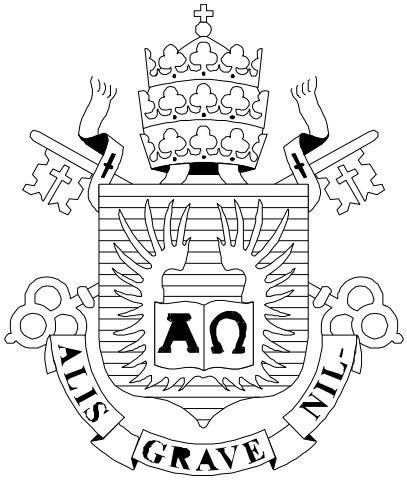

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° MCC09/2017

A Multiagent System to Train Machine Learning Models

Jefry Sastre Pérez
Marx Leles Viana
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. MCC09/2017 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena June, 2017

A Multiagent System to Train Machine Learning Models

Jefry Sastre Pérez, Marx Leles Viana, Carlos José Pereira de Lucena

{jperez, mleles, lucena}@inf.puc-rio.br

Abstract. This research presents a multiagent system to be used for training machine learning models; more precisely, to automate the training process. It also presents tools to combine, compare and propose new models that might have a good performance, using the results from previous experiments. This approach integrates concepts from Multiagent Systems (MASs), Machine Learning (ML) and more specific supervised learning techniques. As proof of concept, we first present the training model in the IRIS dataset. Then, we show how our approach allows the training of different models by using software agents. Finally, we discuss how the system enhances the training process, using the finished experiments as examples and demonstrate that the more the experiments are executed, the more accurate the proposed model becomes. Last but not least, for future work, we aim at: (i) using more accurate optimization techniques and (ii) extending the system to analyze the features of the dataset by using data science algorithms.

Keywords: Multiagent Systems; Machine Learning; Supervised Learning; Self‐ train.

Resumo. Esta pesquisa apresenta um Sistema Multiagente a ser utilizado para o treinamento de modelos de aprendizado de máquina. Mais precisamente, para automatizar o processo de treinamento. Também apresenta ferramentas para combinar, comparar e propor novos modelos que possam ter um bom desempenho, usando os treinamentos anteriores. Esta abordagem integra conceitos de Sistemas Multiagentes (MASs), Aprendizado de Máquinas (ML) e técnicas de aprendizagem supervisionadas. Como prova de conceito, apresentamos o proceso de treinamento no conjunto de dados IRIS. Depois, mostramos como nossa abordagem permite o treinamento de diferentes modelos usando agentes de software. Finalmente, discutimos como o sistema melhora o processo de treinamento através dos experimentos, demonstrando que quanto mais vezes executamos os experimentos, mais preciso se torna o modelo proposto. Por último, mas não menos importante, para trabalhos futuros, pretendemos: (i) usar técnicas de otimização mais precisas e (ii) estender o sistema para analisar as características do conjunto de dados usando algoritmos de selecçao de features.

Palavras-chave: Sistemas Multiagentes, Aprendizado de Máquinas, Aprendizado Suupervisionado, Auto-entrenamiento.

---

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil (CAPES)

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Table of Contents

1 Introduction 1

2 Background 2

2.1 Multiagent Systems 2

2.2 Multiagent Systems and Machine Learning 2

3 Related Work 3

4 Proposed Solution 4

4.1 The Architecture 4

4.2 Data Model 5

4.3 Mapping the Data 6

4.4 Agents Model 6

4.4.1 Trainer Agent 7

4.4.2 Optimizer Agent 7

4.5 Optimizers 8

4.6 Details of the API 8

5 Application Domain 9

5.1 Overview 9

5.2 The Dataset 10

5.3 Results 10

6 Conclusion and Future work 12

References 12

1 Introduction

Computers have enormous processing capabilities, in addition to fast read and write operations, and the ability to manage resources efficiently. Together, the hardware and the software capabilities turn a computer into a very powerful piece of equipment, capable of solving problems that lie beyond human limits. The evolution in mathematics and computer science can be seen as a pyramid, where new strategies are based on existing strategies (Nonaka & Toyama, 2003). This means that it is possible to create a chain of dependencies from the simplest to the most complex ideas. Currently, the existing algorithms in supervised learning are very powerful and can be used to solve a problem accurately, but everything comes with a price — in this case, time (Lim, Loh, & Shih, n.d.).

Machine Learning is a field of computer science surrounded by mystery and uncertainties due to the fact that machines do not really "learn," according to its original definition: "artificial generation of knowledge from experience" (Holzinger, 2016). Supervised Learning is a technique of machine learning (Kotsiantis, Zaharakis, & Pintelas, 2007). Generally, to solve a problem, it is common to follow a certain strategy that results in a solution; but in the case of supervised learning, we know the correct answer beforehand — it is about creating a strategy to infer a dataset of solved examples of the problem. There are many strategies that do this (decision trees, support vector machines, neural networks, etc.) and each one has its own process for training a mathematical model that can be evaluated later. The process to find an acceptable model can be extenuating, since each model has several parameters that are directly related to accuracy. In addition, the time required to compute results depends on the size of the dataset to be processed. An example is the huge volume of data available online (Ranganathan, 2011) and the effects of this phenomenon have direct repercussions on both accuracy and computational time.

Another detail in the training process is that the instances in the training dataset are examples and may not represent the reality. Therefore, even when the model correctly predicts these examples, there may have a very different and unexpected behavior with new instances, but that is not necessarily garbage. Sometimes the model is biased by outliers and rare examples — in such cases, the model might not be able to classify new instances accurately. These models can be assembled into a committee of models to take advantage of the overfitting and the outliers detection. In short, the process of training an accurate model, or ensemble, is slow and time-consuming. Furthermore, it is based on the choices that were made during the process and the models selected to be trained. Therefore, we have a serious problem; we spend a lot of time training the models that will be used in our applications.

This work aims at reducing 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. Another contribution is that the system proposes new models that might have a 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.

This paper is organized as follows. Section 2 gives an overview of the concepts used in this research. Section 3 shows the related work. Section 4 presents the proposed solution. Section 5 describes the application domain. Finally, Section 6 shows the conclusion and future work.

2 Background

This section describes the main concepts related to agents and multiagent systems. First, we will discuss multiagent systems, agents and their properties. We will also discuss the relation between multiagent systems and machine learning.

2.1 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), (Lucena & Nunes, 2013). These agents are abstractions with the following properties (Wooldridge, 2009): (i) autonomy — it is the capability of taking their own actions within their environment; (ii) reactivity — it is the capability of response to the changes in the environment, which involves a notion of perception of the environment; (iii) social ability — it is the capability of interaction with other agents and possibly humans, and (iv) proactive ability — it is the capability to take actions towards the agent's goals.

The exploratory study in Section 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 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 take (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).

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 learn-

ing algorithms are combined to solve a problem from another domain. Nevertheless, our approach is a multiagent system applied to a machine learning domain.

3 Related Work

Many authors (Garner & others, 1995; Kraska et al., 2013; Kunft, Alexandrov, Katsifodimos, & Markl, 2016; Luo, 2016; Sparks et al., 2013) approach the idea of creating 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's 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.

Within the context of non-experts, (Abadi et al., 2016) presents a simulation tool that aims at creating an initial intuition 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 Google Prediction (Green & others, 2011) and Azure Machine Learning (Barnes, 2015). Both are on line services and provide support to the data mining process by means of an intuitive interface and a huge collection of ready to use algorithms. However, Azure is not free and Google Prediction's dataset size is limited to 250 megabytes.

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. By combining all these tools together, WEKA has proved essential to the analysis process and as an introductory tool for educational environments. However, all these algorithms are presented as black boxes and do not focus on distributed ways to improve the data mining processes.

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. ML Base (Kraska et al., 2013) is another solution that provides a Domain Specific Language (DSL) with high level abstractions to simplify the process. It creates very elaborated plans — logical and physical — that come with several optimizations to gain performance and accuracy. 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.

The work (Kunft et al., 2016) presents LARA, a DSL to reduce problems created when the pre-processing and the algebra are done by using different programming paradigms. It includes optimizations that are normally loose in the mismatch of the paradigms. In addition, LARA compiles to an intermediate representation to enable optimizations and finally compiles to different languages. On the other hand, it is embedded into Scala and it is focused on coding. 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.

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.

4 Proposed Solution

This section describes the main elements required to understand the solution proposed in this paper. 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 Architecture

The application is implemented using the software agents, as illustrated in Figure. 1. 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.

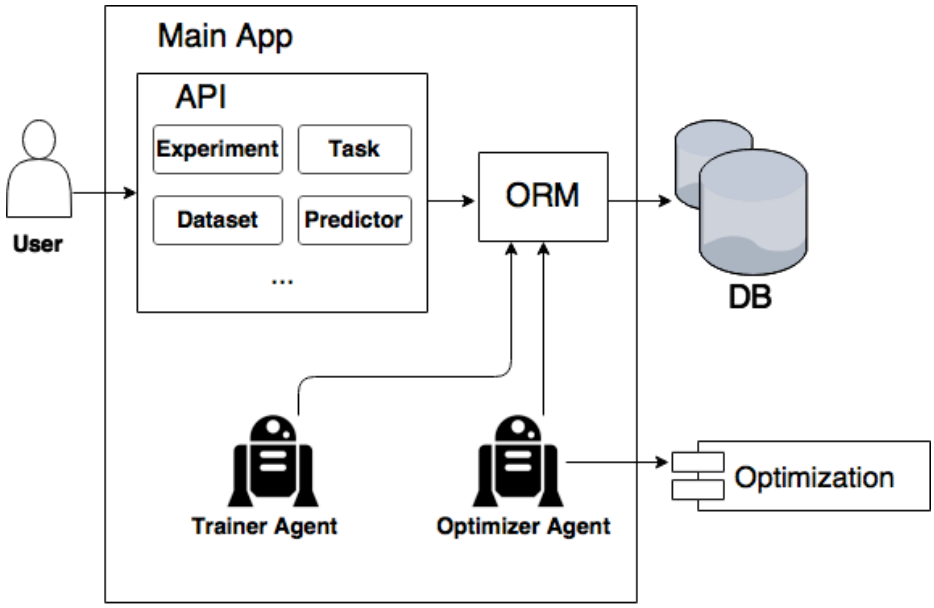

Figure. 1. The architecture proposed.

The users interact with the system via the API and it 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 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 us-

ers (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. 2 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.

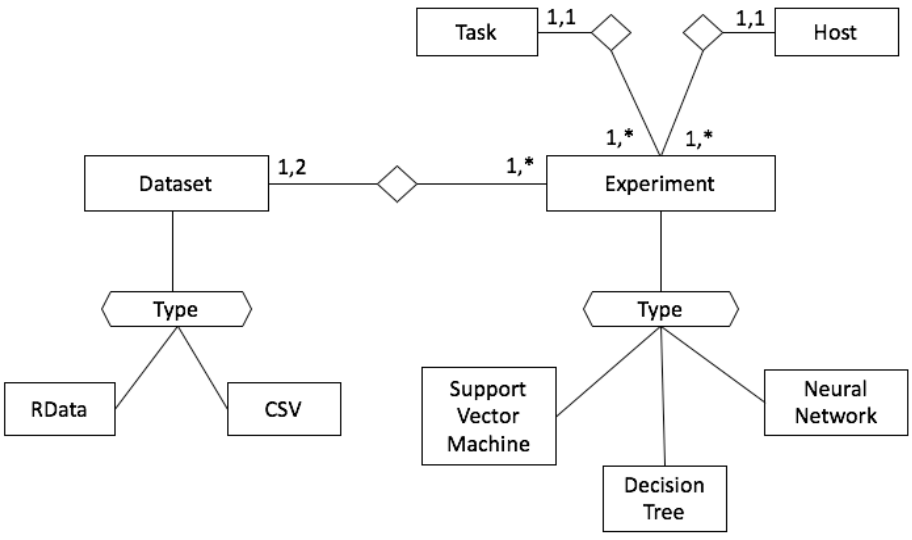

Figure. 2. ERM Model.

- 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 Figure 1. These agents are able to execute the experiments stored in the data model.

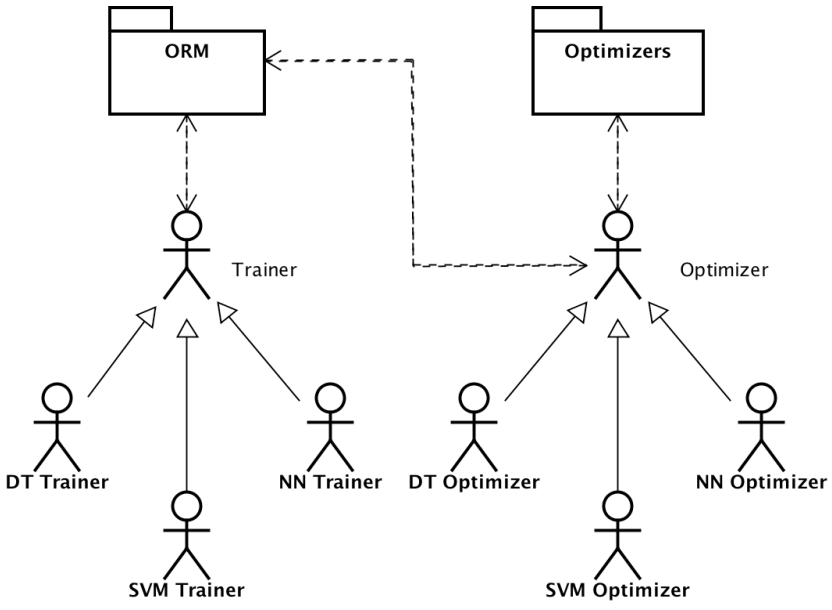

Figure. 3. Agents Model.

Figure. 3 shows and details 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 Section 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 ("Design Patterns by Gamma," n.d.) 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. 4 describes this process.

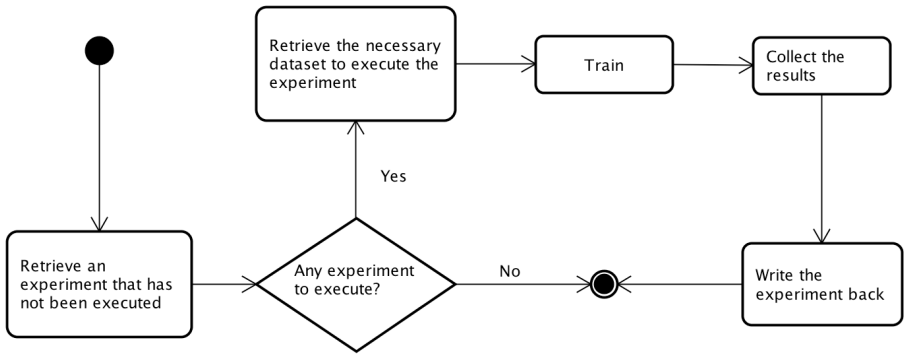

Figure. 4. TrainerAgent Activities Diagram.

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 might 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. 5 describes the workflow of the OptimizerAgent.

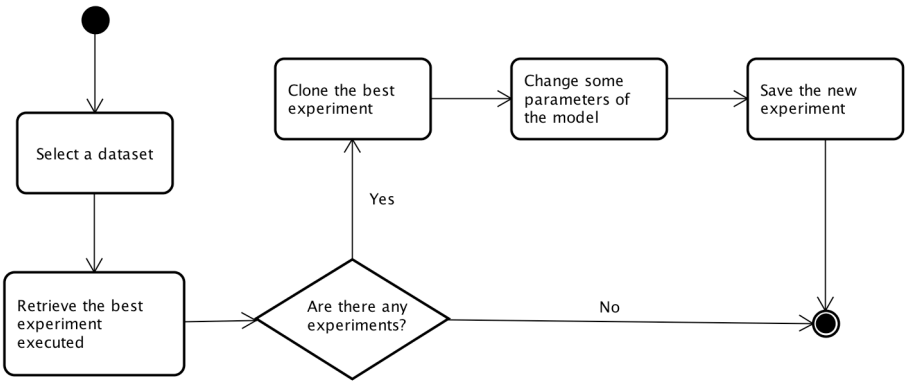

Figure. 5. Optimizer Agent Activities Diagram.

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., that 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.

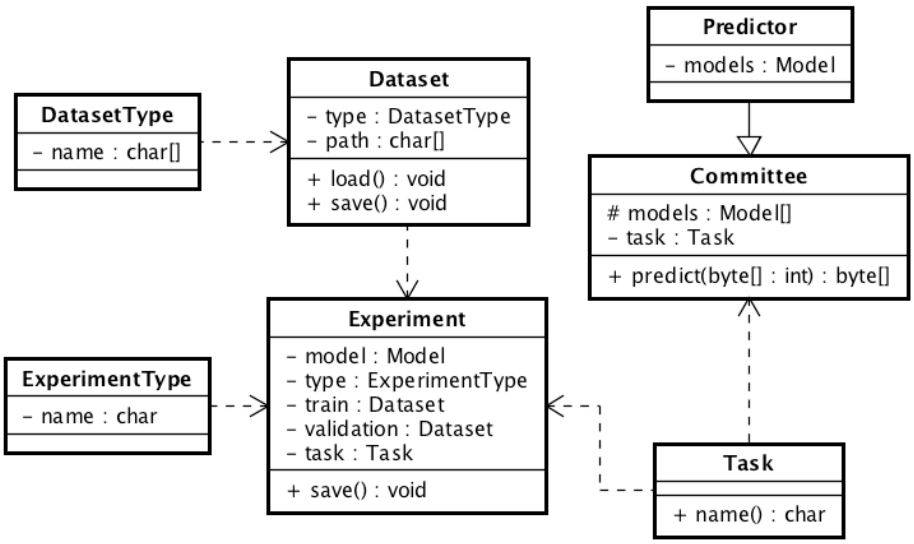

Figure. 6. API Class Diagram.

Figure. 6 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, 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 Application Domain

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 model 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.

5.1 Overview

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 exper-

iment. 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.

5.2 The 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.

IRIS	Sepal Length	Sepal Width	Petal Length	Petal Width
Min	4.3	2	1	0.1
Median	5.8	3	4.35	1.3
Mean	5.843	3.057	3.758	1.199

Table 1. Summary of the IRIS Dataset

5.3 Results

It is possible to choose the desired start configurations and create new models at any point during the training process. In particular, we choose a support vector machine (Cortes & Vapnik, 1995) with the following parameters, as shown in Table 2:

Parameter	Value
Kernel	Poly-nomial
C	1.0
Degree	1.0
Coef0	0
Gamma	auto
Probability	False
Shrinking	1
Max Iterations	- 1

Decision Function	odr

Table 2. Initial experiment setup

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.

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

Table 3. Initial experiment setup

Id	Output	Started	Ended	Time	Validation Accuracy
1	./out/example1.svm	2017-02-27 19:09:43	2017-02-27 19:09:43	0.006163	0.96
2	./out/example2.svm	2017-02-27 19:09:53	2017-02-27 19:09:53	0.004834	0.96
3	./out/example3.svm	2017-02-27 19:10:03	2017-02-27 19:10:03	0.005739	0.97

Table 4. Initial experiment setup

The first row in table 3 shows the beginning of the second stage where 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.

6 Conclusion and Future work

This paper 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 paper 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.

For future work, we have two goals. First, Minimization Optimizers: The proposition of new models based on the previously executed experiments is a challenging task, but a highly profitable tool. However, even if naive approaches may have good, or at least not the worst, results it would be interesting to use an optimization technique to solve the problem – we could ask what the parameters of the new proposed model are, for example. One possible answer to this question is to interpolate the function that receives all the parameters and returns the accuracy, and then, find the maximum value of the interpolated function. Second, Features Selection: Another 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).

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

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.

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

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

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

Design Patterns by Gamma: Pearson India 9789332555402 Paperback - A - Z Books. (n.d.). Retrieved April 12, 2017, from https://www.abebooks.com/Design-Patterns-Gamma-Pearson-India/17320714110/bd

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

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.

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.

Holzinger, A. (2016). Interactive machine learning for health informatics: when do we need the human-in-the-loop? Brain Informatics, 3(2), 119–131.

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

Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Retrieved from https://books.google.com/books?hl=en&lr=&id=vLiTXDHr_sYC&oi=fnd&pg=PA3&dq=Kotsiantis,+Sotiris+B.%3B+Zaharakis,+I.%3B+Pintelas,+P.+Supervised+machine+learning:+A+review+of+classification+techniques.+2007&ots=CXtstyYGfr&sig=N7Gb9al_ztidWdyRt-qG55pDbuw

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

Lim, T.-S., Loh, W.-Y., & Shih, Y.-S. (n.d.). A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms. Machine Learning, 40(3), 203–228. https://doi.org/10.1023/A:1007608224229

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

Nonaka, I., & Toyama, R. (2003). The knowledge-creating theory revisited: knowledge creation as a synthesizing process. Knowledge Management Research & Practice, 1(1), 2–10. https://doi.org/10.1057/palgrave.kmrp.8500001

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

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

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