shows a list in which each row represents a different validation test with a the unique identifier number, a name chosen by the user, the model used, the validation method (*e.g.*, Rolling Base Forecast or K-Fold Cross Validation), the status (*i.e.*, ready, validating, or validated), and the actions available to start the validation test, remove the validation, and visualize the results in the charts. In the validation charts section, users can visualize the performance of each validation

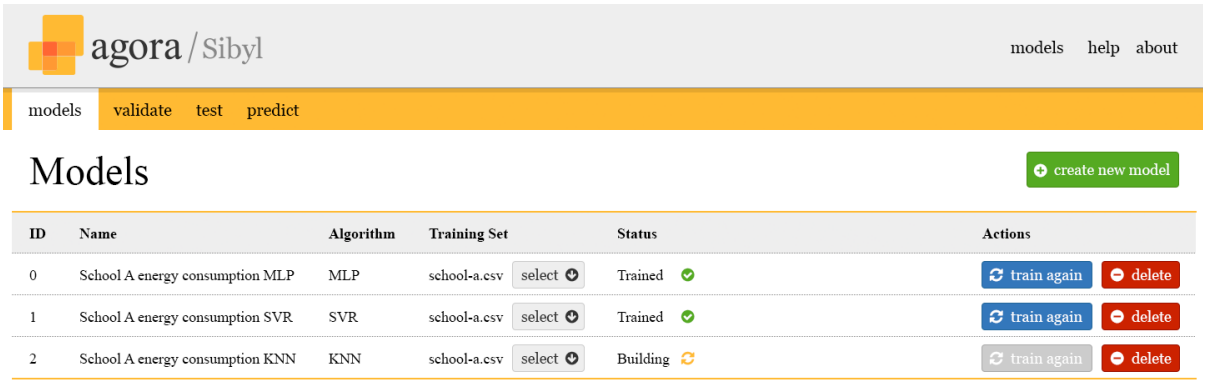


Figure 5.9: Sibyl's models screen.

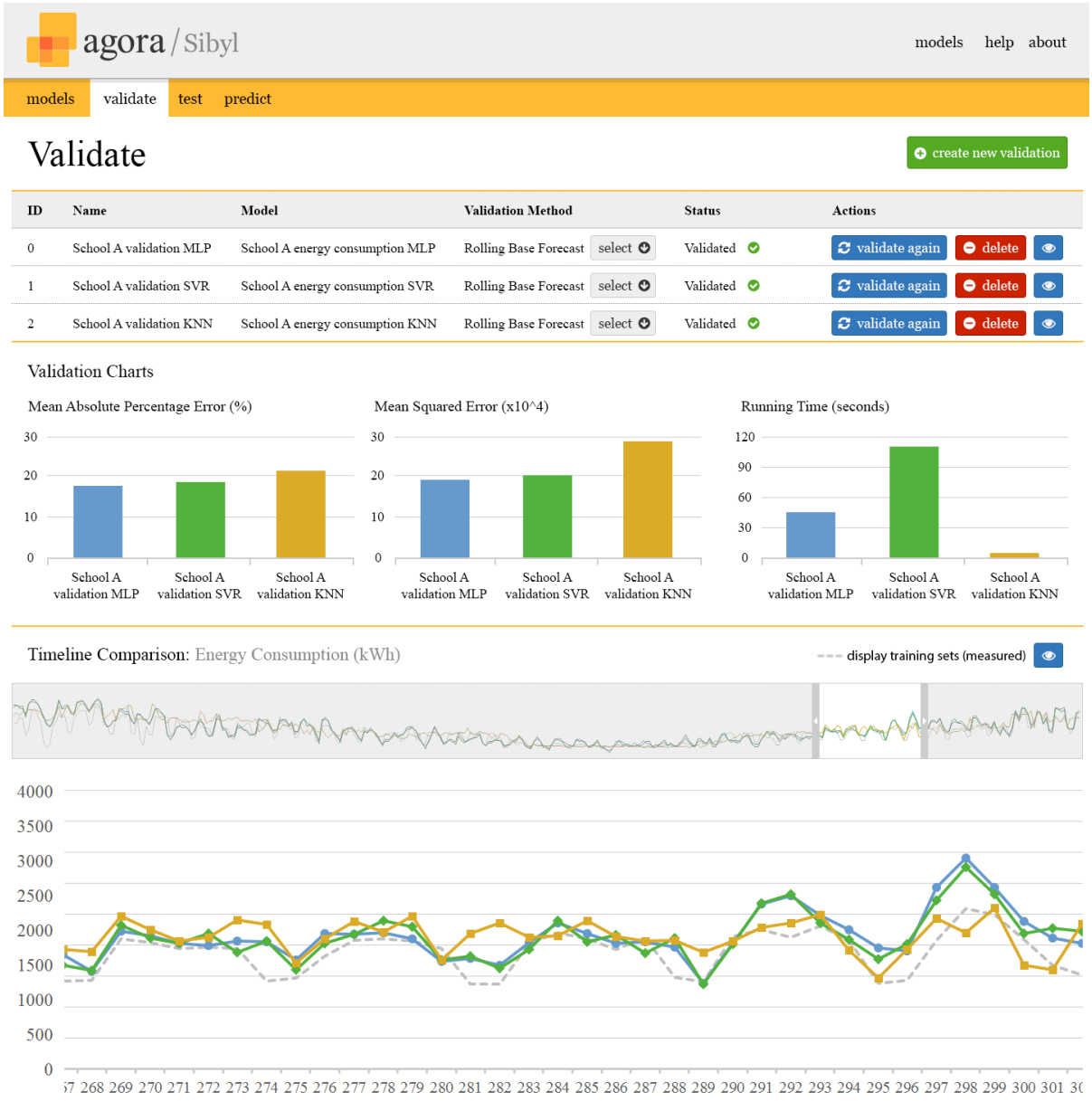


Figure 5.10: Sibyl's validate screen.

test, including the mean absolute percentage error, the mean square error, and the running time. In the timeline comparison, users can compare the predictions from different models against real measured data from the training set.

On the validate screen, the MLP model performed better — it had the lowest mean absolute error and mean square error — whereas the KNN model performed worst. However, when considering running speed, the KNN model was the fastest, and the SVR model was the slowest. In this experiment, all the validation tests started at the same time without blocking each other, with the MLP model finishing first, the SVR model second and the KNN model last. The timeline comparison section illustrates how these models performed differently: the MLP and SVR models performed similarly, but the KNN model performed significantly differently.

Figure 5.11 shows the test screen, which is similar to the validate screen and enables users to upload a test set to compare with the prediction results instead of running a validation method.

For the test screen, the results were slightly different. This time, the SVR model performed better in relation to the test set, and the KNN model maintained the worst position. As for running time, the KNN model had the best speed.

Finally, Figure 5.12 shows the predict screen, which is similar to the validate and test screens. It differs from the test screen by the absence of measured results.

In the predict screen, the results are very similar to those from the test screen because the test and predictive sets are similar despite the absence of labels in the predictive set. However, in this case, the models' performance cannot be measured because no measured label is provided.

5.4 Summary

With the growing amount of data available, companies and researchers are demanding feasible and affordable ways to extract knowledge from all this data. This thesis has presented a novel architecture for machine learning as a service based on SCA and focusing on predictive modeling. The proposed architecture can support multiple data sources and create various models with different algorithms, parameters, and training sets.

To prove the concept, we build Sibyl and used it to predict energy consumption using real-

world data. Once the main architecture is working and at least one algorithm coded, it is simple to implement other algorithms. It is possible to execute multiple models concurrently.

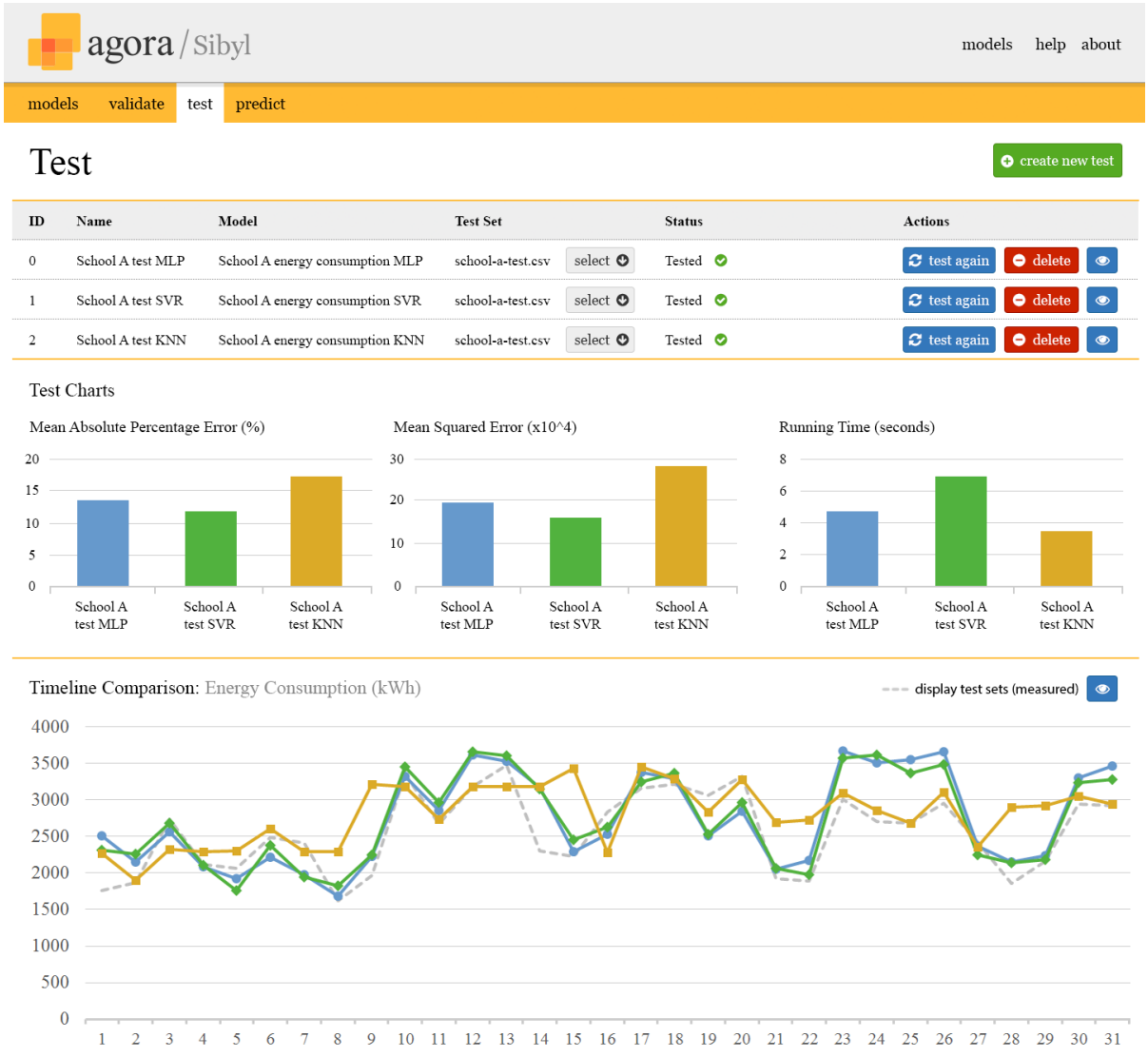


Figure 5.11: Sibyl's test screen.

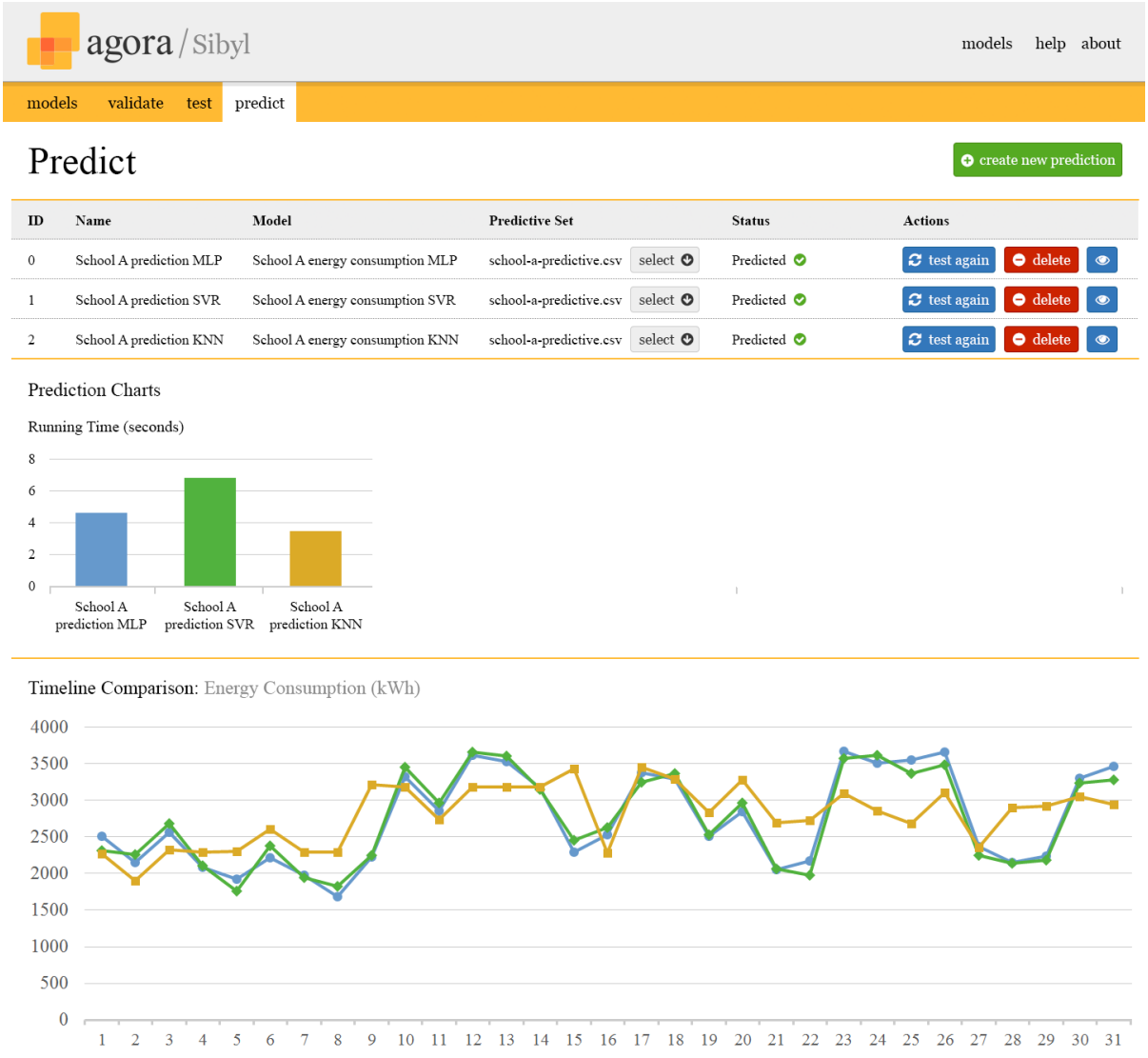


Figure 5.12: Sibyl's predict screen.

Chapter 6

Conclusions and Future Work

This chapter provides a review based on conclusions made about Agora, the knowledge marketplace. In particular, it covers the components discussed in this thesis, which are Hephaestus, the domain adaptation method, and Sibyl, the machine learning as a service. In addition, a description of possible future work involving Agora and its components will be presented.

6.1 Conclusions

This thesis has presented Agora, a knowledge marketplace. Agora is a novel Web-based software as a service platform. Agora enables providers to share their data and machine learning specifications with consumers, which enables consumers with little knowledge and small datasets to make use of providers' data and specifications to create accurate machine learning models. An overview of the architecture has been presented, including its components, their roles, and their relationships, and two of its main components were detailed.

The first component detailed was Hephaestus, a novel transfer learning method for multi-feature regression with seasonal adjustments. Hephaestus works in the pre- and post-processing phases, enabling standard machine learning algorithms to be used. This method can adapt the target variable domain of multiple datasets by removing the effects of time, and adapts features by a simple z-score domain adaptation technique. To demonstrate the efficiency of Hephaestus, we successfully presented a case study on energy consumption prediction for multiple schools, comparing different models using Hephaestus and standard machine learning. Models using

Hephaestus and only one month of data from the target school had similar or better accuracy than standard models using 12 months of data from the target school. For example, one experiment reduced the mean absolute percentage error by more than half, from 21.05% to 9,86%, an improvement of 11.19%. These results show how Hephaestus is capable of improving performance of predictive models with small target domain datasets.

The second component presented was Sibyl, which is a scalable, flexible, and non-blocking machine learning as a service platform for building machine learning models on demand. Sibyl is based on service component architecture (SCA), taking advantages of service oriented architecture (SOA). The proposed architecture can support multiple data sources and create various models with different algorithms, parameters, and training sets. To prove the concept, the system was built to predict electricity demand using real-world data. Once the main architecture is working and at least one algorithm coded, it is simple to implement other algorithms. It is possible to execute multiple models concurrently.

Both Hephaestus and Sibyl are important parts of the whole Agora platform. Their functionalities and characteristics combined enables Agora to provide a machine learning as a service with transfer learning capabilities. The results obtained from both Hephaestus and Sibyl are highly satisfactory, proving the concept of Agora. Consumers can share the same platform to find the knowledge to create many predictive models concurrently without interfering with predictive models from other consumers.

6.2 Future Work

The scope of this thesis was reduced to ensure the quality of the content within the time available for its the development.Because of this, many aspects or conditions were not taken into account when doing this research and were instead left open as new research opportunities.

This thesis has focused on predictive regression models, which is by itself an interesting topic with a wide range of possible applications. For example, it is possible to extend Agora to other machine learning tasks such as classification and clustering, developing more variations of Hephaestus or other methods and improving Sibyl to support them.

This thesis has presented Hephaestus as a transfer learning method for multi-feature regres-

sion models with seasonal adjustment. To extend Agora to work with other machine learning tasks, Hephaestus must be adapted or extended. As previously described in related studies, many transfer learning methods for machine learning tasks other than regression models have already been proposed in the literature. These methods from the literature could be added to the Hephaestus component to support other machine learning types such as classification and clustering.

In addition, the Hephaestus method proposed in this thesis works only in a transductive transfer learning setting, leaving aside inductive transfer learning and unsupervised transfer learning settings. As future work, new methods to implement regression models in these two settings could be developed.

In this research, Sibyl was designed to support multilayer perceptron (MLP), support vector regression (SVR) and k-nearest neighbors (KNN) algorithms. For future work, it would be interesting to add other regression algorithms as well as algorithms from other machine learning categories such as classification and clustering. Because Sibyl is modularized and new services can be coupled to it, it is possible to create and attach new components with new machine learning algorithms. Some algorithms require different pipelines from the standard proposed in this thesis, which is an issue that can be addressed by Sibyl's flexible and modularized architecture.

As mentioned earlier, the Argus component, which is responsible for finding similar datasets, was left out of the scope of this work for now because of limited time. As future project, Argus should be discussed. Once Argus receive a target dataset from a consumer, it should be able to perform a search in the providers database to find the most similar datasets to use in a new prediction. A simplified solution could be to use metadata describing the dataset and what is to be predicted. However, this naive approach could result in inaccurate predictions because even if the same target variable (*e.g.*, energy consumption) is analyzed in different domains, its distributions are not necessarily the same because of the different aspects they may represent (*e.g.*, comparing the energy consumption of a school and a house). This issue could be solved by adding a huge number of properties and descriptions inside the metadata. However, this would be time-consuming for users and covering all the possibilities would be difficult. A better solution would be to measure the distance between distributions and check for those with smaller distances from each other.

Two main challenges can be identified for Argus. The first is how to verify the distance between two or more datasets. As was seen in related work described in section 2.2.1, some approaches already exist to calculate the distance between distributions. These methods could be used as a basis to create Argus. The second question is how to make it fast enough to compare one target dataset with a lot of others from the providers' database. Once metadata are available, the number of datasets to be compared can be drastically decreased depending on the number of properties that are described in the metadata. There must be an optimal and efficient number of properties that should be considered in metadata to balance the issues of the number of datasets involved in the comparison and the complexity of the metadata themselves.

Also left out of scope, the database must be properly designed in the future. In this thesis, Hephaestus and Sibyl were implemented without access to any database, being directly fed with the data. Some considerations must be kept into account when designing this database. For example, how much data must the database support? If considering too many users with big datasets each, a Big Data approach can be considered. Another consideration is how to address relationships between the entities proposed in the architecture that are important to the user and to the system. The relationships may require extra table attributes storing foreign keys for 0xN relationships (e.g. a provider can have many datasets) or even extra tables for NxN relationships (e.g. many global data can be used in many different models).

When dealing with data, challenges in security and privacy arise. Some security issues can be discussed in future work. For example, only registered consumers can visualize Agora content or make use of Agora features and only consumers that have paid a provider or have authorization from a provider can make use of a provider's knowledge. These can be addressed using current technologies for database encryption, Web security, user authentication and user access control. In this way, only registered users with proper authorization can access specific content.

Other security and privacy aspects can be discussed including: providers want to share knowledge, but not data; and piracy. In other words, they do not want other users to have access to their data, but only to use their data and their machine learning specifications. One reason for this, for example, could be the providers' need to hide data for security reasons or to keep them secret from their competitors. A possible hypothesis is that machine learning

algorithms can be seen as a one-way process that does not let someone to reverse-engineer the results to build the original dataset. Moreover, the Agora architecture allows consumers to retrieve data only from the Hephaestus post-processing phase, which is already encrypted, meaning that consumers do not have direct access to providers' data. With both this hypothesis and these facts in hand, it is clear that gaining access to provider's data and copying them would be very difficult. As future work, this hypothesis could be verified.

Privacy also depends on the way Argus, the similarity search component, is designed. For example, in a simple design where Argus verifies if two datasets from a consumer and a provider contain the same exact values, a consumer could find out the provider's data through various attempts guessing the provider's dataset until finding a fit. Despite, at least in this case, the chances are very small, the possibility shows that this privacy issue must be addressed when designing Argus.

Another privacy issue is that, in some cases, data can be extremely personal or secret and cannot be shared. For example, in some fields such as health and finances, data cannot be sent to external resources such as Agora due to high restrictions established by their owners or organizations. However, some data are allowed to be published by de-identifying the data (*e.g.*, removing parts of the data or aggregating values by location). It is necessary to investigate if it would impact the performance of predictive models in Agora.

The contribution of this thesis considers the technical aspects of Agora only. The business and economic aspects can be investigated regarding how to attract users to the marketplace and how to monetize it. Populating Agora is a problem similar to the chicken and the egg dilemma: consumers will come up once providers join the marketplace and vice-versa. How to bring the first users to Agora? One idea would be to start from a specific niche (e.g. energy consumption) that data is easily available and have a greater potential to attract users. Once early adopters and first users are using Agora, it must have an attractive business model. Freemarket and opendata models can possibly work, attracting non-profit organizations and altruistic providers. However, to attract providers and compensate for sharing their knowledge, an evident business model is to reward providers with cash, which will probably impact users' pocket and interest in using the system.

One feature that was not considered in this thesis but can be in future work is to use con-

sumers' data to improve providers' model. So far, the idea of Agora was to use the data from providers to create or improve consumers' models. What if consumers can be sending data to Agora that may be potentially valuable for providers to enhance the accuracy of their models? In this way, providers would be acting as consumers and vice-versa. Hence, this would probably bring two major impacts: new privacy issues, considering that may be consumers are not open to sharing their data; new business models where providers would pay to have this benefit in case of a paid market, or even a free market where both providers and consumers are benefited.

In this thesis, the Hephaestus' case study was based a one-month time frame for the consumer's data. Hephaestus could be evaluated and adapted if necessary to use smaller time frames such as seven days only. Because the case study implementation in Hephaestus works with seasonal adjustment and considers weekly seasons, a small number of days do not have enough data to calculate seasonality. This issue, however, can be solved by using the seasonal indexes from the most similar domain.

Although there are many possibilities for future work, this thesis proves the concept of Agora as a marketplace for machine learning to transfer the knowledge from one user to another. This thesis provides a starting point to build a marketplace for any kind of knowledge for machine learning. For example, it could be expanded for other learning types (*e.g.*, clustering and classification) or adapted for specific purposes beyond a public marketplace (*e.g.*, within a retail organization to predict sales for a new branch using data from other branches). It is clear that many applications exist for Agora that end up creating new research opportunities.

Bibliography

[1] Service Component Architecture Assembly Model Specification Version 1.1. http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.html. [Online; accessed 30-April-2015].

[2] Microsoft Time Series Algorithm Technical Reference. https://msdn.microsoft.com/en-us/library/bb677216.aspx, 2016. [Online; accessed: 06-April-2016].

[3] A. S. Ahmad, M. Y. Hassan, M. P. Abdullah, H. A. Rahman, F. Hussin, H. Abdullah, and R. Saidur. A review on applications of ANN and SVM for building electrical energy consumption forecasting. *Renewable and Sustainable Energy Reviews*, 33:102–109, 2014.

[4] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[5] Alejandro Baldominos, Esperanza Albacete, Yago Saez, and Pedro Isasi. A scalable machine learning online service for big data real-time analysis. In *Computational Intelligence in Big Data (CIBD), 2014 IEEE Symposium on*, pages 1–8, Orlando, FL, December 2014. IEEE.

[6] Peter C Bruce. *Introductory Statistics and Analytics: A Resampling Perspective*. John Wiley & Sons, 2014.

[7] Thomas Calmant, João Claudio Américo, Didier Donsez, and Olivier Gattaz. A dynamic sca-based system for smart homes and offices. In *Service-Oriented Computing-ICSOC 2012 Workshops*, pages 435–438, Shanghai, China, November 2012. Springer.

[8] Simon Chan, Thomas Stone, Kit Pang Szeto, and Ka Hou Chan. PredictionIO: a distributed machine learning server for practical software development. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2493–2496, San Francisco, CA, October 2013. ACM.

[9] Corinna Cortes and Mehryar Mohri. Domain adaptation and sample bias correction theory and algorithm for regression. *Theoretical Computer Science*, 519:103–126, 2014.

[10] Hal Daumé III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting*, pages 256–263, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[11] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Pearson Education India, 2005.

[12] Hans Fischer. *A history of the central limit theorem: From classical to modern probability theory*. Springer Science & Business Media, 2010.

[13] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2066–2073, Providence, RI, June 2012. IEEE.

[14] Samuel B Green. How many subjects does it take to do a regression analysis. *Multivariate behavioral research*, 26(3):499–510, 1991.

[15] Wei-Chiang Hong. Electric load forecasting by seasonal recurrent svr (support vector regression) with chaotic artificial bee colony algorithm. *Energy*, 36(9):5568–5578, 2011.

[16] Hamidreza Hosseinzadeh, Farbod Razzazi, and Ehsanollah Kabir. A weakly supervised large margin domain adaptation method for isolated handwritten digit recognition. *Journal of Visual Communication and Image Representation*, 38:307–315, 2016.

[17] Wenping Hu, Yao Qian, Frank K Soong, and Yong Wang. Improved mispronunciation detection with deep neural network trained acoustic models and transfer learning based logistic regression classifiers. *Speech Communication*, 67:154–166, 2015.

[18] Xuegang Hu, Jianhan Pan, Peipei Li, Huizong Li, Wei He, and Yuhong Zhang. Multi-bridge transfer learning. *Knowledge-Based Systems*, 97:60–74, 2016.

[19] Svend Hylleberg. *Seasonality in regression*. Academic Press, 1986.

[20] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2014.

[21] Rob J Hyndman and Andrey V Kostenko. Minimum sample size requirements for seasonal forecasting models. *Foresight*, 6:12–15, 2007.

[22] Svetlana Kiritchenko, Xiaodan Zhu, Colin Cherry, and Saif Mohammad. Nrc-canada-2014: Detecting aspects and sentiment in customer reviews. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 437–442, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University Dublin, Ireland.

[23] Pieter M Kroonenberg. *Applied multiway data analysis*. John Wiley & Sons, 2008.

[24] Wen Li, Lixin Duan, Dong Xu, and Ivor W Tsang. Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(6):1134–1148, 2014.

[25] Feng-Cheng Lin, Lan-Kun Chung, Wen-Yuan Ku, Lin-Ru Chu, and Tien-Yin Chou. Service component architecture for geographic information system in cloud computing infrastructure. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 368–373, Barcelona, Spain, March 2013. IEEE.

[26] Chi-Chun Lo, Ding-Yuan Chen, and Kuo-Ming Chao. Dynamic data driven smart home system based on a service component architecture. In *Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on*, pages 473–478, Shanghai, China, April 2010. IEEE.

[27] Ying Ma, Guangchun Luo, Xue Zeng, and Aiguo Chen. Transfer learning for cross-company software defect prediction. *Information and Software Technology*, 54(3):248–256, 2012.

[28] Christopher Meek, David Maxwell Chickering, and David Heckerman. Autoregressive tree models for time-series analysis. In *Proc. 2nd Intl. SIAM Conf. on Data Mining*, pages 229–244, Arlington, VA, April 2002. SDM.

[29] Leandro L Minku and Xin Yao. How to make best use of cross-company data in software effort estimation? In *Proceedings of the 36th International Conference on Software Engineering*, pages 446–456, New York, NY, May 2014. ACM.

[30] Azadeh Sadat Mozafari and Mansour Jamzad. A svm-based model-transferring method for heterogeneous domain adaptation. *Pattern Recognition*, 56:142–158, 2016.

[31] Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 382–391, San Francisco, CA, May 2013. IEEE Press.

[32] Jeroen Ooms. The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns. *arXiv:1406.4806*, (2000):1–23, 2014.

[33] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *Neural Networks, IEEE Transactions on*, 22(2):199–210, 2011.

[34] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[35] Kyounghyun Park, Minh Chau Nguyen, and Heesun Won. Web-based collaborative big data analytics on big data as a service platform. In *2015 17th International Conference on Advanced Communication Technology (ICACT)*, pages 564–567, Pyeongchang, South Korea, July 2015. IEEE.

[36] Fernando Silva Parreiras, Gerd Gröner, Daniel Schwabe, and Fernando de Freitas Silva. Towards a marketplace of open source software data. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on*, pages 3651–3660, Koloa, HI, January 2015. IEEE.

[37] Vishal M Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual domain adaptation: A survey of recent advances. *Signal Processing Magazine, IEEE*, 32(3):53–69, 2015.

[38] Dorian Pyle. *Data preparation for data mining*. Morgan Kaufmann, 1999.

[39] Soheil Qanbari, Navid Rekabsaz, and Schahram Dustdar. Open government data as a service (godaas): Big data platform for mobile app developers. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 398–403, Rome, Italy, August 2015. IEEE.

[40] Jiang Qifa. A solution to seasonal adjustment forecasting for hydraulic smes in investment decision-making. In *Control and Decision Conference (2014 CCDC), The 26th Chinese*, pages 724–729, Changsha, China, May 2014. IEEE.

[41] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[42] Mauro Ribeiro, Katarina Grolinger, and Miriam AM Capretz. Mlaas: Machine learning as a service. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 896–902. IEEE, 2015.

[43] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[44] Thomas Stewart and Clare Ruckdeschel. Intellectual capital: The new wealth of organizations, 1998.

[45] Biao Sun, Peter B Luh, Qing-Shan Jia, Ziyan Jiang, Fulin Wang, and Chen Song. Building energy management: integrated control of active and passive heating, cooling, lighting, shading, and ventilation systems. *Automation Science and Engineering, IEEE Transactions on*, 10(3):588–602, 2013.

[46] Jianzhou Wang, Wenjin Zhu, Wenyu Zhang, and Donghuai Sun. A trend fixed on firstly and seasonal adjustment model combined with the ε-svr for short-term forecasting of electricity demand. *Energy Policy*, 37(11):4901–4909, 2009.

[47] Makoto Yamada, Leonid Sigal, and Yi Chang. Domain adaptation for structured regression. *International Journal of Computer Vision*, 109(1-2):126–145, 2014.

[48] Mehmet Yesilbudak, Seref Sagiroglu, and Ilhami Colak. A new approach to very short term wind speed prediction using k-nearest neighbor classification. *Energy Conversion and Management*, 69:77–86, 2013.

[49] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the twenty-first international conference on Machine learning*, page 114, Banff, Canada.

[50] Yu Zheng. Methodologies for cross-domain data fusion: an overview. *Big Data, IEEE Transactions on*, 1(1):16–34, 2015.

[51] Fan Zhu and Ling Shao. Weakly-supervised cross-domain dictionary learning for visual recognition. *International Journal of Computer Vision*, 109(1-2):42–59, 2014.

Curriculum Vitae

Name:	Mauro Ribeiro
Year of Birth:	1985
Place of Birth:	Campinas, SP - Brazil
Post-Secondary Education and Degrees:	University of Western Ontario London, ON 2014 - 2016 MESc University of Campinas Campinas, SP - Brazil 2004 - 2008 BSc
Related Work Experience:	Teaching Assistant The University of Western Ontario 2014 - 2016 Front-End Developer Strategy Manager (Brazil) 2014 Co-founder and Director of Technology Empreendemia (Brazil) 2009 - 2016 Designer and Front-End Developer Pinuts Studios (Brazil) 2008 - 2009 Web-Developer and Web-Designer Pinuts Studios (Brazil) 2006 - 2007

Publications:

M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "*MLaaS: Machine Learning as a Service*". In IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pages 896–902, Miami, Florida, December 2015.

M. Ribeiro, K. Grolinger, H. F. ElYamanya, and M. A. M. Capretz, "*Hephaestus: Domain Adaptation for Multi-Feature Regression with Seasonal Adjustment*". Submitted to Elsevier Applied Soft Computing.