Gonçalo Brito Infante

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Online platform for building, testing and deploying predictive models

Dissertação para Obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Tiago Oliveira Machado de Figueiredo Cardoso, Professor Doutor, FCT-UNL

Júri:

Presidente: Paulo da Costa Luís da Fonseca Pinto, Professor Doutor, FCT-UNL

Arguentes: Ana Inês Oliveira Machado de Figueiredo Cardoso, Professor Doutor, FCT-UNL

Vogais: Tiago Oliveira Machado de Figueiredo Cardoso, Professor Doutor, FCT-UNL

Dezembro 2017

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Online platform for building, testing and deploying predictive models

Copyright © Gonçalo Brito Infante, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

A Faculdade de Ciências e Tecnologia e a Universidade Nova de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Para família e amigos

Agradecimentos

Quero começar por agradecer a todas as pessoas que me motivaram e ajudaram ao longo da realização desta dissertação.

Gostaria de agradecer também ao meu orientador, o Professor Doutor Tiago Cardoso, pela possibilidade realizar esta dissertação nesta área específica e de ter a sua orientação.

Um enorme agradecimento também aos meus amigos mais próximos que sempre me ajudaram e acompanharam durante todo o percurso académico, partilhando comigo momentos inesquecíveis. Um agradecimento especial também a todos os colegas que se cruzaram comigo durante estes seis anos.

Por fim, agradeço à minha família, em especial aos meus pais e irmão, que me apoiaram e motivaram desde o inicio desta caminhada, permitindo a chegada a bom porto.

A todos vós,

O meu sincero e enorme Muito Obrigado!

Resumo

Tipicamente, “aprendizagem de máquina” e inteligência artificial têm sido implementadas e executadas “manualmente” apartir de uma única máquina local, utilizando ferramentas como R ou Weka. Os tempos estão a mudar e na era dos serviços em tempo-real e da big data, estes metodos estão a ficar obsolentos, uma vez que limitam severamente a aplicabilidade e executabilidade das técnicas de “aprendizagem máquina”. Muitas companhias como a Microsoft, Amazon e Google têm tentado mitigar este problema desenvolvendo a suas soluções de MLaaS(*Machine Learning as a Service),* que são aplicações online escaláveis e que automatizam o densenvolvimento de modelos preditivos.

Apesar da existência de algumas plataformas de “aprendizagem de máquina” na nuvem que permitem ao utilizador desenvolver e executar processos de “aprendizagem máquina”, estes não estão otimozados para o utilizador prototipar e executar modelos preditivos, uma vez que alguns passos complexos têm de ser feitos antes de se utilizar a plataforma, como configuração de ambientes, configuração de contas e ultrapassagem da curva de aprendizagem dos mesmo.

Neste trabalho apresentamos o MLINO, um conceito de uma plataforma online que permite ao utilizador a rápida prototipagem e execução de processos simples de “aprendizagem máquina”, de uma forma intuitiva e fácil. Embora a implementação do protótipo não tenha sido a mais indicada devido a limitções de software e de infrastrutura, através de uma série de testes, foi demonstrado que a performance do protótipo foi satisfatória. Ao compararmos a solução desenvolvida com o Microsoft Azure ML, concluímos que o protótipo é mais facil de utilizar e demora menos tempo a construir e a correr um modelo preditivo simples.

Palavras-chave: Aprendizagem máquina; Previsões; Plataforma online; Sistema flexível; Aprendizagem Supervisionada; Classificação.

Abstract

Machine Learning (ML) and Artificial Intelligence (AI) have been traditionally built and deployed manually in a single machine, using tools such as R or Weka. Times are changing and in the real-time service and big data era, this methods are being obsoleted, as they severely limit the applicability and deployability of ML. Many companies such as Microsoft, Amazon and Google have been trying to mitigate this problem developing their MLaaS (Machine Learning as a Service) solutions, which are online platforms capable to scale and automate the development of predictive models.

Despite the existence of some ML platforms available in the cloud, that enable the user to develop and deploy ML processes, they are not suitable for rapidly prototype and deploy predictive models, as some complex steps need to be done before the user starts using them, like configuration of environments, configuration of accounts and the overcome of the steep learning curve.

In this research project, it's presented MLINO, which is a concept of an online platform that allows the user to rapidly prototype and deploy basic ML processes, in an intuitive and easy way. Even though the implementation of the prototype wasn't the optimal, due to software and infrastructure limitations, through a series of experiments it was demonstrated that the final performance of the prototype was satisfactory. When benchmarking the devised solution against the Microsoft Azure ML, the results showed that MLINO tool is easier to use, and takes less time when building and deploying a basic predictive model.

Keywords: Machine learning; Predictive models; Online platform; Flexible system; Supervised-learning, Classification.

Acronyms

1-NN	1-nearest-neighbor
AI	Artificial Intelligence
BI	Business Intelligence
BPN	Back Propagation Neural Network
EPL	English Premier League
GUI	Graphic User Interface
ML	Machine Learning
MLaaS	Machine Learning as a Service
PSI	Protocols and Structures for Inference
REST	REpresentational State Transfer
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SOM	Self-Organizing Map
SVM	Support Vector Machine
XML	eXtensible Markup Language

Terminology

MLINO – Special word created to name the concept developed in this research work. As this work was focused in developing a concept for an online framework for prototyping machine learning solutions, MLINO is a mix between machine learning and Arduino (platform for prototyping hardware) words.

Table of Contents

1 Introduction 1
1.1 Motivation 1
1.2 The Problem 3
1.3 Contributions 5
1.4 Document Organization 7
2 State of Art 9
2.1 Machine Learning Fundamentals 10
2.2 Offline Machine Learning solutions 14
2.3 Online Machine Learning solutions 18
2.4 Discussion 20
3 System Architecture 21
3.1 Principles and Assumptions 22
3.2 High level architecture view 22
3.3 ML engine 25
3.4 GUI platform (web application) 28
3.5 Data Storage 30
3.6 ML Workflows structure 31
3.7 Model Performance Evaluation 32

3.8 Final Considerations 33
4 Implementation 35
4.1 Used Technologies 35
4.2 Prototype Architecture 37
4.3 Machine Learning Algorithms 39
4.4 GUI Platform 39
4.5 ML Engine 46
4.6 ML Workflow Structure 54
4.7 Class Diagram 56
4.8 Final Considerations 58
5 Evaluation 59
5.1 Performance Test 60
5.2 Benchmarking 64
5.3 Discussion 66
6 Conclusions and Future Work 67
6.1 Conclusions 67
6.2 Future Work 68

List of Tables

Table 1 Roles of each web page 41
Table 2 Data model tables description 44
Table 3 GUI platform REST methods overview 45
Table 4 ML Engine REST methods overview 46
Table 5 Assets stored in each warehouse 54
Table 6 Test result for a small dataset 65
Table 7 Test result for a bigger dataset 65
Table 8 User average number of mouse clicks to accomplish the test 66

List of Figures

Figure 1 The spectrum of BI technologies (Eckerson, 2007) 2
Figure 2 Responder who have implemented predictive analytics (Eckerson, 2007) 10
Figure 3 Categorization of the different ML learning methods (Ribeiro et al., 2015) ... 12
Figure 4 The learning process 13
Figure 5 Process of predictive model creation (Hung, Yen and Wang, 2006) 16
Figure 6 Comparison of error rates between models 17
Figure 7 High level framework architecture 23
Figure 8 Relation between experiment, model and prediction 24
Figure 9 ML engine overview 25
Figure 10 Interaction of the Workflow Build with the other system components 27
Figure 11 Interaction of the Workflow Executor with the other system components 28
Figure 12 Interaction of the GUI platform with the other system components 29
Figure 13 Conceptual experiment workflow (Fallis, 2013) 32
Figure 14 Architecture overview of the MLINO prototype 38
Figure 15 Screenshot example of the MLINO prototype visual interface. 40
Figure 16 High level pages disposal diagram 41
Figure 17 Example of implementation of read operation 43
Figure 18 Creation of a new experiment 43
Figure 19 Web application data model 44
Figure 20 Supported REST methods by the GUI platform 45
Figure 21 Supported REST methods by the ML Engine API 46
Figure 22 Activity diagram of get file job 48

Figure 23 Activity diagram of upload dataset job 49
Figure 24 Activity diagram of process label job 50
Figure 25 Activity diagram of run experiment job 52
Figure 26 Activity diagram of run prediction job 53
Figure 27 Experiment workflow 55
Figure 28 Prediction workflow 56
Figure 29 ROC chart workflow 56
Figure 30 ML engine class diagram 57
Figure 31 Performance test scenario 1 60
Figure 32 Performance test scenario 2 61
Figure 33 Performance test scenario 3 62
Figure 34 Performance test scenario 4 63
Figure 35 Performance test scenario 5 64

1

Introduction

1.1 Motivation

Technology have a huge role in modern society, where evolution is constant. From our smartphone, to wearable devices or to the sensors in our cars, data is plentiful and can be found everywhere. So we can say the world contains an incredible vast amount of digital data which is increasing exponentially over time. This turns possible to do things that previously couldn't be done: detect fraud, optimize marketing campaigns, improve operations, predict future infectious diseases or even predict sport events and so on. Managed well, the data can be used to propel businesses, improve life and help people.

The need to understand past events become a subject that today we call Business Intelligence (BI), which supports planning and decision making based on statistics gathered from historical data (Eckerson, 2007). Typical examples are:

- How many clients left my company in the past months?
- Where are my best clients located?
- How a disease is affecting a patient relative to normal care delivery processes?
- Why there's a shortage of flu vaccine?
- How data gathered from sport players can help to prevent injuries?

As a result of the increasing complexity of the actual business scenario, it's safe to say that using the traditional BI techniques are not enough. The insights provided by the historical data still lead to empiric and intuition based decisions. While in a stable business scenario this way of acting might be acceptable, on other hand, in a dynamic scenario ever evolving, probably this method of decision making is not enough and makes harder and complex the task of turning data into useful insights. One might say that other methods to process it, in a faster, accurate and more reliable way need to be used.

The Figure **1** exposes the spectrum of BI technologies, mapping business value to complexity.

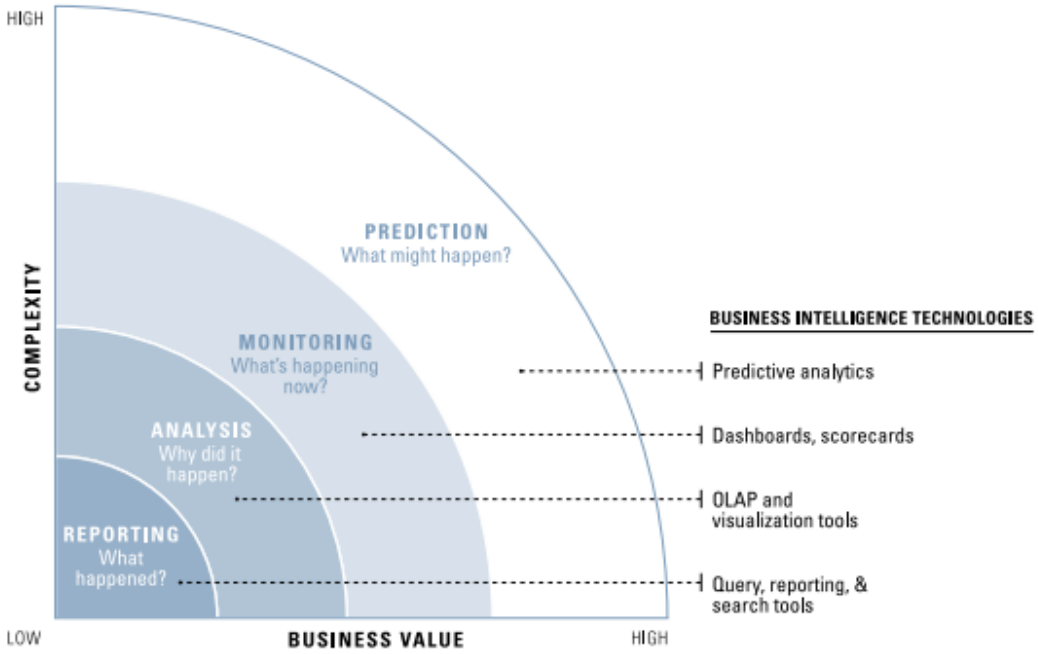

Figure 1 The spectrum of BI technologies (Eckerson, 2007)

This is where predictive analysis might have a big impact. This technique has been around for some time now in the business world, but only now is being widely used due to several factors:

- growing volumes and types of data;
- Faster data processing capability,
- Easier-to-use software;
- Organizations need and are interested in producing valuable insights in order to potential secure a competitive advantage versus their competitors;

1.2 The Problem

Large organizations have enough resources to create and invest in predictive analysis solutions adapted to their problems, however small companies, startups, students and researchers in general have difficulty in building their predictive model, mainly due to the learning curve of working with machine learning or because of impracticable costs of the required hardware systems (Ribeiro *et al.*, 2015).

Machine Learning (ML) and Artificial Intelligence (AI) have been traditionally built and deployed manually in a single machine using tools such R or Weka. Time are changing and in the real-time service and big data era, this methods are becoming obsoleted, as they severely limit the applicability and deployability of ML. Many companies such as Microsoft, Amazon and Google have been trying to mitigate this problem developing their MLaaS (Machine Learning as a Service) solutions, which are online platforms capable to scale and automate the development of predictive models (Li *et al.*, 2017). they are not suitable for rapidly prototype and deploy predictive models

All the studied ML online platforms are not suitable for rapidly prototype and deploy of predictive models, as before they can be used, some complex steps need to be done, like configuration of environments, configuration of accounts and the overcome of the steep learning curve.

Despite the existing variety of techniques available to build and test machine learning models (predictive models), there is still room for innovation in this area. Considering that an online ML solution with the purpose of prototyping and testing machine learning models should be easy to use and should only take a few minutes since the account creation in the platform until

the first model is deployed. This project doesn't pretend to be another MLaaS solution, but a prototyping ML platform that is easier to use and takes less time to build, test and run basic predictive models, if comparing to the other ML tools.

Different models have different performances and outcomes, depending on the dataset, algorithms and parameters used. One of the project goals is to devise a solution that allows the user to run multiple supervised ML algorithms and compare their performances, so the most suitable algorithms can be selected and deployed as a predictive model.

Furthermore, the platform seeks to abstract away most of the complexity of building, testing and running predictive models, making it easier and faster to do it.

1.3 Contributions

This work focused on developing MLINO: an online ML concept where users can get started quickly with machine learning, without having to install additional software or to have their own infrastructure. This solution abstracts away most of the technical complexity of implementing a ML system, and allows in a matter of minutes prototype, test and deploy simple classifications models, automatically from the platform, without the need to use any code, or to manually train or tune the ML learning algorithms.

The project was divided in two different phases:

1. The objective of the first phase was to develop an environment that would allow the user to dynamically build a ML model, which given a ML algorithm and a test dataset, the system would automatically train it, validate it and present the final results to the user. Also in this step, was implemented the method of deploying a predictive model given a trained ML model.
2. In the second phase the goal was to devise an online platform, in which the user would use as a control center of his experiments. An experiment is the name given to the website object that is composed by up to two ML models, respective performances and associated deployed predictions. The user can setup as many experiments as he wants.

After setting up an experiment, the user can start the process of prototype and compare the performance up to two different ML models. The ML predictive model can be built using the available ML supervised algorithms available in the platform. In each experiment, after the user build and test the ML models, according to its performance a model can be chosen, deployed and used to provide predictions. Additionally, for the same experiment and given a prediction dataset[1], the user can deploy several predictions and download their result as a excel file.

Using technologies available to anyone in the internet, the MLINO prototype was implemented to prove the feasibility of the solution. Finally, the project was submitted to several performance tests and benchmarked against other solutions available online. The results suggested that although the platform doesn't allow users to solve complex predictive problems and doesn't allow the user to set advanced parameters of the ML algorithms (because it is an automatic process and the goal was to abstract the technical complexity behind a ML predictive

[1] Dataset used to run the predictive model and get predictions.

system), it showed that the MLINO platform is easier to use and gives faster results compared to other platforms.

1.4 Document Organization

The remaining of this document is organized in the following schema:

- Chapter 2 – Covers the State of Art and so presents the work related to this project. The initial sub-section covers some basic notions about machine learning. The remaining sub-sections distinguish different machine learning online and offline approaches.

- Chapter 3 – Covers the conceptual architecture proposed for the system. In this chapter is detailed the architecture and goals behind the GUI platform, ML engine and data storage modules, that are the three main components of this project.

- Chapter 4 – Presents a detailed description of the MLINO implementation, as well the reasoning behind it. It is presented the technologies used, then the implantation detail behind each architecture component.

- Chapter 5 – In this section is present the validation results of prototype developed. Starting to present the system behavior when exposed to different workload and then benchmarking the solution performance to other tools available in the internet.

- Chapter 6 – Summarizes the conclusions of this project and cover a perspective for the future work that wasn't cover in this essay.

2

State of Art

Predicting the results of an event is an interesting area to many, from enthusiastic to organizations. Predictive analytics is part of the BI technologies spectrum that makes use of past events to anticipate the future, uncovering hidden data trends and discovering new data patterns and insights within large volume of data (Eckerson, 2007).

Nowadays, technology management, planning and decision making based in predictive analytics can be an essential business tool, where the intelligence models create predictions by integrating several techniques such as data mining, machine learning and statistical modeling. This techniques can be applied to a wide range of business strategies and has been a key player in search advertising and recommendation engines, providing executives and manager with decision-making insights.

As exposed in the Figure 2, 66% of the responders who implemented predictive analytics are “high” or “very high” satisfied with the value that it provided for their businesses.

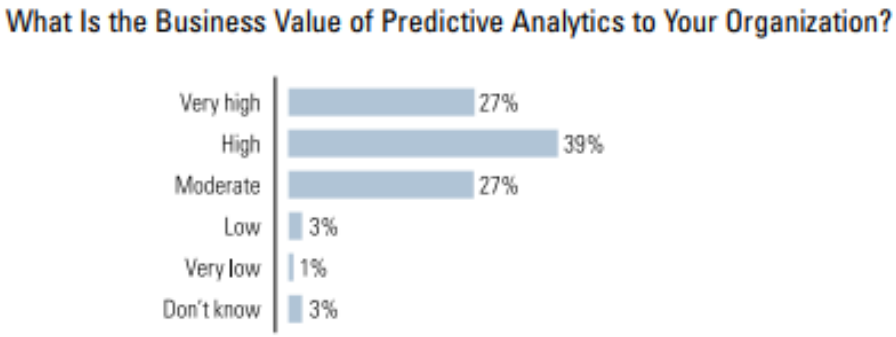

Figure 2 Responder who have implemented predictive analytics (Eckerson, 2007)

For the above reasons, machine learning and predicative analytics domains have led to a growing interest among researchers and organizations, which have been investigating and developing different approaches to make this systems faster, reliable and scalable in order to solve the modern society challenges. In this chapter are presented a set of systems and approaches related to the theme e somehow related to the work developed.

2.1 Machine Learning Fundamentals

As stated in the last chapter ML have innumerous applications that range from businesses to people's daily life. In past years all the growing interest in this research area made ML one of the fastest growing field in computer science (Ethem and Alpaydin, 2004). In 1969, Arthur Samuel defined ML as *Field of study that gives computers the ability to learn without being explicitly programmed.* Being part of Artificial Intelligence area, ML is a set of statistical techniques used to build mathematical models that make inferences from data samples (training dataset) (Ribeiro *et al.*, 2015).

Three main types of learning are distinguished in machine learning, as seen in (Stuart and Peter, 2003; Ethem and Alpaydin, 2004):

- Supervised Learning: task of learning from a labeled[2] dataset. This method can handle discrete or continuous labels, using classification and regression algorithms respectively.

[2] When a dataset has the attribute that the model is trying to predict.

Classification attempts to estimate a category of each individual example and it can be used for prediction tasks like pattern recognition, outlier detection or even credit scoring. When there are only two possible categories to be predicted its known as binomial or two-class classification, if there are more categories then is called multi-class classification. When the value being predicted is a number, it is called regression, and this algorithms are mainly used for prediction and ranking tasks, such as foreseeing the price of a used car, predicting stock prices or forecasting the future demand for a company's products.

- Unsupervised Learning: task of learning from an unlabeled dataset, where the goal is to find regularities on the input in order to describe its structure.
- Reinforcement Learning: task of learning by trial-and-error, based in the feedback provided in terms of reward and punishments, leading to sequence of actions that change the environment.

The Figure 3 summarizes the usage of each learning types:

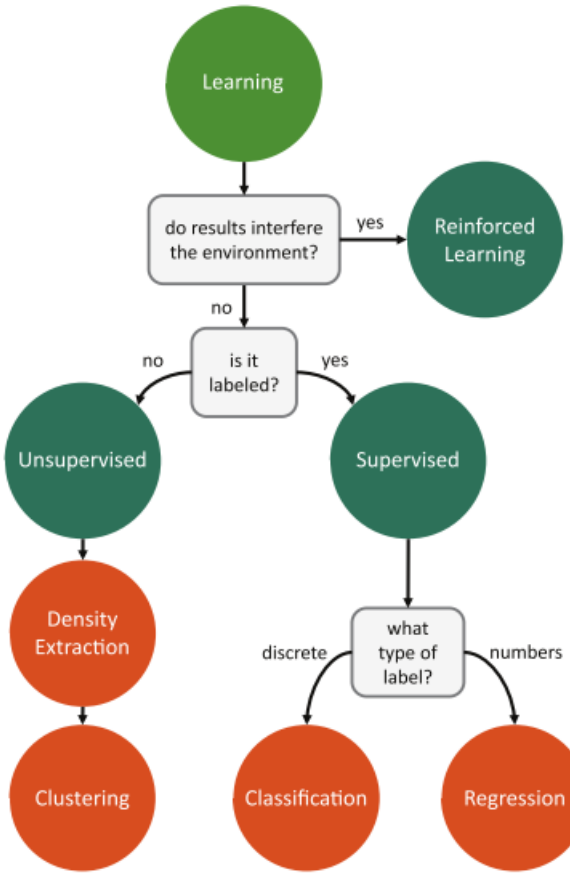

Figure 3 Categorization of the different ML learning methods (Ribeiro *et al.*, 2015)

Although all the different applications just described, machine learning algorithms work in similar way. In general, a ML model is the output generated when a ML algorithm is trained using an example set (dataset). A ML model should then become able to make inferences for a new dataset (Murphy and Kaiser, 2007).

The Figure 4 show an overview of the generic learning process.

Figure 4 The learning process

Making predictions is a tough and complex challenge. Predictions can rely on multiple factors, depending of the application area or depending the magnitude of the problem. The attempt of using ML to develop predictions in commercial, industrial and government applications is called predictive analytics (Siegel, 2013).

The powerhouse organizations of the Internet era, which include Google and Amazon ... have business models that hinge on predictive models based on machine learning.

-Professor Vasant Dhar, Stern School of Business

New York University

After building a predictive model, one last phase should be achieved, the validation phase. This step helps us to estimate the expected error for new predictions and to test the accuracy of the model for the chosen parameters. As seen in (Ethem and Alpaydin, 2004), there are several ways of validating a ML model, among the most popular are:

- K-Fold Cross-Validation: Method used to split the dataset in randomly K parts of the same size. For each K iterations, one of the K folds is used to calculate the training errors while the K-1 is used to train the algorithm. This process is repeated K times, each round using a different fold for validation. This method guarantees that the entire dataset is validated, giving an accurate estimation of the model prediction performance.
- Bootstrapping: Method that create samples with replacement from the original data, and take the "not chosen" data points as test cases. This is done several times, calculating the average score as estimation of the model performance.

It's important to note that for the same dataset, different algorithms produce different outcomes through their respective models, so is imperative to run multiple algorithms and

compare their performances, so the most suitable algorithm can be selected and successfully be applied to solve a predictive problem.

2.1.1 Predictive Model

A predictive model is a mathematic function to best predict the probability of an outcome. A number of modeling methods from machine learning, artificial intelligence, and statistics are available in predictive analytics. The model is chosen on the basis of testing, validation an evaluation using the detection theory to guess the probability of an outcome given set amout of input data. Each model has its own strengths and weakeness and is best suited for particular types of problems.

2.1.2 Workflow

A workflow consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information.

In ML workflows are used to automate learning and deployment tasks, such as applying ML algorithms to the training and test sets, validate and deploy predictive models. In the architecture and implementation sections, it is exposed the role of the workflows in the MLINO concept and prototype.

2.2 Offline Machine Learning solutions

Traditionally ML algorithms are implemented, tested and deployed across a wide range of different specialized toolkits such as Weka, Orange, R, Matlab, Knime, Scikit-learn, RapidMiner, among many others that help the user in all the steps of the data analysis process. This process is characterized to make use of a single machine, where all ML steps like building, validating/testing and deploying an ML model are performed manually by the user in an offline system. Some of these softwares are used standalone, whereas other solutions can be integrated into third party softwares (Ooms, 2014; Li *et al.*, 2017).

Canadian Environmental Hazards Detections System, as the name suggests is a system composed by several tools and techniques that aims to detect environmental hazards. One of its features is to use ML for detection of oil spills in satellite radar images. Using the C4.5 and 1-nearest-neighbor rule (1-NN) classifiers, the researchers attempted to classify a set of satellite images. The authors facing the issue of working with an imbalanced dataset (41 positive examples against 896 negative examples), and because induced classifiers tend to have a bad performance on imbalanced dataset's class distribution, being highly accurate on negative examples and inaccurate on the positive ones, they had to develop a new algorithm that would mitigate this problem and have a better performance (Kubat, Holte and Matwin, 1998).

As seen in (Hung, Yen and Wang, 2006), a framework was developed to assess the performance of various ML techniques when applied to churn[3] prediction. The process of creating a predictive model was divided in the following steps:

- Scope definition: definition of which part of the client population the churn model would focus;
- Exploratory Data Analysis (data extraction and processing and variable selection): data exploration in order to determine the possible variables and behaviors that can differentiate or characterize the costumer behaviors of churning. For this task several interviews were performed to telecommunication consultants, marketing analysts, costumer career and sales provider in order to identify the typical patterns and relationships associated to would take a client to churn.
- Machine learning: The aim of this step was the model creation. Here the authors took two approaches to be assessed. In the approach 1 a K-means method was used to segment the costumer in distinct segments, and then a C5.0 decision tree algorithm was used in each cluster. This implementation aimed to evaluate the churn behavior in the previously defined costumer clusters. In approach 2, Back Propagation Neural network (BPN) was used to segment the costumers, followed by a decision tree model. The goal of this solution was to assess if BPN could improve the decision tree prediction accuracy used in the approach 1.
- Model performance evaluation: this phases was characterized by the evaluation of the prediction model accuracy. To assess the model performance LIFT and Hit ratio were used.

[3] Term used in telecommunications companies to designate the costumer movement from one provider to another.

The Figure 5 exposes the generic process of predictive model creation.

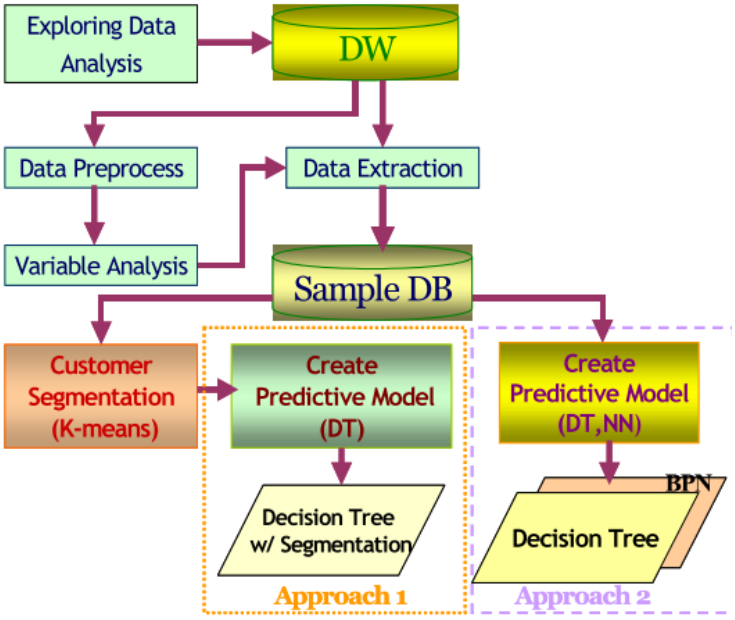

Figure 5 Process of predictive model creation (Hung, Yen and Wang, 2006)

In (Ulmer and Fernandez, 2013), as an attempt to predict the results of soccer matches in English Premier League (EPL), five different ML classifiers were created and its performance tested. The training dataset was composed by 10 seasons of the EPL, ranging from 2002-03 season to 2011-12 season and a testing dataset containing 2 seasons of the EPL, 2012-13 to 2013-14. Using this dataset five ML classifiers were created: Stochastic Gradient Descent (SGD), Naive Bayes, Hidden Markov Model, Support Vector Machine (SVM), and Random Forest. Several techniques such as grid searches and Receiver operating characteristic (ROC) curve were used to improve model parameters and dataset's class imbalance, so the model performance could be increased. Finally, as suggested by the research final results, the eight implemented models have a different behaviors, even though the used dataset is the same. It is possible to conclude that there isn't a standard way to solve a predictive problem, and to build a good predictive model the user should run multiple algorithms and compare their performances, so the most suitable algorithm can be selected.

The Figure 6 shows the comparison of the error rates between the different models used in the above study.

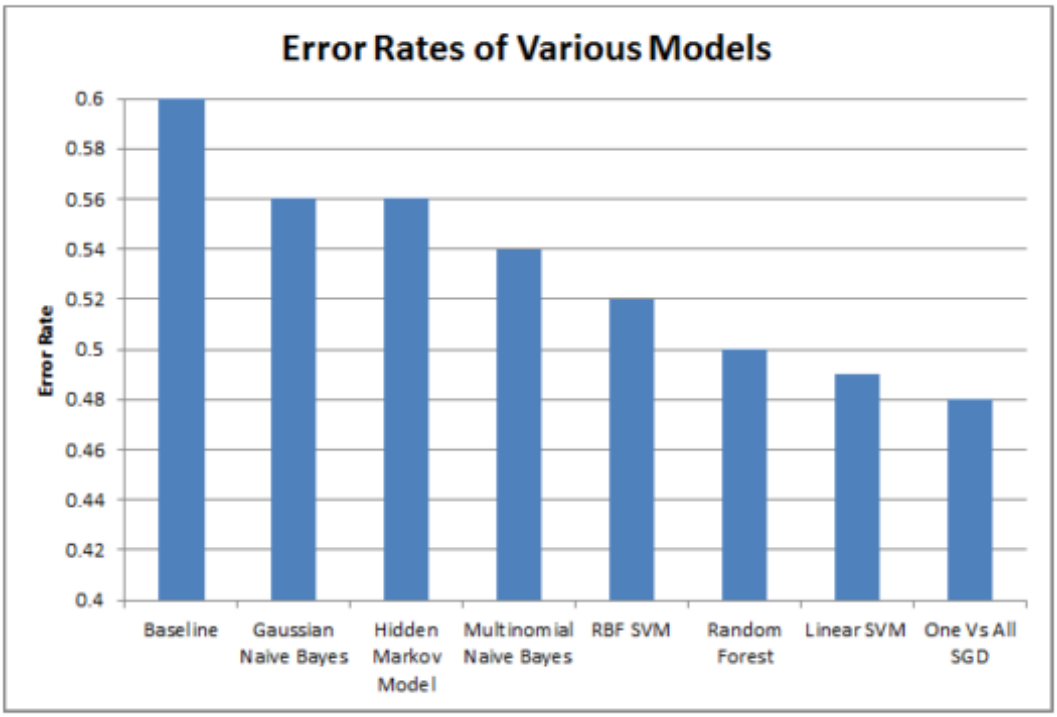

Figure 6 Comparison of error rates between models

Java-ML (Abeel, Peer and Saeys, 2009) is a cross- platform open source machine learning library written in Java that is oriented to developers that want to use machine learning in their own programs. Java-ML is straightforward to integrate in the developers' source code, can be extended and provides several features for exploration task of different models.

ProSOM (Abeel *et al.*, 2008) is a solution to distinguish a promoter[4] sequence from the rest of the DNA, that was devised making use of unsupervised self-organizing map (SOM). This project was implemented using the java language and the Java-ML library. The algorithm training was unsupervised, so the authors didn't train it to specifically recognize the sequences in the training set, it automatically would recognize the patterns and would cluster all the sequences. As result, clusters with high promoter content emerged, making possible to use them for promoter prediction. To measure the performance of the model the team used recall[5] (sensitivity) rate and precision[6] (specificity) rate, where higher the values, the better would be the performance of the model to separate the promoter sequences from the other sequences. This study concluded that

[4] Controlling element of a gene.
[5] Number of correctly predicted promoter divided by the total number of promoter in the dataset.
[6] Number of correct prediction divided by the total number of predictions.

unsupervised SOM algorithms are a very efficient and intuitive way to cluster DNA sequences, having their model achieved an overall precision of 98%.

2.3 Online Machine Learning solutions

Offline ML solutions lack of applicability, scalability and deployability of learning systems. As the ultimate goal is to solve real-world challenges, the industry has shift towards modern ML systems that are ready for real-life problems. As the new demands of the modern society are changing the way we see the machine learning methods, ML can´t be only the ability of applying algorithms on a given dataset, building predictive models and applying new data to them, but also fully integrable, so it can be easily used by desktop, web or mobile applications. This requirements demanded a new approach to Machine Learning, and one that is becoming very popular are the Machine Learning a Service (MLaaS) platforms. MLaaS systems are online ML solutions that allow data scientists and developers to build predictive applications without setting up any infrastructures or without requiring deep machine learning or distributed computing expertise. This systems are a way to abstract away most of the complexity of building and running predictive models, thus making ML easier and quicker to deploy. In this section is present MLaaS and so other online ML solutions that help the users in their prediction challenges.

OpenTox (Hardy *et al.*, 2010), provides a online platform for the creation of predictive toxicology applications. Abstracting the details behind a predictive solution, it provides an end-user oriented tool to non-computational specialist, risk assessors and toxicological experts. Additionally, the framework includes APIs and services for toxicology data management, model building, validation and reporting, which can be combined into multiple applications of a variety end users needs. This tool is composed by two main modules:

- ToxCreate: Build and validates a predictive toxity model based on an input toxicology dataset.
- ToxPredict: Predicts and reports to endpoints, based on data input toxities of a chemical structure.

OpenTox approach allows an efficient way for mapping the incoming new datasets, into a unique structure of shared terminology and representation. This solution can be integrated with

the applications through a set of distributed, interoperable API-compliant REpresentational State Transfer (REST) web services.

As seen in (Reid, Montgomery and Drake, 2016), typically new users to work with traditional ML toolkits like Weka, Orange, Shogun, scikit-learn, may have to learn new programing languages, write new parsers for new data format, and frequently use multiple language platforms, which can make their usage complex and tough task. As so, Protocols and Structures for Inference (PSI) is an attempt to improve the accessibility and interoperability of traditional ML tools, and to address this issue the authors implemented a standard that provides abstraction to their features via web services. The authors claim that this specification for a RESTful API is flexible enough to deal with datasets, attributes, data transformers, learners and predictors allowing users to tackle a broad range of ML problem using a variety of different algorithms.

In (Li *et al.*, 2017), a ML platform was developed for Uber. The authors referred that traditional tools like scikit-learn and R couldn't efficiently process the Uber's big data in real-time. Recognizing that ML is critical for Uber's business, they developed a platform that made extremely easy and fast for any engineer or data scientist to develop and deploy an ML solution in production. This solution solved the problem of training models with big data, running on the cloud this solution was scalable enough to allow fast iteration of model development and frequent model deployment, allowing prediction to happen in real-time. This platform integrates with Uber's internal data infrastructure and application, enabling interaction with people in real-time in the physical world. One of the applications of this platform was in UberEATS application, where it shows the estimated time to delivery of a food order for each restaurant.

AzureML (Raghunathan, 2016), is a MLaaS solution that allows easy development of predictive models and APIs. Compared to other solutions in the market like BigML, Google Prediction API, or DataRobot that enable software developers and data scientists to build predictive applications without requiring deep machine learning or distributed computing expertise, AzureML provides other distinguishing features like collaboration, versioning, possibility to visualize the workflows, integration of user-code (python or R) and easy operationalization. The platform provides the necessary features for uploading assets (datasets, models, transforms), sharing ML experiments and converting ML experiments into web services. On the other hand, the exposed API allows the users to create, migrate and edit the ML experiments. Additionally, the API can also be used to run, test and validate the ML models.

2.4 Discussion

New and disruptive technologies are being invented and developed at an extremely rapid pace, the computational power is increasing year by year and so the amount of data being generated, therefore the needs and challenges of ML in the modern era are hard to fulfil using traditional tools. New ML techniques and method had to be developed, ML can't be just the ability of applying ML algorithms on a given data set , and applying learning predictive models to new data, ML also has to be highly scalable, be available online and be fully integrable with applications in any platform such as desktop, web and mobile.

Due to their relatively easy use, high scalability, and abstraction of infrastructure and technical concerns, the MLaaS platforms are becoming popular among the ML solutions. Despite all the advance made on the MLaaS solutions, there's still a long way to build tools that are standardized, specialized, scalable and yet cost effective in order to achieve the goal of completely automated machine learning utilities, which will become an integral part of the modern software application development toolbox.

Finally, is possible to conclude that machine learning, predictive analytics and MLaaS applications are not technologies of the future, but important technologies of the present.

3

System Architecture

In this chapter is described the MLINO architecture. MLINO is an online platform for prototyping machine learning solutions in a faster and easier way.

In section 3.1, is stated the principles and assumptions of the framework. In section 3.2 is described the general architecture. Afterwards, in section 3.3 is detailed the purpose and relation of the ML engine with the other components. In section 3.4 is explained the goal of the GUI platform and its interaction with the other components. Then, in section 3.5 is explained the data storage solution behind our solution. In section 3.6 is presented the concepts behind the workflows used by the concept and in section 3.7 is referred the techniques used to evaluate the models performance. Finally, the last chapter is ended with few considerations.

3.1 Principles and Assumptions

The main design principle behind MLINO is minimizing the time that takes to build, test and deploy a predictive solution, by using the standart configuration and good practices of building predictive models, abstracting away the technical difficulties that users can face when working with machine learning. This approach aimed to build a platform that could be accessible from any device with internet access, from anywhere, at any time. As a mean to execute ML processes, MLINO doesn't require the installation of additional software or the need of a specific hardware infrastructure, the user through its browser just needs to login in the online platform and make use of its features. The developed concept doesn't provide any feature to pre-process the uploaded datasets, which means the user is in charge of providing a clean and balanced dataset for better results.

3.2 High level architecture view

This section describes the proposed architecture for the MLINO framework. Initially the architecture of the systems is composed by three main components, the ML engine, the Graphic User Interface (GUI) Platform, and the data storage component. For better performance all the three main components should be implemented in the same machine, in order to facilitate resource sharing and communication. Being flexible, and well defined, the MLINO main components can also be implemented in different machines when making use of a good communication channel that doesn't delay the ML processes. The final implementation of the framework should always focus the usability, meaning that it should be accessible from any browser via a device with internet connection.

The general architecture of the MLINO framework is represented in Figure 7.

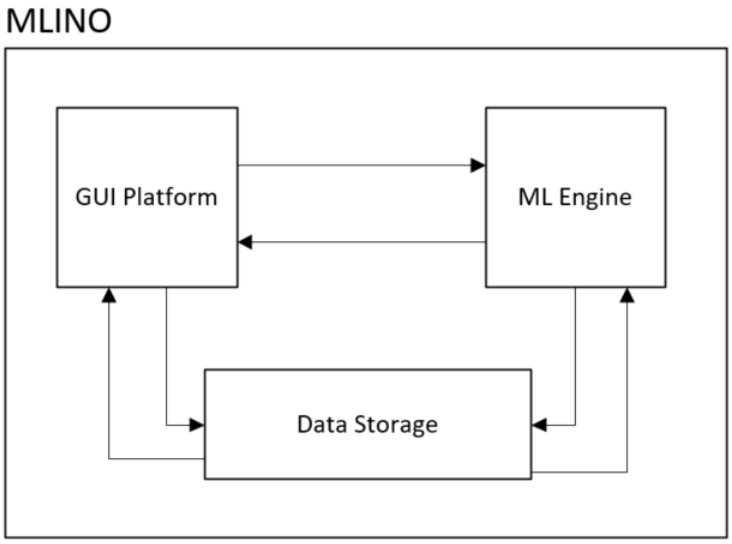

Figure 7 High level framework architecture

The ML engine is the core component of the architecture, because it's responsible for building, training, and deploying the ML models. It is where all the machine learning algorithm source codes are implemented and capable of being enhanced. When requested by the user, the engine have to be able to dynamically generate and run the ML processes according to the user specifications and commands from the GUI platform. Among all the possible user actions that the engine should be able to successfully process are:

- Handle different machine learning algorithms;
- Set the label[7] in the dataset;
- Produce a sub-dataset according to the user choice of features[8];
- Optimize the generated model;
- Generate model performance report (generation of metric and charts).

This component should also be able to run parallel executions, which will allow the processing of several simulations when requested at the same time. The platform should be able to handle different dataset file types.

The GUI platform (web application) is the mean in which the user interacts with the ML engine, using it the user can manage (access, create and remove) their experiments, consult the

[7] Attribute from a dataset in which the prediction model will focus.

[8] Subset of the initial dataset. Technique used to reduce the initial dataset facilitating learning and improving generalization and interpretability.

graphical report of a model execution result (which includes model performance comparison), access/run prediction models and download their outcomes.

The experiment concept, is an abstraction of the primary document type within the framework. Through an experiment the user can run different ML algorithms against a chosen dataset, deploy and generate predictions based on the experiment models, chose the dataset label and features, and consult the performance report generated upon a model execution.

Using this approach the user can create and organize his machine learning creations into experiments. As stated in Figure 8, an experiment can be composed by one or by two models, and each of the models can have associated numerous prediction outcomes which can be downloaded by the user. The user is always prompted to upload a training/test dataset and upload a prediction dataset, when building a model or running a prediction respectively.

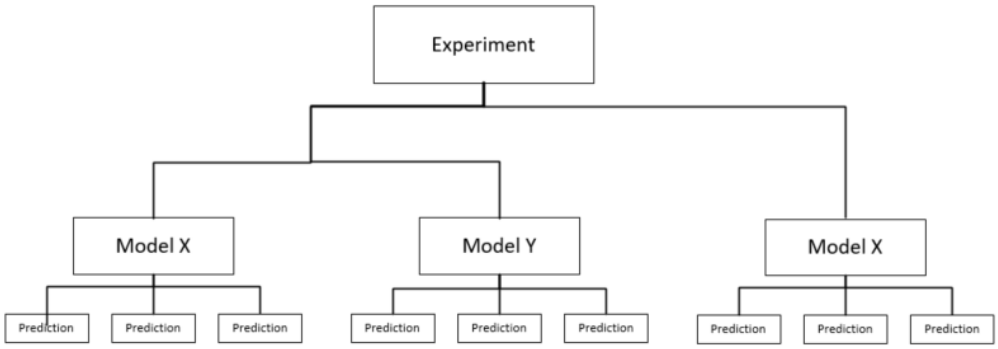

Figure 8 Relation between experiment, model and prediction

The Data Storage component, has the critical mission to manage and store all the generated assets by the ML processes such as models, charts, metrics and predictive outcomes. In other hand, it also has to be capable of receiving and store the uploaded datasets (training dataset, test dataset and the dataset for prediction) from the web application. Additionally, as the GUI platform pages are dynamical and database driven, the pages will have to load information from any place, as so the data storage component will serve as database for it. If the information stored in this database changes, the connect web page will also change accordingly automatically. This component is also responsible to manage the user information and to control the registration and login on the platform.

3.3 ML engine

As the core component of the framework, it's responsible for all the machine learning tasks, from the initial stage of building a model, until its validation and execution as a predictive model. It's also where all the ML techniques and methods such as ML algorithms, validation methods, performance assessment methods, model optimization processes, among other processes, are implemented. This component is made up of three components: controller, workflow builder, workflow executer.

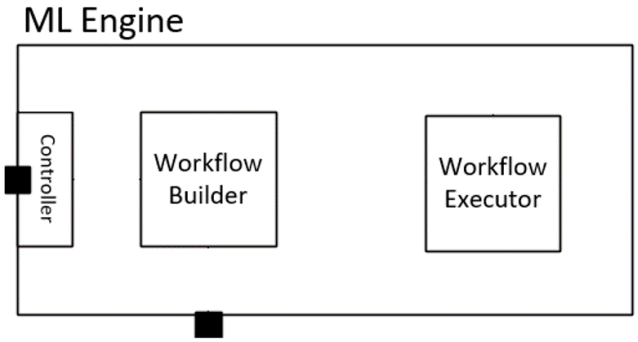

Figure 9 ML engine overview

In the following sub-chapter it is detailed each of them.

3.3.1 Controller

The controller is one of the main components of the architecture. This component controls all the interaction in the ML engine and has a built-in REST API, which is responsible for converting the jobs received via REST API to commands that the remaining component can interpret and execute. In the following sections is presented how the processor handles the different requested jobs. The controller is responsible for the following tasks:

- Process the parameters received upon a job request from the web application;
- Synchronize the workflow builder with the workflow executor;

- Communicate with the GUI platform, receiving jobs or sending the processing results back to it;

3.3.2 Workflow builder

This component receives instructions from the controller (e.g. algorithm, dataset properties) to build a new ML workflow process. The workflow builder, upon a new request has to be able to construct two different ML workflow dynamically. One is the experiment workflow, which is able to build, test and assess the model performance, the other will be the prediction workflow which will be able to run a model and generate predictions. Therefore, it's also responsible to dynamically build a workflow capable of the following features:

- Build a workflow process capable of being executed by the workflow executer;
- Train a model from a given dataset (training set);
- Adjusts the model for given dataset, enhancing the model performance;
- Test the model using a test dataset;
- Asses the model performance;
- Run a predictive model and generate predictions;
- Save the model to the data storage component;
- Save the performance metrics and charts to the data storage component;
- Signal the workflow dependencies (models and datasets to be loaded), this will be useful in the execution phase, because the executor will know which dependencies will have to load.

One of the main goals of the workflow builder is to aggregate several ML steps in just one step, making the job of running a ML system easier, less complex and faster. One of the approaches taken with this architecture, is to create, train, test and asses the model performance in just one machine learning workflow.

In the Figure 10, it is possible to observe the interactions between the MLINO components in the workflow building phase. This steps begins when the user makes a request using the web application. Then, the controller (ML engine) receives the request from the GUI platform and forwards it to the workflow builder, which will build a process capable of being executable by the workflow executor. Finally, when the workflow is generated, it is sent to the data storage component to be saved, and the workflow executor will be notified through the controller to continue the ML process.

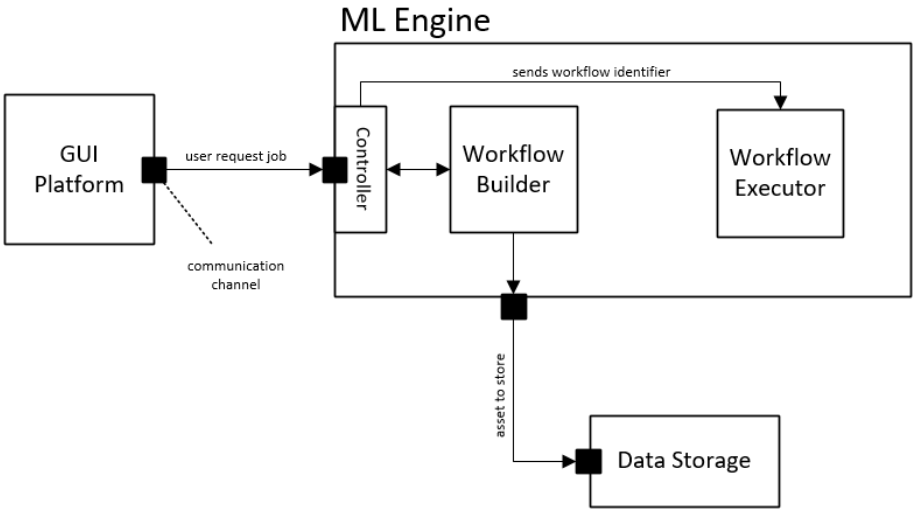

Figure 10 Interaction of the Workflow Build with the other system components

3.3.3 Workflow Executer

The workflow executor is the responsible to run a ML workflow. In our framework two different workflow are capable to be generated and executed: the workflow responsible for building, training, and testing a ML model, and a workflow in charge of running a predictive model in order to generate predictions. When running a workflow, the workflow executor has to be able to retrieve from the data storage the training/test dataset or the prediction dataset, when the task is to generate a ML model or to run a prediction respectively. This component must be able to save the generated assets (e.g. performance charts and metrics, models, predictive outcomes) in the data storage module and notify the controller that the requested action is finished.

In the Figure 11, is shown the different interactions between the MLINO components in the workflow execution phase. It begins when the controller notifies the workflow executer module about the end of the workflow builder task, passing the identifier of workflow to be ran, to the executor. Then, using the workflow identifier, the executor retrieves the workflow from the data storage module and check its dependencies (e.g. datasets or models to be loaded) and integrity (if there's any error in the workflow syntax). Then, the dependencies have to be retrieved and loaded into the ML engine - executor. When the workflow execution is finished, the generated assets like performance charts and metrics, models and predictive outcomes are saved in the data

storage component and the controller is notified that job is done. Then, the controller informs the web application that the job is finished.

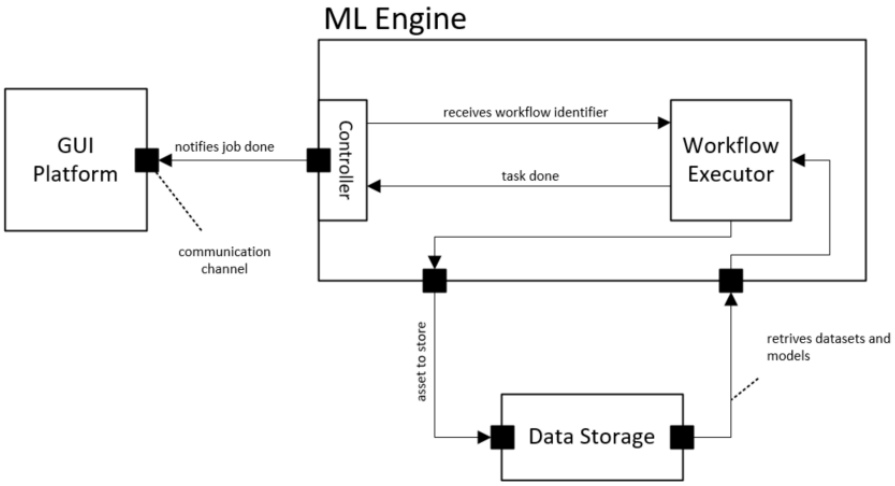

Figure 11 Interaction of the Workflow Executor with the other system components

3.4 GUI platform (web application)

The GUI platform is the only MLINO framework component which the user have direct interaction. As stated before, the GUI platform is a web application that abstracts the technical details of implementing a ML systems. As the MLINO framework is a ready-to use online solution, the user don't need to have deep ML expertise, to install any additional software or to have a specific infrastructure. To use this technology, the user just have to login into the platform, and enjoy the available features to prototype, test and deploy ML systems in an easy and fast way. The GUI platform, focus on being self-explanatory and intuitive providing a smooth user experience when used. Summarizing the web application, has the following objectives:

- Guide the user through the model creation, test, and deployment;
- Report the models performance(metrics and charts);
- Allow user to access and manage their experiments;
- Allow users to run predictions on the built models;

- Allow to download prediction outcomes;
- Validate a dataset when being submitted on the platform;
- Allow users to set the label of their predictions;
- Allow user to choose the features (sub-dataset from the uploaded one) to be used in ML process;
- Notify the user when an experiment or a prediction task is finished;

The GUI platform relies on the data storage module, to retrieve all the information that is shown in its web pages, when is loaded. In contrast, all the user actions and information inserted in the online solution, are sent to the data storage module. The main goal here is to keep track of the user action on the platform and prevent it's lost when the user logs out. An example of utility and usability of this approach, is when a user creates a new experiment and then he logs out, the experiment will not be wiped out. If he logs in again in the platform, his experiment created in the past will still be there and he will be able to perform actions on it.

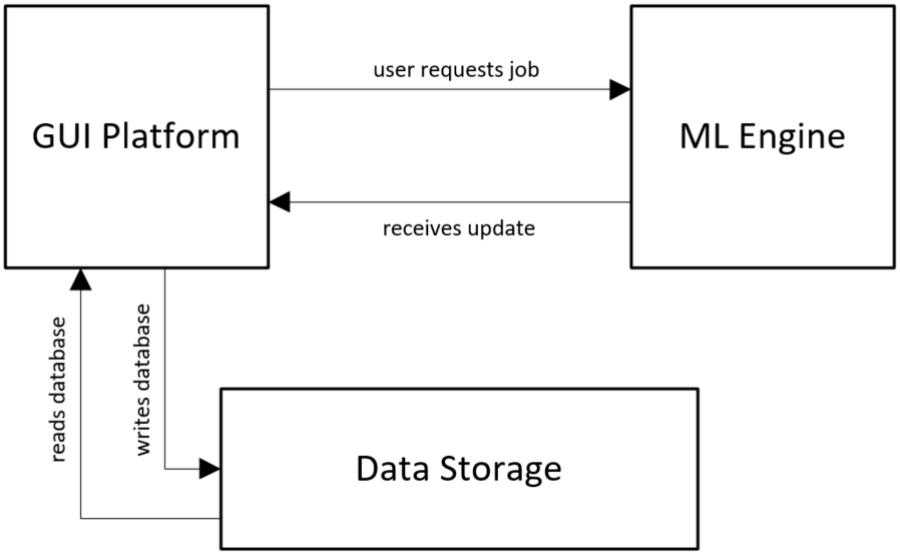

Figure 12 Interaction of the GUI platform with the other system components

In the Figure 12 is shown the interactions between the GUI platform and the other components of the MLINO architecture. When a user request a job (e.g. new experiment or new prediction), the web application sends the job specifications to the ML engine. After finishing, the ML engine notifies the GUI platform that the job is done and the user can now check its results. The information being shown while the user is navigating on the website, is read from the

data storage component. On the other hand, when a user makes a new action on the online platform, the information is written in the database.

3.5 Data Storage

The data storage component is vital in the MLINO architecture, without it all the actions performed by the user would be in vain and the system wouldn't be able to run a ML system. As the name suggests, the data storage module is in charge of saving data. When its referring data, is the data generated by the GUI platform and by the ML engine. This model is also the source model for the information shown in the online platform and the data source to create and run a ML system. The data storage is divided in three components, each of them with the purpose of storing and managing different type of objects used by the framework.

3.5.1 Resource Warehouse

This component is used to store the assets generated during the ML process executions, like predictions results and model performance charts (lift and ROC). When the GUI platform loads the report pages, it retrieves the performance charts from this module and loads them into them website. In a similar way, when then user requests the download of a prediction result via web application, it is retrieved from resource component and sent back to the user. Additionally, this warehouse store all the datasets (training, testing, and prediction) uploaded in the online platform, feeding the data needs of the ML workflow execution, which retrieves from this module the data dependencies.

3.5.2 ML Warehouse

This warehouse stores the ML workflows that are created by the workflow builder module. The workflow executer module when processing a new job, comes here to grab the workflow to be ran. This component is also responsible to store the models generated by other ML processes.

3.5.3 GUI platform database

This database is used to make the online platform interactive. Its primary role is to store and display updated information to the web application. The database can be fed in two ways: by user direct actions or by the execution of experiments or predictions. Every action performed in the online application is saved here. As stated before, when experiment or prediction job ends, their outcome, like performance metrics and status are also save here by the ML engine.

3.6 ML Workflows structure

The workflows generated by the workflow builder are import pieces in MLINO concept, without them there aren't any machine learning processes. There are two main workflow that the workflow builder will be creating: the experiment workflow and the prediction workflow. Both workflow are supposed to be flexible enough, in order to work with any ML algorithm chosen by the user and to support any valid dataset. The workflow structure used in this project, is based in workflows and good practices seen in (Fallis, 2013; Cetinsoy *et al.*, 2016; Studio, 2017). In the above sections is presented an overview about the different workflows that the MLINO can process.

3.6.1 Experiment Workflow

The experiment workflow is executed when the user runs an experiment via client (web application). Typically, to have a ready to deploy ML model, it should pass through three main steps: creation, training and testing. Each of this steps, could correspond to different workflows, and the combination of results of the three workflow would correspond to a model ready to deploy.

As stated in the past chapters, the aim of MLINO is to optimize the existing solutions, in order to have a concept that allows fast prototyping and deploy of ML creations. To achieve this, one of the techniques used is to have a 3-in-1 workflow, where just one workflow is responsible to create, train and test a model. Additionally, the workflow needs to automatically save the generated model in the ML warehouse and to automatically optimize the generated model, in order to get best results for the given dataset. In the end of the process, the workflow executor,

has to be able to communicate the model performance to the controller. The Figure 13 show the conceptual experiment workflow that is used to achieve the tasks referred above.

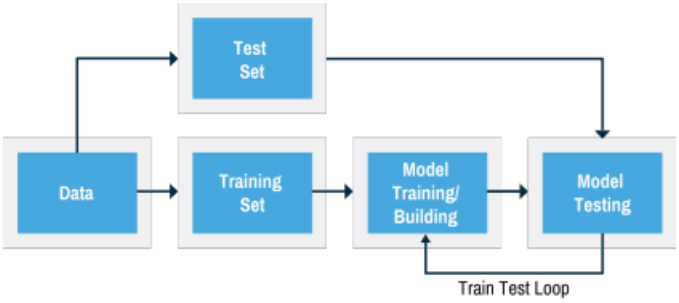

Figure 13 Conceptual experiment workflow (Fallis, 2013)

First the dataset is separated into training data (80 %) and test data (20%). Then, using the training set, a model is created and trained. The process is considered finished after the model is tested using the remaining data (test data). In order to optimize the model, the test process is done multiple times, using different subsets, and each combination of subsets is used only once. This process is called cross-validation (Kohavi, 1995) and the goal here is to estimate how accurately a predictive model will perform in practice.

3.6.2 Prediction Workflow

The prediction workflow is responsible to run predictions. This process has to be capable of executing the models created by the experiment workflow and run the predictive dataset against it, generating prediction results. The prediction outcomes can be either a file (e.g. excel file) or just a value that is stored in the resource warehouse.

3.7 Model Performance Evaluation

Evaluating the model performance is one of the core stages in MLINO concept. Once the model is created and trained, one question comes up: How successful is the model going to be when running against an unseen dataset? Based in the validation techniques seen in (Kohavi, 1995; Kubat, Holte and Matwin, 1998; Hung, Yen and Wang, 2006; Abeel *et al.*, 2008; Hardy *et*

al., 2010; Studio, 2017), for each model, the tool provides the user with several performance metric and charts, namely: accuracy, precision, recall, running time, lift and ROC charts.

This information is later used by the user, in order to help him to choose the most suitable ML algorithm to solve his predictive problem, being able to deploy a predictive model according to the choice.

3.8 Final Considerations

The MLINO concept aims to be a flexible and scalable platform to prototype, test and deploy basic machine learning solutions without need of expertise in machine learning, additional software or specific hardware infrastructure. Using several techniques, this project approach intends to minimize the required time to build and test this kind of applications. In the next chapter we will present some details about the implementation of the prototype.

4

Implementation

In this chapter is covered the most relevant details and technical aspects behind the implementation of the MLINO prototype. It's started by detailing the technologies used and which role they will have in the implementation. Afterwards, is explored the implementation of each MLINO component and its relation with the other components, and the chapter is closed with some final considerations.

4.1 Used Technologies

In this sub-chapter is presented an overview of the technologies used in the implementation of the prototype, as well the reasoning that led to their choice.

4.1.1 OutSystems

OutSystems is a low-code fast development platform that let the users to devise an entire application and integrate it with the existing system in an easy way. As the applications are built using OutSystems visual tools, most of the technical details of developing this kind of solutions are abstracted and facilitated. The applications developed using this technology, are then hosted in their own cloud, which makes the solutions accessible to all the user when connected to the internet.

To develop the web application is chosen the OutSystems technology, which makes the job of develop an online application easier because it's a low code platform with templates that can be used. As the OutSystems applications are hosted in their own cloud, the solution is then always up in the internet, allowing the users to start their ML creations from any internet browser at any time. To develop the prototype it was used the free version, which have some limitations like only two GB of cloud database, and a user capacity one hundred users.

4.1.2 REST

REST is an architectural style, often used in the web applications communications. It relies on the HTTP requests to GET, PUT, POST, DELETE data.

This technology was used to build the communication channel, as it is flexible, easy to implement and fulfills the communication requirements, between the web application and the ML engine, which is running in a local computer.

4.1.3 Java

JAVA is a programing language oriented to objects. As JAVA, has large set of different libraries, its usage in the prototype was critical, allowing to easily integrate the different MLINO components. Also the RESTful controller that was responsible to process the REST API requests, to run some jobs like process training datasets, to run the ML workflows builder, to run the workflow executor (RapidMiner toolkit through its integration with JAVA) was developed using JAVA.

4.1.4 Ngrok

As stated in their website, *ngrok is a reverse proxy*[9] *that creates a secure tunnel from a public endpoint to a locally running web service*. As a personal computer is being used to host the ML engine, it was chosen to use the free reverse proxy ngrok, because it's easy to implement and fulfills the systems requirements, allowing a local web service (ML engine REST API) to be available in the internet.

4.1.5 RapidMiner

RapidMiner is an open source software that provides a graphical interface, permitting user to rapidly design and test their machine learning creations. RapidMiner allows in an intuitive and easy way to develop and test standard ML workflows that would be later integrated in the prototype. Using this software some crucial time during the development phase could be saved, because it's an all-in-one tool that features a wide range of machine learning algorithms and methods that fulfills all the MLINO requirements. Even though RapidMiner software is a free software, it was run under the educational license, in order to get rid of dataset size limitation.

4.2 Prototype Architecture

The Figure 14 presents the general architecture of the MLINO prototype. At first glance, is easily to realize that the architecture here presented is slightly different from the concept architecture (Figure 7) presented in the past chapter.

[9] A reverse proxy takes requests from the Internet and forwards them to servers in an internal network.

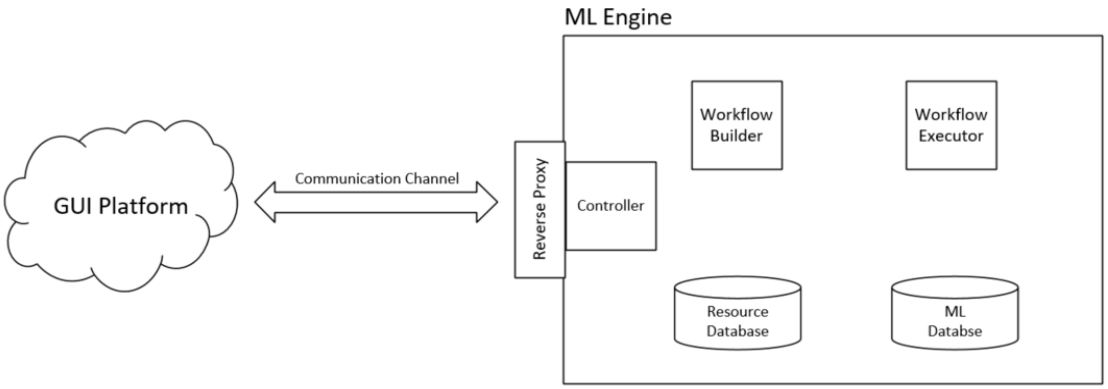

Figure 14 Architecture overview of the MLINO prototype

The reasons behind this adaptation of the MLINO concept, were the limited time for the prototype development and the limited resources available. An optimal implementation of the theoretical concept, would require specific resources and much more time spent. With this approach is intended to build a budget prototype that would be easy to implement, but yet with optimal performance and still follow the MLINO concept.

Instead of having three main components, this approach have just two: the GUI platform (web application) and the ML Engine. The web application is developed and runs in the cloud. In contrast, the ML Engine is implemented in the local computer that will act as a normal server. Both components communicate through a communication channel that will make use of a reverse proxy to enable the communications (port forwarding) with the localhost. In this implementation, each of the main components have a built-in data storage components.

As the ML engine would be a lot more complex than the web application, it was chosen to be developed in a personal computer and then connect it to the GUI platform through a special API (detailed in the next chapters). To implement the purposed ML engine architecture is used the JAVA language alongside the RapidMiner machine learning toolkit.

The communication channel was developed using the REST API and it's the mean, which makes possible the communications between the GUI platform and the ML engine via internet. First, the REST API was configured in the OutSystems platform, and then ML engine RESTful API was developed. As a personal computer was used to host the ML engine, a reverse free proxy, NGROK was used in order to create a secure tunnel from a public endpoint to the localhost (ML engine REST API), making it available in the internet.

4.3 Machine Learning Algorithms

As referred several times before, one of the main tasks of the MLINO concept is to test the performance of several different ML algorithms, providing the user with their performance report. So, as this prototype is a proof of concept and due to the wide range of ML algorithms available, it was implemented only four algorithms. The chosen algorithms are classifiers that belong to the group of supervised machine learning algorithms. Using the prototype the user can run the following algorithms:

- Decision tree;
- Naïve Bayes;
- K-NN;
- H2O Deep Learning

Through this set of ML algorithms, the prototype is capable of running binominal or polynomial classifications (predictions). Choosing this algorithms, four big groups of classifications algorithms were covered: decision trees, Bayesian (Naïve Bayes), lazy (K-NN), and neural nets (Deep learning).

4.4 GUI Platform

The GUI platform is the front-end of the prototype. This module was implemented using the OutSystems technology. As it runs in their cloud, the web application is accessible from any web browser. The web application is the only contact point that the user has with the prototype. Using it, the user can manage experiments, check experiment results, run predictions and download their outcome.

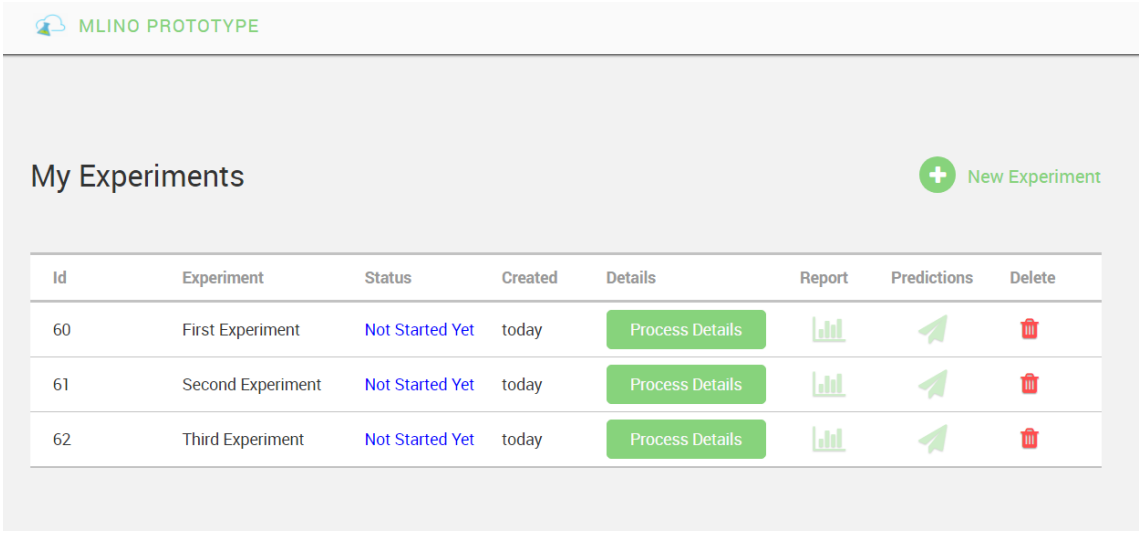

Figure 15 Screenshot example of the MLINO prototype visual interface.

In the Figure 15, is presented an example of the web interface that the user will have to interact in order to use the MLINO prototype. In the above section is discussed some implementation highlights.

4.4.1 Website pages

The web application is composed by different pages, each of them with different purposes. In the Figure 16 is exposed how the main web pages are organized and their interactions.

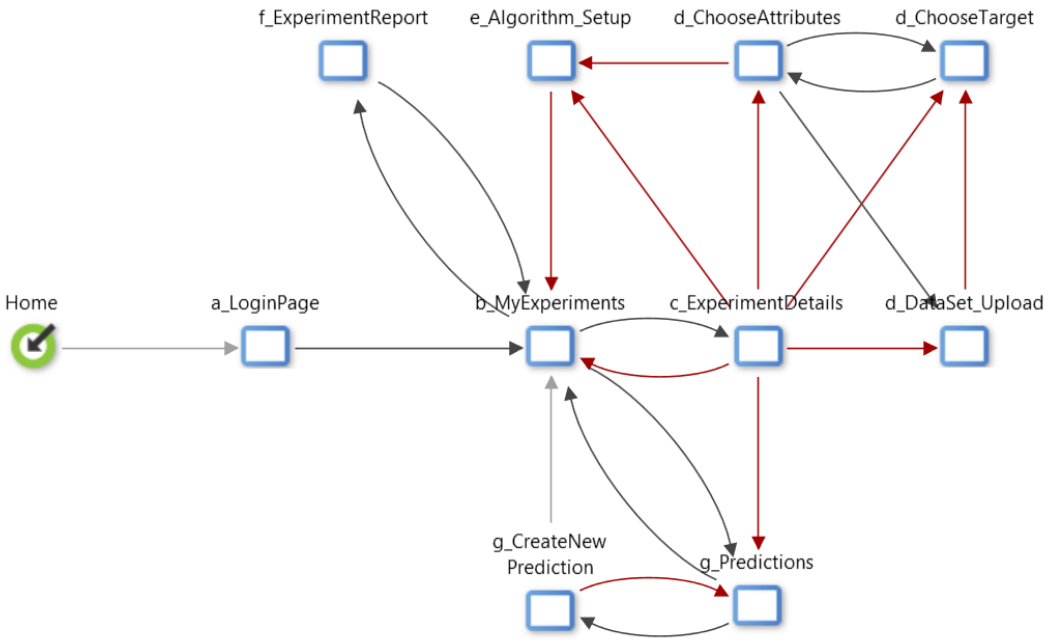

Figure 16 High level pages disposal diagram

As it is seen in the picture above, the visual interface of the web application is composed by several web pages, and each of them connects to a different set of pages. In the Table 1 is detailed the purpose of each web page.

Table 1 Roles of each web page

Page name	Description
LoginPage	Page where the user can login into the web application.
MyExperiments	Here are listed all user experiments. Also in this page the user can manage their experiments.
ExperimentDetails	Page where the user creates/edit an experiment. In this page the user has to provide some experiment info such as name and description.
Dataset_Upload	Page where the user uploads the dataset to train and test the experiment algorithm(s).
ChooseTarget	In this page the user has to choose label (target) of the classification process.
ChooseAttributes	In this page the user has to choose the features (attributes) that he will use from the dataset, to train and test the experiment algorithm(s).
Algorithm_Setup	Page where the user chooses one or two ML algorithms to be used in the experiment.

ExperimentReport	Page where the user can see the experiment results. In this page, some metrics and charts regarding the ML models performance are shown.
CreateNewPrediction	In this page the user can use the previously created models to run predictions. Also here the user have to upload the dataset in which the prediction process will run.
Predictions	Here are listed all prediction processes of a chosen experiment. In this page the user can manage their predictions processes and download their outcome.

4.4.2 Front-end interaction with database

The web application as referred before, it's dynamic and database driven, which means that whenever a web page is loaded, the system have to access the database, fetch its information and present it in the web page. In contrast, some processes access (create, delete or edit) the database through the web pages action or via the built-in API. Summarizing, the front-end application can perform two kind of operations in the database: read and write.

Read operations

One of the most common read operations that occur in the front-end platform is to fetch data from the database and to display it on screen whenever a new web page is loaded. In the Figure 17 is presented an example of the read operation when the page "Experiment Report" is loaded. OutSystems makes very easy to implement the read operation of the database, using its visual development environment, for each page it was configured a workflow that runs whenever a web page is loaded. In the referred example are used two aggregate connectors to retrieve data from the "Experiment" table and from the "UploadedDatasetStats" table. Therefore, after getting the information is used an "Assign" connector to associate this information to the page variables. This variables would then be used to feed the web page upon its load.

Figure 17 Example of implementation of read operation

Write operations

When a user interacts with the web application or the API receives any update/request, the system writes that interaction in the database. The interaction is saved in the database either to keep track of the user actions (e.g. creation of a new prediction process, edition of an experiment info), or to manage any database object (e.g. update experiment results). In the Figure 18 is presented a case where the user creates a new experiment using the web app. In this example, after the user fulfils the form fields and clicks save, the system will perform a write operation, in order to save the experiment data.

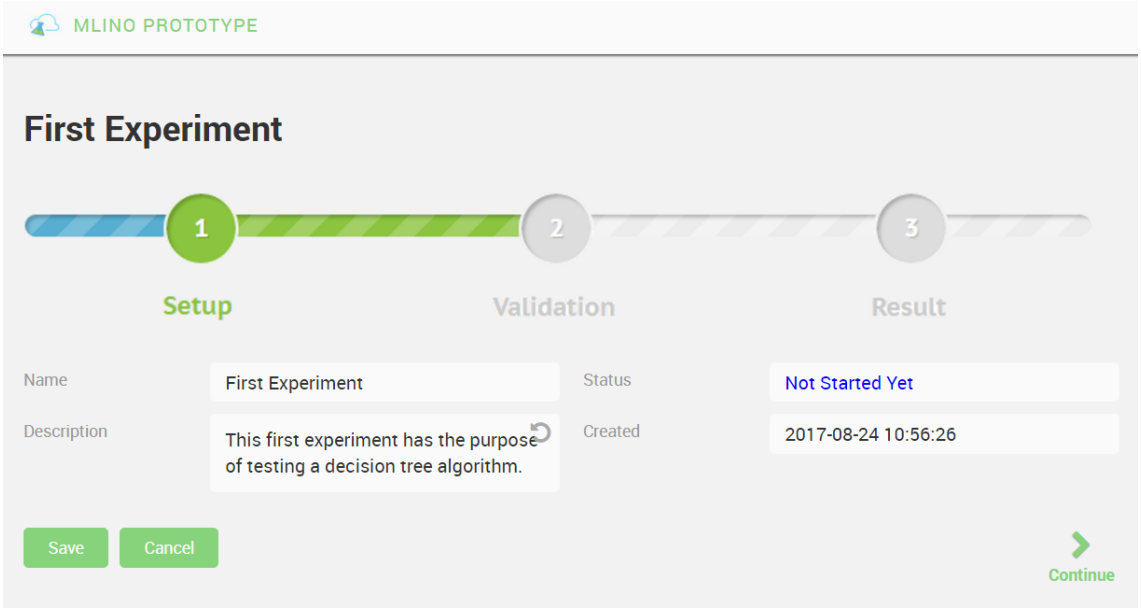

Figure 18 Creation of a new experiment

The saved experiment data will be presented to the user whenever he loads a web page that is fed with this information (e.g. list my experiments).

4.4.3 Data model

The GUI platform makes use of a database to support all the interaction available in the web application. Thus, a data model was created (tables and relations between them) in order to support them.

The Figure 19 the data model used by the OutSystems platform.

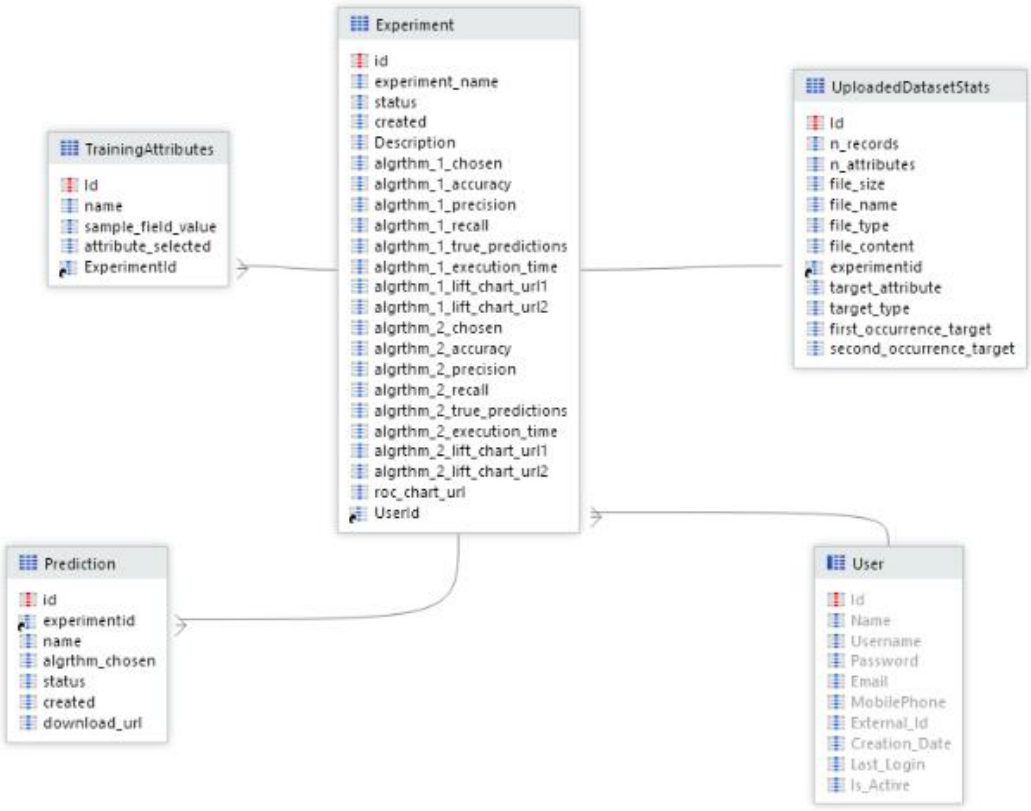

Figure 19 Web application data model

Table 2 Data model tables description

Table name	Description
Experiment	One of the main tables of the data model. It's responsible to store all the information regarding an experiment.
User	This table is used to control the user logins into the platform and to store the user information.

Prediction	The role of this table is to save all the information regarding a prediction process.
TrainingAttributes	Auxiliary table which stores the training attributes of the dataset used to train and test a ML model in an experiment.
UploadedDatasetStats	Auxiliary table which stores the properties of the dataset used to train and test a ML model in an experiment.

4.4.4 REST API structure

The built-in REST API provides interoperability between the GUI platform and the ML engine. The ML engine communicates with the web application using the REST methods presented in Figure 20.

Figure 20 Supported REST methods by the GUI platform

Table 3 GUI platform REST methods overview

REST method	Description
PUT /UpdateExperiment	The ML engines uses this method in order to get the web application database updated with the experiment data, after processing it.
PUT /UpdatePrediction	The ML engines uses this method in order to get the GUI platform database updated with the prediction data, after getting it processed.

4.5 ML Engine

4.5.1 ML Engine RESTful API structure

The built-in REST API provides interoperability between the GUI platform and the ML engine. This RESTful implementation allowed the web application to communicate with the ML engine, using the REST methods presented in Figure 21.

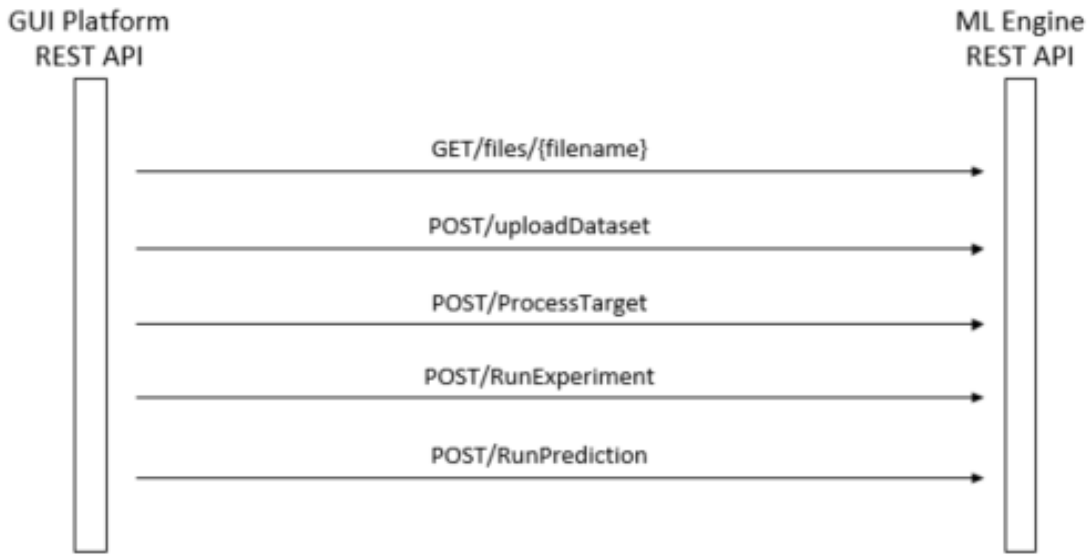

Figure 21 Supported REST methods by the ML Engine API

Table 4 ML Engine REST methods overview

REST method	Description
GET /files/{filename}	The GUI platform uses this method to get (download) files (e.g. prediction outcomes), from the resource warehouse.
POST /uploadDataset	This method is used to upload datasets from the web application to the resource warehouse in the ML engine.
POST /ProcessTarget	Method used to process the label attribute. When the API processor receives this job, its aim is to check whether the label attribute is binary and multiclass.

POST /RunExperiment	The GUI platform uses this method to transfer the job of running an experiment to the ML engine. Its body contains all the parameter needed for the execution.
POST /RunPrediction	The GUI platform uses this method to transfer the job of running a prediction to the ML engine. Its body contains all the parameter needed for the execution.

4.5.2 Controller

The controller is one of the main components of the architecture. As the name indicates, this component controls all the interaction in the ML engine. Having a built-in REST API the controller is responsible for converting the jobs received via REST API to commands that the remaining component can interpret and execute. In the following sections is presented how the processor handles the different the jobs.

Job get file

This jobs starts when the controller receives the request through the method *GET /file/{filename},* via its build-in REST API. Then, the controller invokes the java class *StorageService.LoadASResource()* to load the file from the resource warehouse, wraps it in a Hypertext Transfer Protocol (HTTP) response and sends back to the client (web application).

The Figure 22 presents the activity diagram of get file job.

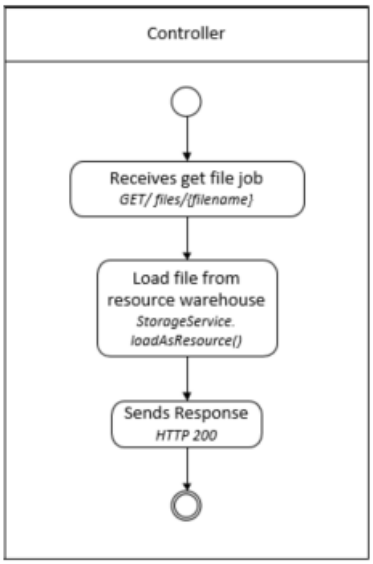

Figure 22 Activity diagram of get file job

Job upload dataset

This jobs starts when the controller receives the request through the method *POST/ uploadDataset* via its REST API. First, the controller unpacks the dataset from the request body and saves it in the resource warehouse, using the java method *StorageService.store()*. To finish the job, the controller replies back to the client, using a HTTP message of success.

The Figure 23 presents the activity diagram of upload dataset job.

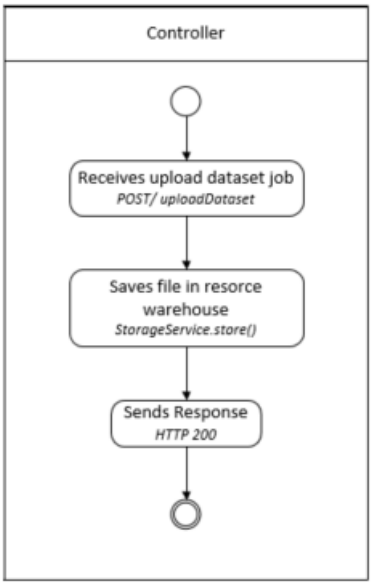

Figure 23 Activity diagram of upload dataset job

Job process label

When the controller receives this job request through its API, unwraps the parameters from the request body and processes it using the java class *ProcessTargetAttribute().* When the processing is done, it wraps the results in the HTTP body and sends back to the client.

The Figure 24 presents the activity diagram of process label job.

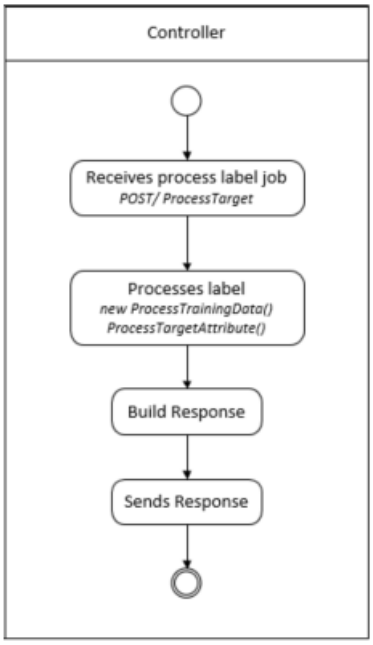

Figure 24 Activity diagram of process label job

Job run experiment

This job starts when the processor receives the request in its built-in REST API. First, the processor unwraps the experiment parameters. Then, creates a java thread that will be responsible to execute this job. Using threads, the processor can process and execute jobs in parallel. Afterwards, the controller communicates the experiment parameters to the workflow builder. In this step, the workflow builder, which is a java class, builds a ML workflow according to the parameters received. Therefore, the processor using the java API of RapidMiner, invokes it to run the workflow. After, an experiment workflow is executed, several asset are created. The generated model is stored in the ML warehouse and the lift charts images are stored in the resource warehouse. This two type of assets are automatically stored in their respective warehouses by the RapidMiner.

After running the workflow for the algorithm 1 (remember that an experiment can have two algorithms chosen), the processor check if there is an algorithms 2 in the parameters received, via job request. If there isn't another second algorithm, the results are wrapped in a HTTP body and sent back to the client, otherwise the previously described procedure is repeated, but this time using the algorithm 2 parameters.

When the execution of the algorithm 2 workflow is finished and the assets stored, the processor check if the dataset label is binary. If negative, wraps the result of both workflow executions and replies the result back to the client. If positive, it means that a ROC chart can be built (two algorithms were chosen and the label is binary). So once again, the processor invokes the workflow builder to create the ROC chart workflow, and send it to the RapidMiner to be executed. When the execution is finished, the ROC chart image is stored in the resource warehouse and the final results are wrapped up in a HTTP body and sent back to the client.

The Figure 25 presents the activity diagram of run experiment job.

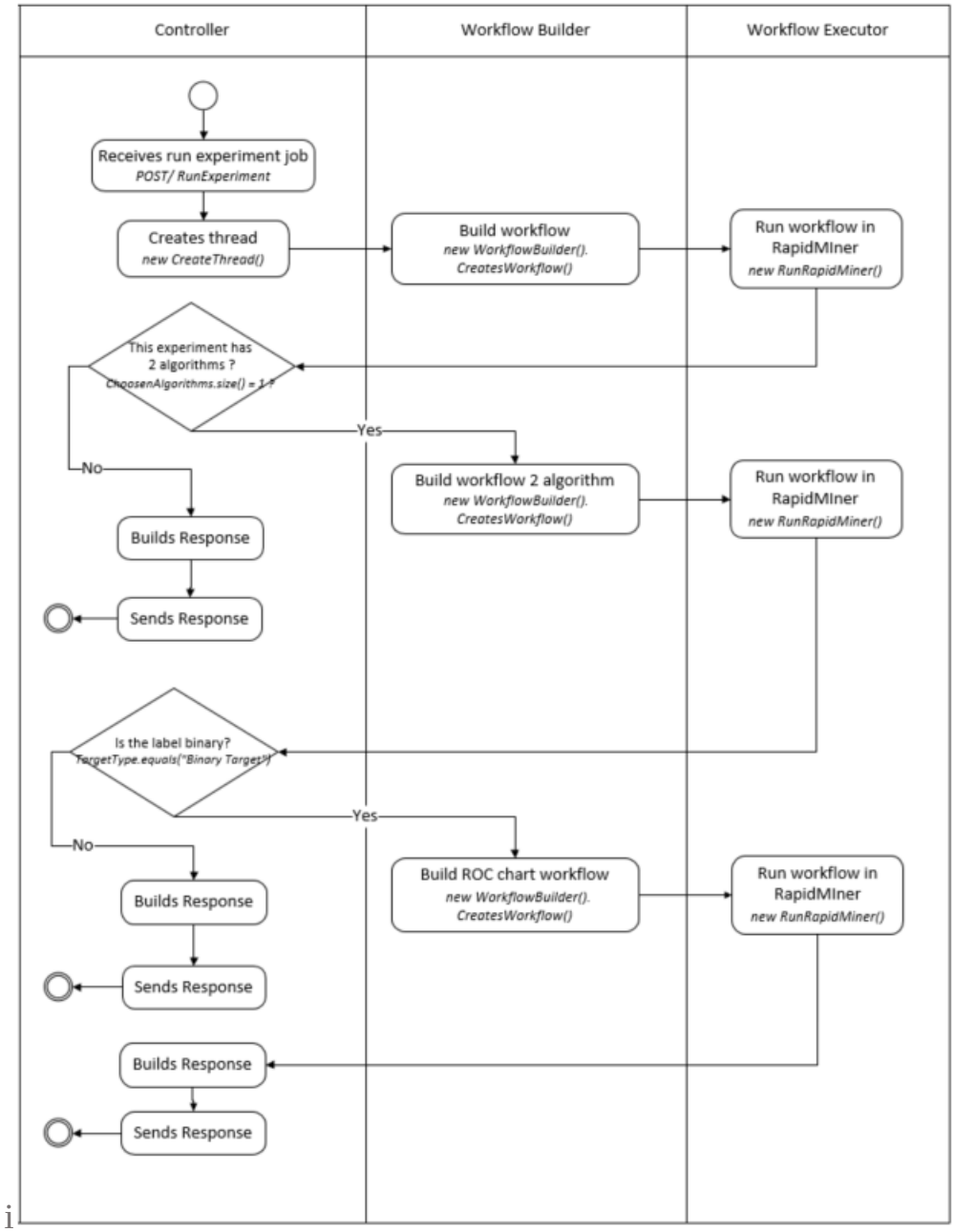

Figure 25 Activity diagram of run experiment job

Job run prediction

This job starts when the controller receives the request through its REST API. To process the job the processor creates a thread, enabling in this way other jobs to be processed in parallel. Then, the workflow builder receives the job parameters and builds the respective workflows. Therefore, this workflow is executed by the workflow executor (RapidMiner), and the generated prediction outcome, an excel file, its saved in the resource warehouse. Having the job finished the processor builds the answer and replies it back to the client.

The Figure 26 presents the activity diagram of get file job.

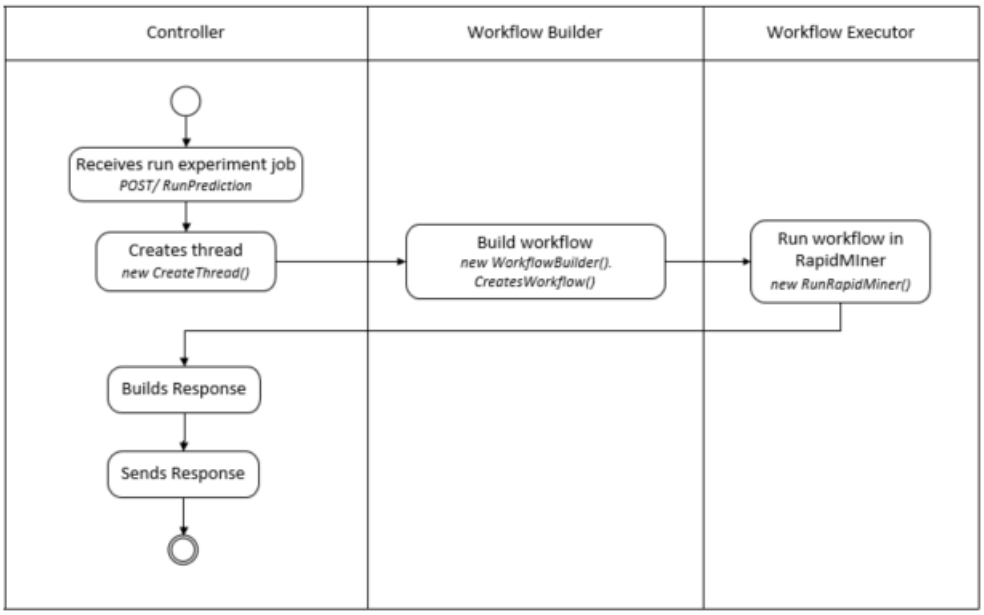

Figure 26 Activity diagram of run prediction job

4.5.3 Resource warehouse e ML warehouse

The concept behind this two warehouses, is to store generated assets upon a workflow execution. In the prototype this two warehouses are just two folders, located in the ML engine host machine. Both folders are accessible by the ML engine components and by the web application, via processor's REST API. In the Table 5 Assets stored in each warehouseis summarized the assets stored in each warehouse.

Table 5 Assets stored in each warehouse

Warehouse	Asset
Resource Warehouse	Training/test datasets, prediction datasets, lift chart images and ROC chart images.
ML Warehouse	Experiment workflows, prediction workflows, ML models.

4.6 ML Workflow Structure

The workflow builder is the component responsible for building the workflows, which generated them in eXtensible Markup Language (XML). As referred before, this XML file is later capable of being run by the RapidMiner toolkit. To accomplish all the requirements to implement the ML architecture and due to limitations of RapidMiner, the prototype is capable of creating three different workflows: experiment workflow, prediction workflow and ROC chart workflow. Based in the concepts presented in 3.6 and in 4.3, the above sub-chapter presents the generic workflows developed in the prototype. The workflows where presented, were cleaned to show only the most import logical objects that make part of it.

4.6.1 Experiment Workflow

The Figure 27 shows the workflow used to the run experiment job. The workflow is composed by the following logical objects:

- Read Excel: retrieves the training/test dataset from the resource warehouse;
- Set Role: selects the label in the given dataset;
- Guess Types: assigns the value types of all attributes dataset, but label;
- Filter Examples: prevents missing values in the label;
- Select Attributes: sets the chosen features to be used in the model creation;
- Cross Validation: trains, tests and optimizes the model. It was configured with K = 10 (number of iterations) in order to avoid a pessimistically biased model;
- Create Lift Chart: creates the lift chart of the model;
- Write Model: saves the generated model in the ML warehouse;

- Report: saves the lift chart image in the resource warehouse;

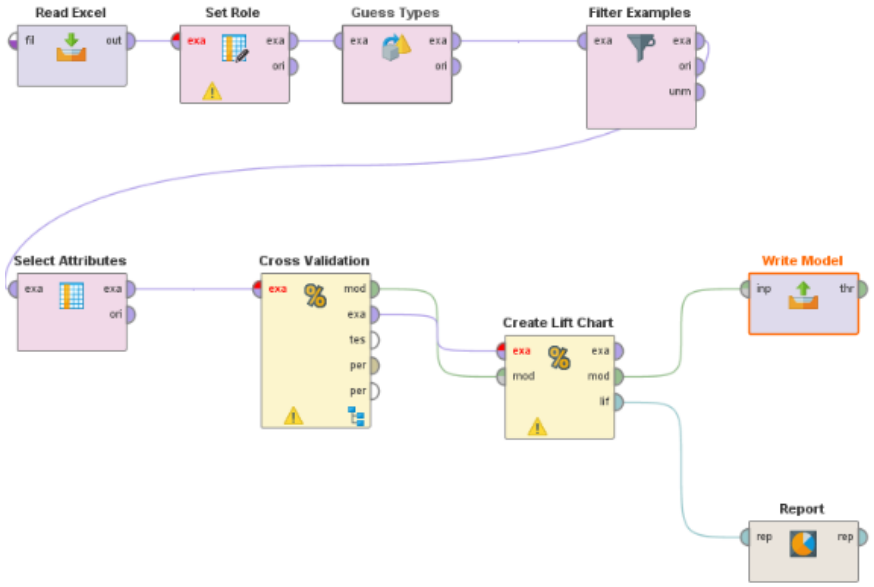

Figure 27 Experiment workflow

4.6.2 Prediction Workflow

The Figure 28 presents the prediction workflow used to compute predictions. The logical operators that make part of it are (the referred above components are not described again here):

- Read Excel: retrieves the prediction dataset from the resource warehouse;
- Read Model: retrieves the previously generated model from the ML warehouse;
- Apply Model: applies an already trained model on the given dataset;
- Write Excel: saves the generated prediction result (excel file) in the resource warehouse;

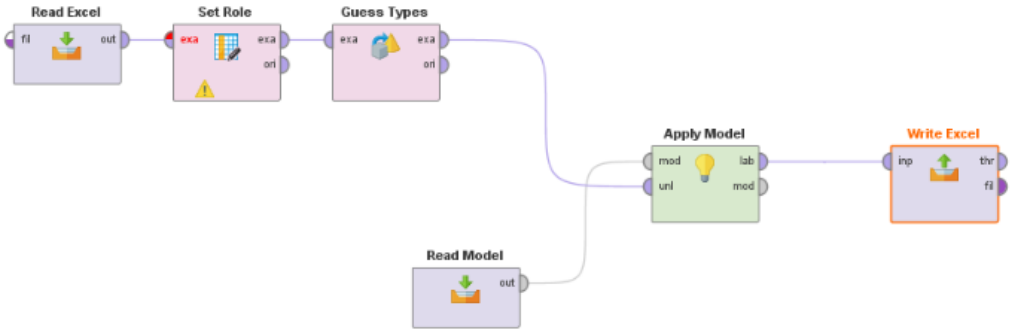

Figure 28 Prediction workflow

4.6.3 ROC chart workflow

This generation of the ROC chart workflow is not included in the generic experiment workflow, because to generate ROC chart, the label has to be binary (only two classes) and the experiment has to be composed by two ML algorithms, as explained before. The operators that make part of this workflow are (the referred above components are not described again here):

- Compare ROCs: Generates ROC charts for the models generated by the experiment workflow;
- Report: saves the ROC chart image in the resource warehouse;

The Figure 29 presents the ROC chart workflow, used by the RapidMiner.

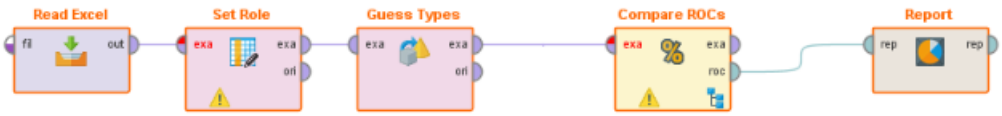

Figure 29 ROC chart workflow

4.7 Class Diagram

The Figure 30 presents the UML class diagram of ML engine – MLINO prototype. The diagram was cleaned to present only the most relevant classes of the system.

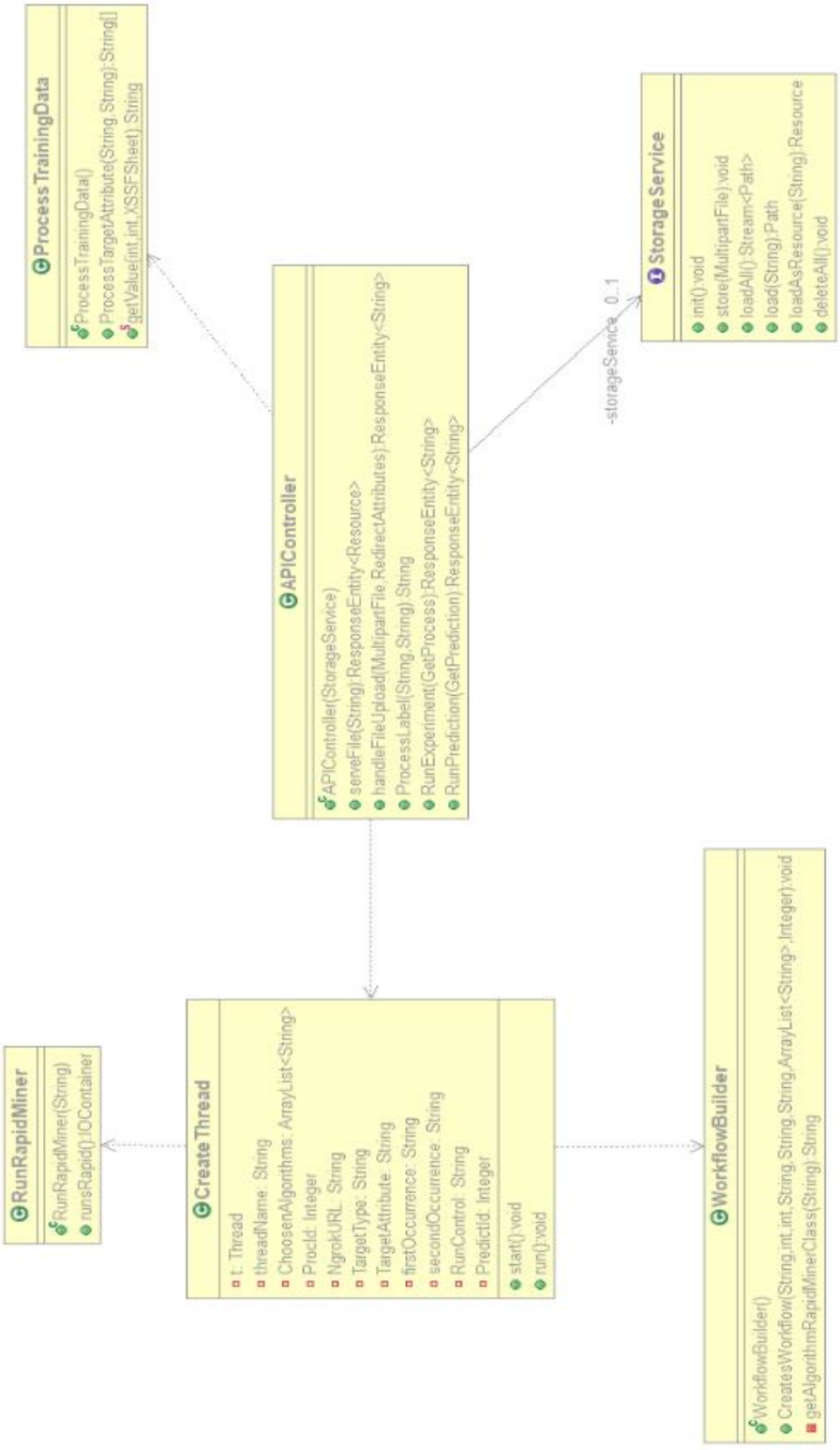

Figure 30 ML engine class diagram

4.8 Final Considerations

In this chapter was presented the details about the implementation of the MLINO prototype. As referred before, the implementation of the MLINO prototype was based in a slightly modified architecture from the one presented in section 3, namely because:

- Limited time to implement the prototype;
- Limitations of the chosen technologies;
- Lack of optimal infrastructures (hardware) to devise the prototype.

Even though the technical challenges to develop such a tool, we consider that the prototype accomplishes the requirements and fully complies with the concept behind MLINO.

5

Evaluation

This chapter presents a detailed evaluation of the MLINO prototype. In this chapter is demonstrated that the concept described in the section 0 is feasible and accomplishes the aim of the project, which was described in section 1.2. In section 5.1 is presented the behavior of the system when under different workloads. In section 5.2 is compared the prototype performance to the other available tools in the internet. Finally, in section 5.3 is discussed the results.

All the tests were performed in a single computer with the following technical aspects:

- Intel Core i7-6700HQ (6M Cache, up to 3,50 GHz)
- 16 GB RAM
- Operative System: Windows 10 Pro 64-bit

Even though an optimal implementation of the prototype should have been in the cloud, which is a scalable and flexible environment, this chapter aims to demonstrate that using a personal computer to implement the concept, is possible to get satisfactory results.

5.1 Performance Test

The aim of this tests was to evaluate the system's behavior under both normal and unusual peak loads when used by multiple user at the same time. To execute the performance tests an open source tool, Apache JMeter, was used. This tools is one of the most used to analyze the overall performance of a system under different workload types. To test the performance of the prototype five different scenarios were defined:

5.1.1 Scenario 1

In this first scenario, the performance of the system was tested when 1, 10 and 40 users at same time load the report page in the web application, which then loads an image from the resource warehouse. The imaged used in this test had 35 KB.

In the Figure 31 are presented the results of the performance test.

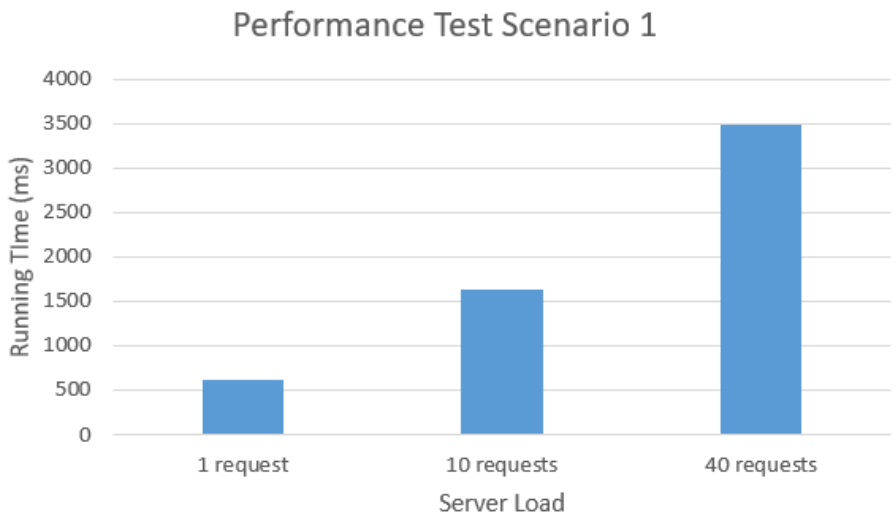

Figure 31 Performance test scenario 1

In the Figure 31 is possible to observe that the load of 40 simultaneous requests took less than 4 seconds to be resolved. As predicted, the response time increased considerably with the amount of load, which was expected.

5.1.2 Scenario 2

In this second scenario is evaluated the system response when 1, 10 and 40 users simultaneously upload a dataset through the GUI platform. In this test we used a excel file of 32KB.

In the Figure 32 are presented the results of the performance test.

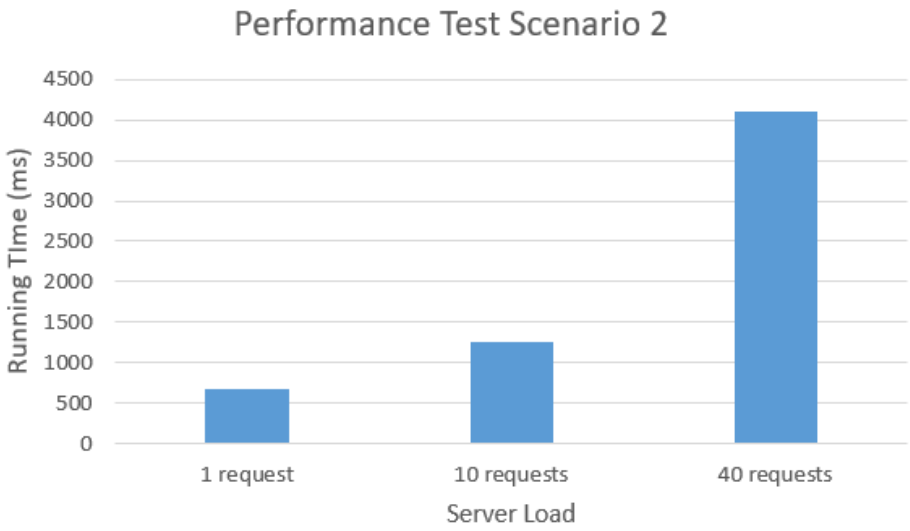

Figure 32 Performance test scenario 2

As is it can observed in the Figure 32, the load of 40 simultaneous requests took less than 4,5 seconds to be resolved. Comparing the time that 1 request takes to be processed to the time that 10 requests take, it is to be concluded that the system automatically adapts when the load requests increases, optimizing the process. As predicted, the response time increases with the amount of load, which was expected.

5.1.3 Scenario 3

This is the scenario that assess the system performance when the users through the web application, chose the label. Then, the online client sends the label information to the ML engine, which processes it and replies back, with the information if the label is binary or multiclass. In this test is evaluated the system response to 1, 10 and 40 simultaneous requests.

In the Figure 33 are presented the results of the performance test.

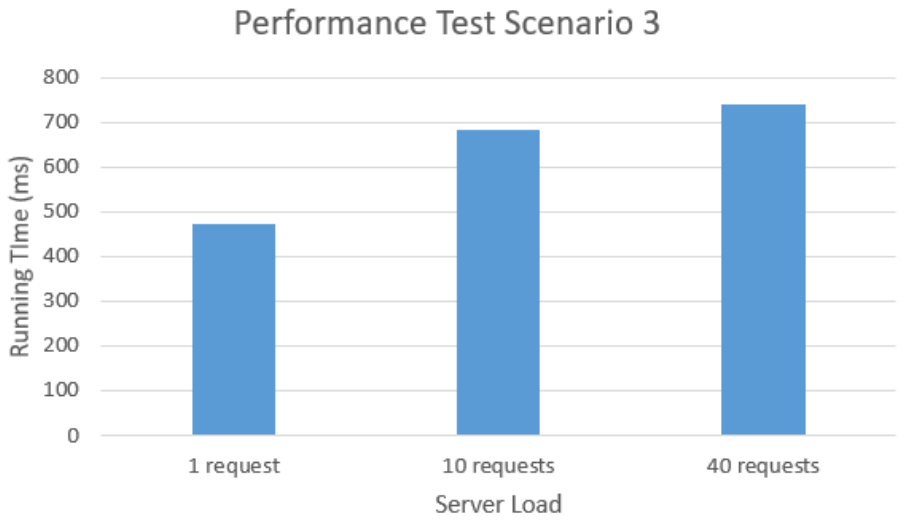

Figure 33 Performance test scenario 3

In the Figure 33, is demonstrated that one request takes almost the same processing time than 10 or even 40 simultaneous requests. This happens because the label processing by the ML engine is an easy task and so, the system is so fast processing it that almost can't feel the increase of load.

5.1.4 Scenario 4

In this scenario is simulated the system behavior when 1,10 and 40 prediction jobs are requested simultaneously in the web application. To run this test, is used the decision tree algorithm and a prediction dataset of 35 KB.

In the Figure 34 are presented the results of the performance test.

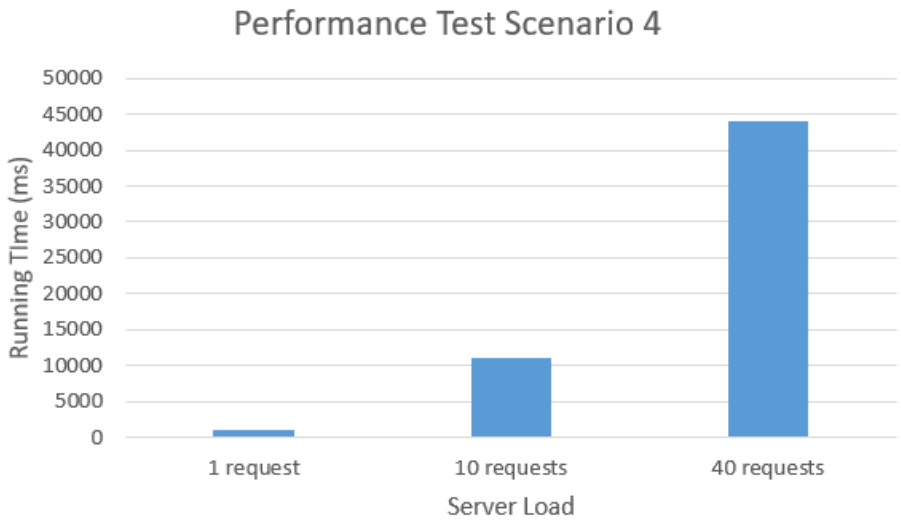

Figure 34 Performance test scenario 4

As is stated through the above graphic, running a prediction requires way more processing time if compared to the past scenarios. In contrast, this one uses the workflow executor (RapidMiner) to run predictions, which consume more computational resources and time, as this is a more complex task. As expected with the increase of load, the running time increased considerably.

5.1.5 Scenario 5

The fifth scenario, evaluates the system performance when 1, 10 and 40 simultaneously experiments are dispatched to the ML engine to be processed. This test was ran training and testing the models with a training dataset of 32KB, and the chosen ML algorithms were decision tree and K-NN.

In the Figure 35 are presented the results of the performance test.

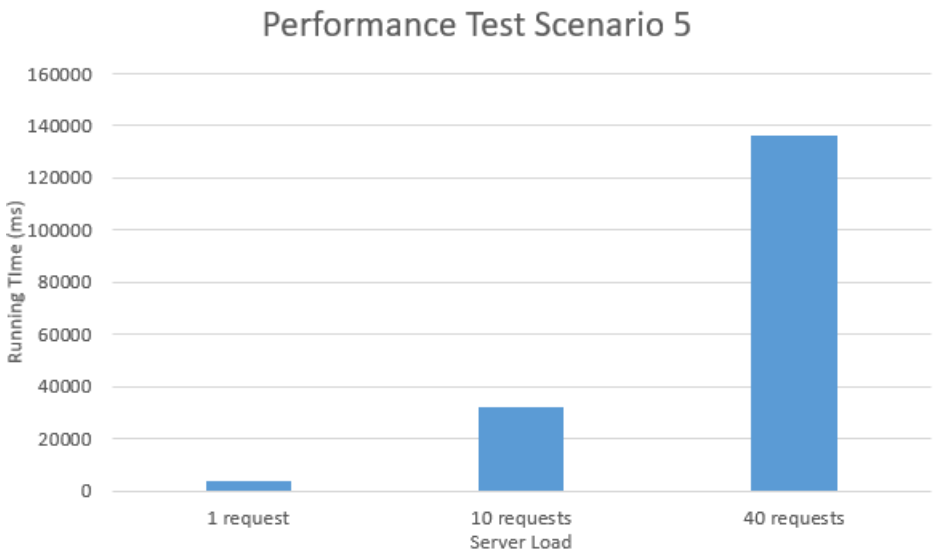

Figure 35 Performance test scenario 5

As is possible to be observed in the Figure 35, running an experiment is the job that takes more time to be processed, because it is the most complex task that the prototype can execute. In this test is confirmed once again, that the running the workflow executor (RapidMiner) creates more overhead in the system when comparing to the other relatively easy tasks, and so it needs more time to finish the job. As foreseen due to the infrastructure limitations, the running time required to finish the request, increased a lot with the server load.

5.2 Benchmarking

As referred in 1.2, one of the issues the users face when using the available ML tools online, is their lack of readiness[10]. To use the available solutions the users have to overcome the steep learning curve, because they are not optimized to rapidly allow the user to prototype and deploy his ML creations. So, from the beginning of this project that the premise was develop a truly prototyping ML solution that would enable the user to rapidly build and deploys predictive models in an easy and intuitive way. To support the premise aforementioned, several tests were performed against the project prototype and to compare its viability, the same tests were performed against the Microsoft Azure ML (Raghunathan, 2016). Therefore, a comparison

[10] How suitable is the tool to rapidly prototype and deploy predictive models

between both results was made. The idea was also to test the Amazon ML, which is one of the main solutions available in the internet, but free trials are not available.

To assess the readiness of both tools, two tests were performed: for the first test a small dataset (32 KB) was used to train and test the models, and for the second one a bigger dataset (670 KB) was used. In this evaluation was recorded the time needed for a user to build (create and test) two predictive models, one using a decision tree and other using neuronal network, to deploy them and generate a prediction.

The following table presents the result for the first test, where a small dataset was used to train and test the predictive models:

Table 6 Test result for a small dataset

Tool	Running Time
MLINO Prototype	0min 58s 35ms
Microsoft Azure ML	4min 01s 55ms

Analyzing the results presented in the Table 6 Test result for a small dataset**Error! Reference source not found.**, is possible to conclude that for small data sets the MLINO solution allows the user to build and deploy a predictive model in around $\frac{1}{4}$ of time the required when using the Microsoft Azure.

The results for the second test, where a bigger dataset was used to train and test the predictive models are exposed in the following table:

Table 7 Test result for a bigger dataset

Tool	Running Time
MLINO Prototype	1min 38s 78ms
Microsoft Azure ML	4min 47s 66ms

Through the results exposed in the Table 7, is concluded that the MLINO prototype allows the user to build and deploy a predictive model in around $\frac{1}{3}$ of time the required when using the Microsoft Azure. This result shows that for bigger datasets, the prototype creates a bigger overhead in the infrastructure, taking more time to process it.

Additionally, to support the results aforementioned, is also recorded the user average mouse clicks required to accomplish the test. The results are present in Table 8.

Table 8 User average number of mouse clicks to accomplish the test

Tool	Avg. mouse clicks
MLINO Prototype	23
Microsoft Azure ML	55

With this last test, we can conclude that the user to realize the proposed task, when using the MLINO prototype requires about half of the interactions with the tool, than when using the Microsoft Azure ML.

5.3 Discussion

In this chapter are presented some tests performed to the prototype, and is also exposed the benchmark results of MLINO tool when compared with the Microsoft Azure ML, in terms of readiness to prototype and deploy predictive models. The experimental results suggests that even using a personal computer to process the ML jobs, the MLINO solution have a satisfactory performance. When processing multiple requests at the same time, the prototype take more time, to process them, due to the RapidMiner usage, which creates a big overhead in the computational resources. We believe when implementing the MLINO concept in a scalable environment this issue would be easily overcome, and the performance would be better.

Even though the limitations of the prototype, it still has a reasonable performance, being a platform which is more suitable for fast building and deployment of basic predictive models than the Microsoft Azure ML.

6

Conclusions and Future Work

6.1 Conclusions

Machine learning and predictive techniques have a big role solving modern society challenges. The usage of this techniques in businesses is increasing and by pulling powerful insights, companies are taking advantage against their nearest competitors. This research work concluded that although there's a wide range of solutions that can be used in the most various challenges, still there's lack of diversification and powerful ML solutions that run in the cloud, in order to be more flexible when solving actual business challenges, to be easier to integrate with other systems and to abstract away the technical difficulties when implementing a ML system, thus making the job of creating ML processes easier and faster. We believe that first step towards constructing and operationalizing ML processes in the cloud have been given, several companies

like Microsoft and Amazon have been releasing their own proprietary platforms, Azure ML and Amazon ML respectively.

Despite the existence of some recent ML platform available in the cloud that enable the user to develop and deploy ML processes, this project aimed to develop a tool which is easier and more intuitive to be used, allowing faster prototyping and deployment of basic ML creations. Even though the implementation of the prototype wasn't the optimal, due to software and infrastructure limitations, through a series of experiments, is demonstrated that the final performance of the prototype was satisfactory. When benchmarking MLINO solution against the Microsoft Azure ML, is possible to conclude that the MLINO tool is easier to use, and takes less time when building and deploying a simple predictive model.

6.2 Future Work

As future work, it would be interesting to provide the MLINO platform with some intelligence, like instead of the user having to setup the experiment and choose the algorithms, the user would then set up his problem, and the system would automatically build a predictive model totally adapted to the problem, providing the user with the best results.

Another interesting approach would be to allow the integration of the MLINO, with other systems. To accomplish this, the API would have to be adapted in order to be possible to run jobs not only from the GUI platform, but also from other web application that can communicate through rest.

In order to make the MLINO concept more flexible and complete, some pre-processing data features could be implemented, like data formatting, data cleaning or data sampling. Integrating this features in the solution would enable the user to get the datasets ready for the machine learning processes using MLINO, instead of having to do it using other tools.

Finally as referred before, the MLINO concept aims to solve predictive classification problems, for future work the solution could be extended to other machine learning approaches for example clustering, anomaly detection or even image recognition.

Bibliography

Abeel, T. *et al.* (2008) 'ProSOM: Core promoter prediction based on unsupervised clustering of DNA physical profiles', *Bioinformatics*, 24(13), pp. 24–31. doi: 10.1093/bioinformatics/btn172.

Abeel, T., Peer, Y. Van De and Saeys, Y. (2009) 'Java-ML: A Machine Learning Library', *Journal of Machine Learning Research*, 10, pp. 931–934. Available at: http://www.jmlr.org/papers/volume10/abeel09a/abeel09a.pdf.

Cetinsoy, A. *et al.* (2016) 'The Past , Present , and Future of Machine Learning APIs', *PAPIs 2015*, pp. 43–49. Available at: http://jmlr.org/proceedings/papers/v50/cetinsoy15.pdf.

Eckerson, W. (2007) 'Extending the Value of Your Data Warehousing Investment', *Advances*, pp. 1–36.

Ethem and Alpaydin (2004) *Introduction to Machine Learning Second Edition*. Available at: http://cs.du.edu/~mitchell/mario_books/Introduction_to_Machine_Learning_-_2e_-_Ethem_Alpaydin.pdf.

Fallis, A. . (2013) 'Getting started with Apache Spark', *Journal of Chemical Information and Modeling*, 53(9), pp. 1689–1699. doi: 10.1017/CBO9781107415324.004.

Hardy, B. *et al.* (2010) 'Collaborative development of predictive toxicology applications', *Journal of Cheminformatics*, 2(7), pp. 1–29. doi: 10.1186/1758-2946-2-7.

Hung, S.-Y., Yen, D. C. and Wang, H. (2006) 'Applying data mining to telecom churn', *Expert Systems with Applications*, 31, pp. 515–524. doi: 10.1016/j.eswa.2005.09.080.

Kubat, M., Holte, R. C. and Matwin, S. (1998) 'Machine learning for the detection of oil spills in satellite radar images', *Machine Learning*, 30(2–3), pp. 195–215. doi: 10.1023/A:1007452223027.

Kohavi, R. (1995) 'A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection', *Appears in the International Joint Conference on Articial Intelligence (IJCAI)*, 5, pp. 1–7. doi: 10.1067/mod.2000.109031.

Li, L. E. *et al.* (2017) 'Scaling Machine Learning as a Service', *Proceedings of Machine Learning Research*, pp. 14–29. Available at: http://proceedings.mlr.press/v67/li17a.html.

Murphy, C. and Kaiser, G. (2007) 'An Approach to Software Testing of Machine Learning Applications'. doi: 10.1.1.103.3244.

Ooms, J. (2014) 'The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns', *arXiv*, (2000), pp. 1–23. Available at: http://arxiv.org/abs/1406.4806.

Raghunathan, S. (2016) 'AzureML : Anatomy of a machine learning service', *PAPIs 2015*, pp. 1–13. Available at: http://jmlr.org/proceedings/papers/v50/azureml15.pdf.

Reid, M. D., Montgomery, J. and Drake, B. (2016) 'Protocols and Structures for Inference : A RESTful API for Machine Learning', *PAPIs 2015*, pp. 29–42. Available at: http://jmlr.org/proceedings/papers/v50/montgomery15.pdf.

Ribeiro, M. *et al.* (2015) 'MLaaS: Machine Learning as a Service', *the IEEE International*

Conference on Machine Learning and Applications, (c). Available at: http://publish.uwo.ca/~kgroling/papers/MLaaS.pdf.

Siegel, B. Y. E. (2013) 'Predictive analytics', *Harnessing the power of big data. BY*. Available at: http://www.predictiveanalyticsworld.com/book/Predictive Analytics - The Power of Big Data - Siegel in Analytics Magazine - July-Augus 2013.pdf.

Stuart, J. R. and Peter, N. (2003) *Artificial Intelligence a Modern Approach, Artificial Intelligence*. doi: 10.1017/S0269888900007724.

Studio, R. (2017) 'How to Correctly Validate Machine Learning Models Basics of Predictive Models and Validation'.

Ulmer, B. and Fernandez, M. (2013) 'Predicting Soccer Match Results in the English Premier League', p. 5. Available at: http://cs229.stanford.edu/proj2014/Ben Ulmer, Matt Fernandez, Predicting Soccer Results in the English Premier League.pdf.