Pontifícia Universidade Católica
do Rio de Janeiro

Jefry Sastre Pérez

An Agent-based Software Framework for Machine Learning Tuning

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduacão em Informática of the Departamento de Informática do Centro Técnico Científico da PUC– Rio as partial fullfilment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Carlos José Pereira de Lucena

Rio de Janeiro

March, 2018

Pontifícia Universidade Católica
do Rio de Janeiro

Jefry Sastre Pérez

An Agent-based Software Framework for Machine Learning Tuning

Dissertation presented to the Programa de Pós-Graduação em Informática, of PUC-Rio, in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the undersigned Examination Committee.

Prof. Carlos José Pereira de Lucena
Advisor
Departamento de Informática – PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática - PUC-Rio

Prof. Simone Diniz Junqueira Barbosa
Departamento de Informática - PUC-Rio

Dr. Marx Leles Viana
Pesquisador Autônomo

Prof. Márcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Tecnico Cientifico – PUC-Rio

Rio de Janeiro, March 22th, 2018

Jefry Sastre Pérez

B.Sc. in Computer Science, University of Havana 2015. He is a researcher member of the Software Engineering Laboratory at the Pontifical Catholic of Rio de Janeiro since 2016. His main studies are related to the area of software engineering, artificial intelligence and multiagent systems.

Bibliographic data

Sastre Pérez, Jefry

An Agent-based Software Framework for Machine Learning Tuning / Jefry Sastre Pérez; advisor: Carlos José Pereira de Lucena. Rio de Janeiro: PUC-Rio, Departamento de Informática, 2018.

55 f. : il. (color); 30 cm

1. Dissertacão (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2018.

Incluí bibliogrfia.

1. Informática – Teses. 2. Sistema Multiagente. 3. Engenharia de Software. 4. Aprendizado de Máquina. I. Lucena, Carlos José Pereira de. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgements

I would like to thank Professor Lucena for his expert advise throughut the research process.

I would like to thanks all my colleagues for their wonderful collaboration

This project would have been imposible without the support of CNPQ and CAPES. Thank you.

Abstract

Sastre Pérez, Jefry; Lucena, Carlos José Pereira de. (Advisor). **An Agent-based Software Framework for Machine Learning Tuning**, 2018. 55p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Nowadays, the challenge of knowledge discovery is to mine massive amounts of data available online. The most widely used approaches to tackle that challenge are based on machine learning techniques. In spite of being very powerful, those techniques require their parameters to be calibrated in order to generate models with better quality. Such calibration processes are time-consuming and rely on the skills of machine learning experts. Within this context, this research presents a framework based on software agents for automating the calibration of machine learning models. This approach integrates concepts from Agent Oriented Software Engineering (AOSE) and Machine Learning (ML). As a proof of concept, we first train a model for the Iris dataset and then we show how our approach improves the quality of new models generated by our framework. Then, we create instances of the framework to generate models for a medical images dataset and finally we use the Grid Sector dataset for a final experiment.

Keywords

Agent-oriented Software Engineering (AOSE); Machine Learning; Calibration Process.

Resumo

Sastre Pérez, Jefry; Lucena, Carlos José Pereira de. **Um Framework Baseado em Agentes para a Calibragem de Modelos de Aprendizado de Máquina**. Rio de Janeiro, 2018. 55p. Dissertação de Mestrado - Departamento de informática, Pontifícia Universidade Católica do Rio de Janeiro.

Hoje em dia, a enorme quantidade de dados disponíveis online apresenta um novo desafio para os processos de descoberta de conhecimento. As abordagens mais utilizadas para enfrentar esse desafio são baseadas em técnicas de aprendizado de máquina. Apesar de serem muito poderosas, essas técnicas exigem que seus parâmetros sejam calibrados para gerar modelos com melhor qualidade. Esses processos de calibração são demorados e dependem das habilidades dos especialistas da área de aprendizado de máquinas. Neste contexto, esta pesquisa apresenta uma estrutura baseada em agentes de software para automatizar a calibração de modelos de aprendizagem de máquinas. Esta abordagem integra conceitos de Engenharia de Software Orientada a Agentes (AOSE) e Aprendizado de Máquinas (ML). Como prova de conceito, foi utilizado o conjunto de dados Iris para mostrar como nossa abordagem melhora a qualidade dos novos modelos gerados por nosso framework. Além disso, o framework foi instanciado para um dataset de imagens médicas e finalmente foi feito um experimento usando o dataset Grid Sector.

Palavras-chave

Sistemas Multiagentes; Aprendizado de Máquina; Calibragem de Modelos.

Content

List of Figures

List of Tables

Introduction

According to ("Data Never Sleeps 5.0 | Domo," 2017) as shown in Figure 1, every minute on the internet over 4 million queries are made on Google, more than 200 million emails are sent and users share almost 2.5 million pieces of content on Facebook, among other actions. The amount of data generated is growing exponentially (Ranganathan, 2011) and we need to be prepared to face all the challenges upfront. Peter Norvig at Google's Zeitgeist Conference (2011) refers to this matter, stating:

"We don't have better algorithms. We just have more data".

Indeed, the huge volumes of data available are a massive challenge to be accepted, but at the same time a vast opportunity to learn from the data to generate more expert artificial intelligent softwares and to enhance the knowledge discovery processes (KDD).

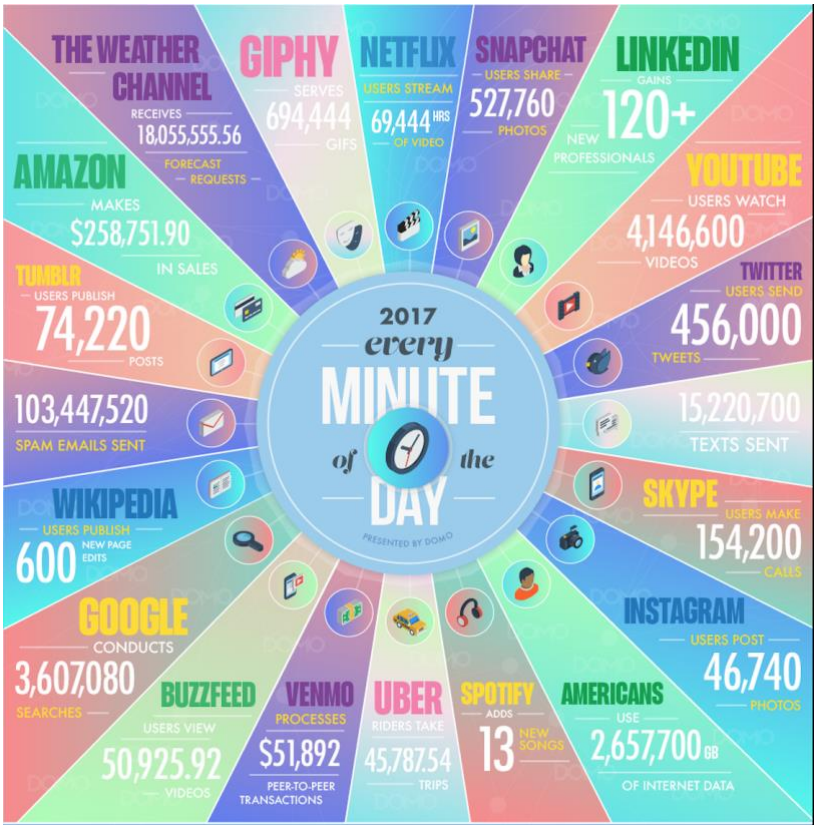

Figure 1. Data generated every minute in 2017

One of the most popular and widely used approaches to generate knowledge is through machine learning techniques. In order to obtain better results from machine learning algorithms, we need to adjust their parameters. This calibration process makes the resulting models more accurate and, certainly, more profitable;

but the drawback at stake is time. The tuning process is generally done by hand, is highly time consuming and strongly relies on the skills of machine learning experts, turning it into an extenuating process.

Within this context, we foresee a chance to incorporate knowlegde from the Agent Oriented Software Engineering (AOSE) area to automate the process of tuning the models prone to the generation of more accurate models and, at the same time, reduce efforts dedicated to produce more accurate models. Agents are software components with autonomy, reactiveness, proactiveness and social capabilities (Lucena & Nunes, 2013). Agent autonomy comes in handy when a system needs to make its own decisions. The agents could also use their proactiveness to guide the tuning process. As a result, it is possible to see how multiagent systems can contribute to the automation of machine learning. To solve this problem, we propose a framework based on software agents to handle the tuning of machine learning models.

1.1 Research Problem

Taking into account the context involving this research, our research question is: "How can agent-oriented software engineering contribute to the machine learning area making a suitable use of well-established methodologies?"

In order to solve the main problem, we define the following questions:

RQ1: Is it possible to design a framework for generating machine learning models?

RQ2: Can software agents be created to train and tune machine learning models?

RQ3: Will the agents be able to improve the performance of the models?

Inside the automatic machine learning area, we target the problem of calibrating the parameters of a machine learning model, not the selection of the first model.

1.2 Objective

To tackle the research question, we define our objective as:

Propose a framework embedded into a multiagent system to handle the tuning of data mining processes in conformance with existing methodologies.

To achieve this goal, we define the following specific objectives:

- Design an environment to save the related entities in a knowledge discovery process;

- Develop a process to allow users to create and plan their experiments;
- Create a software agent capable of training the scheduled experiments and collecting results;
- Create a software agent capable of tuning the hyperparameters of the trained models and propose new models based on the best parameters selection.
- Allow users to test and exploit the generated models.

1.3 Expected Contributions

Some contributions are:

- The system proposes new models that may have good performance based on the models previously trained. This includes a new set of possibilities in the selection of the ways and strategies that will guide the optimizations. The system allows the creation of a committee of models to predict and negotiate a consensus among all the predictors in order to deliver a solution. In addition, the results of the system do not depend on a single trained model, but on a set of models that might be specialized at detecting specific characteristics.
- The framework could reduce the time spent by the user to train a successful model with a multiagent system to support the training process. The idea is to configure some of the training and allow the system to handle the training results, the timing and the long wait for the end of the training and the start of a new one without human interference.
- Validate three case scenarios: (i) Iris, this case is an exploratory study taken as a proof of concept but instantiating the framework and exploiting the agents to generate new models; (ii) Lung Detection, this dataset contains medical images provided by specialized equipment. In the images, the lung tissue can be detected and the goal of this experiment is to provide the framework with low performance models and evaluate the performance of the models genererated by the agents; (iii) Grid Sector, this dataset contains images from Google Earth with constructed areas with different geographical characteristics from 12 areas of Brazil.

1.4 Work Structure

This dissertation is organized as follows. Chapter 2 gives an overview of the main concepts, methodologies, and patterns used along the research. Chapter 3 shows the related work divided into three categories: domain expert focus, code oriented and the AutoML Challenge. Chapter 4 presents the framework and explains each part of the development process in details. Chapter 5 describes the experiments. Finally, chapter 6 offers the conclusion and future work.

2 Background

This chapter describes the main concepts related to agents and multiagent systems. First, we discuss frameworks, multiagent systems, agents and their properties. We also discuss the relation between multiagent systems and machine learning. We review some of the most widely used methodologies in literature for the KDD process. Finally, we provide a brief explanation of the gamma correction algorithm.

2.1 Frameworks

A framework is not an application in itself: it is a generator of specific domain related applications. Those applications are gathered together in a family of problems (Markiewicz & de Lucena, 2001). The common framework development process contains three remarkable stages: (i) domain analysis includes a survey of requirements, expectations and experiences. During this stage, existing solutions and possible standards related to the problem in question are examined; (ii) framework design is to design the project and highlight the flexibility points of the architecture taken into account the requirements and discoveries gathered in the initial stage. In this stage, the design patterns to be used when coding are shaped; and (iii) framework instantiation is to generate applications by implementing the extension points made available and using the shared code that solve the common aspects of the family of problems; the same aspects, that gathered together in the first place.

2.1.1 White, Gray and Black boxes

A framework can be seen as a box that has inside functionalities to solve common aspects of a group of problems. In the literature those boxes can be classified in white, gray, or black, according to their extensibility capabilities. A white box framework is extensible only by coding the missing parts of the program. In general, this kind of framework requires an advanced understanding of the fixed internal code of the framework because the developers have to finish the implementation of the application. A black box framework, on the other hand, generates applications using configuration scripts, and an automated tool

generates the source code. These kinds of frameworks are often simpler. A gray box is a framework somewhere in the middle between white and black boxes. This kind requires both configuration files and coding to generate instances. The complexity of this kind of framework varies depending on how much is needed to configure or to be extended.

In our proposed solution we adopt a white box approach because some of the hot spots are not available at design and must be extended from code.

2.1.2 Hot spots and frozen spots

During the modeling of the framework the flexible points to be extended are highlighted and the fixed modules are shared among instances. The parts of the framework that must be implemented to generate new instances are called hot spots; the imutable parts are called frozen spots. The hot spots are the way a framework has to call instace-specific code. Frameworks are occasionally described as "old code calls new code". The frozen spots are the functionalities common to the family of problems. All frozen spots combined form the kernel of the framework and will be present in all the instances.

2.2 Multiagent Systems

A multiagent system can be defined as an environment shared by autonomous entities that live, interact, receive information and can act in the environment (Khalil, Abdel-Aziz, Nazmy, & Salem, 2015). These agents are abstractions with the following properties (Wooldridge, 2009): (i) autonomy — the capability of taking their own actions within their environment; (ii) reactivity — the capability of responding to the changes in the environment, which involves a notion of perception of the environment; (iii) social ability — the capability of interacting with other agents and possibly humans, and (iv) proactive ability — the capability of taking actions towards the agent's goals.

The exploratory study described in Chapter 5 demonstrated that the agents' properties were useful in the simulation of the training process to optimize the parameters of a model based on the previously trained models and in proposing new models that might be more accurate.

2.2.1 Multiagent Systems and Machine Learning

The idea of joining these two areas seems very natural. In artificial intelligence, we consider that software agents are autonomous entities and are

capable of making decisions without human interference. On the other hand, learning is a crucial part of the autonomy: the more skilled the agent, the better decisions it will make (Alonso, D'inverno, Kudenko, Luck, & Noble, 2001)(Alonso, D'inverno, Kudenko, Luck, & Noble, 2001)(Alonso, D'inverno, Kudenko, Luck, & Noble, 2001)(Alonso, D'inverno, Kudenko, Luck, & Noble, 2001). Indeed, in most dynamic domains it is extremely hard to predefine the agents' actions, which mostly emerge with new behaviors in order to adapt themselves to the current situation.

There are several aspects to take into account when dealing with machine learning in multi‑agent systems. First, the coordination of agents — there must be some coordination mechanism for agents to engage and interact in some way. Note here that the coordination is supposed to happen at runtime, therefore, it has to be part of the agent's internal activity cycle (Khalil et al., 2015). Second, dealing with cooperation can be a problem when agents need to team up to achieve some goals. Third, the noisy environment — specifically, how to deal with supervised learning when the result can be biased by the noise. Finally, together with the noisy environment comes the partial knowledge; to deal with it, agents use strategies and metaheuristics to guide the search as in (Nouri, Driss, & Ghédira, 2015)(Nouri, Driss, & Ghédira, 2015)(Nouri, Driss, & Ghédira, 2015)(Nouri, Driss, & Ghédira, 2015).

Some approaches use a machine learning model in the agents' activities cycle to take actions (Khalil et al., 2015). Other approaches use a multiagent system — known as multiagent learning (MAL) — to learn (Shoham, Powers, & Grenager, 2007), (Stone, 2007). In the latter approaches the integration of the agents' capabilities and the learning algorithms are combined to solve a problem from another domain. Nevertheless, our approach is a multiagent system applied to a machine learning domain.

2.3 KDD Methodologies

In the data mining field, there have been some efforts to establish the bases and patterns to guide the process. Within this context, SEMMA and CRISP-DM raise as industrial methodologies to mine the data (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). This paper offers a comparative overview of the two standards as implementations of the knowledge discovery process (KDD). The attention paid in this area comes from the recent data explosion on the internet and the large databases generated from numerous companies.

The authors chose these two very well-established standards because they are considered to be the most popular among the industrial and research areas. The authors state that "the five stages of the SEMMA process can be seen as practical implementation of the five stages KDD process". They based this affirmation on the idea that SEMMA is directly linked to the Enterprise Miner software. On the other hand, in CRISP-DM the authors say that it involves pre and pos KDD stages. Those stages are linked to the understanding of the business domain and the deployment for the client.

2.3.1 Knowledge Discovery Process (KDD)

The term "was coined in 1989 to refer to the broad process of finding knowledge in data, and to emphasize the *high-level* application of particular data mining methods." (Azevedo & Santos, 2008).

The KDD process contains five stages as shown in Figure 2, namely: Selection, Preprocessing, Transformation, Data Mining, Interpretation/Evaluation.

- Selection: This stage aims to precisely define a target dataset. It can be done by directly selecting a dataset or a subset of features.
- Pre-processing: This stage focuses on cleaning the data. It means that the data generated most of the times is fiilled with nulls and inconsistencies and need to be cleaned in order to become profitable.
- Transformation: This stage aims at applying some transformation algorithms to generate the final dataset to explore. It is common to use dimensionality reduction algorithms, normalize the data, etc.
- Data Mining: This stage focuses on the search for the required patterns in the data according to the mining objectives.
- Interpretation/Evaluation: "This stage consists of the interpretation and evaluation of the mined patterns." (Azevedo & Santos, 2008)

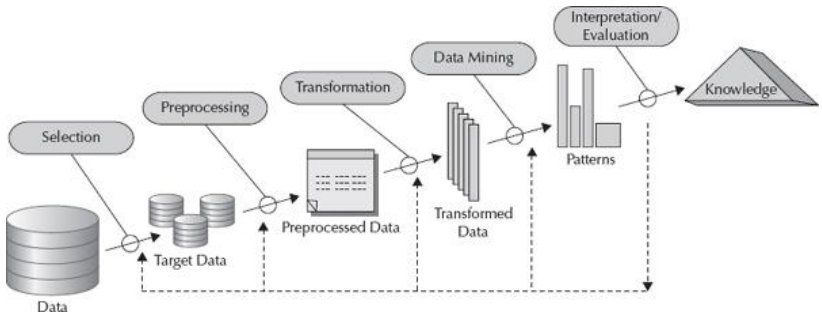

Figure 2 The KDD process

2.4 DICOM Standard

DICOM stands for Digital Imaging and Communications in Medicine and it was created with the purpose of solving the communication and exchange problem between different manufacturers of medical images (Association & others, 2009). The core of the standard is a file format and a network protocol based on TCP/IP. Nowadays, it has almost global acceptance among medical equipment manufacturers and guarantee the interconnectivity in a multi-vendor hospital environment.

The DICOM object is a collection of data elements. A data element is a data structure containing the following fields:

- Tag: It is a unique identifier for each data element.
- VR: It is called representation value and it is the data type of the data element.
- Length: It is the Length of the data in bytes.
- Value: It is the data stream.

The data elements are organized in a hierarchy. At the top of the hierarchy there is the patient and it contains data elements such as Patient's Name, Patient's ID, and Patient's Age. Below the patient there is the exam. The data elements related to the exam are Study Date, Modality, among others. The exams are splitted into Series. A Serie is a logical division of the exam. For instance, an image exam can have a frontal and a sagittal series. The series have data elements related such as Samples Per Pixel, Rows, Columns, and Bits Allocated. A Serie is a collection of images. Images hava data elements associated such as Pixel Data, Image Position, among others.

The DICOM objects are transferred using the DICOM Networking Protocol. This protocol defines nodes and operations through the network. A node in a DICOM network is called Application Entity and is defined by the following properties:

- Host: IP address of the equipment in the network
- Port: Port to establish a connection. By default, the DICOM protocol uses the port 104, but it can be changed.
- AE Title: It stands for Application Entity Title and contains an alphanumeric combination of fixed length of 16 chars.

To communicate different application entities the DICOM protocol define services to support basic and complex operations. For instance, basic operations

such as create, delete, set, and get; and complex operations such as find, move, and echo. Using these services to specify the operation, the DICOM nodes apply the following communication steps:

- Association Establishment: In this step, the DICOM nodes uses the A-Associate-RQ (DICOM association request), the A-Associate-AC (DICOM request acceptance), and the A-Associate-RJ (DICOM association rejection) to start a connection between two application entities.
- Data Transfer: In this step are sent the P-Data-TF (DICOM block data transfer) from one application entity to another. The P-Data-TF message contains the DICOM objects with the data elements relatives to the exams.
- Finalization: In this step, the application entities uses the A-Release-RQ (finish association request) and the A-Release-RP (answer to a finish association request) to safely close the connection between the two application entities.

At any point during the communication the A-Abort message to abort any invalid association can be sent.

2.5 Quality metrics for the classification problem

In this section, we will discuss the metrics used in the Experiments chapter to analyze the results. In this work we will focus only on the classification problem.

	Real Positives	Real Negatives
Prediction Positives	True Positives	False Positives
Prediction Negatives	False Negatives	True Negatives

Figure 3 Traditional notation in a binary classification confusion matrix

Figure 3 shows a relation between the true classification in the columns and the predicted classification in the rows. The True Positives value (TP), holds the number of instances predicted as true and in fact were true. The True Negatives value (TN) is the number of instances predicted as false and they were really falses. The False Positives value (FP), counts the number of false values predicted

as true while the False Negatives value (FN) counts the number of true values predicted as false.

Based on this representation, there are several metrics commonly used to evaluated prediction results:

- Accuracy: It is the number of correctly classified instances (Powers, 2011).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision: This measure takes into consideration the number of false positives. The higher the precision, the fewer errors the model make when it predicts true (Fawcett, 2006).

$$Precision = \frac{TP}{TP + FP}$$

- Recall: This measure takes into consideration the number of false negatives. The higher the recall, the fewer errors the model make when it predicts false (Fawcett, 2006).

$$Recall = \frac{TP}{TP + FN}$$

- F1: This measure combines Precision and Recall and balance their problems. For instance, a maximum Recall can be obtained by always predicting true (Fawcett, 2006).

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

3 Related Work

Many authors (Garner & others, 1995; Kraska et al., 2013; Kunft, Alexandrov, Katsifodimos, & Markl, 2016; Luo, 2016; Sparks et al., 2013) have created systems to support the data mining process. A common discussion among all authors is about the target user and the environment — some systems are designed to be used by domain experts and others by data mining experts. Systems dedicated to non-experts normally focus on the analysis of the domain-specific features while other systems propose educational environments for novices to learn and interact. On the other hand, systems designed to be used by data mining experts focus on performance, optimizations and coding capabilities.

3.1 Systems focused on domain experts

Within the context of non-experts, (Abadi et al., 2016) present a simulation tool that aims at creating an initial insight on neural networks with a very user friendly interface and it has proven to be a great choice for educational purposes. However, the datasets available for analysis are fixed and focused only on gaining some understanding of the learning process.

There are some commercial solutions, such as Azure Machine Learning (Barnes, 2015), Amazon Machine Learning (Nketah, Gabriel Uchechukwu, 2016) and Rapidminer (Klinkenberg, 2013). All of them are online services and provide support to the data mining process by means of an intuitive interface and a huge collection of ready to use algorithms.

3.1.1 Azure Machine Learning

Microsoft Azure offers cloud computing services designed for both developers and data experts to provide them with build, deploy and management services on the cloud. By using the Azure services the platform incorporates a layer of security, safety and stability provided by their data centers. Azure integrates with Xamarin for mobile development and with the Bot Framework to enable the interaction between users and bots. Among their products there is a machine learning solution called Azure ML Studio. It is a very complete drag and drop application that integrates algorithms, statistics and analysis in a combined

environment. To save time they also offer a set of pre-built templates with the most widely used and common strategies.

To set the cloud programming environment it is necessary to have a valid Microsoft account, a valid credit card and an updated web browser. Azure offers a trial period of 30 days with $200 to spend in services. The system is designed to be used just with drag-and-drop operations and to easily integrate within the Microsoft platform. Programming turns out to be connecting the objects in the board with arrows to set up the workflows of the experiment and setting all the required parameters during the workflow, like the amount of data to be separated and saved for the testings at the end.

Due to the fact of the visual programming and workflows creation the Azure ML Studio is well suited for domain experts or newbies on machine learning and data analysis.

3.1.2 Amazon Machine Learning

Amazon Machine Learning is a service offered by Amazon on their Amazon Web Services (AWS) platform to create machine learning models and generate predictions. This service is based on a premise, support scalability. Users can start consuming few resources and grow along with the applications. The process with Amazon comprises three stages: data analysis, model training, and evaluation. The data analysis process is to understand the data and to offer possible transformations to optimize the training model. The model training step finds patterns on the dataset and the evaluations stage calculate the accuracy of the predictions.

The Amazon solution offers key features, such as integration with AWS to facilitate the work with data already in the cloud. It provides interactive charts to help you understand the data and interpret the results of the models. The models used within the solution can be binary or multi-class classification models or regressions models. The data sets often require some pre-processing and Amazon Machine Learning offers the most common data transformations ready to be used and which will be suggested when the users upload their data. All the necessary resource to run the models are auto-managed internally by the Amazon services.

Amazon Web Service offers a 12-month free trial with limited resources and number of service calls.

3.1.3 Rapidminer

Rapidminer is an online platform to support data science and machine learning experiments on the cloud. They offer a visual solution with more options than the current existing solutions, it is open (not free) and extensible to cover specific needs. The solutions offer a unified platform to support the whole process, from the data preparation to the deployment of the results. They based the solution on visual programming, allowing domain experts to create workflows and use the data mining algorithms. They support the idea that drag-and-drop programming is easy to learn, easy to use, and time saving. This platform offers a wide number of predefined models to extract patterns and analyze and accept the use of open technologies well accepted by the data-science community. They guarantee the ability and the capabilities to work with big data and to perform statistical analysis with a cost-effective performance.

3.1.4 WEKA

WEKA (Garner & others, 1995; Hall et al., 2009) is a system that offers a collection of algorithms to explore real world datasets. It has three well defined categories of algorithms: (i) dataset processing, (ii) machine learning schemes, and (iii) output processing. One of the most remarkable advantages is that it contains many of the existing algorithms implemented and tested, ready to be used. By combining all these tools together, WEKA has proved essential to the analysis process and as an introductory tool for educational environments. However, this tool does not focus on the challenges of use ML in distributed ways; all its algorithms are presented as black boxes and do not focus on distributed ways to improve the data mining processes, and WEKA requires some expertise to choose the most profitable algorithm.

3.2 Code-Oriented Solutions

All these solutions focus on reducing usage complexity, tuning hyper parameters and gaining some understanding of the data, but none of the previous approaches aims at creating a safe shared environment to enhance the interaction between the users and the system. By using the agent's capabilities, users and agents can both solve the data mining process, complementing each other's weaknesses.

3.2.1 Google Prediction API

Google Prediction (Green & others, 2011) offers a cloud computing solution that includes computing, storage, network, security and easy access. Among the modules in the solution there is a Cloud Machine Learning Engine, which provides services to build machine learning models. The models create a Google Cloud Machine Learning Engine uses the Tensorflow framework to create the models and easily integrates with Google Photos and Google Cloud Speech. Models are rapidly available for prediction and the services are integrated with the Google Cloud Dataflow for pre-processing.

Google's solution offers integration among their services, manage all the necessary resources for training and publishing services to consume the predictions of the trained models. The user creates the hardware configurations to run the experiments, which are easily scalable to new configurations, or to enable some parallel processing. Finally, through the use of the Tensorflow framework, they support locally training and integration with the cloud platform anytime, as well as the other way around: It is possible to train a model online and then download and test locally.

To configure the Google Prediction API, the users need a valid Google account, a valid credit card, and a project on the Google Cloud Platform services with both services activated: the Google Prediction API and the Google Cloud Storage. It has a credit of $300 of free trial to use in 12 months on the services of the platform. From that point, all the process is made by using the cloud services provided by google. Google Prediction API requires web knowledge and understanding of the RESTful web services to be fully used and is best suited for data mining experts and code first programmers “Code first, ask later”.

3.2.2 ML Base

ML Base (Kraska et al., 2013) provides a Domain Specific Language (DSL) with high level abstractions to simplify the process. This solution is focused on the optimization capabilities and, in both phases, the model selection and the optimization capabilities. The authors state that the extraction of the valuable data in the Big Data context is a great concern and that the layman users are overwhelmed by the theory behind the existing algorithms.

Within the architecture of the MLBase, there is a Logical Learning Plan, which performs sub-samplings of the data, feature analysis, creates possibilities

of parameters combinations and tries to eliminate some of them with the help of some optimizer and some predictions of the resulting performances. After the optimizations, then a physical execution plan is created. The focus of the work is on the optimizations for ML and they put aside the integration with relational models. ML Base presents a DSL providing a high-level abstraction to create a system more accessible to non-experts. The core of the solution is the optimization, capable of turning a declarative language into a very elaborated execution plan. The solution aims at solving a problem with a single model; however, the composition of models that create ensembles has been proven to outperform single models, and according to (Luo, 2016), many algorithms and large datasets can be slow and limited.

3.2.3 LARA

The work (Kunft et al., 2016) is motivated by the problems generated in the data analysis, due to the fact that the preprocessing process and the algebra are usually done in separated languages. The need for two different languages not only affects the productivity, but also reduces the optimization capabilities and often requires additional code to transfer the datasets from the relational environment to the algebraic. Within this context the authors present LARA, an embedded DSL in Scala, which offers suport for both relational and algebraic operations. LARA compiles into an intermediate representation to enable optimizations and finally compile to different languages.

The solution includes data types such as Bag, Matrix, and Vector. All of them are intended to be generic and built to be extensible. Among the optimizations, there are Matrix *Blocking Through Joins*, which allows distributed matrix operations, *Row-wise Aggregation Pushdown* and *Caching Throughout Iterations*, bringing several improvements.

On the other hand, it is embedded into Scala and requires knowledge of Scala to be used. Also, this tool requires expert knowledge to perform the KDD process.

3.2.4 Other Solutions

There are some solutions that target data mining experts and focus on tools to improve the techniques. MLI (Sparks et al., 2013) presents an API to easily code machine learning algorithms, using their proposed operations for data loading and linear algebra to boost the performance; but it relies on the expertise

of the programmers rather than the use of previously tested and well established implementations of the algorithms

Predict-ML (Luo, 2016) is a software that uses big clinical data to build predictive models automatically. It presents techniques to automatically select algorithms, hyper parameters and temporal aggregations of the clinical data, but the innovations are focused on the clinical area and the system is still in the design phase.

3.3 AutoML

AutoML is a new area in computer science, pursuing the progressive automation of the machine learning process (Guyon et al., 2015). This area addresses all aspects regarding machine learning automation, such as search and selection of model, hyperparameters optimization, feature engineering, meta learning and transfer learning, among others.

Within this context, a challenge to boost new solutions towards the AutoML goals was created. This challenge includes a novel design element: code submission. The code runs in an open-source platform ensuring there is no human intervention during testing phases and that all proposed solutions run on hardware equality. The challenge contains six phases in which the dataset difficulty is progressively increased. After each phase, the competitors have a Tweakathon time to improve their method with access to the previously tested datasets. This challenge aims at advancing the theoretical state of the art about model selection, implementing useful automation solutions, a chance to compare results of the automatic software and the Tweakathon phase and to disseminate the top solutions and papers.

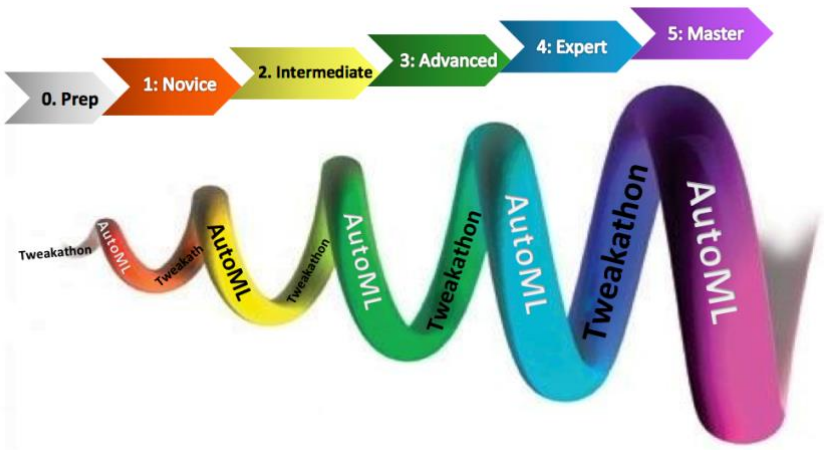

Figure 4 Design of the AutoML challenge

There are five AutoML phases as shown in Figure 4: Novice, Intermediate, Advanced, Expert and Master. The novice phase is the easiest because the prediction variable is binary, has less than 2,000 features, the classes are balanced and there are no missing data categorical variables. Nevertheless, the data matrix can be sparse and there are irrelevant features. The second phase can have either binary or multiclass classification problems, unbalanced classes, fewer than 5,000 features, missing values and some categorical variables. In the third phase, the competitor can face any kind of classification problem and the number of features skyrockets up to 30,000. In the fourth phase, there are no data complexity restrictions and regression problems are also included. In the fifth phase, the datasets used have never been cited nor used in previous works. So, the final challenge is to learn from completely new dataserts. The data is extracted from multiple domains and some ongoing developmnt technologies, such as: speech recognition, medicine, etc. All the datasets are pre-processed and prepared into fixed length feature-based representation.

For each task, there are limited computational resources. The participants willing to submit code share the same hardware configuration and time limits. The participants only submitting results are not able to participate in the AutoML phases (only the Tweakathon phases), but there are no time or hardware restrictions. According to the kind of problems, different metrics are used to evaluate the results. The metrics are also rescaled in a way that the baseline results are 0 and the best possible results are 1.

4 Proposed Solution

This chapter describes the main elements required to understand the solution proposed in this work. In addition, we will provide an overview of the architecture and discuss the different components, including the data model and the software agents.

4.1 The basic functionality understanding: The frozen spots architecture

The application is implemented as a framework using software agents, as illustrated in Figure 5. It contains a module for: (i) data storage (DB); (ii) data access (ORM); (iii) agents; (iv) optimizations (OPT); and (iv) API layer — which will bring the functionalities to the final user.

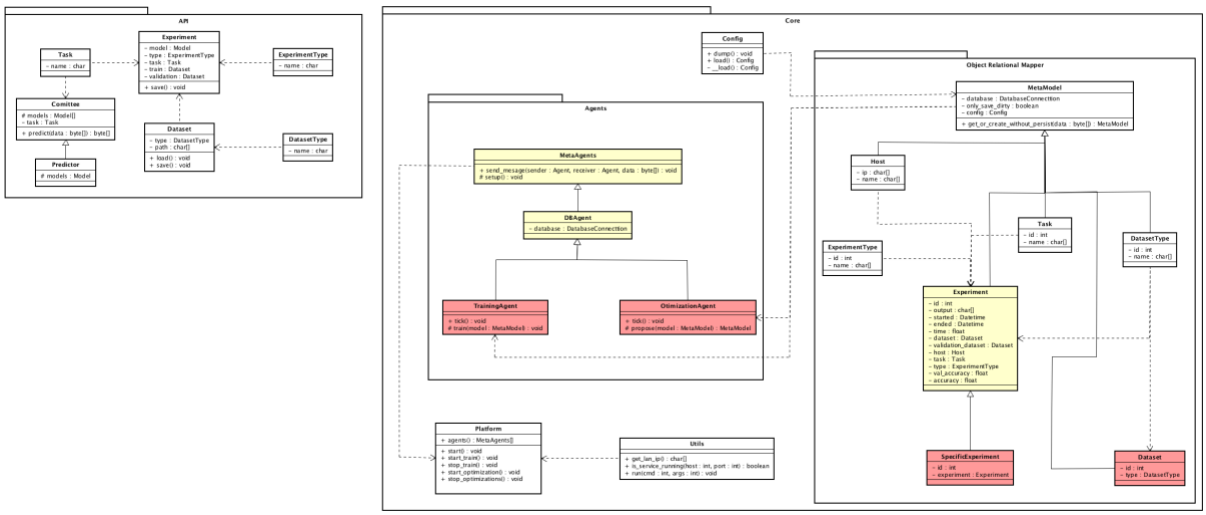

Figure 5 The proposed architecture

The users interact with the system via the API. The system contains several classes to support the knowledge discovery process. The API is directly connected to the ORM. The ORM is in charge of all the operations that require data access. It allows the system to be independent from the physical data storage and it is also the only way to interact with the data. The data refers to the relevant concepts that appear in the domain and their relationships. All of them are physically saved in the DB module. Considering the user's experience, the main flow of the application only involves the API, the ORM and DB modules. The software agents interact in this flow via the ORM module and expertly use the main application flow the same way as normal users do. They retrieve, run and propose new

experiments in a collaborative environment. By working together, the users (as domain experts) and the agents (as machine learning experts) increase the number of experiments, searching for a better model to identify the desired patterns. The agents in charge of the optimizations trust most of the algorithmic analyses in the fifth and last module dedicated to the Optimizations. The TrainerAgent and the OptimizerAgent are both hot-spots (Wooldridge & Jennings, 1998). Therefore, it is possible to add new models into the system by creating subclasses and implementing the particular details of the new model.

By using the API, the users can evaluate the results, that is, they can check if the results meet the initial objective. This phase is crucial, because the models selected to be deployed will finally be in contact with non-controlled environments and real-life mining examples. Nevertheless, if the users determine that the models are not ready to be used, they can define a new experiment or allow the agents to search for better models. At all times, the users can monitor the results obtained, then, analyze, retrieve and compare several of the model's parameters.

4.2 Data Model

Figure 6 presents the data model of the concepts involved in the problem. We used the entity-relationship model (ERM) (Chen, 1976). The description of the entities is shown below:

• Task: Aims at capturing the training process of a successful model for a machine learning problem, i.e., it is a collection of experiments.

• Experiment: Defines an experiment, but this concept just contains the common aspects, such as running_time, train_accuracy, etc.

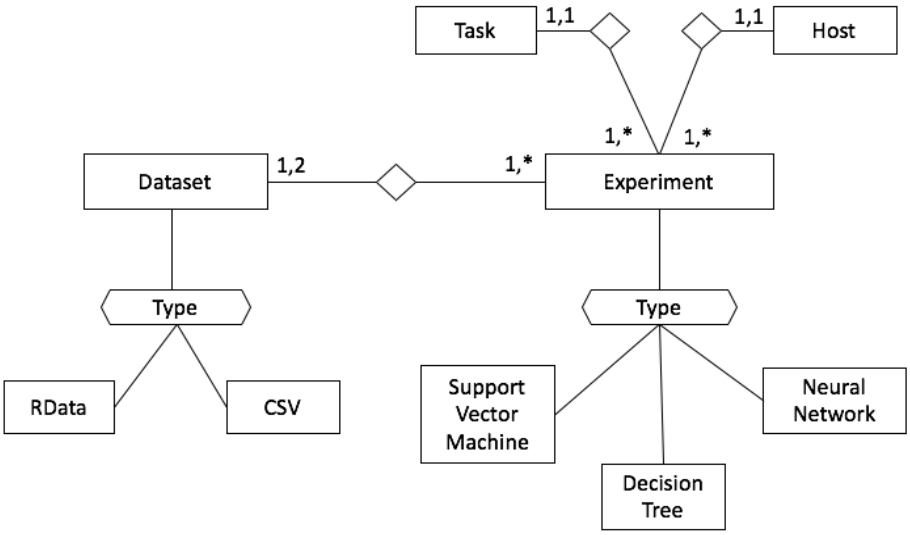

Figure 6 ERM Model

• Decision Tree: Defines a specific kind of experiment. In fact, it defines an experiment to train a decision tree and contains aspects such as max_depth.

• Support Vector Machine: Defines a support vector machine type of experiment and contains attributes such as kernel.

• Neural Network: Defines a neural network type of experiment and contains attributes such as model that specifies the structure of the network.

• Host: Defines a computer in the network, and basically selects the computer in which the model is going to be trained.

• Dataset: Represents a generic data collection, used as the examples to train a model.

• RData: Represents a particular type of dataset generated from a script executed in R (Gentleman, Ihaka, Bates, & others, 1997) and contains the environment variables at the save point.

• CSV: Represents a standard data exchange format. Most of the time it is a collection of comma-separated fields.

4.3 Mapping the Data

We include a layer to map the concepts to the API classes, in order to facilitate the data access and to abstract the project of the database read and write operations. Basically, it is the idea of an Object Relational Mapper (ORM). This new layer provides stability and independence for the following layers to use, allowing: (i) the change of the data provider without changing the core of the project, and (ii) the design of the logic without specific read, write operations that might bind the solution to a particular data access.

4.4 Agents Model

This section presents the agents model based on the architecture shown in Fig 1. These agents are able to execute the experiments stored in the data model.

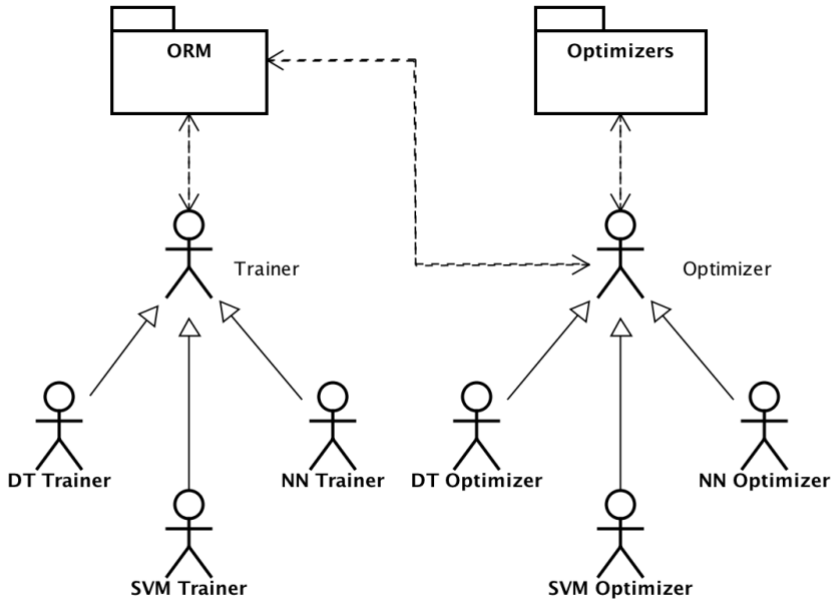

Figure 7 Agents Model

Figure 7 shows the agents-based model proposed. In all the cases, the agent's cyclic behavior was the best option for these software agents – for example, in the application domain presented in Chapter 5 the agents have a cyclic behavior with 10 seconds between iterations.

4.4.1 Trainer Agent

The TrainerAgent is responsible for training an experiment. In order to do so, it has to accomplish several subtasks. First of all, it needs to understand the type of experiment that the agent is going to execute. For each type of experiment, there are different parameters used to set up the training process. Based on these parameters, the agent determines the type of dataset that is going to be used and it loads the data. At this point, the strategy pattern (Gamma, 1995) was used to define which algorithm should be chosen to train and validate the results. After the validation, the agent has to collect all the variables being measured and write the experiment back. Figure 8 describes this process.

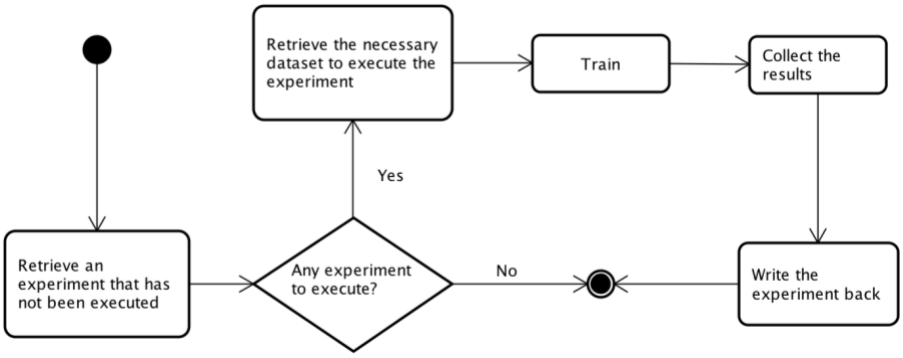

Figure 8 TrainerAgent Activities Diagram

A specific trainer was created to override the specificities of each model and to set up some initialization variables such as the type of experiment.

To run an experiment, both the experiment and the datasets to be used in the training and testing must be previously defined. This process only runs the experiments and collects the results. On the other hand, due to the characteristics of the agent's cyclic behavior, if there are no experiments programed to run, the agent waits a few seconds and asks again. Therefore, once a new experiment is added to the database, it will be automatically detected and executed at the right time.

Another important detail is that the experiments are executed as if they were on a queue — one at a time in each host. But it is possible to program a set of experiments that the agents will automatically run until all the experiments have been executed.

4.4.2 Optimizer Agent

The OptimizerAgent is responsible for generating new models that may have good performance and accuracy based on the previously executed experiments of the same type. To complete this task, the agent starts by selecting a dataset, because the performance and the accuracy are directly related with the dataset used in the training process. Once the dataset is selected the agent retrieves the best experiments of a given type and, based on the parameters, it generates and saves a new model. Notice here that for each type of experiment the OptimizerAgent was extended in order to create specific agents which selected the correct algorithm in each case. Figure 9 describes the workflow of the OptimizerAgent.

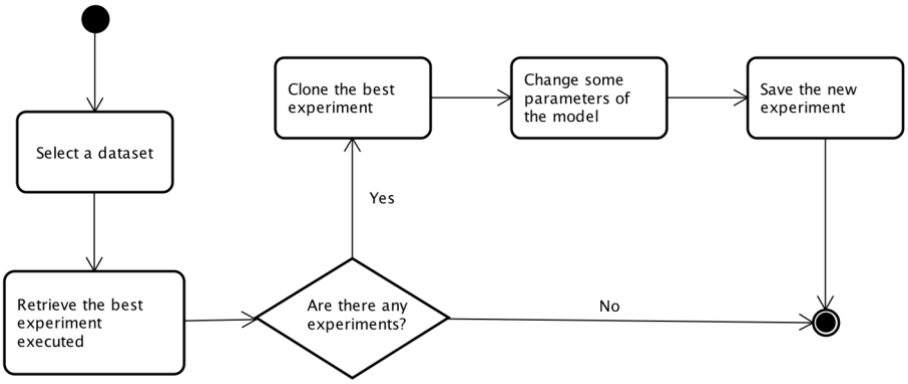

Figure 9 Optimizer Agent Activities Diagram

Observe that the OptimizerAgent needs a different strategy to create the new model, depending on the type of the experiment.

4.5 Optimizers

Each machine learning strategy comes with a lot of tricks and techniques to improve the performance of the model. Some of the techniques can include mathematical operations, such as transpose, reverse, etc., which can increase the dataset and have a direct impact on the performance as a result. Other techniques aim at increasing the number of features in the dataset to facilitate the training process and obtain a better model. Some examples include multiplication of numeric fields or the use of trigonometrical functions. In addition, there is a group of techniques that filter the outliers to obtain a more general model. All these approaches work directly on the dataset, but our focus here is to work with the existing data and tune the model's parameters.

Each one of the techniques has its own unique parameters, so it was necessary to create an optimizer for each one, namely: SVMOptmizer, DTOptimizer and NNOptimizer.

The SVMOptimizer takes advantage of the kernel trick (Scholkopf, 2001) and creates a new model based only on the best SVM experiment executed. If the best model memorizes the dataset, it then decreases the kernel to compact the data. On the other hand, if the model's accuracy is low, then the agent increases the kernel to separate the data by adding new dimensions.

The DTOptimizer uses a similar criterion to increase or decrease the max_depth of the decision tree while the NNOptimizer creates a new model by randomly combining the two best experiments executed.

4.6 Details of the API

Finally, we created an Application Programming Interface (API) that contains the new objects and functionalities required to set up an environment: create, train and validate the experiments; test the results, and use the best models for prediction.

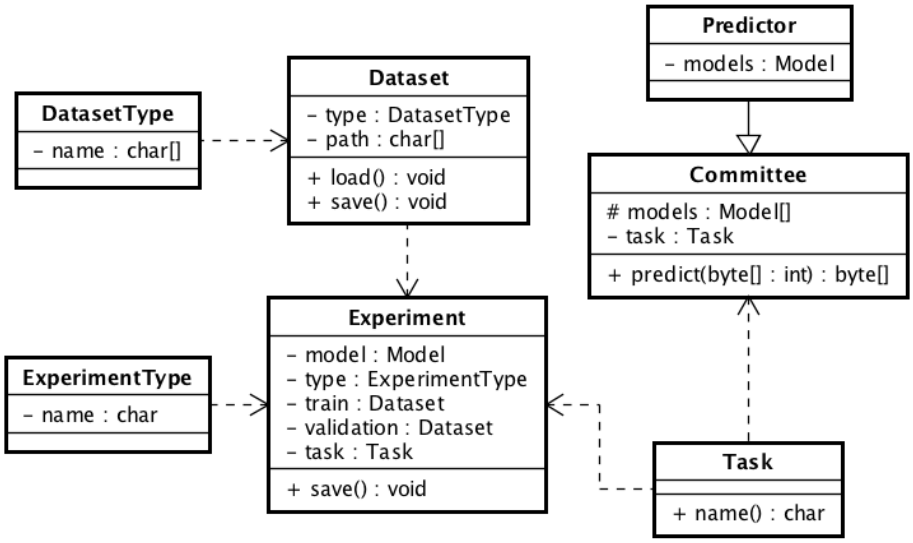

Figure 10 API Class Diagram

Figure 10 shows the API class diagram. The Task class defines a collection of experiments of the same problem and refers to the same machine learning problem. Every machine learning problem requires the analysis of data. The Dataset class represents a collection of data to be used and contains features such as the path in which it is stored. The data can be stored in different file formats. For this reason, each Dataset contains a DatasetType class to specify its type, such as RData, CSV, etc. An Experiment class represents the training process of a model and contains general variables being measured, such as time. It also contains more specific features, depending on the particular model being trained. In order to specify the types of experiments allowed to run within the platform all the Experiments contain an ExperimentType class. The Predictor class defines an object to evaluate a model and the Committee class defines a collection of Predictors and contains a parameter to set the number of members.

First, to use the API, we need to select a Task to work with and after that, the experiments can be created and linked to the selected task. Each Experiment has a type defined in ExperimentType and can have training, validation and testing datasets associated to it, respectively. Each Dataset has a type defined in DatasetType. Finally, to predict, based on previously trained models, there are two possible classes: (i) Predictor, which selects the best trained model based on

accuracy and uses it to predict, and (ii) Committee, which has a collection of predictors and returns a consensus among them.

5 Experiments

The need to build platforms to assist both domain and data mining experts in creating a safe common environment to enhance the interaction between the users and the system to train machine learning models is currently a critical problem, especially when the training process can iterate over several models and the new models depend on the results of the previously executed experiments. The metrics used to evaluate the results were accuracy detailed in (Provost, Fawcett, & Kohavi, 1998) and precision, recall and F1-score detailed in (Powers, 2011).

In this chapter, the experiments made on three datasets are presented: IRIS, Lung Images, and Grid Sector. First, we discuss relevant aspects of the datasets. Then, the framework is instantiated and finally the results are discussed.

5.1 Iris Experiment

The experiment is divided into two stages. First, we set up the environment and create the proper conditions to run the experiment — in this case, it was necessary to launch the agents' platform, to configure the database access and to establish the initial experiment. Second, the agents start their work by training the first model and writing the results. The variables that were measured were the training and validation accuracy, as well as the start and end time. At this point, the OptimizerAgent analyzes the results of the finished experiments and proposes a new experiment using the same dataset.

5.1.1 Iris Dataset

The data used in this example was the IRIS dataset found in the UCI Machine Learning Repository (Bache & Lichman, 2013). It contains 150 instances of three classes of iris plants. The predictable attribute is the type of plant, based on four other attributes: sepal length, sepal width, petal length and petal width — all the measurements are in centimeters (cm). This dataset has no missing values and two of the three types of iris are not linearly separable. Table 1 shows a brief summary of the data.

Table 1 Summary of the Iris Dataset

	Sepal Length	Sepal Width	Petal Length	Petal Width
Min	4.3	2.0	1.0	0.1
Median	5.8	3.0	4.4	1.3
Mean	5.8	3.1	3.8	1.2
Max	7.9	4.4	6.9	2.5

5.1.2 Results

The framework was instantiated as shown in Figure 11. The TrainingAgent and the OptimizationAgent were extended into the SVMTrainingAgent and the SVMOptimizationAgent respectively in order to implement specificities about how to train and optimize an SVM model (Cortes & Vapnik, 1995). In this case, for the optimization agent, we use a grid search approach allowing the parameter C the values 1.0, 1.5, and 2.0 and for the Degree the values 1, 2 and 3. The class SVMExperiment inherits for the SpecificExperiment hotspot and adds the parameters needed to train an SVM experiment. Finally, the FileData class and the CSVData are classes created to store a reference to the dataset.

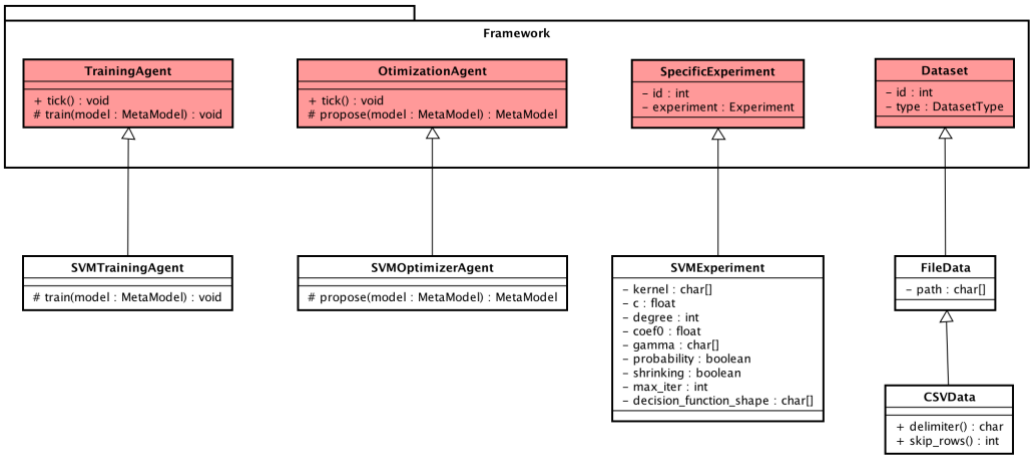

Figure 11 Framework instance for the Iris experiment

The starting point is an instance of the SVMExperiment class and we choose the following parameters, as shown in Table 2:

Table 2 Initial experiment setup for the Iris Experiment

Parameter	Value
Kernel	Polynomial
C	1.0
Degree	1.0

Coef0	0
Gamma	auto
Probability	False
Shrinking	1
Max Iterations	-1
Decision Function	odr

The training agents were essentially training the new models proposed, while the optimizer agents were trying to tune the parameters of the previously executed models and proposing new ones that might have a good accuracy. Table 3 shows the experiments proposed by the OptimizerAgent and Table 4 shows the variables measured.

Table 3 Experiments ran and proposed by the agents for the Iris Experiment

Id	*Kernel*	*C*	*Degree*	*Coef0*	*Gamma*	*Probability*	*Shrinking*	*Max Iterations*	*Decision Function*
1	poly	1.0	1	0	Auto	0	1	-1	odr
2	poly	1.5	1	0	Auto	0	1	-1	odr
3	poly	1.5	2	0	Auto	0	1	-1	odr

Table 4 Metrics recorded by the framework for the Iris Experiment

Id	Time (in milisecods)	Validation Accuracy
1	0.006163	0.96
2	0.004834	0.96
3	0.005739	0.97

The first row in Table 3 shows the beginning of the second stage, when only the first model had been proposed. Then, the TrainerAgent trained the model, resulting in an accuracy of 0.96 (first row in Table 4). The OptimizerAgent performed a query to retrieve the trained models and based on the best one, it modified the allowed error (parameter C in Table 3) from 1.0 to 1.5 and proposed the second model. The TrainerAgent realized that there was a model to train and then trained it, resulting in an accuracy of 0.96 as well. Once again, the OptimizerAgent modified the degree of the function to propose the third model (parameter degree in Table 3) based on the first and the second models. As a result, the TrainerAgent trained the new model and obtained a better accuracy of 0.97.

To obtain new models, the OptimizerAgent balanced the allowed error and the degree of the polynomial function. It is possible to see in Table 4 that the last trained model performed better in the validation. Thus, in the next KDD phase the prediction algorithm will use the best models, based on their accuracy. This

experiment answers our RQ1 and RQ2 because the software agents were able to train and propose machine learning models.

5.2 Lung Images Experiment

The experiment is divided into three stages. The first stage explains the dataset construction process. The second stage describes the initial parameters configurations and the framework instanciation. Finally, we briefly discuss the results.

This experiment aims to identify lung tissue in computerized tomography exams. Those kinds of exams contain several images without lung tissue from upper and lower regions of the body. With a model capable of identify the lung tissue, it would be possible to prune the exams and show to the specialist oly the images containing lung tissue.

5.2.1 Lung Images Dataset

This section is dedicated to the process of get the data ready to be used. This process involves the DICOM protocol to extract data as well as image manipulation techniques to prepare and normalize the data (Section Preprocess) and the annotation process (Section Bitmap Generation).

5.2.1.1 Preprocess

The images of the exams were stored in DICOM format and came from real anonimaized exams. One of the DICOM tags is the PixelArray, this field contains a two-dimension array of densities. Inside the DICOM file there are two other fields BitsStored and BitsAllocated specifying the number of bits used to save each density and reserved respectively. Using those DICOM tags it is possible to normalize the densities. First of all, we shift the densities to fix the minimum in 0 by the equation:

$$PixelArray_{i,j} = PixelArray_{i,j} - \min(PixelArray)$$

where i, j is a position in the matrix.

At this point, and due to the fact that this density map is going to be saved as an image, a sigmoid transformation was applied and it is described in the following equation:

$$PixelArray_{i,j} = \frac{1}{1 + e^{-10 * PixelArray_{i,j}}}$$

where i, j is a position in the matrix.

These sigmoid transformations, also known as contrast adjustment, are commonly used to overcome lightness and contrast loss from rescaling images using dynamic ranges (Braun & Fairchild, 1999).

Note that to apply the sigmoid function the PixelArray must be in the 0 to 1 range. To overcome this problem the array was rescaled using the following transformation before the sigmoid correction:

$$PixelArray_{i,j} = PixelArray_{i,j} * \frac{BitsAllocated}{BitsStored}$$

where i, j is a position in the matrix.

After the sigmoid correction, the resulting array remains between 0 and 1. So, to save it as an image, it has to be rescaled again to values between 0 and BitsAllocated.

$$PixelArray_{i,j} = PixelArray_{i,j} * BitsAllocated$$

where i, j is a position in the matrix.

The final PixelArray is saved using png format. In the rest of the work we will refer these images as the original images.

5.2.1.2 Bitmap Generation

The most important and expensive data are images in which the lungs are completely identified. Indeed, those images are very expensive because are made by humans one by one in order to create an annotated dataset. So, for each original image, we create another image with the lungs highlighted in red as shown in Figure 12.

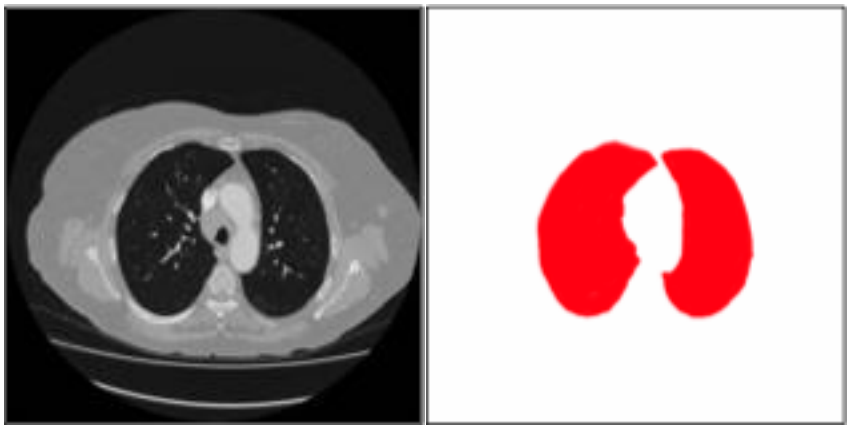

Figure 12 Original and annotated images

The annotated images were in png format, more specific, using an RGBA (Red, Green, Blue, Alpha) configuration. For the purposes of this research, we only needed a bitmap with the same shape of the original image and 1s in the lungs. To do so, the annotated images were transformed into a grayscale image,

resulting into a single byte matrix and finally applied a transformation for all the pixels:

$$bmp_{i,j} = \begin{cases} 1, & if\ grayscale_{i,j} > 127 \\ 0, & otherwise \end{cases}$$

where i, j is a position in the bmp matrix

The bmp image is a matrix of the same dimensions of the original image containing only bits 0 or 1 as shown in Figure 13.

Figure 13 Annotated bitmap image

In the rest of the work we will refer to these images as the bitmap images.

The dataset contains 258 images, 129 original images and 129 bitmap images. In order to use the dataset in a UNet Neural Network (Ronneberger, Fischer, & Brox, 2015) model the images were rescaled. The original images where rescaled from 512x512 pixels to 572x572 pixels. The bitmap images were cropped from 512x512 pixels to 388x388 pixels.

In Figure 14 it is possible to see the whole dataset construction process. Figure 14 (a) is the density map. Figure 14 (b) is the image with the constrast adjusted and rescaled using the BitsStored and BitsAllocated DICOM tags. Figure 14 (c) is the reshaped image with 572x572 pixels. The annotated image is shown in Figure 14 (d). In Figure 14 (e) is shown the cropped image and finaly the bitmap image is shown in Figure 14 (f).

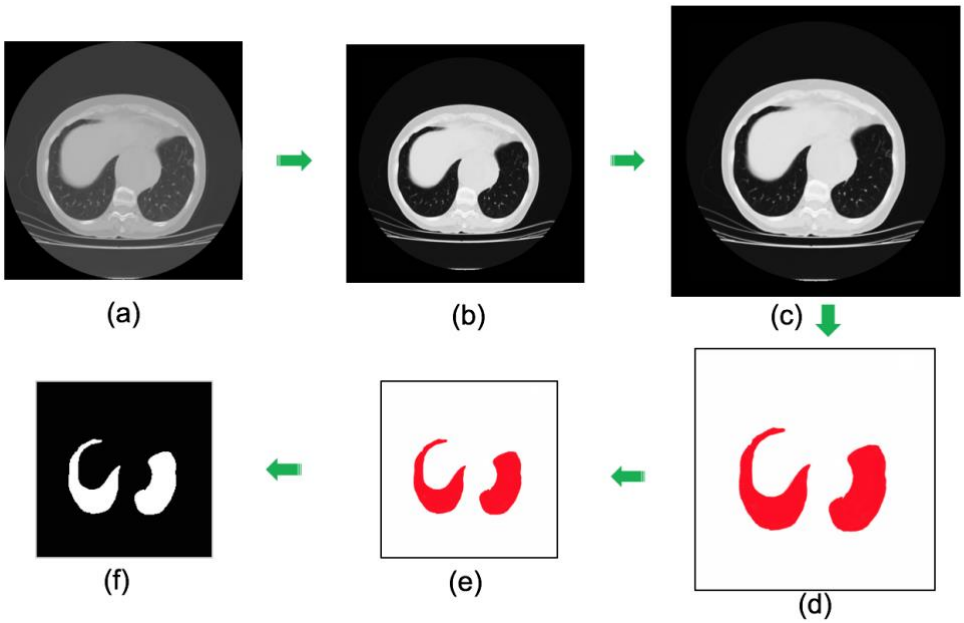

Figure 14 Dataset construction process

5.2.2 Results

The framework was instantiated by extending the corresponding hotspots as shown in Figure 15.

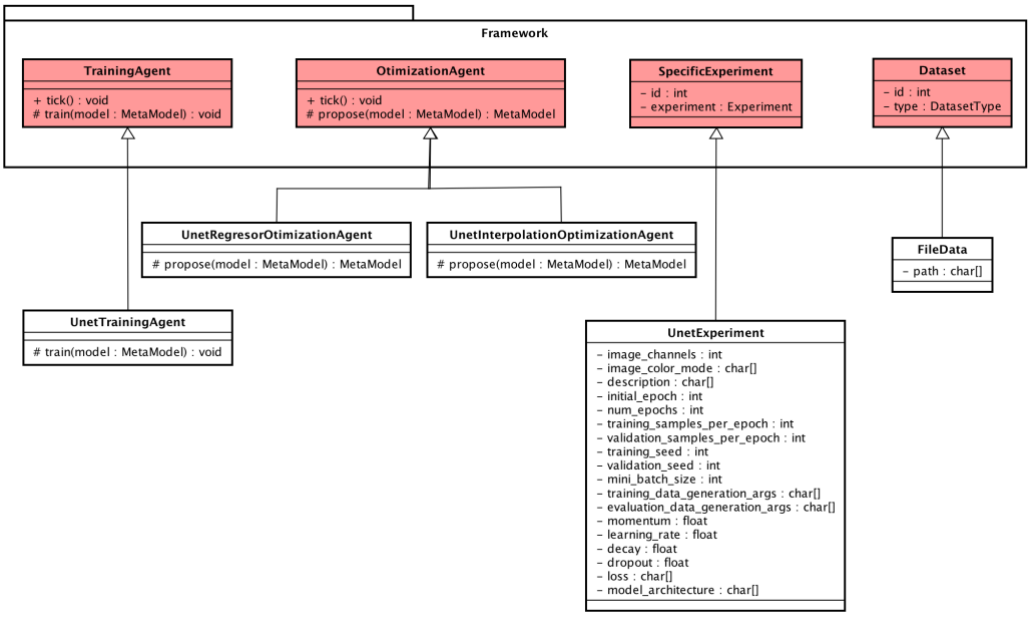

Figure 15 Lung dataset framework instance

In Figure 15 it is possible to observe that the TrainingAgent was extended in the UnetTrainingAgent and inserted in the train method, the specificities about how to train a UNet neural network. To optimize in this instance, we use two different combined approaches. The UnetRegresorOptimizationAgent extended the OptimizationAgent and uses a linear regression approach to calibrate the hyperparameters while the UnetInterpolationOptimizationAgent uses an

interpolation approach to estimate the hyperparameters space of solutions and propose new models. The optimizers were trying to improve the F1 score. The class UnetExperiment extends the SpecificExperiment and adds the parameters needed to train a UNet neural network and there was used the FileData class to reference the dataset.

The starting point were two instances of the UnetExperiment with the following parameters:

Table 5 Initial experiment setup for the Lung Images Experiment

Parameter	Value	Value
Mini Batch Size	3	3
Initial Epoch	0	0
Num Epoch	50	40
Training Samples per Epoch	999	999
Validation Samples per Epoch	888	888
Momentum	0.5	0.5
Learning Rate	0.001	0.01
Decay	0. 000001	0. 000001
Dropout	0.1	0

Figure 16 shows the 56 models proposed sorted by the F1 metric in the rows, the columns val_accuracy, val_fmeasure, val_precision and val_recall represent the metrics recorded during the experiment; and the remaining columns represent parameters of the models. It is possible to see one of the starting points in the 23rd position and the other is in the 45th highlighted in blue. The metric values in red are values close to 0 while the values in green are close to 1.0.

	val_accuracy	val_fmeasure	val_precision	val_recall	num_epochs	decay	learning_rate	dropout	momentum
1	0,336421	0,470238	0,310987	0,997904	232	2,09844E-08	6,04948E-06	0,00397761	0,0948092
2	0,310753	0,464535	0,302866	0,996421	250	1,39935E-08	5,15915E-06	0,00406702	0,422924
3	0,302067	0,459059	0,299083	0,98699	250	1,29517E-08	5,47351E-06	0,00406501	0,430834
4	0,299748	0,457587	0,29807	0,984415	250	2,75497E-08	2,67876E-09	0,00454871	0,432794
5	0,29398	0,450066	0,293667	0,962859	223	1,84011E-08	4,8498E-06	0,00340566	0,540395
6	0,324724	0,44542	0,291542	0,972783	174	2,09593E-08	5,11929E-06	0,00173961	0,657402
7	0,317199	0,445418	0,296037	0,932566	206	5,01011E-09	1,36519E-05	0,00117277	0,405511
8	0,315952	0,439285	0,284858	0,990548	155	1,77726E-08	5,07078E-06	0,00245645	0,633844
9	0,0000001	0,435249	0,281498	1	184	1,89623E-08	2,34956E-06	0,00328823	104.005
10	0,314796	0,432701	0,2814	0,982659	222	1,51387E-08	3,16534E-06	0,00460285	0,271263
11	0,0000001	0,429698	0,276858	1	162	9,48388E-09	1,8103E-07	0,0041695	112.931
12	0,0000001	0,415804	0,265269	1	161	1,24313E-08	1,31409E-07	0,00172713	11.194
13	0,30884	0,403596	0,269983	0,846504	250	1,65594E-08	3,5165E-06	0,00218642	0,67938
14	0,344669	0,383138	0,260207	0,758731	173	1,39212E-08	2,60965E-07	0,00969793	0,725092
15	0,289217	0,37367	0,247864	0,799936	117	1,30748E-08	4,18039E-06	0,00346214	0,546722
16	0,29455	0,366875	0,247904	0,750684	217	1,4529E-08	5,20698E-06	0,00244398	0,608755
17	0,254054	0,36183	0,235457	0,794284	139	1,23421E-08	6,41803E-06	0,00456226	0,425752
18	0,299191	0,359203	0,241701	0,738518	165	2,09277E-08	4,64582E-06	0,00521571	0,0396652
19	0,27743	0,356372	0,237207	0,756545	146	2,01515E-08	2,14319E-05	0,0245061	0,224331
20	0,681072	0,346476	0,398193	0,314943	250	1,13054E-08	3,49222E-06	0,00334274	0,355667
21	0,275882	0,345433	0,231033	0,72216	139	1,33027E-08	4,28777E-06	0,00318443	0,450637
22	0,290242	0,343342	0,231171	0,702187	250	6,65847E-09	4,89191E-06	0,00230896	0,246468
23	0,709026	0,319306	0,39404	0,297279	50	0,000001	0,001	0,1	0,5
24	0,685818	0,31859	0,414203	0,26608	182	2,14085E-08	2,17135E-07	0,00289675	0,541297
25	0,302166	0,316146	0,216144	0,618184	153	2,07504E-08	2,13347E-05	0,0238666	0,203407
26	0,285021	0,312728	0,211474	0,633738	250	1,87521E-08	4,14061E-06	7,53635E-05	0,626351
27	0,737796	0,289797	0,547478	0,198788	198	1,55818E-08	4,45049E-06	0,0027846	0,471932
28	0,641541	0,285924	0,31494	0,270109	174	6,78437E-09	4,48295E-06	0,00242655	0,770533
29	0,717862	0,271653	0,497187	0,18871	20	2,69059E-08	4,44464E-06	0,00233161	0,345036
30	0,714786	0,266418	0,446044	0,192329	249	2,18004E-08	7,04063E-06	0,000925675	0,003591
31	0,682067	0,24194	0,326668	0,194673	250	2,42365E-08	4,64823E-06	0,00175201	0,00262914
32	0,277403	0,241516	0,167294	0,456443	117	9,62236E-09	3,61274E-06	0,00295407	0,910985
33	0,269449	0,239976	0,168053	0,446954	230	1,08338E-07	0,000090917	0,000772714	0,153836
34	0,294917	0,213192	0,15343	0,365338	173	1,63869E-08	5,37635E-06	0,000802095	0,931575
35	0,686002	0,181724	0,295657	0,134188	1	1,55127E-08	5,01281E-06	0,00390181	0,448278
36	0,720845	0,143934	0,426959	0,0873235	111	1,35698E-08	4,54871E-06	0,00296605	0,531987
37	0,738052	0,125586	0,500189	0,0724743	221	2,45612E-08	4,68387E-06	0,00178797	0,298232
38	0,733961	0,0845086	0,598592	0,0455753	202	2,07528E-08	2,11996E-05	0,0250876	0,170646
39	0,730624	0,0735213	0,459133	0,0399685	103	2,12964E-08	5,83547E-06	0,00400471	0,377702
40	0,731543	0,0711478	0,613012	0,0383448	250	1,44559E-08	5,02255E-06	0,00324809	0,32482
41	0,69763	0,067315	0,451903	0,0363661	213	3,35112E-08	6,01885E-06	0,00167431	0,240308
42	0,726059	0,065019	0,570335	0,0345741	132	1,6277E-08	4,42231E-06	0,00333361	0,905116
43	0,693498	0,0542346	0,365705	0,0292891	250	3,04884E-08	4,61143E-06	0,0016765	0,334477
44	0,699895	0,0518904	0,498254	0,0273705	210	2,00833E-08	5,98905E-06	0,00348655	0,392436
45	0,72837	0,0429007	0,529015	0,0224083	40	0,000001	0,0001	0,1	0,5
46	0,725001	0,0192492	0,349655	0,00993579	206	6,29999E-09	3,07329E-06	2,86517E-05	0,929415
47	0,737471	0,0128294	0,494523	0,00652551	250	9,66686E-09	6,00649E-07	0,00323646	0,993148
48	0,718714	0,0106524	0,372165	0,00544008	176	3,99058E-10	5,04627E-06	0,00608299	0,33828
49	0,719571	0,00909779	0,568204	0,00459062	202	1,32695E-08	3,29094E-06	0,000242642	0,974674
50	0,719875	0,00749227	0,678072	0,00376916	212	2,72495E-08	2,7519E-07	0,00448616	0,842033
51	0,727957	0,00471847	0,41836	0,00237765	174	2,01366E-08	4,78518E-06	0,00242823	0,323985
52	0,729244	0,00427089	0,690729	0,00214304	71	5,86473E-07	8,63587E-05	0,0596232	0,0717155
53	0,729189	0,00365795	0,682835	0,00183444	55	7,09232E-07	7,96593E-05	0,056044	0,0735951
54	0,732838	0,0021463	0,51442	0,00107541	193	2,65048E-08	4,03666E-06	0,0057874	0,22253
55	0,732766	0,00183752	0,230627	0,000922434	184	3,79E-08	4,3529E-06	0,00279335	0,0688594
56	0,737307	0,000764093	0,466897	0,000382587	170	1,29917E-08	4,41224E-06	0,00310689	0,457972

Figure 16 Models proposed by the software agents for the Lung Image Experiment

It is also possible to see in Figure 16 that the models 9, 11, and 12 did not learn because the recall is 1.0 and the accuracy is 0.0000001. This means that those models always predict the same class.

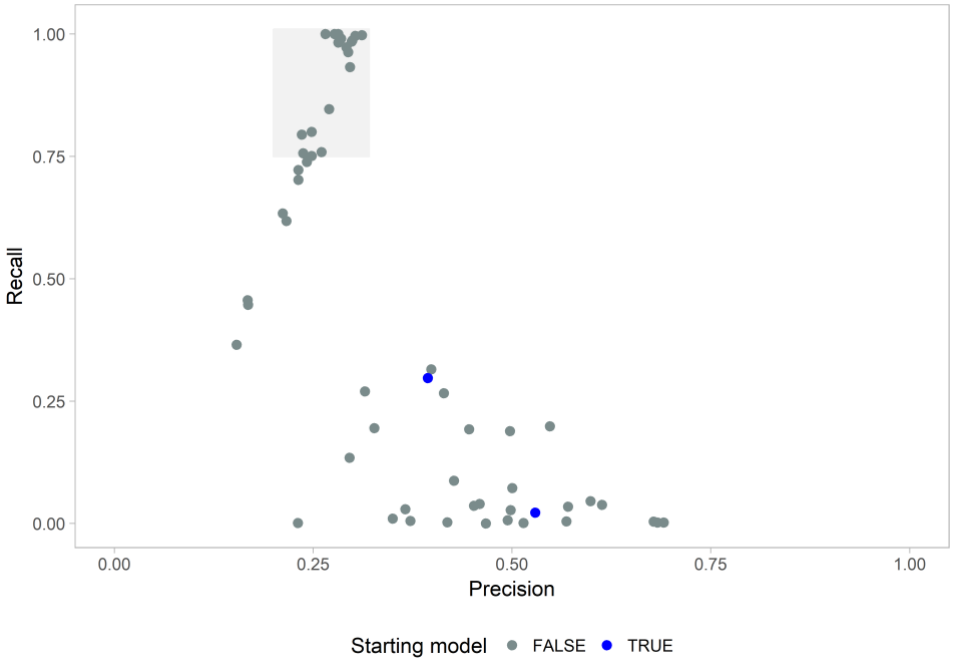

Figure 17 Models trained by the software agents in terms of precision and recall

Figure 17 shows each model trained in the precision versus recall space. The values used to plot the models were the ones obtained during the validation stage. In blue, we represent the starting points; and in gray, we represent the solutions generated by the software agents. It is possible to see two major goups, the first is trying to increase the precision while the second is focused on improving the recall. In this experiment we want to be sure that images without lungs are safely eliminated. So, the models could be used to prune the output of the exams and only show the images that have lung tissue. Following this idea, we prefer a model located in the group improving the recall. If we tried in this example to improve the precision, we might miss some images containing lung tissue. The gray area is zoomed in Figure 18.

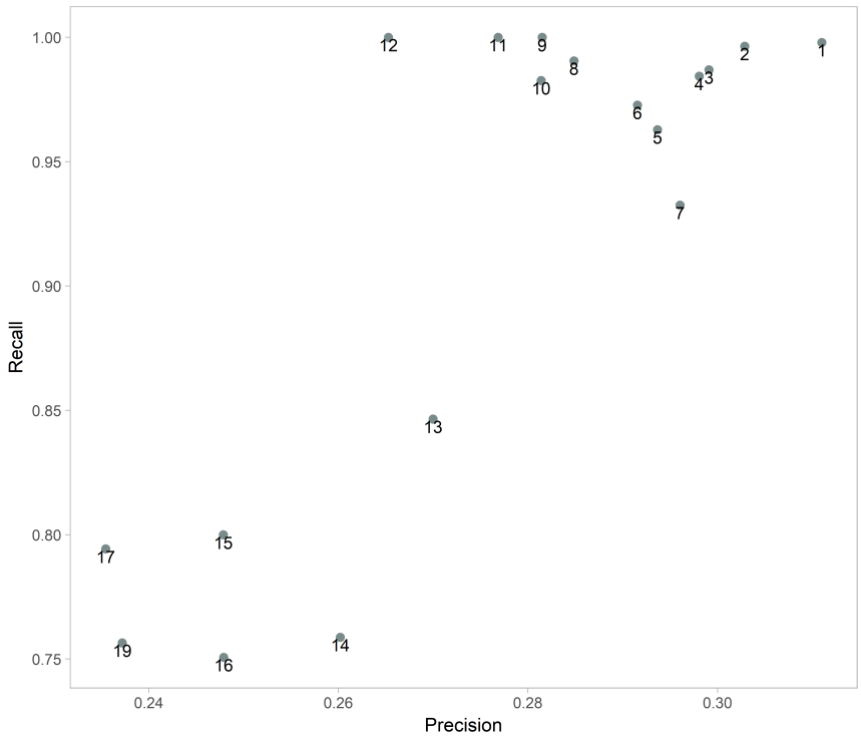

Figure 18 Models with recall greater than 0.75

Figure 18 presents a zoom of the models with recall greater than 0.75. The numbers in each model represent the first column of the models presented in Figure 16. Removing the 9, 11, and 12; due to the fact that the models did not learn, we could use solutions between 1 and 10 as final solutions to filter the exams removing images that do not contain lung tissue. This experiment answers our RQ3 because the models proposed by the software agents were more accurate over time.

5.3 Grid Sector Experiment

This experiment was intended to be different form the above experiemets. The Iris Experiment and the Lung Images Experiment were designed in such manner that the software agents must find solutions from misleading start points, in this experiment the software agents will have a good starting point and the goal is whether the agents will be able to improve the metrics or not.

5.3.1 Grid Sector Dataset

In this experiment, we use the GridSector Dataset. This dataset was proposed in (ALMEIDA, Cassio F. P, 2017) and it contains 252 images, 126 images were obtained from Google Earth (江宽, 2008)(江宽, 2008)(江宽, 2008)(江宽, 2008) and 126 were annotated during the construction process. The objective of the task is to identify the built areas inside the images. So, the

annotated images are bitmaps highlighting the constructed areas. Image sizes vary around 900x900 pixels and the images were collected from 12 different regions of Brazil, taking into account geographical differences in the soil, light and darkness, occlusion, and cloud shadows.

5.3.2 Results

Figure 19 shows how the framework was instantiated for the GridSector experiment. Due to the coincidence of the machine learning model, we were able to reuse most of the classes from the Lung Images Experiment. However, in this experiment we only use the UnetInterpolationOptimizationAgent to calibrate the hyperparameters to improve de F1 score and the FileData class pointed to another dataset of the same type.

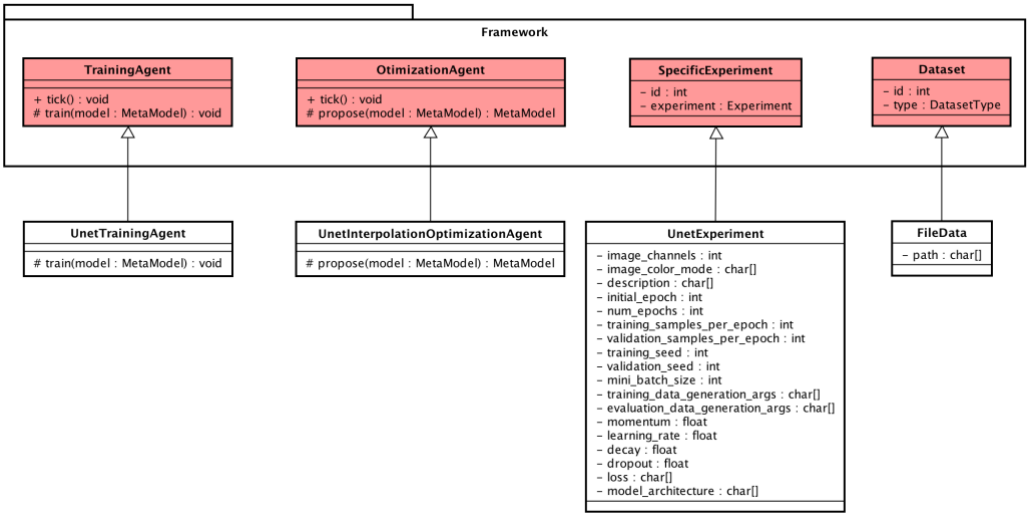

Figure 19 Framework instance for the Grid Sector Experiment

The initial configuration for this experiment can be found in (ALMEIDA, Cassio F. P, 2017) as the better solution found for the problem. The parameters used are shown in Table 6:

Table 6 Initial experiment setup for the Grid Sector Experiment

Parameter	Value
Mini Batch Size	3
Initial Epoch	0
Num Epoch	450
Training Samples per Epoch	447
Validation Samples per Epoch	98
Momentum	0.5
Learning Rate	0.01
Decay	0. 000001
Dropout	0

Using these parameters the accuracy was 0.9581, the precision was 0.9378, the 0.9411, and F1 score was 0.9393. The results obtained from the software agents are shown in Figure 20.

	val_accuracy	val_fmeasure	val_precision	val_recall	num_epochs	decay	learning_rate	dropout	momentum
1	0,9581	0,9393	0,9378	0,9411	450	0.000001	0.01	0	0.5
2	0,7786	0,6814	0,6052	0,8091	450	0.000002	0.02	0	0.2
3	0,7721	0,6882	0,5949	0,8334	500	0.00000280902	0.0281402	0.2175	0.817705
4	0,0000	0,5068	0,3585	1,0000	500	0.00000387342	0.0388766	0.340928	1.29231
5	0,0000	0,5068	0,3585	1,0000	500	0.00000518402	0.0521348	0.522542	1.96392
6	0,0000	0,5068	0,3585	1,0000	500	0.00000677904	0.0683121	0.003569	2.89078
7	0,0000	0,5068	0,3585	1,0000	500	0.00000869263	0.0877661	0.704978	4.13829
8	0,0000	0,5068	0,3585	1,0000	500	0.0000109571	0.110837	0.938426	5.77985
9	0,0000	0,5068	0,3585	1,0000	500	0.0000136039	0.137856	1.21128	7.89732
10	0,0000	0,5068	0,3585	1,0000	500	0.0000166636	0.169149	1.52671	10.5814
11	0,0000	0,5068	0,3585	1,0000	500	0.0000201667	0.205037	1.88784	13.9317
12	0,0000	0,5068	0,3585	1,0000	500	0.0000241431	0.24584	2.29777	18.0574
13	0,0000	0,5068	0,3585	1,0000	500	0.0000286225	0.291875	2.75955	23.0771
14	0,0000	0,5068	0,3585	1,0000	500	0.0000336345	0.343457	3.27624	29.1192
15	0,0000	0,5068	0,3585	1,0000	500	0.0000392084	0.4009	3.85086	36.3221
16	0,0000	0,5068	0,3585	1,0000	500	0.0000453735	0.464517	4.48641	44.8343
17	0,0000	0,5068	0,3585	1,0000	500	0.0000521587	0.53462	5.1859	54.8149
18	0,0000	0,5068	0,3585	1,0000	500	0.000059593	0.61152	5.95231	66.4333

Figure 20 Results obtained from the software agents from the Grid Sector Experiment

In Figure 20 the two starting points set up manually can be seen highlighted in blue. The columns val_accuracy, val_fmeasure, val_precision and val_recall represent the metrics recorded during the experiment; and the remaining columns represent paramenters of the models. The metric values in red are values close to 0 while the values in green are close to 1.0. The models listed from 3 to 18 were generated by the UnetInterpolationOptimizationAgent. It is possible to see that the models generated by the agents were not able to improve the starting point metrics by appling interpolation techniques on the hyperparameters domain proving that the starting solution was a good one for the problem.

6 Conclusion and Future work

This chapter proposes a MAS to set up a battery of experiments and tools to help in the training and prediction processes. We conclude that it is possible to take advantage of the characteristics of the software agents to train machine learning models, and also to make decisions about new models that might have good accuracy. The API presented in this work is a tool to demonstrate that a multiagent learning approach is reasonable and decreases the models' training time. The multiagent system inside the proposed solution is the core of the application because it requires autonomy to make decisions, proactivity to create new experiments, and reactivity to deal with overfitting and low accuracy. By automating this process, the users only need to set up the initial battery of experiments, which reduces the time dedicated to train a successful model.

Three experiments in three different datasets were carried out. In each one it was possible to instantiate the proposed framework and the instances were able to generate machine learning models for the different problems. Each new model was intended to enhance the knowledge about the hyperparameter space of solutions in order to generate more accurate models. In the case of the Lung Experiment, a dataset of 258 images was created, 129 images from specialized equipment and 129 handmade annotated images highlighting the lung tissue.

For future work, we have three goals:

First, Features Selection: An interesting problem is how to improve the performance of the training by first selecting the most important attributes. This could significantly impact the time spent to train a model. Other possible approaches to improve performance include the use of heuristics such as Principal Features Analysis (PFA) (Lu, Cohen, Zhou, & Tian, 2007) or methods, such as Sequential Forward Selection (SFS) (Doak, 1992) and Sequential Backward Selection (SBS) (Doak, 1992).

Second, Initial Model: It would be very interesting that the framework could propose the starting models for a given problem. To do this, one possible solution could be made by mapping the features and their characteristics to the problem to the algorithms used to generate the models.

Third, Feedback: The instances do not communicate with the user to give a feedback of the results. For instance, the system could send alerts to the user via email, short messages, among others to inform the results of the trained experiments, or when it is obtained a model that improves the metrics of the previous models. This future work will close the relation between the software agents and users.

7 References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … others. (2016). **Tensorflow: Large-scale machine learning on heterogeneous distributed systems**. *arXiv Preprint arXiv:1603.04467*. Retrieved from https://arxiv.org/abs/1603.04467

ALMEIDA, Cassio F. P,. (2017). *Mapeamento da Distribuição Populacional Através da Detecção de Áreas Edificadas em Imagens de Regiões Heterogêneas do Google Earth Usando Deep Learning.* (Tesse). Departamento de Informática, Pontifícia Universidade Católica., Rio de Janeiro.

Alonso, E., D'inverno, M., Kudenko, D., Luck, M., & Noble, J. (2001). **Learning in multi-agent systems**. *The Knowledge Engineering Review*, *16*(3), 277–284.

Association, N. E. M., & others. (2009). *Digital imaging and communications in medicine (DICOM)*. NEMA.

Azevedo, A. I. R. L., & Santos, M. F. (2008). **Kdd, semma crisp-dm: a parallel overviee**. *IADS-DM*. Retrieved from https://recipp.ipp.pt/handle/10400.22/135

Bache, K., & Lichman, M. (2013). **UCI machine learning repository**.

Barnes, J. (2015). **Azure machine learning: Microsoft azure essentials**.

Braun, G. J., & Fairchild, M. D. (1999). **Image lightness rescaling using sigmoidal contrast enhancement functions**. *Journal of Electronic Imaging*, *8*(4), 380–394.

Chen, P. P.-S. (1976). **The Entity-relationship Model—Toward a Unified View of Data**. *ACM Trans. Database Syst.*, *1*(1), 9–36. https://doi.org/10.1145/320434.320440

Cortes, C., & Vapnik, V. (1995). **Support-vector networks**. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Data Never Sleeps 5.0 | Domo. (2017). Retrieved March 25, 2018, from https://www.domo.com/learn/data-never-sleeps-5

Doak, J. (1992). **CSE-92-18 - An Evaluation of Feature Selection Methodsand Their Application to Computer Security**. *UC Davis Dept of Computer Science Tech Reports*. Retrieved from http://escholarship.org/uc/item/2jf918dh

Fawcett, T. (2006). **An introduction to ROC analysis**. *Pattern Recognition Letters*, *27*(8), 861–874.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). **From Data Mining to Knowledge Discovery in Databases**. *AI Magazine*, *17*(3), 37. https://doi.org/10.1609/aimag.v17i3.1230

Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

Garner, S. R., & others. (1995). **Weka: The waikato environment for knowledge analysis**. In *Proceedings of the New Zealand computer science research students conference* (pp. 57–64). Citeseer.

Gentleman, R., Ihaka, R., Bates, D., & others. (1997). **The R project for statistical computing**. *R Home Web Site: Http://Www. R-Project. Org*.

Green, T., & others. (2011). **Prediction API: Every app a smart app**. *Google Developers Blog, Apr*, *21*.

Guyon, I., Bennett, K., Cawley, G., Escalante, H. J., Escalera, S., Ho, T. K., … Viegas, E. (2015). **Design of the 2015 ChaLearn AutoML challenge**. In *2015 International Joint Conference on Neural Networks (IJCNN)* (pp. 1–8). https://doi.org/10.1109/IJCNN.2015.7280767

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). **The WEKA data mining software: an update**. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18.

Khalil, K. M., Abdel-Aziz, M., Nazmy, T. T., & Salem, A.-B. M. (2015). **MLIMAS: A Framework for Machine Learning in Interactive Multi-agent Systems**. *Procedia Computer Science*, *65*, 827–835. https://doi.org/10.1016/j.procs.2015.09.035

Klinkenberg, R. (2013). *RapidMiner: Data mining use cases and business analytics applications*. Chapman and Hall/CRC.

Kraska, T., Talwalkar, A., Duchi, J. C., Griffith, R., Franklin, M. J., & Jordan, M. I. (2013). **MLbase: A Distributed Machine-learning System.** In *CIDR* (Vol. 1, pp. 2–1). Retrieved from http://cidrdb.org/cidr2013/Papers/CIDR13_Paper118.pdf

Kunft, A., Alexandrov, A., Katsifodimos, A., & Markl, V. (2016). **Bridging the gap: towards optimization across linear and relational algebra** (pp. 1–4). ACM Press. https://doi.org/10.1145/2926534.2926540

Lu, Y., Cohen, I., Zhou, X. S., & Tian, Q. (2007). **Feature Selection Using Principal Feature Analysis**. In *Proceedings of the 15th ACM International Conference on Multimedia* (pp. 301–304). New York, NY, USA: ACM. https://doi.org/10.1145/1291233.1291297

Lucena, C., & Nunes, I. (2013). **Contributions to the emergence and consolidation of Agent-oriented Software Engineering**. *Journal of Systems and Software*, *86*(4), 890–904. https://doi.org/10.1016/j.jss.2012.09.016

Luo, G. (2016). **PredicT-ML: a tool for automating machine learning model building with big clinical data**. *Health Information Science and Systems*, *4*(1). https://doi.org/10.1186/s13755-016-0018-1

Markiewicz, M. E., & de Lucena, C. J. (2001). **Object oriented framework development**. *Crossroads*, *7*(4), 3–9.

Nketah, Gabriel Uchechukwu. (2016, June 15). *Comparison of Cloud Machine Learning Services*.

Nouri, H. E., Driss, O. B., & Ghédira, K. (2015). **Hybrid Metaheuristics within a Holonic Multiagent Model for the Flexible Job Shop Problem**. *Procedia Computer Science*, *60*, 83–92. https://doi.org/10.1016/j.procs.2015.08.107

Powers, D. M. (2011). **Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation**.

Provost, F. J., Fawcett, T., & Kohavi, R. (1998). **The case against accuracy estimation for comparing induction algorithms.** In *ICML* (Vol. 98, pp. 445–453).

Ranganathan, P. (2011). *The data explosion*. IEEE Computer Society Press.

Ronneberger, O., Fischer, P., & Brox, T. (2015). **U-net: Convolutional networks for biomedical image segmentation**. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 234–241). Springer. Retrieved from http://link.springer.com/chapter/10.1007/978-3-319-24574-4_28

Scholkopf, B. (2001). **The kernel trick for distances**. *Advances in Neural Information Processing Systems*, 301–307.

Shoham, Y., Powers, R., & Grenager, T. (2007). **If multi-agent learning is the answer, what is the question?** *Artificial Intelligence*, *171*(7), 365–377. https://doi.org/10.1016/j.artint.2006.02.006

Sparks, E. R., Talwalkar, A., Smith, V., Kottalam, J., Pan, X., Gonzalez, J., … Kraska, T. (2013). **MLI: An API for distributed machine learning**. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on* (pp. 1187–1192). IEEE. Retrieved from http://ieeexplore.ieee.org/abstract/document/6729619/

Stone, P. (2007). **Multiagent learning is not the answer. It is the question**. *Artificial Intelligence*, *171*(7), 402–405. https://doi.org/10.1016/j.artint.2006.12.005

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. John Wiley &

Sons.

Wooldridge, M., & Jennings, N. R. (1998). **Pitfalls of Agent-oriented Development**. In *Proceedings of the Second International Conference on Autonomous Agents* (pp. 385–391). New York, NY, USA: ACM. https://doi.org/10.1145/280765.280867

江宽. (2008). *Google API 开发详解: Google Maps 与 Google Earth 双剑合璧*. 电子工业出版社.