A Scalable and Automated Machine Learning Framework to Support Risk Management

Luís Ferreira[1,2](✉), André Pilastri[2], Carlos Martins[3], Pedro Santos[3], and Paulo Cortez[2]

[1] EPMQ - IT Engineering Maturity and Quality Lab, CCG ZGDV Institute, Guimarães, Portugal
{luis.ferreira,andre.pilastri}@ccg.pt

[2] ALGORITMI Centre, Department of Information Systems, University of Minho, Guimarães, Portugal
pcortez@dsi.uminho.pt

[3] WeDo Technologies, Braga, Portugal
{carlos.mmartins,pedro.santos}@mobileum.com

Abstract. Due to the growth of data and widespread usage of Machine Learning (ML) by non-experts, automation and scalability are becoming key issues for ML. This paper presents an automated and scalable framework for ML that requires minimum human input. We designed the framework for the domain of telecommunications risk management. This domain often requires non-ML-experts to continuously update supervised learning models that are trained on huge amounts of data. Thus, the framework uses Automated Machine Learning (AutoML), to select and tune the ML models, and distributed ML, to deal with Big Data. The modules included in the framework are task detection (to detect classification or regression), data preprocessing, feature selection, model training, and deployment. In this paper, we focus the experiments on the model training module. We first analyze the capabilities of eight AutoML tools: Auto-Gluon, Auto-Keras, Auto-Sklearn, Auto-Weka, H2O AutoML, Rminer, TPOT, and TransmogrifAI. Then, to select the tool for model training, we performed a benchmark with the only two tools that address a distributed ML (H2O AutoML and TransmogrifAI). The experiments used three real-world datasets from the telecommunications domain (churn, event forecasting, and fraud detection), as provided by an analytics company. The experiments allowed us to measure the computational effort and predictive capability of the AutoML tools. Both tools obtained high-quality results and did not present substantial predictive differences. Nevertheless, H2O AutoML was selected by the analytics company for the model training module, since it was considered a more mature technology that presented a more interesting set of features (e.g., integration with more platforms). After choosing H2O AutoML for the ML training, we selected the technologies for the remaining components of the architecture (e.g., data preprocessing and web interface).

Keywords: Automated machine learning · Distributed machine learning · Supervised learning · Risk management

A. P. Rocha et al. (Eds.): ICAART 2020, LNAI 12613, pp. 291–307, 2021.
https://doi.org/10.1007/978-3-030-71158-0_14

1 Introduction

Nowadays, Machine Learning applications can make use of a great amount of data, complex algorithms, and machines with great processing power to produce effective predictions and forecasts [11]. Currently, two of the most important features of real-world ML applications are distributed learning and AutoML. Distributed learning is particularly useful for ML applications in the context of Big Data or when there are hardware constraints. Distributed learning consists of using multiple machines or processors to process parts of the ML algorithm or parts of the data. The fact that it is possible to add new processing units enables ML applications to surpass time and memory restrictions [29]. AutoML intends to allow people that are not experts in ML to efficiently choose and apply ML algorithms. AutoML is particularly relevant since there is a growing number of non-specialists working with ML [31]. It is also important for real-world applications that require constant updates to ML models.

In this paper, we propose a technological architecture that addresses these two ML challenges. The architecture was adapted to the area of telecommunications risk management, which is a domain that mostly uses supervised learning algorithms (e.g., for churn prediction). Moreover, the ML models are constantly updated by people that are not experts in ML and may involve Big Data. Thus, the proposed architecture delineates a set of steps to automate the typical workflow of a ML application that uses supervised learning. The architecture includes modules for task detection, data preprocessing, feature selection, model training, and deployment.

The focus of this work is the model training module of the architecture, which was designed to use a distributed AutoML tool. In order to select the ML tool for this module, we initially evaluated the characteristics of eight open-source AutoML tools (Auto-Gluon, Auto-Keras, Auto-Sklearn, Auto-Weka, H2O AutoML, Rminer, TPOT, and TransmogrifAI). We then performed a benchmark to compare the two tools that allowed a distributed execution (H2O AutoML and TransmogrifAI). The experiments used three real-world datasets from the domain of telecommunications. These datasets were related to churn (regression), event forecasting (time series), and fraud detection (binary classification).

This paper consists of an extended version of our previous work [14]. The main novelty of this extended version is the technological architecture that is presented in Sect. 6. This section describes the particular technologies that were used to implement the components of the proposed AutoML distributed framework apart from model training. Also, this section describes the REST API that was developed to mediate the communication between the end-users and the proposed framework.

The paper is organized as follows. Section 2 presents the related work. In Sect. 3, we detail the proposed ML architecture. Next, Sect. 4 describes the analyzed AutoML technologies and the datasets used during the experimental tests. Then, Sect. 5 discusses the experimental results. Section 6 details the technological architecture. Finally, Sect. 7 presents the main conclusions and future work directions.

2 Related Work

In a Big Data context, it is critical to create and use scalable ML algorithms to face the common constraints of memory and time [29]. To face that concern, classical distributed ML distributes the work among different processors, each performing part of the algorithm. Another current ML problem concerns the choice of ML algorithms and hyperparameters for a given task. For ML experts, this selection of algorithms and hyperparameters may use domain knowledge or heuristics, but it is not an easy task for non-ML-experts. AutoML was developed to combat this relevant issue [22]. The definition of AutoML can be described as the search for the best algorithm and hyperparameters for a given dataset with minimum human input.

In recent years, a large number of AutoML tools was developed, such as AutoGluon [3], Auto-Keras [23], Auto-Sklearn [15], Auto-Weka [24], H2O AutoML [21], Rminer [10], TPOT [27], and TransmogrifAI [30]. Within our knowledge, few studies directly compare AutoML tools. Most studies compare one specific AutoML framework with state-of-the-art ML algorithms [15], do not present experimental tests [12,35], or are related to ML automation challenges [18–20].

Recently, some studies focused on experimental comparisons of AutoML tools. In 2019, [17,32] compare a set of AutoML tools using different datasets and ML tasks. In 2020, a benchmark was conducted using publicly available datasets from OpenML [33], comparing different types of AutoML tools, which were grouped by their capabilities [36]. None of the mentioned comparison studies considered the distributed ML capability for the AutoML tools. Furthermore, none of the studies used datasets from the domain of telecommunications risk management, such as churn prediction or fraud detection.

3 Proposed Architecture

This paper is part of “Intelligent Risk Management for the Digital Age” (IRMDA), a R&D project developed by a leading Portuguese company in the area of software and analytics. The purpose of the project is to develop a ML system to assist the company telecommunications clients. Both scalability and automation are central requirements to the ML system since the company has many clients with diverse amounts of data (large or small) and that are typically non-ML-experts.

The ML technological architecture that is proposed by this work identifies and automates all typical tasks of a common supervised ML application, with minimum human input (only the dataset and the target column). Also, since the architecture was developed to work within a cluster with several processing nodes, the users can handle any size of datasets just by managing the number of cluster nodes. The architecture is illustrated in Fig. 1.

3.1 Phases

The proposed architecture assumes two main phases (Fig. 1): a training phase and a testing phase.

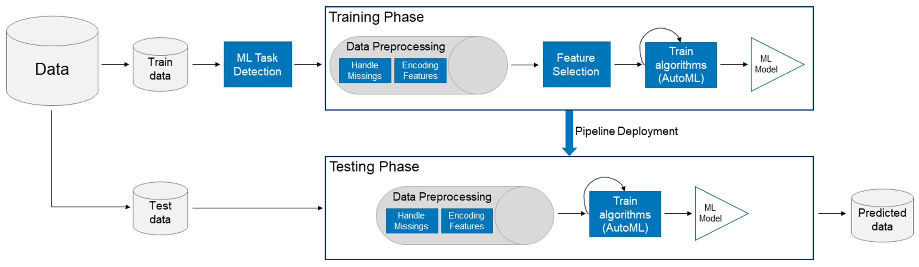

Fig. 1. The proposed automated and scalable ML architecture (adapted from [14]).

Training Phase: The training phase includes the creation of a pipeline instance and the definition of its stages. The only human input needed by the user is the selection of the training dataset and the identification of the target column. Depending on the dataset columns, each module defines a set of stages for the pipeline. Each stage either transforms data or also creates a model based on the training data that will be used on the test phase to transform the data. When all stages are defined, the pipeline is fitted to the training data, creating a pipeline model. Finally, the pipeline model is exported to a file.

Testing Phase: The execution of the testing pipeline assumes the same transformations that were applied to the training data. To execute the testing pipeline the user only needs to specify the test data and a pipeline model (and a forecasting horizon in the case of time series forecasting task). The last stage of the testing pipeline is the application of the best model obtained during training, generating the predictions. Performance metrics are also computed and presented to the user.

3.2 Components

The proposed architecture includes five main components: task detection, data preprocessing, feature selection, model training (with the usage of AutoML), and pipeline deployment.

Machine Learning Task Detection: Set to detect the ML task of the pipeline (e.g., classification, regression, time series). This detection is made by analyzing the number of levels of the target column and the existence (or not) of a time column.

Data Preprocessing: Handles missing data, the encoding of categorical features, and the standardization of numerical features. The applied transformations depend on the data type of the columns, number of levels, and number of missing values.

Feature Selection: Deletes features from the dataset that may decrease the predictive performance of the ML models, using filtering methods. Filtering methods are based on individual correlations between each feature and the target, removing several features that present the lowest correlations [4].

Model Training: Automatically trains and tunes a set of ML models using a set of constraints (e.g., time limit, memory usage). The component also identifies the best model to be used on the test phase.

Pipeline Deployment: Manages the saving and loading of the pipelines to and from files. This module saves the pipeline that will be used on a test set, ensuring that the new data will pass through the same transformations as the training data. Also, the component stores the best model obtained during the training to make predictions, discarding all other ML models.

4 Materials and Methods

4.1 Experimental Evaluation

For the experimental evaluation, we first examined the characteristics of the open-source AutoML tools. Then, we used the tools that could be implemented in our architecture to perform a benchmark study. In order to be considered for the experimental evaluation, the tools have to implement distributed ML.

4.2 AutoML Tools

We first analyzed eight recent open-source AutoML tools, to verify their compliance with the project requirements.

Auto-Gluon: AutoGluon is an open-source AutoML toolkit with a focus on Deep Learning. It is written in Python and runs on Linux operating system. AutoGluon is divided into four main modules: tabular data, image classification, object detection, and text classification [3]. In this article, only the tabular prediction functionalities are being considered.

Auto-Keras: Auto-Keras is a Python library based on Keras [6] that implements AutoML methods with Deep Learning algorithms. The focus of Auto-Keras is the automatic search for Deep Learning architectures and hyperparameters, usually named Neural Architecture Search [13].

Auto-Sklearn: Auto-Sklearn is an AutoML Python library based on Scikit-Learn [28] that implements methods for automatic algorithm selection and hyperparameter tuning. Auto-Sklearn aims to free the user from the choice of an algorithm and the tuning of its hyperparameters using Bayesian optimization, meta-learning, and Ensemble Learning [16].

Auto-Weka: Auto-Weka is a module of WEKA, a ML tool that provides data preprocessing functions and ML algorithms that allow users to quickly compare ML models and create predictions using new data [34]. Auto-Weka aims to solve the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem, first established in [31].

H2O AutoML: H2O AutoML is one of the open-source modules of H2O, a ML analytics platform that uses in-memory data and implements a distributed and scalable architecture [7]. H2O AutoML uses H2O's infrastructure to provide functions to automate algorithm selection and hyperparameter optimization [21].

Rminer: Rminer is a package for the R tool, intending to facilitate the use of Machine Learning algorithms. The focus of Rminer are the CRISP-DM phases of Modeling and Evaluation [8,9]. In the most recent version, Rminer uses more than 20 classification and regression algorithms. Also, since version 1.4.4, Rminer implements AutoML functions.

TPOT: Tree-Based Pipeline Optimization Tool (TPOT) is an open-source AutoML written in Python. TPOT automates the phases of feature selection, feature engineering, algorithm selection, and hyperparameter tuning. It uses algorithms such as Decision Trees, Random Forest, and XGBoost, most of them from the Scikit-Learn library [25, 27].

TransmogrifAI: TransmogrifAI is a tool that uses Apache Spark framework to automate ML applications. It is written in Scala and focused on the automation of several phases of the ML workflow, such as algorithm selection, feature selection, and feature engineering [30].

AutoML Tool Comparison: Table 1 presents the characteristics of the analyzed AutoML related to interface language, associated platforms, current version and if it contains a Graphical User Interface and distributed ML mode.

For the experimental study, we selected H2O AutoML and TransmogrifAI, as these were the only tools from Table 1 that meet the distributed ML requirement. Table 2 presents the ML algorithms implemented by both tools. The last two rows are related to the stacking ensembles implemented by H2O AutoML: all, which combines all trained algorithms; and best, which only combines the best algorithm per family.

4.3 Data

For the benchmark study, we used three real-world datasets from the domain of telecommunications, provided by the IRMDA project analytics company. The datasets are related to customer churn prediction (regression), event forecasting (univariate time series), and telecommunications fraud detection (binary classification).

Table 1. Main characteristics of the analyzed AutoML tools (extended from [14]).

	Interface language	Associated platforms	Current version	Graphical user interface	Distributed mode
Auto-Gluon	Python	-	0.0.14	-	-
Auto-Keras	Python	-	1.0.9	-	-
Auto-Sklearn	Python	-	0.10.0	-	-
Auto-Weka	Python R	WEKA	2.6.1	✓	-
H2O AutoML	Python R Scala	AWS Azure Google Cloud Apache Spark	3.30.1.3	✓	✓
Rminer	R	-	1.4.6	-	-
TPOT	Python	-	0.11.5	-	-
TransmogrifAI	Scala	Apache Spark	0.7.0	-	✓

Table 2. Algorithms implemented by H2O AutoML and TransmogrifAI (adapted from [14]).

Algorithm	H2O AutoML	TransmogrifAI
Decision Trees	-	✓
Deep Learning	✓	-
Extremely Randomized Forest	✓	-
Gradient-Boosted Trees (GBT)	-	✓
Gradient Boosting Machine (GBM)	✓	-
Generalized Linear Model (GLM)	✓	-
Linear Regression	-	✓
Linear Support Vector Machine	-	✓
Logistic Regression	-	✓
Naive Bayes	-	✓
Random Forest (RF)	✓	✓
XGBoost	✓ (only fully supported in Linux)	-
Stacking All (SA)	✓	-
Stacking Best (SB)	✓	-

Churn Prediction: The churn dataset contains 189 rows and 21 attributes. The attributes of each row characterize a client and the probability for canceling the company's analytics service (churn), as defined by the company. Table 3 describes each attribute of the churn dataset.

Table 3. Description of the attributes of the churn dataset (adapted from [14]).

Attribute	Description
tenure	Time passed since the beginning of the contract
streaming_quality	Contractualized display resolution
ott_video	If OTT video is contractualized or not
contract	Duration of the contract
payment_method	Contractualized method of payment
product_name	Identification of the product
platform	Type of connectivity present in the contract
financial_status	If the payment is late or regularized
service_latency	Latency of the service
dropped_frames	Number of dropped frames
volume	Information about volume
duration	Information about duration
account_number	Account identification number
service_latency category	Category of the service latency attribute
dropped_frames category	Category of the dropped frames attribute
volume category	Category of the volume attribute
duration category	Category of the duration attribute
tenure category	Category of the tenure attribute
account_segment	Age segment of the client
equipment	Equipment used by the client
churn_probability	Probability of canceling the service ($\in [0, 1]$)

Table 4. Description of the attributes of the event forecasting dataset (adapted from [14]).

Attribute	Description
Time	Timestamp (format: yyyy-mm-dd hh:mm)
Datapoints	Number of events

Event Forecasting: The event forecasting dataset contains 1,418 rows that correspond to records about telecommunication events of a certain type (e.g., phone calls). The events occurred from February to April of 2019, aggregated on an hourly basis, ranged from 3,747 to 56,320. The only attributes are the timestamp and the number of events in that interval, as described in Table 4.

Fraud Detection: Each row of the fraud detection dataset contains the identification of A (sender) and B (receiver), and the classification of the phone call ("fraud" or "normal"). The dataset contains more than 1 million examples, which correspond to one day of phone calls from one of the company clients. The dataset attributes are described in Table 5.

Table 5. Description of the attributes of the fraud dataset (adapted from [14]).

Attribute	Description
A	Identification of the call sender
B	Identification of the call receiver
Result	Classification of the call ("fraud" or "normal")

5 Results

5.1 Experimental Setup

The benchmark consisted of several computational experiments that used three real-world datasets to compare the selected AutoML tools (H2O AutoML and TransmogrifAI). The benchmark was executed on a machine with an i7-8700 Intel processor with 6 cores. Every AutoML experiment considered a holdout split that used 3/4 of the data as training set and 1/4 as test set. The split between train and test sets was random for two of the datasets (churn and fraud). For the event forecasting dataset, the division between train and test was ordered in time (since the data is ordered in time). Every AutoML execution implemented a 10-fold cross-validation during the training of the algorithms.

Each AutoML tool optimizes a performance metric to select the best algorithms and tune the hyperparameters. We selected the Mean Absolute Error (MAE) for the regression tasks and Area Under Curve (AUC) for the classification data. Also, we computed additional metrics for the test data in order to further compare the tools.

Additionally, we disabled time limits to allow the execution of all selected ML algorithms. From the algorithms presented in Table 2, we only disabled Deep Learning from the experiments, from H2O AutoML. First, because it required a greater computational effort, especially for the fraud detection dataset. Second, to achieve a more fair comparison with TransmogrifAI, since this tool does not include Deep Learning algorithms.

For the churn dataset, the performance of the AutoML tools was measured with two scenarios. The first scenario (1) considered all the attributes of the dataset as input features for the ML algorithms. The second scenario (2) only uses a subset of the attributed as input features, derived from a previous feature selection phase. For TransmogrifAI the intention was to test the automatic feature selection characteristic. For H2O AutoML, the features of scenario 2 were the most relevant features identified by the best performing algorithm of scenario 1.

For event forecasting, we transformed the dataset, creating time lags as inputs for a regression task. The dataset could not be used as a univariate time series, since neither H2O AutoML nor TransmogrifAI implement native time series algorithms (e.g., ARIMA, Holt-Winters). We created three scenarios with different combinations of time lags: 1 – with time lags $t-1$, $t-24$, and $t-25$, where t is the current time (corresponding to the previous hour, day, and hour before that day); 2 – with all the time lags from the last 24 h (from $t-1$ to $t-24$); and 3 – with the time lags $t-12$, $t-24$, $t-36$, and $t-48$.

For the fraud detection dataset, we designed three training scenarios. Since the fraud detection dataset only has around 0.01% of illegitimate calls, we used the Synthetic Minority Oversampling Technique (SMOTE) technique [5] to balance the two classes in two of the scenarios. Scenario 1 used a simple undersampling that considered all "fraud" records and a random selection (with replacement) of "normal" cases. Scenarios 2 and 3 used SMOTE to generate extra fraud examples (100% and 200%, respectively). For each training scenario, we also considered three test scenarios of unseen data with different class balancing (with "normal"/"fraud" ratio): A – 50%/50%, thus balanced; B – 75%/25%; and C – 80%/20%.

5.2 Discussion

A summary of the overall results is presented in Table 6. For each AutoML tool, the execution times and test error metric values were aggregated by considering the average of the dataset scenario executions.

Table 6. Summary of the experimental results, best values in **bold** (adapted from [14]).

Dataset	Number of scenarios	AutoML tool	Avg. execution time (mm:ss)	Used metric	Avg. test metric
Churn prediction	2	H2O AutoML	**00:27**	MAE	**0.119**
		TransmogrifAI	03:40	MAE	0.160
Event forecasting	3	H2O AutoML	**02:25**	MAE	**2467**
		TransmogrifAI	04:41	MAE	2765
Fraud detection	9	H2O AutoML	07:11	AUC	**0.973**
		TransmogrifAI	**01:46**	AUC	0.963

The experimental results show that both AutoML tools require a small execution time to select the best ML model, with the highest average execution time being slightly higher than 7 min. The low training time can be justified with the usage of distributed ML, datasets with a small number of rows or columns, and the removal of Deep Learning algorithms. However, if the benchmark included datasets with more examples or attributes, an addition of machines or cores to the cluster would maintain the execution time low.

The metrics obtained during the predictions show that H2O AutoML obtained the best average results for all three datasets. In particular, H2O AutoML was better on three of the five regression scenarios and in seven of the nine classification scenarios. TransmogrifAI obtained the best predictive results in two regression scenarios and two classification scenarios. Although the AutoML tools present minor predictive differences, the results of all scenarios can be considered of high quality.

After analyzing the results, the risk management software and analytics company decided to select H2O AutoML for the model training module of the architecture. This choice was supported by two main reasons. First, H2O AutoML obtained better predictive results for most of the scenarios. Second, the analytics company considered H2O AutoML a "more mature" technology. This classification was due to the fact that H2O AutoML is available in more programming languages than TransmogrifAI (as shown in Table 1), it can be integrated with more platforms and it provides an easy to use Graphical User Interface.

6 Technological Architecture

After the comparative ML experiments, the analytics company selected the H2O AutoML tool for the model training component. The remaining technological modules were then designed in cooperation with the company. Since one of the prerequisites of the architecture is that it is distributed, we tried to identify technologies with distributed capabilities. Given that H2O can be integrated with Apache Spark (using the Sparkling Water module) and that Spark provides functions for data processing, we relied on Spark's Application Programming Interface (API) functions to implement the remaining components of the architecture. The updated architecture, with references to the technologies used, is illustrated in Fig. 2.

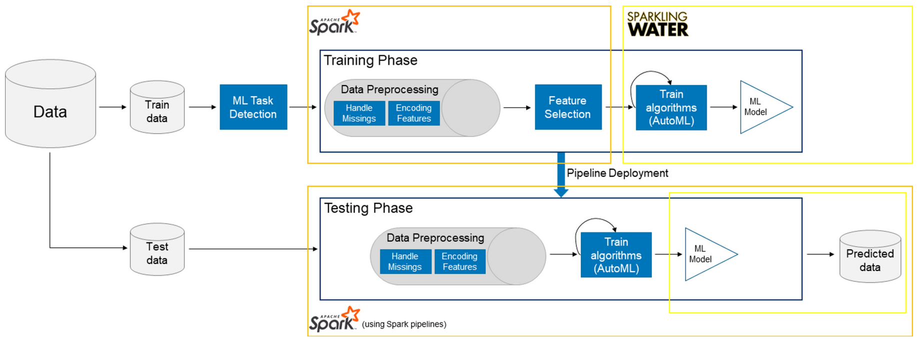

Fig. 2. The technological automated and scalable ML architecture (adapted from [14]).

6.1 Components

This subsection describes the current implementation of each module of the architecture. The updated technological architecture changed some of the modules initally described in Sect. 3. These changes were related to feedback received from the analytics company or due to technological restrictions.

Machine Learning Task Detection: Currently set to detect if the ML pipeline should be considered a binary classification, multi-class classification, pure regression, or a univariate time series task since these are the typical telecommunications risk management ML tasks used by the company.

The detection of the ML task can be overridden by the user. This is due to the fact that it could be useful to consider an ML task different than the one suggested by the module. For example, the end-user might want to consider a regression task, although the target column of the dataset only has a few number of levels, which could be automatically considered a multi-class classification. If the user specifies an ML task before running the pipeline, this component is skipped.

The type of supervised tasks handled will be expanded according to feedback provided by the software company clients and the AutoML tools capabilities. Interesting future possibilities of tasks to be addressed are multivariate time series, ordinal classification, or multi-target regression.

Data Preprocessing: Currently, the preprocessing transformations (e.g., dealing with missing data, the encoding of categorical features, standardization of numerical features) are done using Apache Spark's functions for extracting, transforming and selecting features [1].

To deal with missing data in numerical columns we use the `Imputer` function from Spark. This function replaces the unknown values of a column with its mean value. For categorical columns, we replace the unknown fields with a predefined tag (e.g., "Unknown"). The encoding of categorical features is done by default using Spark's one-hot Encoding function. If the categorical column has a high cardinality (a vast number of levels), instead of the one-hot encoding we apply the `String Indexer` function. This function replaces the values of the column by numerical indices. The standardization of numerical features uses the `Standard Scaler` function from Spark. This function normalizes the column to have mean zero and standard deviation one.

Feature Selection: Currently, this module uses the Chi-Squared feature selection function from Apache Spark. This method decides what features to keep based on Chi-Squared statistical test. Depending on the dataset and the ML task, we filter a fixed number of features or a percentage of features with the most correlation.

Additionally, we added the possibility for the user (usually a domain expert) to influence this step. Thus, the user can specify beforehand the features that will be used as inputs by the model training module. Such features cannot be removed by the feature selection step, although other features can be added to the ones that the user selected. If no features were chosen by the user, this component works without restrictions. Also

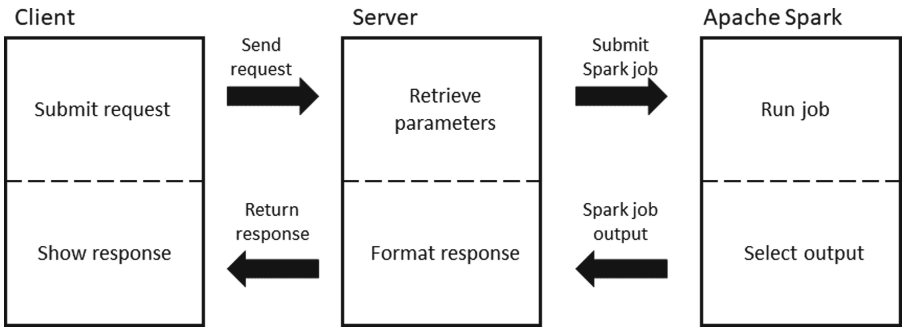

Fig. 3. Adopted scheme for handing of requests and responses.

by request of the company, we created an auxiliary pipeline that performs a simple feature filtering, outputting a list of the most relevant features for a particular supervised learning dataset but without fitting an ML model (e.g., usage of the simple correlation statistic).

Model Training: Currently, this module uses one of two AutoML approaches we implemented, depending on the ML task that is being considered. For classification (binary or multi-class) and regression tasks, we use H2O AutoML to automatically find and tune the best model. Since none of the AutoML tools we analyzed support native univariate time series forecasting algorithms, we implemented our own AutoML for the time series task.

In order to create the AutoML for time series, we used the algorithms implemented by the GitHub repository scalaTS[1] as a base. The repository includes a set of time series algorithms, such as autoregressive integrated moving average (ARIMA), autoregressive moving average (ARMA), autoregression (AR), and moving average (MA). Also, the package includes hyperparameter optimization capabilities, with the algorithms Auto ARIMA, Auto ARMA, Auto AR, and Auto MA, which pick the best parameters for each algorithm. The repository is built on top of Apache Spark using the distributed DataFrames objects, allowing distributed training and forecasting.

In order to select the best algorithm for a time series task, we run each Auto algorithm with the training data and select the one that performs best on the validation data by using a rolling window validation [26].

Pipeline Deployment: Currently, the pipeline management module uses an Apache Spark API related to ML pipelines [2]. To create a Spark ML pipeline it is necessary to detail a list of stages and then fit the pipeline to the training data. After fitting the pipeline to the training data, the Spark API allows the export of the pipeline to the disk. This process is applied during the training phase of the architecture.

To apply a pipeline to test data it is necessary to load the model from a file. Then,

[1] https://github.com/liao-iu/scalaTS/.

using the transform function, it is possible to apply the pipeline to previously unseen data. This process is applied during the test phase of the architecture, generating a set of predictions.

6.2 API

In order to facilitate the execution of the architecture, we also created a REST API to mediate the communication between the end-users and the pipelines. The development of the API resulted in two main endpoints: one to run the train pipeline and the other to run the test pipeline.

Since the execution of each request consists of one Apache Spark job (using H2O's capabilities through the Sparkling Water module), the API works as an intermediary between the end-user and the execution of the code inside Spark. This way, the API server receives the client's requests and uses the parameters of the body of the request to initiate a Spark job inside the server (using the spark-submit command). After the execution of the application that was submitted to Spark, the server receives the output of the job (e.g., metrics of training, predictions). The server formats the response to the appropriate format (e.g., XML, JSON) and sends the response to the client interface.

Figure 3 depicts this process. We highlight that the current version of the overall architecture, which received positive feedback from the Portuguese software company of the IRMDA project, is expected to be incrementally improved in future research. In particular, we intend to evolve and test the non AutoML components by using more real-world datasets and feedback from the analytics company clients.

7 Conclusions

This paper proposes a ML framework to automate the typical workflow of supervised ML applications without the need for human input. The framework includes the modules of task detection, data preprocessing, feature selection, model training, and pipeline deployment. The framework was developed within project IRMDA, a R&D project developed by a leading Portuguese software and analytics company that provides services for the domain of telecommunications risk management. The company clients work with datasets of variable sizes (large or small) and are mostly non-ML-experts. Thus, the proposed framework uses distributed ML to add computational scalability to the process and AutoML to automate the search for the best algorithm and hyperparameters.

In order to assess the most appropriate AutoML tools for this model training module, we initially conducted a benchmark experiment. First, we analyzed the features of eight open-source AutoML tools (Auto-Gluon, Auto-Keras, Auto-Sklearn, Auto-Weka, H2O AutoML, Rminer, TPOT, and TransmogrifAI). Then, we selected the tools that allowed a distributed execution for the experiments (H2O AutoML and TransmogrifAI). The benchmark study used three real-world datasets provided by the software company from the domain of telecommunications risk management. The proposed framework was positively evaluated by the analytics company, which selected H2O AutoML as the best tool for the model training module.

After the selection of H2O AutoML for the model training module, we developed the technological architecture. We selected technologies with distributed capabilities for the remaining modules of the initially proposed framework. Most of the remaining modules were implemented using Apache Spark's API functions. Then, we describe the current implementation of each module of the architecture. Finally, we describe the REST API that was created to facilitate the communication between the end-users (the company clients) and the implemented pipelines.

In future work, we intend to use more telecommunications datasets to provide additional benchmarks for the model training module. Moreover, new AutoML tools can be considered, as long as they provide distributed capabilities. Besides, we intend to add more ML tasks to the framework, such as ordinal classification, multi-target regression, or multivariate time series. For the remaining modules, we expect to conduct similar studies to evaluate the most appropriate technologies to use (e.g., for handling missing data, for choosing the best features). Finally, even though the framework was developed specifically for the telecommunications risk management domain, we intend to study the applicability of the framework to other areas.

Acknowledgements. This work was executed under the project IRMDA - Intelligent Risk Management for the Digital Age, Individual Project, NUP: POCI-01-0247-FEDER-038526, co-funded by the Incentive System for Research and Technological Development, from the Thematic Operational Program Competitiveness of the national framework program - Portugal2020.

References

1. Apache Spark: extracting, transforming and selecting features - Spark 2.4.5 documentation (2020) https://spark.apache.org/docs/latest/ml-features
2. Apache Spark: ML pipelines - Spark 2.4.5 documentation (2020). https://spark.apache.org/docs/latest/ml-pipeline.html
3. Auto-Gluon: AutoGluon: AutoML toolkit for deep learning — AutoGluon documentation 0.0.1 documentation (2020). https://autogluon.mxnet.io/
4. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. Artif. Intell. **97**(1–2), 245–271 (1997). https://doi.org/10.1016/s0004-3702(97)00063-5
5. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002). https://doi.org/10.1613/jair.953
6. Chollet, F., et al.: Keras (2015). https://keras.io
7. Cook, D.: Practical Machine Learning with H2O: Powerful, Scalable Techniques for Deep Learning and AI. O'Reilly Media, Inc., Sebastopol (2016)
8. Cortez, P.: Data mining with neural networks and support vector machines using the R/rminer tool. In: Perner, P. (ed.) ICDM 2010. LNCS (LNAI), vol. 6171, pp. 572–583. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14400-4_44
9. Cortez, P.: A tutorial on using the rminer r package for data mining tasks, Technical report, Universidade do Minho, Escola de Engenharia (EEng) (2015)
10. Cortez, P.: Package 'rminer' (2020). https://cran.r-project.org/web/packages/rminer/rminer.pdf
11. Darwiche, A.: Human-level intelligence or animal-like abilities? Commun. ACM **61**(10), 56–67 (2018). https://doi.org/10.1145/3271625

12. Elshawi, R., Maher, M., Sakr, S.: Automated machine learning: state-of-the-art and open challenges. arXiv preprint arXiv:1906.02287 (2019)
13. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. arXiv preprint arXiv:1808.05377 (2018)
14. Ferreira, L., Pilastri, A., Martins, C., Santos, P., Cortez, P.: An automated and distributed machine learning framework for telecommunications risk management. In: Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART, pp. 99–107. INSTICC, SciTePress (2020). https://doi.org/10.5220/0008952800990107
15. Feurer, M., et al.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, 7–12 December 2015, Montreal, Quebec, Canada, pp. 2962–2970 (2015). http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning
16. Feurer, M., Springenberg, J.T., Hutter, F.: Initializing Bayesian hyperparameter optimization via meta-learning. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 25–30 January 2015, Austin, Texas, USA, pp. 1128–1135. AAAI Press (2015). http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10029
17. Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., Vanschoren, J.: An open source autoML benchmark. arXiv preprint arXiv:1907.00909 (2019)
18. Guyon, I., et al.: Design of the 2015 chalearn autoML challenge. In: 2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, 12–17 July 2015, pp. 1–8. IEEE (2015). https://doi.org/10.1109/IJCNN.2015.7280767
19. Guyon, I., et al.: A brief review of the chalearn automl challenge: any-time any-dataset learning without human intervention. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Proceedings of the 2016 Workshop on Automatic Machine Learning, AutoML 2016, co-located with 33rd International Conference on Machine Learning (ICML 2016), New York City, NY, USA, 24 June 2016. JMLR Workshop and Conference Proceedings, vol. 64, pp. 21–30. JMLR.org (2016)
20. Guyon, I., et al.: Analysis of the AutoML challenge series 2015–2018. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Automated Machine Learning. TSSCML, pp. 177–219. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5_10
21. H2O.ai: H2O AutoML, June 2017. http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html, h2O version 3.30.0.1
22. He, X., Zhao, K., Chu, X.: AutoML: a survey of the state-of-the-art. arXiv preprint arXiv:1908.00709 (2019)
23. Jin, H., Song, Q., Hu, X.: Auto-keras: an efficient neural architecture search system. In: Teredesai, A., Kumar, V., Li, Y., Rosales, R., Terzi, E., Karypis, G. (eds.) Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, 4–8 August 2019, pp. 1946–1956. ACM (2019). https://doi.org/10.1145/3292500.3330648
24. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-weka 2.0: automatic model selection and hyperparameter optimization in WEKA. J. Mach. Learn. Res. **18**, 25:1–25:5 (2017). http://jmlr.org/papers/v18/16-261.html
25. Le, T.T., Fu, W., Moore, J.H.: Scaling tree-based automated machine learning to biomedical big data with a feature set selector. Bioinformatics **36**(1), 250–256 (2020)
26. Oliveira, N., Cortez, P., Areal, N.: The impact of microblogging data for stock market prediction: Using twitter to predict returns, volatility, trading volume and survey sentiment indices. Expert Syst. Appl. **73**, 125–144 (2017). https://doi.org/10.1016/j.eswa.2016.12.036

27. Olson, R.S., Urbanowicz, R.J., Andrews, P.C., Lavender, N.A., Kidd, L.C., Moore,J.H.: Automating biomedical data science through tree-based pipeline optimization. In: Squillero, G., Burelli, P. (eds.) EvoApplications 2016. LNCS, vol. 9597, pp. 123–137. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31204-0_9
28. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011). http://dl.acm.org/citation.cfm?id=2078195
29. Peteiro-Barral, D., Guijarro-Berdiñas, B.: A survey of methods for distributed machine learning. Prog. Artif. Intell. **2**(1), 1–11 (2013). https://doi.org/10.1007/s13748-012-0035-5
30. Salesforce: Transmogrifai (2019). https://docs.transmogrif.ai/en/stable/
31. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 847–855 (2013). https://doi.org/10.1145/2487575.2487629
32. Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, B., Farivar, R.: Towards automated machine learning: Evaluation and comparison of autoML approaches and tools. arXiv preprint arXiv:1908.05557 (2019)
33. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. SIGKDD Explor. **15**(2), 49–60 (2013). https://doi.org/10.1145/2641190.2641198
34. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, Amsterdam (2016)
35. Yao, Q., et al.: Taking human out of learning applications: a survey on automated machine learning. arXiv preprint arXiv:1810.13306 (2018)
36. Zöller, M.A., Huber, M.F.: Benchmark and survey of automated machine learning frameworks. Technical report. https://www.researchgate.net/publication/332750780