An Agent-based Software Framework for Machine Learning Tuning

Jefry Sastre[1], Marx Viana[1], Carlos Lucena[1]
[1]Laboratory of Software Engineering (LES) - Pontifical Catholic University - PUC-Rio
Rio de Janeiro, RJ - Brazil
{jperez, mleles, lucena}@inf.puc-rio.br

Abstract— 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.

Keywords. Agent oriented Software Engineering (AOSE); Machine Learning; Time-consuming

I. INTRODUCTION

The big data era is coming! According to [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 [2] and we need to be prepared to face all the challenges upfront. Peter Norving 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 volume of data available is a massive challenge to be accepted, but at the same time a vast opportunity to learn from the data to generate more expert artificial intelligence software and to enhance the knowledge discovery processes (KDD). One of the most popular and widely used approaches to generate knowledge is through the machine learning techniques. In order to be better rewarded 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 extenuated, endless process. We foresee a chance to incorporate the Agent Oriented Software Engineering (AOSE) [23] 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 profitable models.

DOI reference number: 10.18293/SEKE2018-074

Agents are software components with autonomy, reactiveness, proactiveness and social capabilities [3]. Agent autonomy comes in handy when a system needs to make its own decisions. The agents can 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 the machine learning models.

The contributions are: (i) the framework will facilitate building new models that might display good performance based on the previously trained models. This includes a new set of possibilities in the selection of the ways and strategies that will guide the optimizations; (ii) the framework allows the creation of an ensemble 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 rather on a set of models that might be specialized at detecting specific characteristics; (iii) the framework reduces 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 framework 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, and (iv) to validate a case scenario, IRIS. This test case is an exploratory study taken as a proof of concept but instantiating the framework and exploiting the agents to generate new models.

This paper is organized as follows. Section 2 gives an overview of the main concepts. Section 3 shows the related work. Section 4 presents the framework. Section 5 describes an exploratory study. Finally, Section 6 offers the conclusion and future work.

II. BACKGROUND

First, we will discuss the relation between multiagent systems and machine learning. After, the KDD process.

A. Multiagent Systems and Machine Learning

A multiagent system can be defined as an environment shared by autonomous entities that live, interact, receive information and can act in the environment [5]. These agents are abstractions with the following properties [6]: (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 shows how agents' properties are useful in the simulation of the training process to optimize the parameters of a model based on the previously trained models. It also evidences how the software agents are able to propose new models that might be more accurate. The idea of joining together 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 [7]. 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 multiagent systems. First, the coordination of agents — there must be some coordination mechanism for agents to engage and interact in some way. 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 [8]. Some approaches use a machine learning model in the agents' activities cycle to take actions [5]. Other approaches use a multiagent system — known as multiagent learning (MAL) — to learn [9] [10]. 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.

B. KDD Methodologies

The KDD process [11] [12] contains five stages: (i) Selection: This stage is to precisely define a target dataset. It can be done by directly selecting a dataset or a subset of features; (ii) Pre-processing: This stage focuses on cleaning the data. It means that the data most of the times is generated crowded with null values and inconsistencies and needs to be cleaned in order to became profitable; (iii) 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; (iv) Data Mining: This stage focuses on the search of the required patterns in the data according to the mining objectives, and (v) Interpretation/Evaluation: This stage consists of the interpretation and evaluation of the extracted patterns.

III. RELATED WORK

Many authors [13, 14, 15, 16, 17] broach 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, [18] 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 [19] and Azure Machine Learning [20]. 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 [17], [21] 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 [15] 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 [14] 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 [16] many algorithms and large datasets can be slow and limited. The work [13] 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 with different languages. On the other hand, it is embedded into Scala and it is focused on coding. Predict-ML [16] 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 focuses 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 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.

A. Auto ML

Auto ML is a new area in computer science pursuing the progressive automation of the machine learning process [22]. This area addresses all the aspects which are related to machine learning automation, such as search and selection of model, hyper parameters optimization, feature engineering, meta learning and transfer learning, among others. Within this

context, a challenge to boost new solutions towards the Auto ML 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.

IV. 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.

A. The Archicteture

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

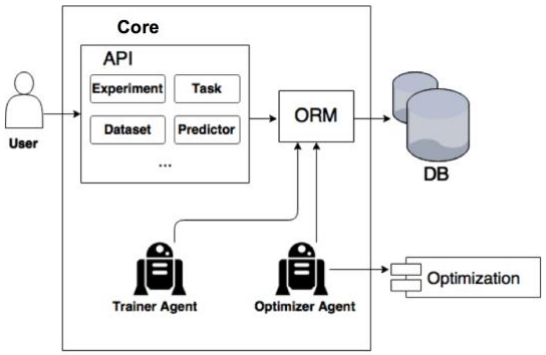

Figure 1. The proposed architecture.

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. ORM 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. 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 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 *Trainer Agent* and the *Optimizer Agent* are both hot spots [23]. 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 and then, analyze, retrieve and compare several of the model's parameters.

B. Data Model

Fig. 2 presents the data model of the concepts involved in the problem. We used the entity-relationship model (ERM) [24]. The entities are: (i) Task: Aims at capturing the training process of a successful model for a machine learning problem, i.e., it is a collection of experiments; (ii) Experiment: Defines an experiment, but this concept just contains the common aspects, such as *running_time*, *train_accuracy*, etc; (iii) 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; (iv) Support Vector Machine: Defines a support vector machine type of experiment and contains attributes such as kernel; (v) Neural Network: Defines a neural network type of experiment and contains attributes such as the model that specifies the structure of the network; (vi) Host: Defines a computer in the network, and basically selects the computer in which the model is going to be trained; (vii) Dataset: Represents a generic data collection, used as the examples to train a model; (viii) RData: Represents a particular type of dataset generated from a script executed in R [25] and contains the environment variables at the save point, and (ix) CSV: Represents a standard data exchange format. Most of the time it is a collection of comma separated fields.

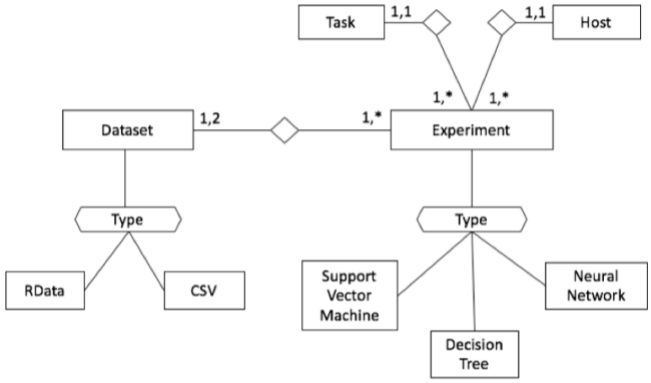

Figure 2. The architecture proposed.

C. Agents Model

Figure 3 shows and details the agent-based model proposed. In all the cases, the agent's cyclic behavior was the best option for these software agents – for instance, in the exploratory study presented in Section 5 the agents have a cyclic behavior with 10 seconds between iterations. The *Trainer Agent* 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 [26] 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. Fig. 4 describes this process.

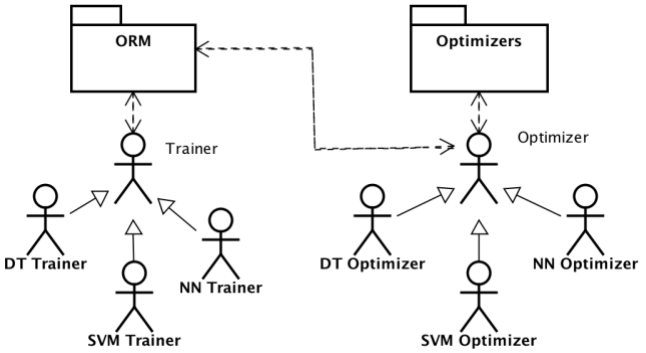

Figure 3. Agents Model.

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.

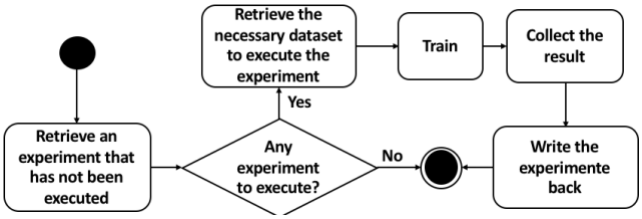

Figure 4. *Trainer Agent* Activity Diagram.

The *Optimizer Agent* is responsible for generating new models that might have good performance and accuracy based on the previously executed experiments of the same type.

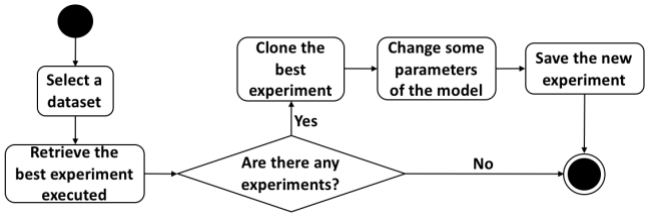

Figure 5. *Optimizer Agent* Activity Diagram.

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 *Optimizer Agent* was extended in order to create specific agents which selected the correct algorithm in each case. Fig. 5 describes the workflow of the *Optimizer Agent*. Observe that the *Optimizer Agent* needs a different strategy to create the new model, depending on the type of the experiment.

D. 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 datasets and calibrate 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: *SVM Optmizer*, *DT Optimizer* and *NN Optimizer*. The SVM Optimizer takes advantage of the kernel trick [27] 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 *DT Optimizer* uses a similar criterion to increase or decrease the *max_depth* of the decision tree while the *NN Optimizer* creates a new model by randomly combining the two best experiments executed.

E. 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.

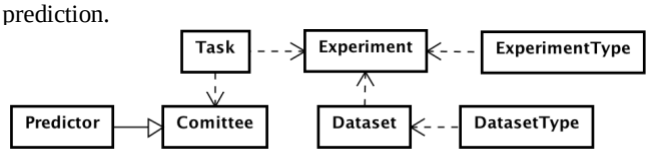

Figure 6. API Class Diagram.

Fig. 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.

V. User Scenario

This section details an exploratory study taken as a proof of concept for the framework. 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 *Optimizer Agent* analyzes the results of the finished experiments and proposes a new experiment using the same dataset.

A. The Dataset

The data used in this example was the IRIS dataset found in the UCI Machine Learning Repository [28]. 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

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
Max	7.9	4.4	6.9	2.5

B. Results

The framework was instantiated as shown in Figure 7. 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 [29]. 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. The starting point is an instance of the SVMExperiment class and we choose the following parameters, as shown in Table 2 (first line).

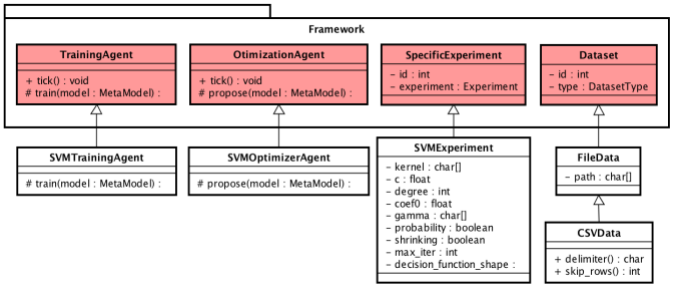

Figure 7 Framework instance for the Iris experiment

Table 2. Parameters of the executed experiments

Id	*Kernel*	*C*	*Degree*	*Coef0*	*Gamma*	*Probability*	*Shrinking*	*Max Iterations*	*Decision Function*
1	poly	1	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 3. Measures of the executed experiments

Id	Started	Ended	Time (in seconds)	Validation Accuracy
1	2017-02-27 19:09:43	2017-02-27 19:09:43	0.006163	0.96
2	2017-02-27 19:09:53	2017-02-27 19:09:53	0.004834	0.96
3	2017-02-27 19:10:03	2017-02-27 19:10:03	0.005739	0.97

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 2 in rows 2 and 3 shows the experiments proposed by the *Optimizer Agent* and Table 3 shows the variables measured. The first row in table 2

shows the beginning of the second stage where only the first model had been proposed. Then, the *Trainer Agent* trained the model, resulting in an accuracy of 0.96 (first row in Table 3). The *Optimizer Agent* performed a query to retrieve the trained models and based on the best one, it modified the allowed error (parameter C in Table 2) from 1.0 to 1.5 and proposed the second model. The *Trainer Agent* realized that there was a model to train and then trained it, resulting in an accuracy of 0.96, as well. Once again, the *Optimizer Agent* modified the degree of the function to propose the third model (parameter degree in Table 2) based on the first and the second models. As a result, the *Trainer Agent* trained the new model and obtained a better accuracy of 0.97.

To obtain new models the *Optimizer Agent* balanced the allowed error and the degree of the polynomial function. It is possible to see in Table 3 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.

VI. Conclusion and Future Work

This paper proposes a software framework based on multiagent systems to automate the process of calibrating machine learning models and reduce the amount of human time dedicated to the parameters adjustment. By means of this framework, the agents will tune the parameters of the models while the data mining experts are focused on providing the framework with potential parameter insights. Together, the agents and the data mining experts can complement each other's' capabilities in a shared software environment. 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 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, Selection of the first model: The framework needs initial models as inputs to begin the calibration processes, but it would be interesting if the system was capable of auto-generate starting models. 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) [30] or methods, such as Sequential Forward Selection (SFS) [31] and Sequential Backward Selection (SBS) [31].

References

[1] J. James, Data never sleeps 2.0. 2014.

[2] P. Ranganathan, The data explosion. IEEE Computer Society Press, 2011.

[3] C. Lucena and I. Nunes, "Contributions to the emergence and consolidation of Agent-oriented Software Engineering," J. Syst. Softw., vol. 86, no. 4, pp. 890–904, Apr. 2013.

[4] M. E. Markiewicz and C. J. de Lucena, "Object oriented framework development," Crossroads, vol. 7, no. 4, pp. 3–9, 2001.

[5] K. M. Khalil, M. Abdel-Aziz, T. T. Nazmy, and A.-B. M. Salem, "MLIMAS: A Framework for Machine Learning in Interactive Multi-agent Systems," Procedia Comput. Sci., vol. 6, Jan. 2015.

[6] M. Wooldridge, An Introduction to MultiAgent Systems. John Wiley & Sons, 2009.

[7] E. Alonso, M. D'inverno, D. Kudenko, M. Luck, and J. Noble, "Learning in multi-agent systems," Knowl. Eng. Rev., vol. 16, no. 3, pp. 277–284, 2001.

[8] H. E. Nouri, O. B. Driss, and K. Ghédira, "Hybrid Metaheuristics within a Holonic Multiagent Model for the Flexible Job Shop Problem," Procedia Comput. Sci., vol. 60, pp. 83–92, Jan. 2015.

[9] Y. Shoham, R. Powers, and T. Grenager, "If multi-agent learning is the answer, what is the question?," Artif. Intell., vol. 171, no. 7, pp. 365–377, May 2007.

[10] P. Stone, "Multiagent learning is not the answer. It is the question," Artif. Intell., vol. 171, no. 7, pp. 402–405, May 2007.

[11] A. I. R. L. Azevedo and M. F. Santos, "Kdd, semma crisp-dm: a parallel overviee," IADS-DM, 2008.

[12] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," AI Mag., vol. 17, Mar. 1996.

[13] A. Kunft, A. Alexandrov, A. Katsifodimos, and V. Markl, "Bridging the gap: towards optimization across linear and relational algebra," 2016, pp. 1–4.

[14] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "MLbase: A Distributed Machine-learning System.," in CIDR, 2013, vol. 1, pp. 2–1.

[15] E. R. Sparks et al., "MLI: An API for distributed machine learning," in Data Mining (ICDM), IEEE 13th International Conference on, 2013..

[16] G. Luo, "PredicT-ML: a tool for automating machine learning model building with big clinical data," Health Inf. Sci. Syst., Dec. 2016.

[17] S. R. Garner and others, "Weka: The waikato environment for knowledge analysis," in Proceedings of the New Zealand computer science research students conference, 1995, pp. 57–64.

[18] M. Abadi et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," ArXiv Prepr., 2016.

[19] T. Green and others, "Prediction API: Every app a smart app," Google Dev. Blog Apr, vol. 21, 2011.

[20] J. Barnes, "Azure machine learning: Microsoft azure essentials," 2015.

[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," ACM SIGKDD Explor. Newsl., vol. 11, no. 1, pp. 10–18, 2009.

[22] I. Guyon et al., "Design of the 2015 ChaLearn AutoML challenge," in International Joint Conference on Neural Networks (IJCNN), 2015.

[23] M. Wooldridge and N. R. Jennings, "Pitfalls of Agent-oriented Development," in Proceedings of the Second International Conference on Autonomous Agents, New York, NY, USA, 1998, pp. 385–391.

[24] P. P.-S. Chen, "The Entity-relationship Model—Toward a Unified View of Data," ACM Trans Database Syst, vol. 1, pp. 9–36, Mar. 1976.

[25] R. Gentleman, R. Ihaka, D. Bates, and others, "The R project for statistical computing," R Home Web Site Httpwww R-Proj. Org, 1997.

[26] "Design Patterns by Gamma: Pearson India 9789332555402 Paperback - A - Z Books." [Online]. Available: https://www.abebooks.com/Design-Patterns-Gamma-Pearson-India/17320714110/bd. [Accessed: 12-Apr-2017].

[27] B. Scholkopf, "The kernel trick for distances," Adv. Neural Inf. Process. Syst., pp. 301–307, 2001.

[28] K. Bache and M. Lichman, "UCI machine learning repository," 2013.

[29] C. Cortes and V. Vapnik, "Support-vector networks," Mach. Learn., vol. 20, no. 3, pp. 273–297, Sep. 1995.

[30] Y. Lu, I. Cohen, X. S. Zhou, and Q. Tian, "Feature Selection Using Principal Feature Analysis," in Proceedings of the 15th ACM International Conference on Multimedia, New York, 2007.

[31] J. Doak, "CSE-92-18 - An Evaluation of Feature Selection Methods and Their Application to Computer Security," UC Davis Dept Comput. Sci. Tech Rep., Jan. 1992.