CompareML: A Novel Approach to Supporting Preliminary Data Analysis Decision Making

Antonio Jesús Fernández-García[1]*, Juan Carlos Preciado[2], Alvaro E. Prieto[2], Fernando Sánchez-Figueroa[2], Juan D. Gutiérrez[2]

[1] Universidad Internacional de La Rioja (Spain)
[2] University of Extremadura (Spain)

Received 8 February 2021 | Accepted 23 July 2021 | Published 2 August 2021

Abstract

There are a large number of machine learning algorithms as well as a wide range of libraries and services that allow one to create predictive models. With machine learning and artificial intelligence playing a major role in dealing with engineering problems, practising engineers often come to the machine learning field so overwhelmed with the multitude of possibilities that they find themselves needing to address difficulties before actually starting on carrying out any work. Datasets have intrinsic properties that make it hard to select the algorithm that is best suited to some specific objective, and the ever-increasing number of providers together make this selection even harder. These were the reasons underlying the design of CompareML, an approach to supporting the evaluation and comparison of machine learning libraries and services without deep machine learning knowledge. CompareML makes it easy to compare the performance of different models by using well-known classification and regression algorithms already made available by some of the most widely used providers. It facilitates the practical application of methods and techniques of artificial intelligence that let a practising engineer decide whether they might be used to resolve hitherto intractable problems. Thus, researchers and engineering practitioners can uncover the potential of their datasets for the inference of new knowledge by selecting the most appropriate machine learning algorithm and determining the provider best suited to their data.

Keywords

Classification, Decision Support System, Knowledge Elicitation, Machine Learning, Regression, Software.

DOI: 10.9781/ijimai.2021.08.001

I. Introduction

The evolution of computing capacities and the democratization of access to cloud computing at reasonable and affordable prices have fostered the appearance of a great number of machine learning applications for different fields. The engineering field in particular is not unaware of this trend. Engineering researchers and practitioners in a variety of areas have been drawn into the use of machine learning with the aim of inferring knowledge from their data. In particular, there are many engineering problems that can be solved using machine learning and artificial intelligence, and this, together with the Internet of Things and Big Data, inter alia, is one of the foundations of the fourth industrial revolution.

In most cases, the lack of knowledge about machine learning hinders engineers' and researchers' ability to analyse the potential that their data may have. Even with such knowledge, they will have to choose to use any from among an ever increasing number of alternatives with respect to algorithms, frameworks, tools, and providers. Given this context, it is hard to properly assess which will be the best way to address a specific engineering problem involving some particular data.

* Corresponding author.
E-mail address: antoniojesus.fernandez@unir.net

Moreover, the problem is compounded because, although well-known machine learning algorithms (such as Linear Regression, Decision Tree, Logistic Regression, Support Vector Machine, or Naive Bayes) are supposed to perform exactly the same, their implementations in third-party libraries and services can yield different results depending on the particular dataset.

Thus, any thorough evaluation of the different algorithms and their different implementations will be a time-consuming task. This ultimately results in most practitioner engineers and researchers in the field usually relying on their own experience to select a particular platform and algorithm, so that they will probably miss others that might well better reveal the whole potential of their datasets.

In this context, given this variety of options, and before taking into account such other factors as affordability, availability, complexity, or expertise, a preliminary study to find the optimal choice for a particular dataset might be helpful even for engineers, researchers, or scientists with no great depth of machine learning knowledge. While there are general-purpose machine learning tools that allow different algorithms to be applied to the same dataset, to the best of our knowledge, there is no solution available that is able to quickly and easily compare algorithm implementations from different providers.

All of this led us to posit the following research questions:

- RQ1: *Has a dataset the potential, i.e., is it possible to infer knowledge from it, to be used in a real-world engineering application or to solve a real-world engineering problem?*

TABLE I. Software Comparison

Software	Regression	Classification	Clustering	Deep Learning	Comparison	Visualization	Multiple Providers	Knowledge
Weka	✓	✓	✓	✓	✓			✓
Orange	✓	✓				✓		✓
RapidMiner	✓	✓	✓	✓		✓		✓
Knime	✓	✓	✓			✓		✓
CompareML	✓	✓			✓		✓	

(Regression: Support for Regression algorithms, Classification: Support for Classification algorithms, Clustering: Support for Classification algorithms, Deep Learning: Support for Deep Learning algorithms, Comparison: Model Comparison, Visualization: Data Visualization, Multiple Providers: Implementation of algorithms from Multiple Providers, Knowledge: Machine Learning Knowledge required.)

- RQ2: *Is it possible to determine, a priori, which are the best algorithms to reveal the potential of a dataset and construct the most appropriate predictive model from it?*
- RQ3: *Is it possible to select from among the wide range of libraries and services that allow the creation of predictive models those which are, a priori, the best to work with a specific dataset and problem?*

With the aim of answering these questions, we present *CompareML*, an approach that allows practitioner or research engineers to make a preliminary analysis of their data by testing different machine learning algorithms from different providers. The main goal of this approach is to allow such users to select the most suitable environment for their datasets without requiring any in-depth knowledge about machine learning. Thus, using *CompareML*, engineers can quickly and effortlessly compare the performance of different algorithms and providers applied to their specific datasets before deciding which to employ in their machine learning models.

The resulting software supporting the *CompareML* approach is licensed under an MIT permissive free software licence for it to be useful, reusable, and customizable in machine learning research areas. It is readily accessible at the following URL: https://compareml.io/. The software is already very intuitive, but nevertheless there is a User Manual[1] available for other researchers to answer any doubts they may have when using *CompareML*. Finally, scripts are available to enable full automated portability of the software.

The rest of this paper is organized as follows. Section II summarizes other approaches, and lists the algorithms and providers supported by *CompareML*. Section III describes the approach. Section IV details the implementation of the approach and the software architecture. Section V provides an illustrative example using *CompareML*. Finally, Section VI draws some conclusions and describes future work.

II. Related Work and Background

This section reviews some related work, describes the fundamentals of the machine learning algorithms compared by *CompareML*, and presents the state-of-the-art of the libraries and services available on the market, describing those supported by *CompareML*.

A. Related Work

There are general-purpose machine learning tools that have dealt with similar problems:

- Weka [1], a machine learning software package ideally suited for teaching and research which allows, inter alia, the comparison of algorithms for a dataset. It has recently been enriched with WekaLearning4j [2], a deep-learning package based on Deeplearning4j. The software is free under GNU GPL 3 for non-commercial purposes.
- Orange [3], a machine learning software package that allows data analysis and data visualization to be performed on datasets. It can be applied by means of Python scripts or through a visual programming interface in which users can make use of its functionalities. The software is developed by the University of Ljubljana, and is open-source released under a GPL licence.
- RapidMiner [4], [5], a data science software platform that implements data mining algorithms as operators that users can visually drag and drop to create customized dataflows. It is proprietary software developed by the RapidMiner firm, although previous versions were open source.
- Knime [4], a data mining tool integrated in Eclipse (the Java Integrated Development Environment) with a visual interface in which users make use of blocks, connecting them to create dataflows that visualize, deploy, and manage machine learning models. It is open source software developed by the University of Konstanz.

Table I presents a comparison of the work mentioned in this section in such aspects as:

a) Support for Regression algorithms;
b) Support for Classification algorithms;
c) Support for Clustering algorithms;
d) Support for Deep Learning algorithms;
e) Model Comparison;
f) Data Visualization;
g) Implementation of Multiple Providers' algorithms;
h) Machine Learning Knowledge required.

A more detailed and specific comparison of these software packages and the Scikit-Learn library and R Programming Language can be found in [6], where the authors compare software and services that are beyond the scope of this present work.

Our approach follows AutoML or Automated Machine Learning principles. AutoML [7] is an idea that consists of automating the entire pipeline or a part of a machine learning project. This is a hot topic of interest in both industry and academia, and the evaluation of its results [8] has led to several AutoML approaches and tools emerging. Some of those most widely used are:

- Automated Machine Learning in PowerBI [9] was launched by Microsoft in 2019 to allow business analysts, economists, and PowerBI users in general to build machine learning models to solve business problems without their needing to have a strong background in machine learning.
- PyCaret [10] is an open-source machine learning library in Python developed by Moez Ali and launched in 2019. Its main characteristic is its low-code orientation, making it simple and easy to use.
- Cloud AutoML [11] is the AutoML platform launched by Google in 2018 that trains custom machine learning models for image, video, and tabular data, as well as making use of pretrained models to create natural language processing, image classification, video recognition, or structured data discovery applications.

[1] User Manual available at http://shorturl.at/yCLX4

In addition to the software analysed, one can find in the literature related work worthy of mention with several tools and libraries, such as:

- LEAC [12], an efficient library for clustering with evolutionary algorithms in the neural networks field.
- Ruta [13], which implements autoencoders, neural networks that perform feature learning on data.
- AutoML-Zero [14], a specific-purpose tool that simultaneously searches for all aspects of a machine learning algorithm, using basic maths operations, with the objective of reducing human bias in the search space.

However, as noted above, to the best of our knowledge, there is no solution that is able to compare algorithm implementations from different providers while requiring no depth of machine learning knowledge on the part of the user.

B. Background

CompareML provides researchers with the possibility of creating models of their data using the following well-known algorithms:

- Linear Regression. This assumes a linear relationship between the input variables and a single output variable. The model learns by estimating the values of the coefficients used in the representation from the data available. The linear regression can be expressed as $y = ax + b$, where a and b are the aforementioned coefficients.
- Decision Tree. Decision tree algorithms build a tree-like structure in which each node represents a question concerning an attribute. The responses to that question create new branches, expanding the structure until the end of the tree is reached, with the leaf node being the one that indicates the predicted class.
- Boosted Decision Tree. This is a general method, not limited to decision trees, which consists of applying a boosting method to combine many classifiers into a new and stabler one with a smaller error. In the boosting, the predictors are made sequentially rather than independently, applying the rationale that the subsequent predictors learn from the mistakes of the previous ones.
- Random Forest. This algorithm is an improvement that creates several decision trees, using bagging or some other technique, and votes for the most popular output that the trees yield. Usually, most implementations do not count the outputs directly, but sum their normalized frequencies to get the label with greatest likelihood.
- Logistic Regression. This algorithm uses a more complex cost function – the 'sigmoid' or 'logistic' function – than the Linear Regression model. Input values are combined linearly using weights or coefficients to predict an output value. A key difference with Linear Regression is that the output being modeled is dichotomous (a 0 or 1) rather than numerical.
- Support-Vector Machine. The data are mapped onto a high-dimensional feature space so that the data points can be categorized even when the data are not otherwise linearly separable. Then, a separator is estimated for the data. The data should be transformed in such a way that a separator can be drawn as a hyperplane. As there are many possible hyperplanes, the Support Vector Machine algorithm finds a hyperplane that represents the largest separation, or margin, between classes.

Of particular importance are the libraries and services from different machine learning providers that *CompareML* supports:

- Turi Create [15], a Python™ [16] package that allows programmers to perform end-to-end large-scale data analysis and data product development. It is a distributed computation framework written in C++, developed at Carnegie Mellon University, and acquired in 2016 by Apple Inc.
- Scikit-Learn [17], one of the most popular machine learning libraries. It is largely written in Python with some core algorithms written in Cython to improve performance. It is supported by several institutional and private grants.
- R [18], a programming language that is an environment for statistical computing software written in C, Fortran, and R itself. It is widely used in machine learning tasks. It is developed by the *R Core Team*. *CompareML* runs R code embedded in Python through the access provided by the *rpy2* library.

III. Approach

A. The Approach in a Nutshell

The approach consists of a software architecture that supports machine learning experiments being carried out very easily, even for practitioners with no in-depth knowledge or skills in machine learning and artificial intelligence. Specifically, the approach allows classification and regression models to be implemented, and evaluates them so that their performance can be very easily compared using metrics that are in line with the models' types. Additionally, none of the machine learning algorithms are constrained to a single library or provider. Instead, several implementations of the algorithms from different libraries, tools, or providers can be built and evaluated, so that, for every scenario (i.e., dataset and problem to solve), it is possible to select the most suitable algorithm and the most appropriate implementation library or provider.

B. Data Analysis Decision Making

After the foregoing description of the main concepts of the approach's architecture, this section will present a discussion of the proposed pipeline on which *CompareML* is built. Engineers and practitioners will follow this pipeline (Fig. 1) to decide whether there is hidden knowledge within their data, and, if so, which are the best algorithms and providers for them to use to build accurate models.

The circumstances begin when engineers are faced with trying to solve some problem, and they think that machine learning presumably can help in looking for a solution. After collecting the data, reasonable doubt arises about the data's suitability and the possible hidden knowledge they contain so that the problem can be solved properly.

In the interest of clarifying the data's potential, researchers or engineers upload their data to an implementation of the *CompareML* approach, indicating the attribute to predict (the label). They then choose the machine learning libraries and services they can work with, and select those machine learning algorithms they want to use to create predictive models. In response, after evaluating and comparing the models generated, *CompareML* provides performance metrics about the models. It is quite common to do that as can be seen in work such as [19]–[22] where authors create several models to find out the one that gets better results. By analysing these performance metrics, engineers can decide whether their data has the potential to build accurate models with which to solve their problem, and, if so, the algorithm and provider combination most likely to produce a reliable solution.

C. Inputs and Outputs

There are certain classes of data that need to be sent to *CompareML*. These input data are listed in Table II.

In version 1.0, the validation software does not automatically recognize the selected label's data type, so that the experiments will be run regardless of the appropriateness of the label data type and the type of algorithms selected. In such cases, the experiments' results will be given according to how each one of the libraries, services, and tools selected handles these situations.

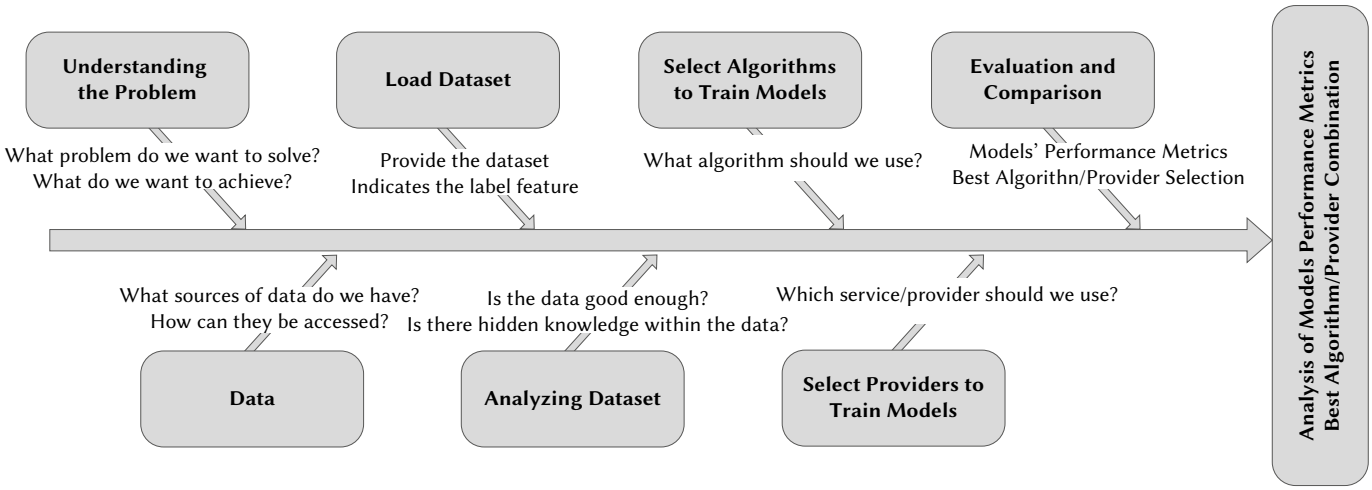

Fig. 1. Pipeline of the approach.

TABLE II. *CompareML* Input Data

Input Name	Description	Observations
`Dataset`	The dataset must be in CSV (comma-separated values) format, and it must contain a header row.	Notice that, since the comma is used as a separator, *CompareML* cannot handle field data containing commas or embedded line breaks. Also, it may not handle other unconventional characters.
`Label`	Feature that models will predict.	After uploading a dataset, users must select the label from among the dataset's existing features.
`Providers`	Machine learning libraries and services available to build models.	Users must select at least one of them. The validation software of the approach supports Turi Create, Scikit- Learn, and R.
`Algorithms`	Regression and classification algorithms available to build models.	Users must select at least one of them. The validation software of the approach allows users to choose between the Linear, Decision Tree, and Boosted Decision Tree regression algorithms, and the Random Forest, Logistic Regression, and Support Vector Machine classification algorithms.

The outputs of *CompareML* are a set of metrices that allow the appropriateness of the models created to be studied using different algorithms from different machine learning tools and services. Those outputs vary depending on the user's algorithm selection.

The outputs for *Regression* algorithms are:

- **RMSE**. This is a measure of the differences between the values predicted by a model and the observed values.

  RMSE can be defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2} \quad (1)$$

  where

  - N is the number of instances;
  - $\hat{y}_1, \hat{y}_2, ..., \hat{y}_n$ are the values predicted by the model; and
  - $y_1, y_2, ..., y_n$ are the observed values

- **R²**. Also known as the Coefficient of Determination, this is a metric that helps to explain the relationship between two variables. It ranges from 0 to 1, with the closer to 1, the better the model being analysed. Although a useful metric, it should be noted that, as more independent variables are added, R^2 will always rise.

  R^2 can be defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (2)$$

  where

  - N is the number of instances,
  - $\hat{y}_1, \hat{y}_2, ..., \hat{y}_n$ are the values predicted by the model,
  - $y_1, y_2, ..., y_n$ are the observed values, and
  - $\bar{y}$ is the mean of the n observed values.

- **Max-Error**. The Max-Error metric is the worst case error between a predicted value and a true value.

  Max-Error can be defined as:

$$Max\text{-}Error(y, \hat{y}) = \max\left(|y_i - \hat{y}_i|\right) \quad (3)$$

  where

  - $\hat{y}_1, \hat{y}_2, ..., \hat{y}_n$ are the values predicted by the model, and
  - $y_1, y_2, ..., y_n$ are the observed values.

- **Raw Data**. Information yielded directly by the provider.

The outputs for *Classification* algorithms are:

- **Accuracy**. An accuracy value, indicating the correctly predicted instances, is given for each algorithm selected. It can be defined as:

$$Accuracy = \frac{Number\ of\ correctly\ predicted\ observations}{Total\ number\ of\ observations} \quad (4)$$

- **Confusion Matrix**. A table layout where each row represents the number of instances of each class and each column represents the class that has been predicted by the model. A confusion matrix is created for each algorithm selected [23].

- **Precision**. This indicates the proportion of predicted positives. A precision value is given for each algorithm selected. Using the confusion matrix, it can be calculated as:

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

  where

  - TP = true positives,
  - FP = false positives.

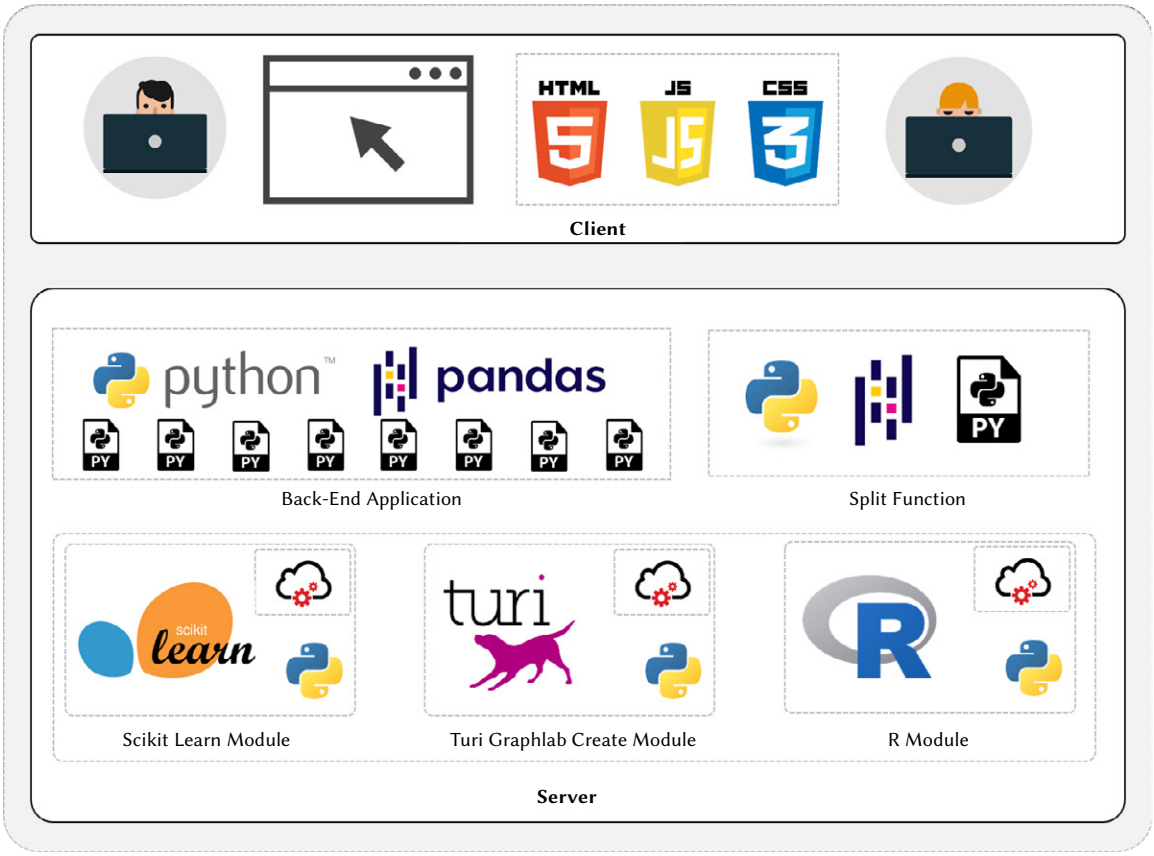

Fig. 2. CompareML Software Architecture.

- **Recall (Sensitivity)**. This indicates the proportion of positives predicted as positives. A recall value is given for each algorithm selected. Using the confusion matrix, it can be calculated as: (number of true positives / (number of true positives + number of false negatives).

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

where

  - TP = true positives,
  - FP = false positives.

- **Raw Data**. Information yielded directly by the provider.

IV. Implementation

A. Software Architecture

CompareML has been implemented following a classical client-server software architecture with an MVC (Model View Controller) approach [24] – a suitable option for this solution.

The server hosts the resources that manage the creation of models and deliver the results to the client. The client interacts with researchers through its user interface, and initiates requests to the server. This architecture is illustrated graphically in Fig. 2, and consists of the following modules:

- **Client**. This module handles the presentation layer, and is responsible for interacting with users. In it, users upload the dataset and set the configuration of the experiments that are sent to the server side. When the experiments are carried out, the results are sent back to this module to be shown to users in a friendly manner. The user interface has been designed to maximize usability, being simple, consistent, and offering cross-browser compatibility. This module was developed using widely used technologies such as HTML5, CSS3, and JavaScript.
- **Back-End Main Application**. This is the core of *CompareML*. It is responsible for coordinating and controlling the software's operational processes. It receives from the user interface the conditions under which the experiments must be carried out, and calls the Turi Create, Scikit-Learn, and R modules required, sending them the conditions of the experiments that affect them (algorithm selection, training dataset, test dataset, ...). When the execution of those modules ends, it receives the results and sends them back to the user interface. This module was developed in Python.
- **Back-End Split Function**. The split function is a special module in the back end that deals with the problem of splitting the dataset uploaded by researchers into two subsets: the training dataset containing 80% of the instances of the total dataset, and the test dataset containing the remaining 20% of the instances. This task is carried out in this module because it is necessary to ensure that the experiment's results are as objective as possible. If each provider module were to divide the dataset itself randomly, the random seeds would be different, and this would have potentially negative implications for any objective comparison of the models created through each of those modules. The split function module was developed using Python and Pandas [25], a powerful open source data analysis and manipulation tool built on top of the Python programming language.
- **Provider Modules**. These contain the implementations of the Random Forest, Logistic Regression, and Support Vector Machine classification algorithms and the Linear, Decision Tree, and Boosted Decision Tree regression algorithms using Scikit-Learn. It makes use of the *sklearn* library to build and evaluate the models

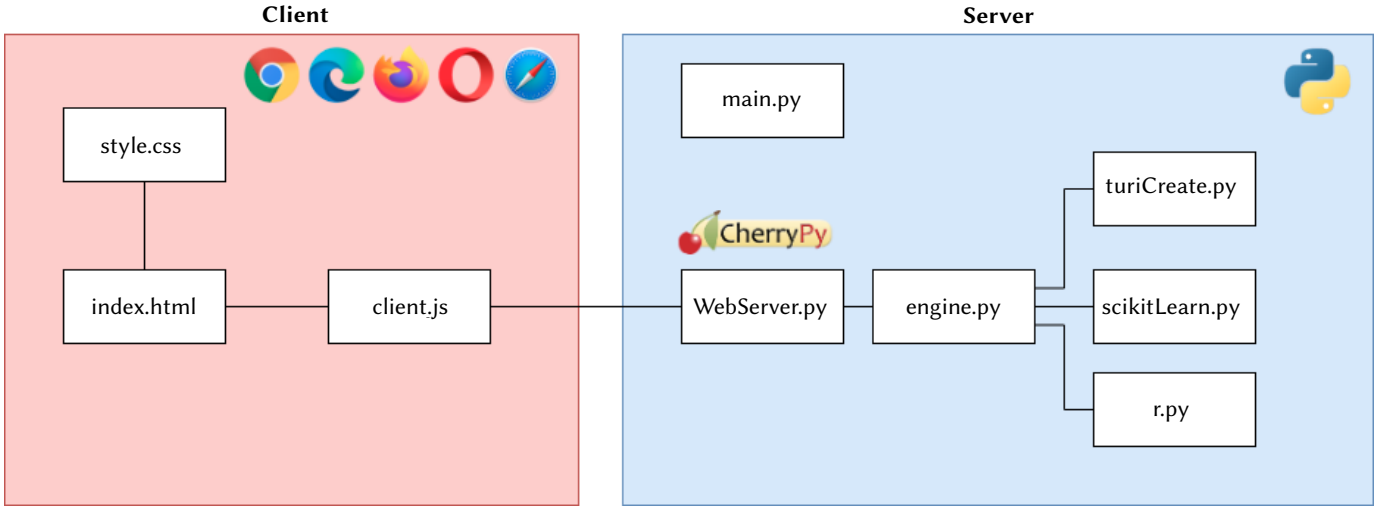

Fig. 3. *CompareML* file structure.

and the *pandas* library to manipulate data using its *DataFrame* data structure and functions. These modules receive as inputs the training and evaluation datasets, the algorithms that need to be used to build models, and the providers. When the experiments are carried out, the results are sent to the Back-End Main Application module. The functionalities of the machine learning providers, i.e., the experiments carried out using each provider's libraries, data structures, and functions, are isolated within the server side so as to facilitate their development.

B. File Structure and Software Business Process

The file structure is illustrated graphically in Fig. 3. The following is a description of the server files:

- `main.py`. This file contains the Web server configuration, defines the resources shown, and links with WebServer.py. It is implemented within the CherryPy framework [26].
- `WebServer.py`. This file receives the input from the client side, and processes it for subsequent handling.
- `engine.py`. This file is the core of *CompareML*. It contains the "split function" module which divides the dataset into training and evaluation, and communicates with the provider modules that need to be called to fulfill the requirements of the experiments defined by the researchers.
- `turiCreate.py`, `scikitLearn.py` and `R.py`. These files create the Turi Create, Scikit-Learn, and R models, respectively.

Fig. 4 is a BPMN (Business Process Model and Notation) diagram of the business processes of the modules in *CompareML* aimed at better illustrating the module relationships and the software framework.

C. Reproducibility and Collaboration

The guidelines on which the approach is based have been described in depth in the preceding subsections. However, in order to facilitate replication of the work and to encourage its implementation, in this subsection additional material is provided so that engineers can easily make use of this approach to perform preliminary data analysis.

The source code of an implementation of the approach is freely available in a GitHub repository under an MIT permissive free software licence[2]. A Developer Manual and a User Manual are provided. The Developer Manual, integrated into the Readme.md[3] file of the GitHub repository, describes clearly how the software is structured and how to make use of the code. It is possible to find Vagrant and Ansible scripts that simplify the software's portability, configuration, and deployment by any party interested in employing their own infrastructure. The User Manual[4] helps engineers without any in-depth knowledge of machine learning or data science to make use of the approach and to get answers to any doubts they may have about how it operates. Moreover, as was noted above, an implementation of the approach has been deployed on the Web[5] so that engineers and researchers can use it without the need to configure it themselves.

Table III provides information relevant for the deployment of software based on the proposed approach.

TABLE III. Information for the Deployment of Software Based on the Proposed Approach

Executable	https://compareml.io/
Licence	MIT licence (MIT)
Platforms	Windows, Linux, MacOS
Installation requirements	The approach can be deployed as a Web application. It is not necessary to install additional software, just a Web browser.
User manual	https://raw.githubusercontent.com/i3uex/CompareML/master/CompareML%20User%20Manual.pdf
Developer manual	https://github.com/i3uex/CompareML/blob/master/README.md
Software used	CherryPy, Python
Compilation requirements	Python dependencies: CherryPy, pandas, 14 sklearn, tensorflow, turicreate. R dependencies: 15 optparse, testthat, ggplot2, randomForest, caret, 16 e1071.

However, it might be useful to illustrate how potential research users could, in their *CompareML* solution, add new algorithms to those already implemented. The *CompareML* software architecture has a simple and elegant design to encourage potential research users to aid in the growth or development of the solution by adding new algorithms or providers to those already implemented. In order to do so, researchers or developers just have to access the files *scikit_learn.py*, *turi_create.py*, or the *R* folders under the providers directory to add the implementation of an algorithm. It is possible to create a new file in that directory to include a new provider. Once the implementations of the new algorithms are done, the *engine.py* file must be updated with

[2] Link to the Source Code: https://github.com/i3uex/*CompareML*/

[3] Link to the Developer Manual: https://github.com/i3uex/*CompareML*/ blob/master/README.md

[4] Link to the User Manual: http://shorturl.at/yCLX4

[5] *CompareML* deployed on the Web: https://compareml.io/

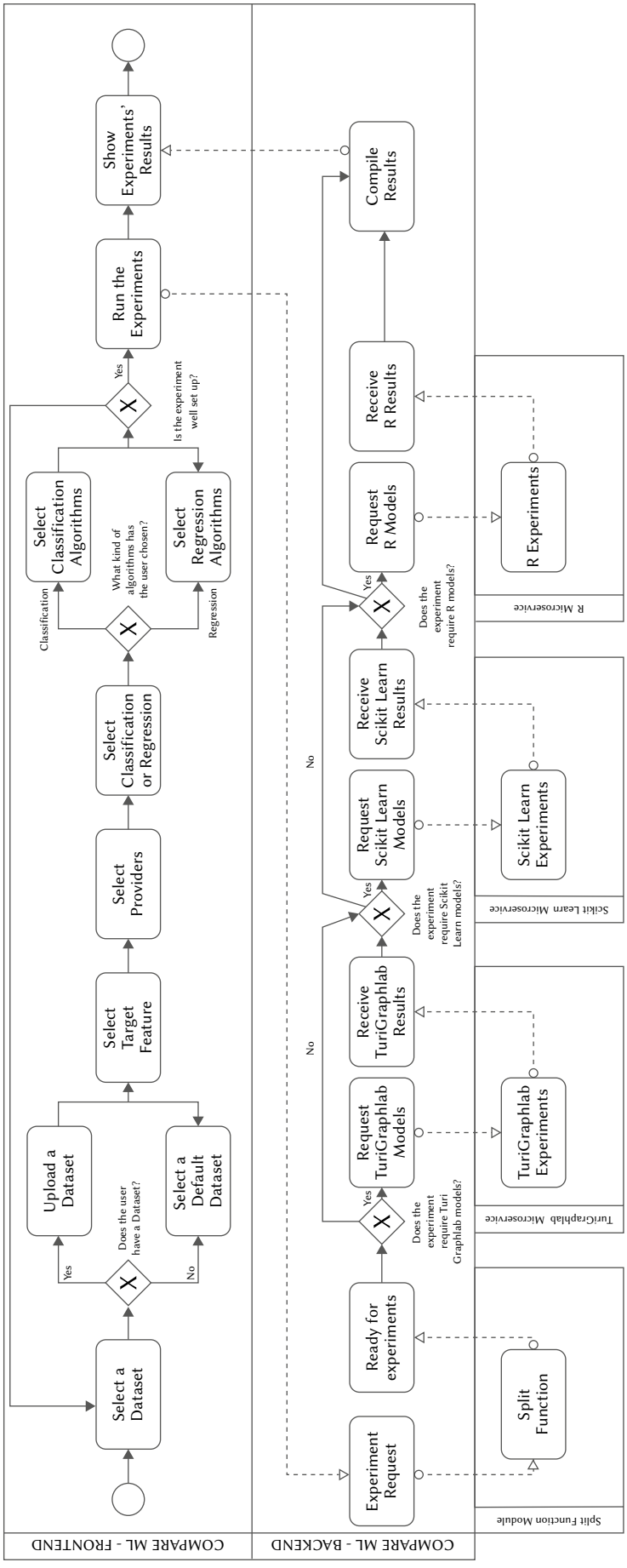

Fig. 4. *CompareML* BPMN diagram.

the list of algorithms and providers as shown in Listing 1. Lastly, the call of the newly implemented methods must be done in the same file.

Listing 1. Code from *engine.py* showing the providers and algorithms currently supported in *CompareML*.

```
PROVIDERS   =   {
   c.TURI_CREATE:   turi_create,
   c.SCIKIT_LEARN:   scikitLearn ,
   c.R:   r,
}

ALGORITHMS   =   {
   'classification':
        [ c.RANDOM_FOREST,
        c.LOGISTIC_REGRESSION,
        c.SUPPORT_VECTOR_MACHINES
   ],
   'regression':
        [ c.LINEAR_REGRESSION,
        c.BOOSTED_ DECISION_TREES,
        c.DECISION_TREE
   ]
}
```

V. Illustrative Example & Results

In this section, as illustrative examples, we make use of *CompareML* to solve two problems – a classification problem and a regression problem. These case studies are good representations of the types of problem for which the approach is useful. The first case study focuses on training classification models, and provides the output metrics of each trained algorithm to allow their evaluation by users. The second case study focuses on training regression models to the same end.

The datasets in the two case studies are in a tabular CSV format, as is supported by*CompareML*. It is important to stress that the current implementation does not allow data preparation operations or the application of feature engineering techniques, so data transformation must be done before loading the dataset into the *CompareML* implementation. Nevertheless, it is easy to rerun the experiments after performing such data transformation operations, and then compare the differences.

A. Classification Example

In this example, the aim is to find the provider which, a priori, is best suited to dealing with the CarEvaluation dataset [27] obtained from the UCI Machine Learning Repository [28]. This dataset comprises labeled data obtained from 1728 cars where the goal is to evaluate car conditions based on certain characteristics related to price and comfort.

The dataset consists of 1726 instances (observations) together with 7 features (variables) which are detailed in Table IV. In contrast with the original dataset, where there are 4 classes of the label feature Acceptability {unacceptable, acceptable, good, verygood}, just 2 categories have been adopted {yes, no} so that we can make use of two-class classification algorithms.

TABLE IV. Set of Features of the CarEvaluation Dataset

Feature Name	Description	Type
Price	Buying price	Categorical
Maintenance	Price of maintenance	Categorical
Doors	Number of doors	Numerical
Seats	Capacity in terms of persons to carry	Numerical
Boot	Size of luggage boot	Categorical
Safety	Estimated safety of the car	Categorical
Acceptability	Car acceptability	Categorical (*Label*)

The steps that need to be followed to carry out the experiment in this implementation of the *CompareML* approach are:

1. Provide the dataset to *CompareML*. To this end, the option of selecting a default dataset is ignored, and the CarEvaluation dataset is directly uploaded from our computer.
2. Select the label feature that we are interested in predicting. After uploading the dataset, the drop-down menu of this section is filled with the names of all the variables. In our case, we select the Acceptability variable, which indicates the level of acceptability of a car according to its characteristics, i.e., the field we wish to predict.
3. Choose the machine learning libraries and services in which we want to run the experiment (Scikit-Learn, Turi Create, and R). At least one of them must be selected. In our example, we are interested in comparing all three providers.
4. Select whether we are going to carry out a *Regression* or a *Classification* experiment. Depending on the type of algorithm selected, we can choose from a variety of algorithms. We are interested in predicting a categorical value, and, for that reason, we mark the *Classification* algorithms checkbox. After selecting this option, we must select at least one algorithm. In our example, we want to perform an experiment including all the algorithms available (Random Forest, Logistic Regression, and Support Vector Machine).

If the experiment is set up properly, we can run it by pressing the "Start" button. If not, an error message will be shown describing how to resolve the problem.

When the experiment has been performed, *CompareML* shows the results below the "Start" button. Fig. 5 is a screenshot of the *CompareML* user interface and the output produced after the classification experiment has been carried out. The results of the experiment are

TABLE V. Results of the Classification Case Study Experiment

		Turi Create	Scikit Learn	R
Ramdom Forest	**Accuracy**	0.9480	0.9220	0.9855
	Precision	0.9091	0.4610	0.9876
	Recall	0.3704	0.5000	0.9969
Logistic Regression	**Accuracy**	0.9855	0.9769	0.9884
	Precision	0.8667	0.9322	0.9968
	Recall	0.9630	0.9027	0.9906
Support Vector Machine	**Accuracy**	0.9855	0.9769	0.9827
	Precision	0.8667	0.9197	0.9906
	Recall	0.9630	0.9197	0.9906

presented in Table V. They include the *Accuracy*, *Precision*, and *Recall* evaluation metrics. *CompareML* also provides the *Confusion Matrix* and raw data with information yielded directly by the provider.

As expected, building a model using the same algorithm and the same data produces similar but slightly different results. This implies that, even when the algorithms are the same, their implementations by different providers impacts the performance of the models built using them. Although in most cases those differences are small, in some cases they are significant. Let us focus for example on the *Accuracy* metric. In this case study, the accuracy of the *Logistic Regression* model with *R* is 6.64 percentage points greater than that of the *Random Forest* model with *ScikitLearn*, which is a considerable difference. In general however, the differences are smaller. With the exception of *Random Forest*, the differences between the models built with the same algorithms from different providers do not reach 1 percentage point. Nevertheless, that percentage point can make a difference.

Looking at the outcomes of the experiment, one can see that, especially with *TuriCreate* and *ScikitLearn*, the accuracy of the *Random Forest* models, regardless of the provider, is poorer than that of the *Logistic Regression* and *Support Vector Machine* models. This may be an indication that the models created with these algorithms are better suited to the CarEvaluation dataset. The greatest accuracy (98.84 percentage points) corresponds to the *Logistic Regression* model built with *R*.

The experimental results of this illustrative example can be reproduced using the CarEvaluation dataset which is preloaded in *CompareML*, by selecting acceptability as target feature (label).

B. Regression Example

In the regression example, we use the Heating dataset [29] obtained from the UCI machine learning Repository [28]. This contains data obtained from energy analyses applied to 12 different building shapes. The dataset consists of 768 instances (observations) together with 9 features (variables), which are detailed in Table VI. The label feature HeatingLoad is a continuous numerical feature that is a measure of the amount of heat energy that would need to be added to a space to maintain the temperature within an acceptable range. The remaining features of the dataset are {Relative Compactness, Surface Area, Wall Area, Roof Area, Overall Height, Orientation, Glazing Area, Glazing Area Distribution, Heating Load}, with this last being the label in our case study.

The steps needed to carry out the experiment are the same as in the previous experiment. The algorithms available for this kind of experiment are Linear Regression, Boosted Decision Tree, and Decision Tree.

Fig. 6 is a screenshot of the *CompareML* user interface and the output of the regression experiment. For readability, the results are listed in Table VII for the *RMSE* and *Max-error* evaluation metrics.

As was the case for the classification example, the experimental results differed slightly depending on the algorithms and providers used to create the model. Overall, the *Decision Tree* models gave uniformly good results, but the best result was with *Scikit-Learn* creating the model using the *Boosted Decision Tree* algorithm.

TABLE VI. Set of Features of the Heating Dataset

Feature Name	Type
Relative Compactness	Categorical
Surface Area	Numerical
Wall Area	Numerical
Roof Area	Numerical
Overall Height	Numerical
Orientation	Numerical
Glazing Area	Numerical
Glazing Area Distribution	Numerical
Heating Load	Numerical (*Label*)

Analysing these results, one observes that on occasions the same algorithm can yield the best and the worst results depending on the provider that implements and deploys the algorithms. Although it is not often the case, it is still of interest that, given a specific problem and a particular dataset, the choice of provider can still have a major impact. For this experiment in particular, the difference in RMSE between the best and the worst models is 5.7872, and these are with the same algorithm – *Boosted Decision Tree*.

For the *Linear Regression* algorithm, the evaluation metrics (for both RMSE and Max-Error) differ little from each other. This suggests that the simpler the algorithm, the greater the similarity of the outcomes.

The experimental results of this illustrative example can be reproduced using the Heating dataset which is pre-loaded in *CompareML*, by selecting HeatingLoad as target feature (label).

C. Application in Education

As can be seen with the illustrative case studies, *CompareML* is very easy to use and the results are provided straightforwardly. This makes it a potentially interesting tool for teaching in that it can facilitate students' interpretation of the results of applying these algorithms to a given problem with specific data. As examples of aspects in which the use of *CompareML* in teaching can generate meaningful knowledge for students, we would emphasize the following:

- One can see how certain algorithms perform better than others with certain data due to the nature of that data and the hidden knowledge it may contain. For instance, there are problems that can be solved with algorithms whose focus is on similarity, others that can be solved with tree-based algorithms which apply decisions based on the values of features, and others that can be modeled through refining.
- The effect that *feature engineering* techniques may have in transforming certain features according to different criteria can be analysed by monitoring the metrics resulting from each *CompareML* execution corresponding to each data transformation.
- One can study the impact of *feature selection* methods that apply *CompareML* to subsets of the original dataset, or, similarly, study the impact on the models' accuracy of adding new features to the dataset.

TABLE VII. Results of the Regression Case Study Experiment

		Turi Create	Scikit Learn	R
Linear Regression	**RMSE**	3.1837	3.1847	3.1778
	Max-Error	8.7520	8.8801	8.8788
Boosted Decision Tree	**RMSE**	6.2619	0.4747	2.1887
	Max-Error	13.2556	3.0332	6.3126
Decision Tree	**RMSE**	2.2976	1.6007	2.7292
	Max-Error	6.4962	4.3373	9.0515

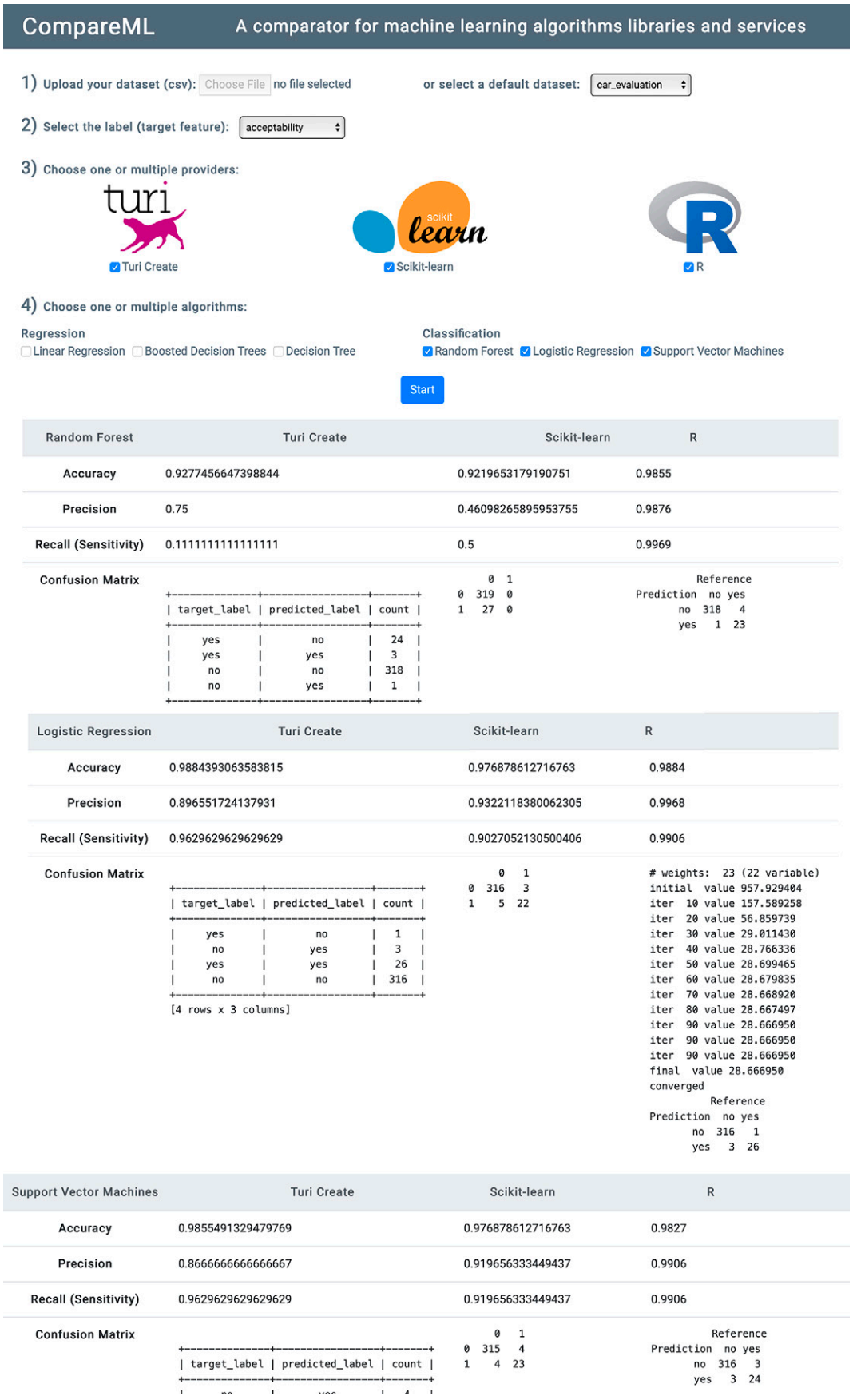

Fig. 5. *CompareML* User interface providing the output of the Classification Case Study.

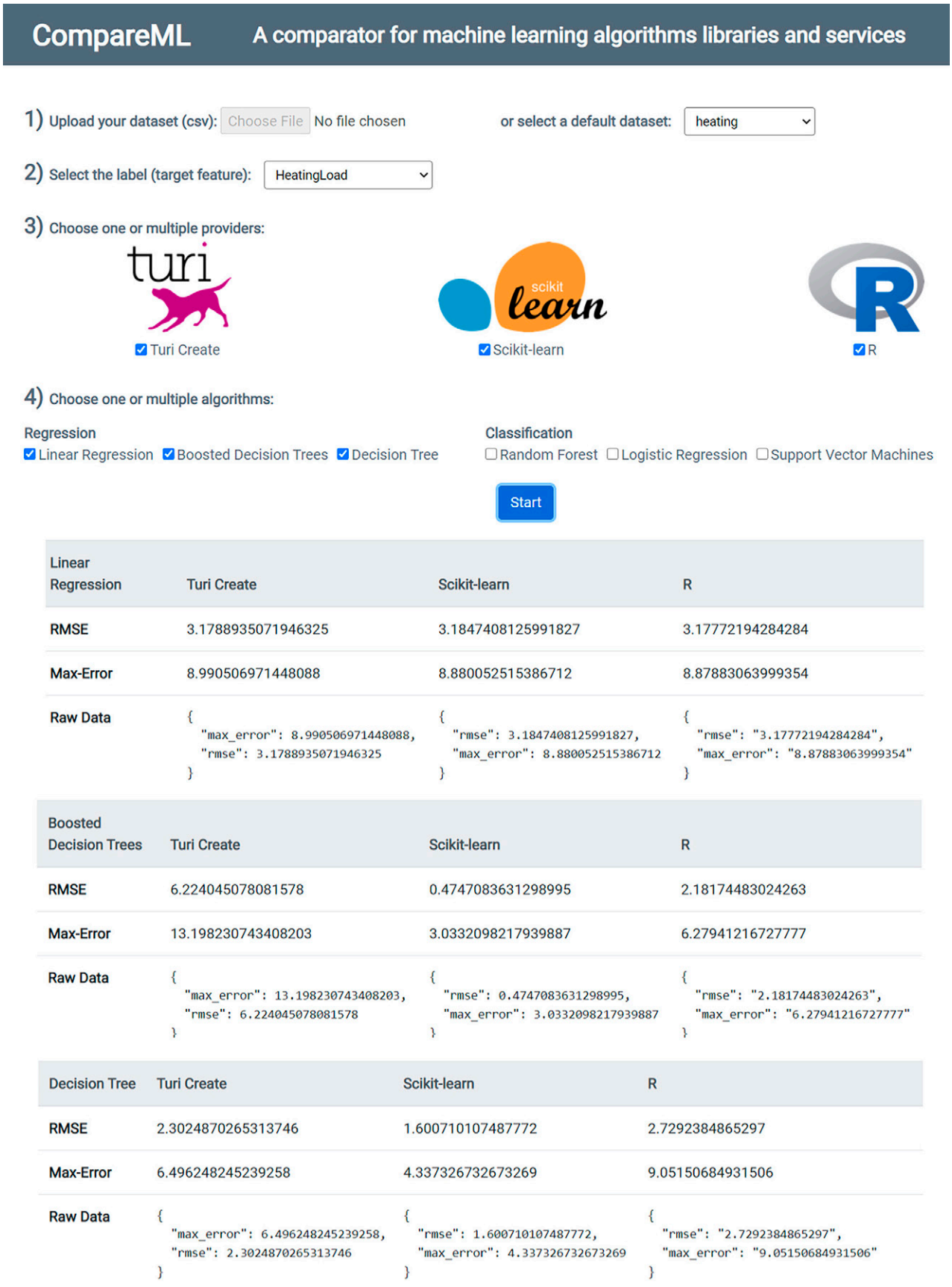

Fig. 6. *CompareML* User interface providing the output of the Regression Case Study.

TABLE VIII. Comparison With Other Approaches

Weka	**Execution environment**	Run locally
	Difficulty	Medium - low
	Algorithms	Wide range of algorithms available. A single implementation per algorithm
	Classification metrics	Accuracy, confusion matrix, precision, recall, F1-score, ROC
	Regression metrics	MAE, RMSE, correlation coefficient
PyCaret	**Execution environment**	Commonly works in a Jupyter environment that can run locally or in the cloud
	Difficulty	Low
	Algorithms	Wide range of algorithms available. A single implementation per algorithm
	Classification metrics	Accuracy, AUC, precision, recall, F1-score, kappa, MCC
	Regression metrics	MAE, MSE, RMSE, R^2, RMSLE, MAPE
PowerBI	**Execution environment**	Runs in the cloud
	Difficulty	Medium. ML concepts are easy to follow, but Power Platform knowledge is required
	Algorithms	Wide range of algorithms available. A single implementation per algorithm
	Classification metrics	AUC, confusion matrix, precision, recall, cost-benefit analysis
	Regression metrics	Model performance explanation, Average Residual Error
CompareML	**Execution environment**	Runs in the cloud
	Difficulty	Low
	Algorithms	Three each for classification and regression. Three implementations per algorithm
	Classification metrics	Accuracy, confusion matrix, precision, recall
	Regression metrics	RMSE, R^2, Max-Error

D. Comparison With Other Approaches

In this subsection, a comparative evaluation is made of *CompareML* with the other approaches discussed in the Related Works section, especially those that, considering their specific features, admit a direct comparison with *CompareML*. These are Weka, PyCaret, and PowerBI AutoML.

Weka needs to be downloaded and installed to run locally. It offers a large number of functionalities. The user interface is built in Java, but, even though it is not hard to understand and use, it is not up-to-date, which is an obstacle to its use. There are a large number of algorithms to choose from, all of them with a single implementation. Experiments provide the key metrics with which to evaluate the models, and it has a CLI interface.

PyCaret is a machine learning library in Python commonly used in the Jupyter Notebooks Environment. It is simple and easy to use due to its low-code orientation. A large number of algorithms are automatically selected to perform experiments, and it is easy to tune the hyperparameters if necessary. The main classification and regression metrics are generated directly after training the models with just one line of code.

PowerBI incorporates the creation of machine learning models with a focus on their explicability. A little background working with PowerBI dataflows is required, but the AutoML process is straightforward and is finely integrated with the PowerBI online app. Data processing is straightforward in the environment, and *Feature Selection* techniques are applied before the models are trained. The main classification and regression metrics are attractively presented in the PowerBI reports.

CompareML differs from the other approaches in that no configuration is required, and it can be used directly. Also, the fact that more than one implementation is available for each algorithm constitutes its principal differentiating element. In particular, since there are three implementations for each algorithm, this allows users to analyse the differences between those implementations.

Table VIII summarizes this comparison in terms of the type of execution environment, of the difficulty in setting up and using the tool, of the metrics that are provided for an evaluation of regression and classification models, and of the number and type of algorithms supported.

VI. Conclusions

In this paper, we have presented a novel approach to supporting preliminary data analysis in the engineering field that enables engineers and researchers to quickly and easily analyse the potential for inferring knowledge that may lie hidden in their data. Similarly, it assists them in comparing machine learning models using different implementations from different providers of well-known algorithms, without their needing prior knowledge about how to create those models with each provider.

To that end, the most widely used machine learning libraries and services and some of the best-known classification and regression machine learning algorithms can be compared by performing a series of experiments. After the experiments have been completed and the models created, the commonest evaluation metrics are presented so that researchers and engineers can properly evaluate and compare the possibilities they have available, giving them all the information they need to make a decision on how to proceed with their work.

As far as we know, there has been no previous intelligent programming environment with the characteristics and objectives of *CompareML*, i.e., using machine learning techniques to construct software with a modular and scalable architecture, and aimed at providing practising engineers with a decision support system that can help them solve hitherto intractable problems by eliciting knowledge from their data, even though they have no in-depth machine learning skills.

From the experiments carried out with *CompareML*, affirmative answers can be given to the three research questions posited. With regard to RQ1, it is possible to analyse the potential for inferring knowledge hidden in a dataset obtained from a real-world engineering application. With regard to RQ2 and RQ3, it is possible to obtain, a priori, the best combination of algorithm and provider with which to construct a predictive model for that specific engineering application. These affirmative answers have their origin in the following contributions of the *CompareML* approach:

- It allows engineers and researchers who have no extensive prior skills in machine learning to generate their own models with which to evaluate the potential knowledge that can be inferred from their data.
- It finds the machine learning algorithm that would build the most appropriate model for a specific problem involving some specific data.
- It determines the best framework, tools, and providers for addressing a specific problem involving some specific data.

In future work, it will be interesting to improve the approach with the following practices:

- Increase the number of libraries and services supported, as well as the number of regression and classification algorithms.
- Support algorithms to create clustering or recommender system models.
- Implement a microservices-based architecture that allows the functionalities of the machine learning providers to be isolated in those microservices [30], thus making it easier to update the algorithms, maintain the code, and add or delete providers.
- Implement new classification and regression model evaluation metrics.

Acknowledgments

This work was developed with the support of (i) Ministerio de Ciencia, Innovación y Universidades (MCIU), Agencia Estatal de Investigación (AEI), and European Regional Development Fund (ERDF): project RTI2018-098652-B-I00, and (ii) European Regional Development Fund (ERDF) and Junta de Extremadura: projects IB16055, IB18034, and GR18112.

References

[1] I. H. Witten, E. Frank, M. A. Hall, "Introduction to weka," in *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, Boston: Morgan Kaufmann, 2011, pp. 403 – 406, third edition ed., doi: https://doi.org/10.1016/B978-0-12-374856-0.00010-9.

[2] S. Lang, F. Bravo-Marquez, C. Beckham, M. Hall, E. Frank, "Wekadeeplearning4j: A deep learning package for weka based on deeplearning4j," *Knowledge-Based Systems*, vol. 178, pp. 48 – 50, 2019, doi: https://doi.org/10.1016/j.knosys.2019.04.013.

[3] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, B. Zupan, "Orange: Data mining toolbox in python," *Journal of Machine Learning Research*, vol. 14, pp. 2349–2353, 2013.

[4] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, B. Wiswedel, "Knime - the konstanz information miner: Version 2.0 and beyond," *SIGKDD Explor. Newsl.*, vol. 11, p. 26–31, Nov. 2009, doi: 10.1145/1656274.1656280.

[5] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler, "Yale: Rapid prototyping for complex data mining tasks," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, New York, NY, USA, 2006, p. 935–940, Association for Computing Machinery.

[6] A. Jovic, K. Brkic, N. Bogunovic, "An overview of free software tools for general data mining," in *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2014, pp. 1112–1117.

[7] X. He, K. Zhao, X. Chu, "Automl: A survey of the state-of- the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021, doi: https://doi.org/10.1016/j.knosys.2020.106622.

[8] H. Song, P. Flach, "Efficient and robust model benchmarks with item response theory and adaptive testing," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, pp. 110–118, 2021, doi: https://doi.org/10.9781/ijimai.2021.02.009.

[9] Microsoft, "Powerbi automated machine learning." https://docs.microsoft.com/en-us/power-bi/transform-model/dataflows/dataflows-machine-learning-integration. Online; last accessed 2 April 2021.

[10] M. Ali, *PyCaret: An open source, low-code machine learning library in Python*, July 2020. PyCaret version 2.3.

[11] Google, "Cloud automl." https://cloud.google.com/automl. Online; last accessed 2 April 2021.

[12] H. Robles-Berumen, A. Zafra, H. M. Fardoun, S. Ventura, "Leac: An efficient library for clustering with evolutionary algorithms," *Knowledge-Based Systems*, vol. 179, pp. 117 – 119, 2019, doi: https://doi.org/10.1016/j.knosys.2019.05.008.

[13] D. Charte, F. Herrera, F. Charte, "Ruta: Implementations of neural autoencoders in r," *Knowledge-Based Systems*, vol. 174, pp. 4 – 8, 2019, doi: https://doi.org/10.1016/j.knosys.2019.01.014.

[14] E. Real, C. Liang, D. R. So, Q. V. Le, "Automl-zero: Evolving machine learning algorithms from scratch," 2020.

[15] C. M. University, "Turi graphlab create." https://turi.com/. Online; last accessed 2 April 2021.

[16] G. van Rossum, the Python Software Foundation, "Python programming language." https://www.python.org/. Online; last accessed 2 April 2021.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[19] A. Gupta, K. Ghanshala, R. C. Joshi, "Machine learning classifier approach with gaussian process, ensemble boosted trees, svm, and linear regression for 5g signal coverage mapping," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, pp. 156–163, 2021, doi: https://doi.org/10.9781/ijimai.2021.03.004.

[20] A. J. Fernández-García, L. Iribarne, A. Corral, J. Criado, J. Z. Wang, "A recommender system for component- based applications using machine learning techniques," *Knowledge-Based Systems*, vol. 164, pp. 68–84, 2019, doi: https://doi.org/10.1016/j.knosys.2018.10.019.

[21] A. J. Fernández-García, R. Rodríguez-Echeverría, J. C. Preciado, J. M. C. Manzano, F. Sánchez-Figueroa, "Creating a recommender system to support higher education students in the subject enrollment decision," *IEEE Access*, vol. 8, pp. 189069–189088, 2020, doi: 10.1109/ACCESS.2020.3031572.

[22] T. H.-Y. Chiu, C. Wu, R. C. C.-H. Chen, "A generalized wine quality prediction framework by evolutionary algorithms," *International Journal of Interactive Multimedia and Artificial Intelligence*, doi: https://doi.org/10.9781/ijimai.2021.04.006.

[23] K. M. Ting, *Confusion Matrix*, pp. 260–260. Boston, MA: Springer US, 2017.

[24] A. Leff, J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, Sep. 2001, pp. 118–127.

[25] W. McKinney, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, vol. 14, 2011.

[26] S. Hellegouarch, *CherryPy Essentials: Rapid Python Web Application Development Design, Develop, Test, and Deploy Your Python Web Applications Easily*. Packt Publishing, 2007.

[27] M. Bohanec, V. Rajkovič, "Knowledge acquisition and explanation for multi-attribute decision," in *8th International Workshop Expert Systems and Their Applications*, 1988.

[28] D. Dua, C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[29] A. Tsanas, A. Xifara, "Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools," *Energy and Buildings*, vol. 49, pp. 560 – 567, 2012, doi: https://doi.org/10.1016/j.enbuild.2012.03.003.

[30] J. Lewis, M. Fowler, "Microservices: a definition of this new architectural term." http://martinfowler.com/articles/microservices.html, 2014.

Antonio Jesús Fernández-García

He received his PhD in Computer Science from the Universidad de Almería in 2019. He is a Researcher and Member of the Applied Computing Group at the University of Almería (UAL) and the Quercus Software Engineering Group at the University of Extremadura (UEX). He has published more than 15 scientific publications in journals and international conferences, and has participated in more than 6 research projects. His research areas include Recommender Systems, Machine Learning, Artificial Intelligence, Data Mining, Data Engineering, and Software Engineering. At present, he is an Associate Professor in the Escuela Superior de Ingeniería y Tecnología at the Universidad Internacional de la Rioja (UNIR).

Juan Carlos Preciado

He is a Professor and member of the Quercus Software Engineering Group in the Department of Computer Science at the University of Extremadura (UEX). He was vice-rector of that university for several years. His research areas include Model-Driven Development, Web and Data Engineering, in which he has published around 100 papers in the software engineering field. He received a PhD in computer science from UEX in 2008.

Alvaro E. Prieto

He is an Assistant Professor of Computer Languages and Systems at the University of Extremadura (Spain). He is a member of the Quercus Software Engineering Group. He received his BSc in Computer Science from the University of Extremadura in 2000 and a PhD in Computer Science in 2013. His research interests include Linked Open Data, Predictive Analytics, and Business Intelligence. He is currently involved in various R&D&I projects.

Fernando Sánchez-Figueroa

He is a Professor in the Department of Computer Science at UEX. His research focuses on Web engineering, big data visualization, and MDD. He holds a PhD in Computer Science from UEX and is co-author of more than 100 publications related to software engineering.

Juan D. Gutiérrez

He is a Computer Engineer for the University of Extremadura. He combines a research staff job at this university with working towards a PhD in visible LED-based light indoor positioning systems (IPS). His skills include different programming languages, system administration, application design, databases, and the Internet. He has also written more than twenty computer science books and translated another ten books from English to Spanish.