Facilitating and Managing Machine Learning and Data Analysis Tasks in Big Data Environments Using Web and Microservice Technologies

Shadi Shahoud(✉), Sonja Gunnarsdottir, Hatem Khalloof, Clemens Duepmeier, and Veit Hagenmeyer

Institute for Automation and Applied Informatics (IAI), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{shadi.shahoud,hatem.khalloof,clemens.duepmeier,veit.hagenmeyer}@kit.edu, sonjabara@gmail.com

Abstract. Driven by the current advances of machine learning in a wide range of application areas, the need for developing easy to use frameworks for instrumenting machine learning effectively for non data analytics experts as well as novices increased dramatically. Furthermore, building machine learning models in the context of Big Data environments still represents a great challenge. In the present article, those challenges are addressed by introducing a new generic framework for efficiently facilitating the training, testing, managing, storing and retrieving of machine learning models in the context of Big Data. The framework makes use of a powerful Big Data software stack platform, web technologies and a microservice architecture for a fully manageable and highly scalable solution. A highly configurable user interface hiding platform details from the user is introduced giving the user the ability to easily train, test and manage machine learning models. Moreover, the framework automatically indexes and characterizes models and allows flexible exploration of them in the visual interface. The performance and usability of the new framework is evaluated on state-of-the-arts machine learning algorithms: it is shown that executing, storing and retrieving machine learning models via the framework results in a well acceptable low overhead demonstrating that the framework can provide an efficient approach for facilitating machine learning in Big Data environments. It is also evaluated, how configuration options (e.g. caching of RDDs in Apache Spark) affect runtime performance. Furthermore, the evaluation provides indicators for when the utilization of distributed computing (i.e. parallel computation) based on Apache Spark on a cluster outperforms single computer execution of a machine learning model.

Keywords: Microservice · Web-based applications · Big Data · Data analytic · Machine Learning

A. Hameurlain et al. (Eds.) TLDKS XLV, LNCS 12390, pp. 132–171, 2020.
https://doi.org/10.1007/978-3-662-62308-4_6

1 Introduction

Data mining is the extraction of implicit, unknown and potentially useful information from data [35]. To this aim, Machine Learning (ML) provides the technical basis including algorithms, metrics and technologies. It is the process of taking an algorithm specification, providing training data and using a training procedure to learn model parameters that optimally fit the training data. The success of ML in many application areas such as text classification [4], speech recognition [5], medical diagnostics [6], energy generation forecasting [7] and load forecasting [8], to name a few, paved the road for more in-depth research on new methodologies as well as an even-growing demand for ready-to-go ML software solutions.

Although ML can be used for solving many complex business problems, there are also some downsides. Applying ML is usually a time-consuming process for the user, in which a lot of hyperparameters need to be configured to achieve the best performance in a so called trial-and-error approach. Such approach is based on the idea that all possible combinations of learning algorithms with their relevant parameters will be tried for each task until a good solution is found. However, this is typically inextricable. It wastes the resources for constructing multiple models which can take a long time especially in the case of large datasets to be forecasted.

Consequently and due to the rapid increase of data, more intelligent solutions utilizing Big Data platforms are becoming one of the hottest topics related to ML [9], where a distributed execution environment is required for the computation of larger datasets. Gaining insightful information, finding patterns and extracting knowledge from big datasets are quite complex tasks. Additionally, the configurations of the underlying Big Data infrastructure introduce more complexity for configuring and running ML tasks. This process consists of multiple steps and is commonly called Machine Learning Pipeline (MLP). Figure 1 shows a simplified MLP encompassing data preprocessing, splitting the data into training and test data, model training and model testing.

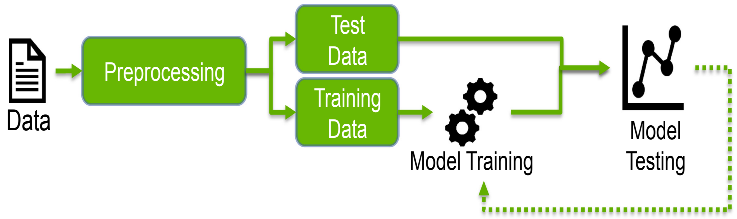


Fig. 1. Simplified methodology of Machine Learning Pipeline (MLP).

The aforementioned challenges are addressed in developing a new microservice-based solution by Shahoud et al. in [34]. They developed a new conceptual framework helping users to solve ML problems in Big Data environments without caring too much about technical issues of the underlying Big Data and cluster computing environment as runtime platform. The goal of this framework is to facilitate training, testing, managing, storing and retrieving ML models in the context of Big Data by using an easy to use web interface which hides the complexity of the underlying runtime environment from the user. For efficient scalable processing, the framework employs a Big Data cluster, a microservices architecture and modern web technologies like REST, React and Spring Boot. As a first exemplary application, smart grid applications are addressed in the evaluation. The proposed framework is able to perform ML tasks on energy time series datasets using a variety of algorithms on different types and size of such data.

In context of ML, the users can be categorized into two main categories, namely expert and non-expert ones as shown in Table 1. On one hand, the expert users have a deep understanding of ML and good programming skills to implement ML models using, for example, some developing tools like Jupiter Notebook[1]. They have worked with ML libraries before and are capable of programming algorithms themselves. On the other hand, non-expert users are grouped into two sub-categories. The first one includes the users who are familiar with statistics and ML but are not able to write the necessary script for training and evaluating ML models particularly in Big Data environments. This sub-category of users will be mainly supported by the current framework presented in this article. The second sub-category of non-expert users is inexperienced and not knowledgeable about statistics and ML. They need to have some analysis results using ML, but they only have the data and seems to be difficult for them to write or build ML models because they also do not have the required ML programming skills.

Table 1. User categories.

Category Nr.	User category		Properties
1	Expert		ML knowledge (+) ML Programming skills (+)
2	Non-expert	**A**	ML knowledge (+) ML Programming skills (−)
		B	ML knowledge (−) ML Programming skills (−)

For the evaluation of the basic concepts of the framework, a first implementation is developed which utilizes Apache Spark as runtime environment for ML on a Big Data cluster and spark.ml as a ML library [18]. The storage layer of the framework utilizes the Hadoop Distributed File System (HDFS) [10] and a PostgreSQL database [17] for storing the required input and the resulting output

[1] https://jupyter.org/.

data. To facilitate the building, training and running of ML models, an easy-to-use web User Interface (UI) which assists non-expert users in performing these tasks is conceptualized and implemented in the current version. The UI utilizes microservices [11] running on the Big Data cluster as background services to hide the complexities of the runtime environment from the user and interfacing to the ML software on the cluster in such a way that it will allow plugging in different ML runtime environments - beside Spark - in the future.

This article is a major extension of [34], with more details on the conceptual microservice-based architecture, related work and fundamental terms as well as technologies. It also provides additional results of a performance evaluation of the framework which are not presented in [34]. I.e., the effect of caching RDDs in Apache Spark is investigated by comparing the execution time of training and testing our benchmark evaluation models, namely Multiple Linear Regression (MLR), Decision Tree (DT), Random Forest (RF) and Gradient Boosted Trees (GBTs) models in case of caching and without caching input time series datasets. Moreover, the execution time and framework overhead are measured for evaluating the efficiency of the framework, highlighting the advantage of storing and retrieving ML models. It is also evaluated at what dataset sizes the calculation of ML models on a computing cluster outperforms calculations on single machines. To this end, the points are defined, referred as thresholds, at which a distributed computing framework based on, e.g., Apache Spark becomes necessary. This is done by comparing the total time required for training and testing different data-driven forecasting models on a computing cluster (using Apache Spark) to the time needed on a single computer for performing the same task.

The remainder of this article is organized as follows. In the next section, state-of-the-art frameworks related to our framework are presented. In Sect. 3, the fundamental terms and technologies used in the presented work are explained. In Sect. 4, the architecture of the proposed microservice-based framework is introduced. Section 5 presents the experimental evaluation of the framework and discusses the obtained results. The last section draws some conclusions and outlines future work.

2 Related Work

ML offers a variety of powerful algorithms and approaches for modeling and decision making from data, but implementing a ML model by yourself is a complex, long lasting and error prone process [12]. To ease the usage of ML, the ML community has developed a variety of powerful frameworks and tools to make its techniques more accessible to end users. Such frameworks and tools can be categorized into data analytic and ML workflow management frameworks.

Frameworks like Apache Spark which is a data analytic framework containing a good library for more traditional ML algorithms, or TensorFlow dedicated to Deep Learning, are low level frameworks that help data scientists in programming ML algorithms which could then be executed on a local computer or even for better performance on a computing cluster. Such frameworks typically don't provide easy-to-use user interfaces for non-experts by themselves but there are additional (Open Source) tools (e.g. Jupiter Notebook) which provide lean web user interfaces to such frameworks for hiding the details of the background cluster runtime environment from the user. Typically, these interfaces are aimed towards a more experienced data scientist programmer and less towards non-expert users who just want to apply ML algorithms. Apache PredictionIO [20] is an open source ML framework for developers. Besides supporting the deployment of ML algorithms, Apache PredictionIO allows expert users to train and test ML models and query results via RESTful APIs. It is built on top of state-of-the-art scalable open source services, e.g. Hadoop, HBase, Elasticsearch and Apache Spark. The drawback here is the non-existence of UI to facilitate performing ML tasks for non-expert users.

Contrary to the data analytics tools aimed for the experienced ML programmers, there are nice User Interface (UI)-based tools targeted to non-experts. Johanson et al. in [13] developed OceanTEA, a framework to analyze time series datasets in climate context. OceanTEA leverages web technology such as microservices and a nice web UI to interactively visualize and analyze time series datasets. It is a cloud-based software platform, consisting of a microservice back-end and a web UI, similar to the framework implemented in this article. Both components communicate with each other through an API gateway utilizing REST and each microservice is deployed independently through a Docker. OceanTEA provides four main UI interfaces for the exploration and analysis of oceanographic times series data including functionalities of time series data management, data exploration, spatial analysis and temporal pattern discovery.

Another project focused on the acceleration of research in energy data analysis is WattDepot presented by Brewer and Johnson in [14]. The software platform is an open source and internet-based one. It supports the collection, storage, analysis and the visualization of data coming from energy meters. The architecture encompasses three types of services, namely sensors, servers and clients. The sensors collect the data from different energy meters and send it to the services which store the incoming data by utilizing the provided RESTful APIs. Since the services are not coupled to a specific database, flexible data storage is provided. For analysis and visualization, the clients request the data from the services in the format XML, JSON or CSV. The applications of WattDepot include a web application for a dorm energy competition and a power grid simulation mechanism.

However, both WattDepot and OceanTEA typically are not generic. They contain dedicated ML based analysis features which are specialized towards the special application domain and therefore e.g. performing ML tasks such as forecasting as needed in the energy application field are not included in them.

Shrestha et al. in [15] developed a user friendly web application to analyze health and education datasets. This tool also includes ML algorithms for the forecasting of time series data. The application also has a nice and easy-to-use user interface that was developed using human-computer interaction design guidelines and principles and targeted at novice and intermediate users. The technologies used were Java, the Play framework and Bootstrap. But only linear regression, logistic regression and back propagation were utilized to perform forecasting on the input datasets. However, this framework is not able to solve ML tasks in the context of Big Data and can only be used as standalone application on a desktop computer.

ML workflow management is a rich area of research that has produced systems to manage the process of building ML models. The process of building a satisfactory ML model by a data scientist is characterized as an iterative trial-and-error procedure, where in each iteration the user reveals essential insights into the effectiveness of algorithms' configurations. Since the models may become numerous, it is important to keep track of the relevant information so a model's performance can easily be analyzed. This leads to the problem of model management which encompasses the storage and retrieval of the models and related metadata (e.g. hyperparameters, evaluation performance, etc.) in order to analyse them collectively [12].

Multiple recent research projects have been introduced addressing the model management as a part of the ML workflow. Vartak et al. in [12] introduced ModelDB, a system for tracking and versioning ML models in form of pipelines. The authors argued that data scientists are reluctant in using other environments than their favored ones, especially those with a GUI and therefore they provide native client libraries for scikit-learn and Spark MLlib which can be used to track and store models, operators and related metadata. The framework consists of a front-end and a back-end encompassing a relational database and custom storage engine. The front-end is implemented as a web UI and supports the review, inspection and comparison of the tracked and indexed models and pipelines through a Tableau-based interface. In addition, the information can be explored and analysed through SQL. The limitations here are that ModelDB is developed as a monolithic application making it difficult to be maintained and further developed. Moreover, ModelDB did not provide the ability to handle problems in the context of Big Data.

To manage ML models and their lifecycle, MLflow is introduced in [19]. Expert users can develop and track ML experiments, share and deploy ML models. MLflow is developed as an open source software system addressing typical problems of the ML workflow particularly experimentation, reproducibility and deployment. It is integrated with Python, Java and R, and provides REST APIs encompassing three main elements. The first one, MLflow Tracking, offers APIs for logging experiments and supports querying the results through APIs as well as visualizing them with a web UI. The second component, MLflow Projects, can be used to create reusable software environments for reproducibility and is defined through YAML files. The last item, MLflow Models, provides the functionality to package ML models in a generic format and deploy them. Those models incorporate similarly to MLflow Projects a YAML file which contains the metadata of the model.

To address the issue of model deployment, a variety of frameworks and tools are developed. Tensorflow serving [21] provides a flexible and powerful system for serving tensorflow models. It allows expert users to achieve an efficient integration of tensorflow models in the production environments. Kubeflow [22] is a cloud platform for ML built on top of Google's internal ML pipelines. It provides expert users with a lot of functionalities including notebooks for training and serving tensorflow models. H2O Flow [23] is another efficient framework for creating and managing ML and Deep Learning workflows including training and testing models. This framework supports Python, R and scala on top of Hadoop/Yarn and Apache Spark.

Table 2 introduces a brief comparison between the aforementioned ML frameworks based on some criteria to precisely highlight the originality of the solution proposed in the present article. In this table, data analytic frameworks are refereed as 1 and ML workflow management frameworks are referred as 2. (2.A) refers to the first sub-category of non-expert users presented in Table 1.

Table 2. Data analytic and ML workflow management frameworks.

Framework	Framework category	Web UI	Microservice architecture	Support Big Data	Support non-expert	Generic
Apache Spark	1	–	–	+	–	+
Tensorflow	1	–	–	+	–	+
Apache PredictionIO	1	–	+	+	–	+
Jupiter Notebook	1	–	+	+	–	+
OceanTea	1	+	+	–	+(2.A)	–
WattDepot	1	+	+	–	+(2.A)	–
ModelDB	2	+	–	–	+(2.A)	+
MLflow	2	+	–	–	–	+
Tensorflow serving	2	+	+	–	–	+
Kubeflow	2	–	+	+	–	+
H2Oflow	2	–	+	+	–	+
Current framework	1 + 2	+	+	+	+(2.A)	+

The framework implemented in this article uses microservice and Apache Spark, including MLlib, in addition to HDFS to provide scalability and simplicity. What differentiates the framework from the aforementioned projects, is the additional abstraction provided by the UI to support non-expert users (category A) in applying ML. Moreover, most of the aforementioned frameworks are intended and developed to mainly support expert users and do not provide an easy to use integrated framework for non-expert users. But the above tools or comparable other tools could be used as building blocks to form a more complete integrated environment such as AutoML[2] which can be seen as a competing

[2] https://www.automl.org/.

approach for automating the process of applying ML to real world problems. The aim of the framework presented in this article can be seen as a first step in the direction of such a more complete environment for even non-experts, designated in a way that will allow plugging in several Machine Learning and Deep Learning runtime environments.

3 Related Fundamental Terms and Technologies

In this section, we introduce the background knowledge necessary to understand the main contributions of this article.

3.1 Machine Learning

With the following definition, Alpaydin et al. in [1] introduced an essential description of machine learning: "Optimizing a performance criterion using example data and past experience". Machine learning, as its name implies, means the ability to make the computers capable to learn from data and use the resulting knowledge to perform further tasks without any guidance from the human side. Precisely, machine learning is a scientific discipline aiming at designing and developing specific algorithms and concepts in order to allow computers to evolve behaviors and react to different actions based on empirical data such as sensor data. Indeed, it can be seen as a core in the field of artificial intelligence, in which the computers can learn from existing data to predict the future behavior, results and trends.

Applying ML to extract useful knowledge from raw data has become increasingly popular in a variety of areas. One such field is the health sector where it helps with medical diagnosis [33,42]. Virtual voice assistance, like siri and alexa, is another example, where ML is used to take voice commands from people like setting the alarm clock or finding specific information on the internet. To ensure better sustainability and economic operation of electricity grids through intelligent decision making in unit commitment of decentralized energy resources and flexible loads at grid level, an accurate prediction of future energy demand and renewable energy generation is required. To this end, ML also takes the advantage for energy load and generation forecasting [30,31,36–40].

Machine Learning Scenarios. Four different scenarios can be distinguished in the field of machine learning, namely supervised, unsupervised, semi-supervised and reinforcement machine learning scenarios. The main distinction between the mentioned scenarios depends on the information they handle. As a result, the behavior of learning algorithms will differ accordingly.

Supervised Machine Learning. It specifies the scenario, in which the examples in the training set are labeled with a significant information called labels. Such labels are missed in the examples in the testing set and need to be predicted [45]. More abstractly, all examples in the training set are labeled explicitly. Each of them consists of attributes or predictors on one side and the corresponding output on the other side. Both predictors and their corresponding outputs could be nominal or numeric depending on the source of data. In the supervised learning settings, we can think of a teacher who provides an extra information i.e. labels to the examples in the training set to predict such information for the unlabeled examples in the testing set.

We distinguish two different problems in the supervised machine learning scenario, namely classification and regression problems. In the current article, regression problems for time series forecasting are implemented and evaluated. Some important examples concerning the supervised machine learning scenario are:

- Linear regression for regression problems.
- Decision Trees (DT), Random Forest (RF) and Gradient Boosted Trees (GBTs) for classification and regression problems.
- Support vector machines for classification and regression problems.

Unsupervised Machine Learning. In contrast to the supervised machine learning scenario, in which the examples are explicitly labeled, the examples here are unlabeled. There is no information in the training set except the training examples containing only the features without the corresponding output [46]. The unsupervised machine learning scenario tries to discover the similar characteristics between the examples and group them into meaningful clusters. Precisely, it aims at discovering and presenting a significant structure in data. Some important examples concerning unsupervised machine learning scenario are:

- k-means algorithm as a clustering algorithm.
- Apriori algorithm as an association rule learning algorithm. This algorithm can be seen as the base of recommendation systems which try to discover the behavior of customers and present the appropriate product to them consequently.

Semi-supervised Machine Learning. It can be seen as a middle point between supervised and unsupervised machine learning scenarios [47]. In this scenario, a part of data is labeled with some supervision information i.e. labels. However, semi-supervised machine learning scenario is cheaper than the supervised one based on the fact that the labeled data is more expensive than unlabeled one. It is hard to get a labeled data because the human annotation of data is expensive and needs the utilization of experts in order to label this data. Hence, semi-supervised machine learning has gained a great advantage in different application fields.

Based on the aforementioned definition of semi-supervised machine learning scenario, the usage of unlabeled data needs some assumptions on the underling distribution of data. The main assumptions of semi-supervised machine learning scenario are:

– Smoothness assumption: in this assumption, the points that are close to each other belong to the same class.
– Cluster assumption: in this assumption, the points are clustered based on the similar characteristics between them. As a result, the points that are in the same cluster belong to the same class.
– Manifold assumption: it is commonly used with high dimensional training data, in which manifolds are learned based on labeled and unlabeled data to get rid of curse of dimensionality and then the learning process is done using distance and density within each manifold.

Reinforcement Machine Learning. In this learning scenario, the model is built based on the interaction with the environment [48]. Reinforcement machine learning scenario aims at maximizing the rewards. It differs from the supervised machine learning scenario in that the input/output pairs are not presented explicitly. On-line performance evaluation is involved in the learning process. As a result, the model will react to the evaluation feedbacks aiming at increasing the rewards and achieving the best performance. Reinforcement machine learning has become more important in the recent years, as it produces the best solutions in a lot world wide applications, for instance helicopter flying, resource-constrained scheduling, robot control systems and playing backgammon.

3.2 Big Data Technologies

With the increasing amount of available data, various libraries and systems have been introduced to enable large-scale distributed/parallel processing. One of the best known open-source frameworks is Apache Hadoop4 which supports Big Data processing and storage in a distributed environment. It encompasses various components including a distributed file system, the data processing tool MapReduce and a cluster resource manager. The Hadoop Distributed File System (HDFS) enables the reliable storage of extensive files in a cluster [25]. It provides fault tolerance by splitting the files into blocks and replicating these blocks multiple times over the cluster.

Figure 2 demonstrates the architecture of HDFS which consists of a Name Node which coordinates file system operations (e.g. opening and closing files, etc.) and multiple Data Nodes which store the file blocks and serve the read and write requests [24]. Hadoop MapReduce [27] is a programming model allowing developers to write programs to process data in parallel. Its motivation is based on the complexity related to computation parallelization, data distribution and fault tolerance. The main functions of MapReduce are Map, responsible for transforming data into key/value pairs and Reduce, which accepts the output from the Map task as input and merges matching pairs.

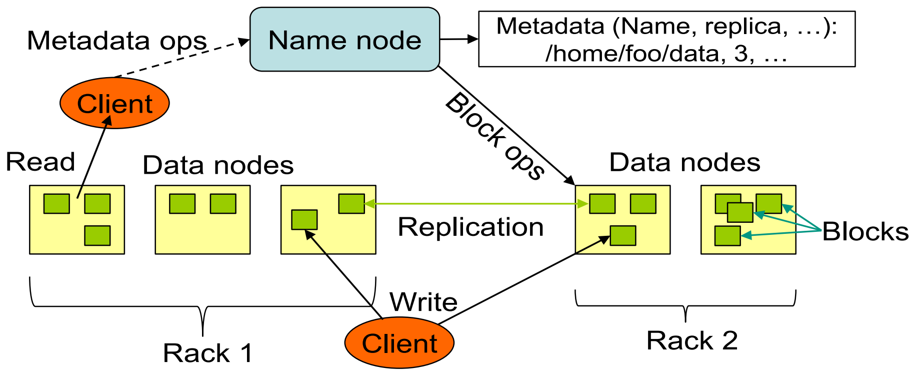


Fig. 2. HDFS architecture [24].

Yet Another Resource Negotiator (YARN) [26] is a technology that decouples the application and the required computational resources (e.g. CPUs, RAM, etc.) for processing from the resource management infrastructure of the cluster. Figure 3 illustrates YARN's architecture which is mainly composed of a Resource Manager (RM), multiple Node Managers (NM) and an Application Master (AM) for each program. When an application is submitted to the RM, the RM allocates a container accommodating the required resources for the application and contacts the related NM to launch this container. The container then executes the Application Master (AM) which coordinates the application scheduling and task execution and sends resource requests to the RM.

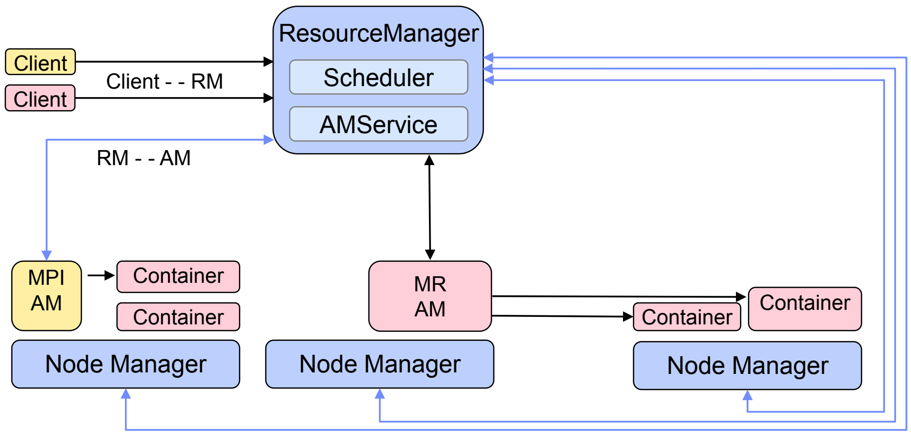


Fig. 3. YARN's architecture [26].

3.3 Microservices

Until recently, the monolithic architecture was a classic approach to implement web applications where the database, the server and the client are maintained in a single codebase. However, with the rising number of application deployments to the cloud, more and more companies like Amazon, Netflix and Zalando have shifted from a monolithic architecture to a newer more scalable architecture called Microservices. For simplicity's sake, the term service in this article refers to microservice.

Characteristics. As the name suggests, the microservices architectural style revolves around implementing an application consisting of multiple small services or entities. These services are built around the application's business functionalities, follow the Single Responsibility Principle (SRP), run in their own process and are independently deployable [32]. By following the SRP which is similar to the UNIX philosophy emphasizing programs to do one thing and doing it well, services become highly cohesive and decoupled, leading to good code maintainability. This is unlike monolithic applications which lack hard boundaries and tend to become, with added functionality, complex and tightly coupled which, in effect, leads to difficulties when changes are made since they often span multiple components.

Another distinction is that the microservices style does not require the redeployment of the whole application when new features are implemented or bugs are fixed. Instead, only the corresponding and affected service needs to be adapted and redeployed. Furthermore, microservices of a single application are not constrained to be implemented with the same set of technologies and frameworks. This allows teams working on different microservices to use independent technology stacks, as well as data storage technologies, suitable to the data they process.

Communication Types. In a microservices architecture, services are isolated from each other and distributed over a network, making communication more complicated than in monolithic applications. It is often said that microservices should have smart endpoints and dumb pipes, meaning that the logic should be inside of the services and only lightweight mechanisms and standards should be used for their communication [28]. Communication styles are usually divided into request/response and event-based techniques [29]:

- Request/response: this method describes how two services can directly communicate with each other, where one service initiates a request to another and in return expects a response.
- Event-based: this type of communication is driven by events, where one service or producer emits an event and all services that have subscribed to the event type will get an update.

REST. A common way to implement the request/response communication style is by using REST (REpresentational State Transfer), a protocol-agnostic architectural style that commonly uses HTTP as a communication protocol. All microservices implemented in this article use REST protocol to communicate between each other. This protocol enforce each service to define some RESTful APIs for transferring the data. The term REST was first coined by Fielding et al. in [2] and is made up of the following 6 constraints.

1. Client-server: to improve the portability of the client i.e. user interface and scalability of the server entities, the client and server should be separated. This constraint enables the independent involvement of both.
2. Stateless: this constraint affects the communication between the client and server and declares that it should be stateless, meaning that the client requests to the server must contain all necessary information.
3. Cache: improving the network efficiency by requiring data within a response to be labeled as cacheable or non-cacheable.
4. Uniform interface: this constraint emphasizes the importance of a uniform interface between components. To this end, the implementations are decoupled from the services they provide and the information is transferred in a standardized form rather than one which is specific to an application's needs.
5. Layered system: to simplify the complexity of an overall system, hierarchical layers should be implemented which constrict the components' behavior.
6. Code-on-demand: this is an optional constraint that allows client functionality to be extended by downloading and executing code in form of applets or scripts.

4 Concept and Architecture

In this section, the basic concepts and architecture of the proposed framework are presented. First, the general framework architecture is introduced. Then, details of the different architectural layers are presented.

4.1 Framework Architecture

Figure 4 describes the conceptual architecture of the presented framework. As seen in this figure, the architecture is layered into three main layers, namely UI layer, service layer and persistence and processing layer. The UI is split into separate sub-parts (e.g. separate web applications) providing dedicated functionalities for data and model management, model training and cluster management which are wrapped into one logical web application forming the UI of the application. The service layer is partitioned into two microservices, where each one is a small and self contained application that can be deployed independently e.g. on the runtime cluster with a single responsibility. One service focuses on data and model management, where models can be seen as special data objects.

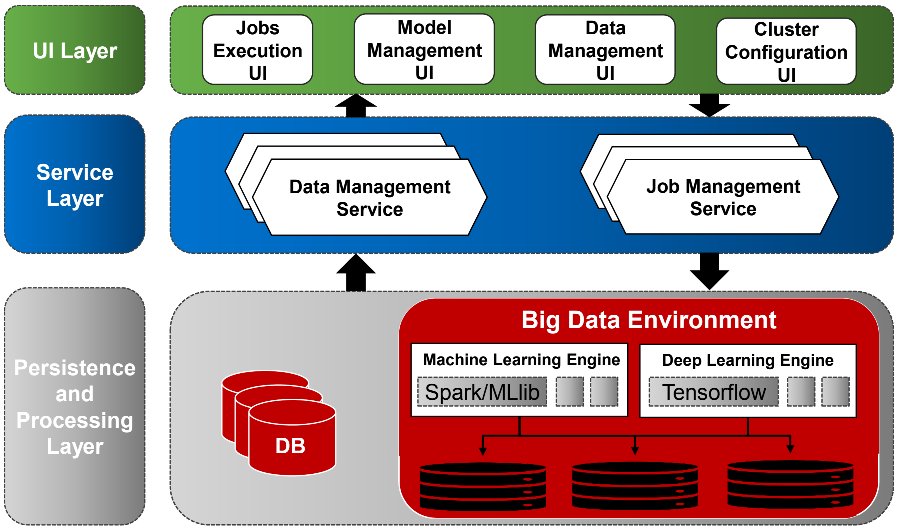


Fig. 4. Basic architecture of the proposed microservice-based framework.

The other service focuses on the management of running ML jobs e.g. for training and testing. The services provide RESTful APIs which are used by the web applications in the UI layer to interact with the runtime environment.

The persistence and processing layer provides the basic model and data storage capabilities according to the underlying runtime computer infrastructure and provides generic interfaces for executing and managing ML jobs on this infrastructure independent of the used low level ML framework. While the current implementation only supports Apache Spark as ML framework, the persistence and processing layer is designed in a way that supports plugging in additional ML frameworks in the future. In the following, the layers will be described in more details.

User Interface (UI) Layer. This layer consists of separate web applications providing dedicated functionalities which interact with the service layer via RESTful APIs. The separate web applications are wrapped into a container application which provides navigation between the views to form the complete UI. To make the user experience of the UI as pleasant as possible, the famous 10 Usability Heuristics for UI Design by Nielsen [3] are applied while conceptualizing and implementing the UI. Multiple technologies including HTML5, CSS and React[3] are utilized to implement the UI. The JavaScript (JS) library from Facebook, React, is chosen because it simplifies the development of complex user interfaces and is very permanent. Its good performance can be attributed to its use of a virtual Document Object Model (DOM) which is a copy of the HTML

[3] https://reactjs.org/.

DOM and enables efficient rendering updates of the otherwise slow HTML DOM. React is based on declarative programming and the concept of encapsulating and reusing of components. Such components are implemented through a specific syntax called JavaScript Syntax Extension (JSX) which is a combination of HTML and JS code.

To simplify the configuration of the build tools and the setup of the React application, Node Package Manager (NPM)[4] is used. For better data management and to organize the side effects related to asynchronous RESTful API calls, Redux[5] and redux-saga[6] are used. To distinguish different functions and to provide good navigability on the website, React Router is utilized. For implementing a responsive and nice web design, the popular framework React Bootstrap[7] which provides easy to use pre-styled components is utilized.

A recent trend in web development has been to develop web UIs as Single Page Applications (SPAs) [49]. Essentially, SPAs are front-end applications that consist of single HTML document that can be dynamically updated through JavaScript (JS). This makes it possible to refresh only particular regions of the screen instead of reloading the whole page when changes take place. This is especially convenient in interactive web pages, since these applications can respond much faster to user input and therefore provide better user experience. Additionally, the number of requests between the SPAs and services is often dramatically decreased, since much of the logic can be implemented in the front-end. For these reasons, the web UI will be implemented as an SPA communicating with the service tier through HTTP requests using the RESTful APIs. In the current version of the concept, the UI contains separate web applications for "data management", "model management", "execution of jobs" (e.g. for training and testing) and "cluster management". Figures 5 and 6 show some web page views related to these applications.

Data Management UI. It allows the uploading, management and configuration of data sources which provide data to ML jobs. Moreover, an interactive visualization besides statistical analysis can be performed on the datasets to achieve a better understanding of their characteristics and properties. For example and in the case of time series datasets, the user has the ability to zoom in/out and select a part of the chart for more detailed view. This allows the user to discover trends and outliers in the selected part of the time series dataset. Additionally, when the user hovers over a specific point in the chart, the related information will appear in a small box, for example the value of the power generation at this point. The interactive visualization of statistic and performance data in our framework is implemented using the HighChart Java-script library [50]. Moreover, dedicated features could be selected in the chosen dataset before performing ML tasks.

[4] https://www.npmjs.com/.
[5] https://redux.js.org/.
[6] https://redux-saga.js.org/.
[7] https://react-bootstrap.github.io/.

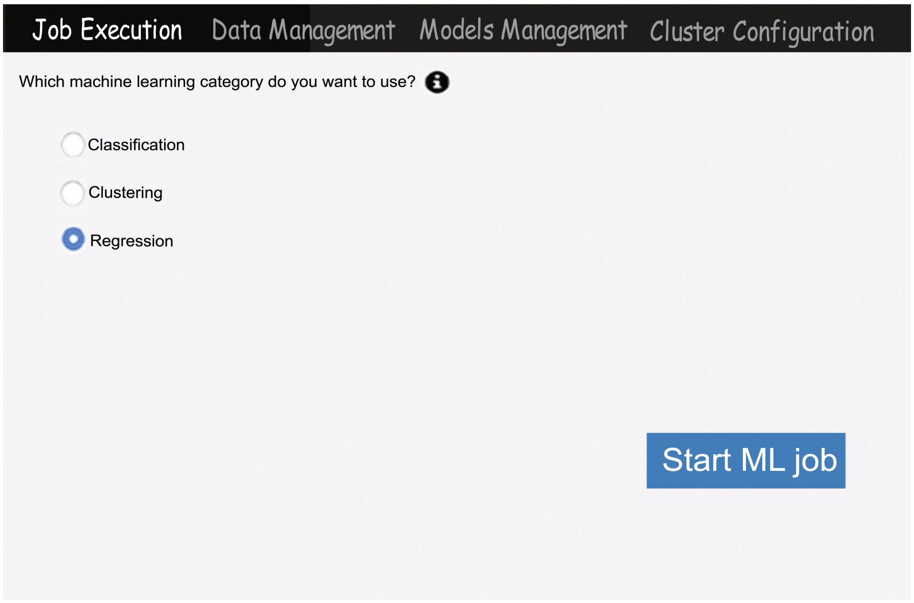
Model Management UI. Analog to data management, the model management UI allows the management of ML models which are (eventually) already pre-trained in the framework. Figure 6a shows a view of this UI which lists the available models. Each model has some associated metadata (e.g. id, creation date, model name, a textual description of what the mode does, etc.) which are shown in the tabular view. Each row (e.g. a pre-trained model) represents a ML pipeline corresponding to a specific ML task. For each task, the related general information resulting from performing this task such as ML algorithm, dataset used for training and testing, hyperparameters and performance results, to name a few, are shown if the user hovers over the model entry in the model list. The user can compare models and select the best one for executing it on a new dataset. Moreover, the user can perform actions on a selected model, namely delete a pipeline, extract the best hyperparameters, extract cluster configurations or extract the whole parameters and use them to build a new ML model.

Job Execution UI. It provides functionalities for executing a job for training and testing a ML model. To ease the usage for non-experts (non-programmers), the UI provides a wizard interface which guides the user through the process of choosing a dataset, a type of analysis to be performed on the dataset, an adequate ML model (e.g. model, either pre-trained or untrained) for performing the wanted type of analysis and afterwards for tuning the execution parameters of the model based on an already existing parameter set.

One of the main advantages of the proposed framework is to be very generic. I.e. in the step of selecting a given type of analysis to be performed on a dataset, the user should be able to select many different types of ML based analysis. But what kind of ML analysis methods and algorithms will be available is directly dependent on what kind of low level ML frameworks will be integrated on the persistence and processing layer.

Because in the present work only Apache Spark is integrated as low level ML framework and Apache Sparks standard ML library mainly provides algorithms for classification, clustering and regression, our framework currently only provides these three categories for choosing an analysis category as shown in Fig. 5a. After choosing one of these categories, the user will be navigated to the datasets tab view in order to select an already uploaded dataset or data source, or directly upload one to perform the ML task. Thereafter, the wizard navigates to the next wizard screen shown in Fig. 5b. Figure 5b shows that a ML framework can provide a variety of ML algorithms for performing a certain analysis category to cover a wide range of ML application scenarios. I.e., it can be seen in Fig. 5b that Apache Spark provides several algorithms for "regression analysis", e.g. "Linear regression", "Decision tree regression" and so on. If at a later time more than one ML framework will be incorporated into the present framework, different algorithms implementing an another analysis category can even be provided.

It can also be seen from Fig. 5b, that the user has the possibility to use an already existing pre-trained model or alternatively create and train a new ML model. Additionally, the user can adapt a given collection of algorithm hyper-

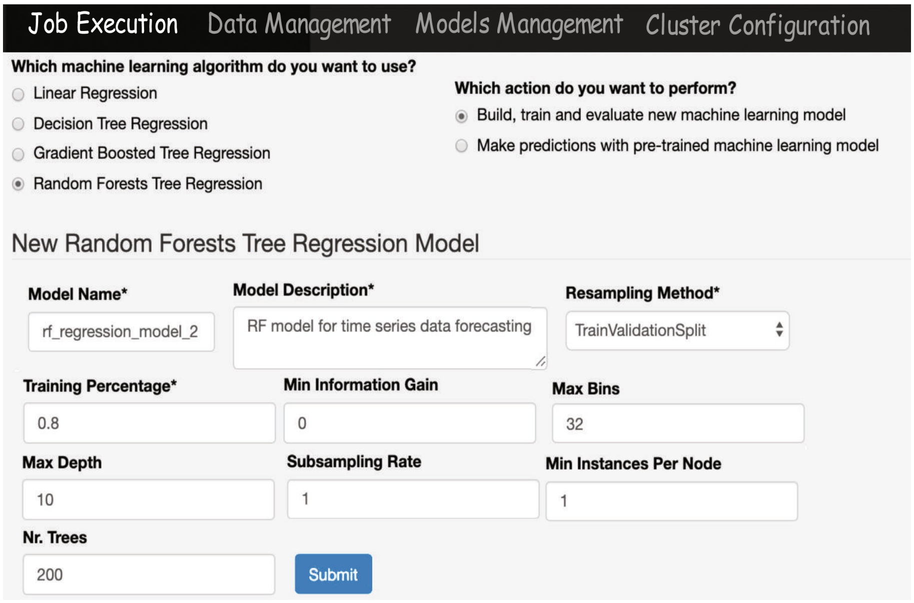

Job Execution Data Management Models Management Cluster Configuration

Which machine learning category do you want to use?

- Classification
- Clustering
- [✓ Regression]

Start ML job

(a) Job Execution UI - Choosing ML Category


Job Execution Data Management Models Management Cluster Configuration

Which machine learning algorithm do you want to use?

- Linear Regression
- Decision Tree Regression
- Gradient Boosted Tree Regression
- [✓ Random Forests Tree Regression]

Which action do you want to perform?

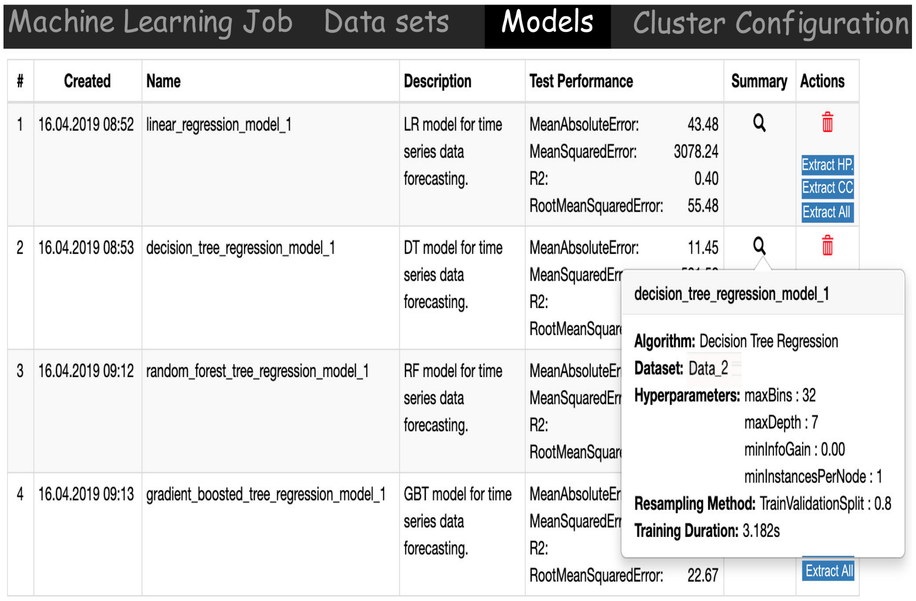
- [✓ Build, train and evaluate new machine learning model]
- Make predictions with pre-trained machine learning model

New Random Forests Tree Regression Model

Model Name*	**Model Description***	**Resampling Method***
rf_regression_model_2	RF model for time series data forecasting	TrainValidationSplit
Training Percentage*	**Min Information Gain**	**Max Bins**
0.8	0	32
Max Depth	**Subsampling Rate**	**Min Instances Per Node**
10	1	1
Nr. Trees		
200	Submit	

(b) Job Execution UI - Building ML Model

Fig. 5. User Interface (UI)

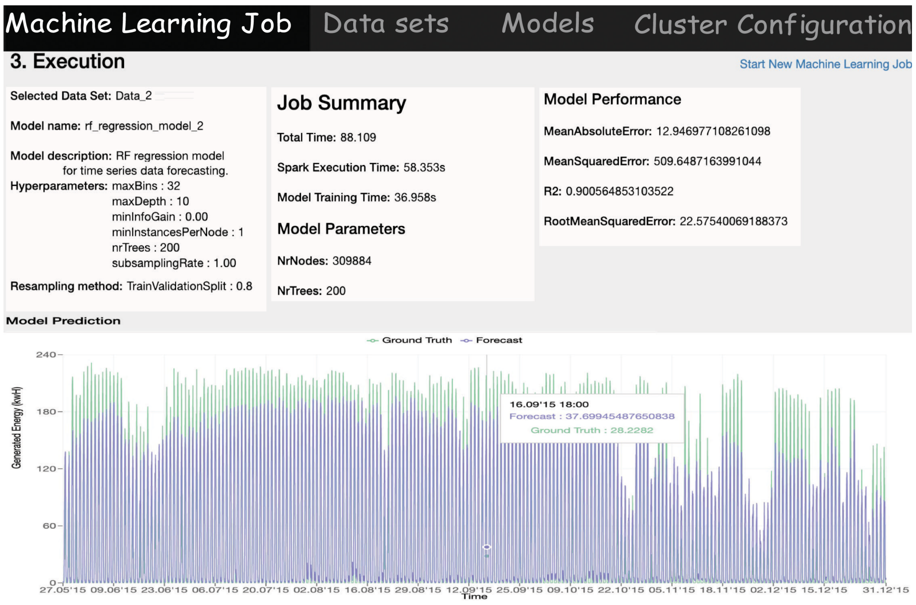


#	Created	Name	Description	Test Performance	Summary	Actions
1	16.04.2019 08:52	linear_regression_model_1	LR model for time series data forecasting.	MeanAbsoluteError: 43.48 MeanSquaredError: 3078.24 R2: 0.40 RootMeanSquaredError: 55.48	🔍	🗑 Extract HP Extract CC Extract All
2	16.04.2019 08:53	decision_tree_regression_model_1	DT model for time series data forecasting.	MeanAbsoluteError: 11.45 MeanSquaredErr R2: RootMeanSquar	🔍	🗑
3	16.04.2019 09:12	random_forest_tree_regression_model_1	RF model for time series data forecasting.	MeanAbsoluteEr MeanSquaredErr R2: RootMeanSquar		
4	16.04.2019 09:13	gradient_boosted_tree_regression_model_1	GBT model for time series data forecasting.	MeanAbsoluteEr MeanSquaredErr R2: RootMeanSquaredError: 22.67		Extract All

decision_tree_regression_model_1

Algorithm: Decision Tree Regression
Dataset: Data_2
Hyperparameters: maxBins : 32
maxDepth : 7
minInfoGain : 0.00
minInstancesPerNode : 1
Resampling Method: TrainValidationSplit : 0.8
Training Duration: 3.182s

(a) Model Management UI



3. Execution

Selected Data Set: Data_2

Model name: rf_regression_model_2

Model description: RF regression model for time series data forecasting.
Hyperparameters: maxBins : 32
maxDepth : 10
minInfoGain : 0.00
minInstancesPerNode : 1
nrTrees : 200
subsamplingRate : 1.00

Resampling method: TrainValidationSplit : 0.8

Job Summary

Total Time: 88.109

Spark Execution Time: 58.353s

Model Training Time: 36.958s

Model Parameters

NrNodes: 309884

NrTrees: 200

Model Performance

MeanAbsoluteError: 12.946977108261098

MeanSquaredError: 509.6487163991044

R2: 0.900564853103522

RootMeanSquaredError: 22.57540069188373

Model Prediction

(b) Job Execution UI - Summary

Fig. 6. User Interface (UI)

parameters for tuning the model performance. The storage and re-usability of pre-trained ML models on new datasets is another advantage of the presented framework. This eases usage and reduces the time the user needs to train and build a new model for each new dataset. After appropriate options are chosen in Fig. 5b, the ML task including learning and testing can be executed on the runtime platform. The wizard will then show a screen which allows to monitor the execution state. When the execution is done, the model and the other results of execution will be saved in the persistence and processing layer and a comprehensive visualization of results as well as an execution summary will be be shown as depicted in Fig. 6b.

Cluster Configuration UI. As mentioned in the introduction, a Big Data infrastructure as runtime environment for ML tasks can introduce great challenges for configuring and running the framework on the cluster with best performance for a given task. To tackle this challenge, the cluster configuration UI implemented in this framework gives the possibility to tune the low level execution framework configurations in relation to the usage of CPU cores, RAM usage and executors instances, to name a few.

Service Layer. This layer abstracts the interface of the UI applications to the ML runtime environment (e.g. computing cluster or single computer, etc.) by providing generic interfaces to the runtime environment via currently two microservices, namely the Job Management Service (J.M.-Service) and the Data Management Service (D.M.-Service) as shown in Fig. 4. Each microservice has dedicated responsibilities and contains a layered architecture based on the Separation of Concerns design principle (SoC). Keeping the code in distinct layers enforces a logical encapsulation of functionalities and dependencies leading to better code maintainability and loose coupling. Figure 7 depicts this architecture, where only upper layers are allowed to access lower layers.

The uppermost layer is the presentation layer which handles HTTP requests and is the entry point of the microservices. It contains controllers which map HTTP URLs and provide Create, Read, Update and Delete (CRUD) functionality to the outside through RESTful APIs. For simple read requests, the layer accesses the persistence layer to acquire the relevant data from the database. However, for complex logic, it communicates with the service layer which contains the business logic. This has the advantage that common operations required by multiple controllers can be abstracted to the service layer. The persistence (i.e. data access) layer consists of repositories and entities. The repositories interact with the underlying data source i.e. database and manage the entities which encapsulate the domain objects.

The following two sections provide a comprehensive description of both microservices, which are called as services for a simplicity's sake. The established RESTful pattern is chosen as the communication tool instead of the event-driven pattern, because the microservices are just two in total and the RESTful communication is easier to implement. In addition, the JSON format is selected for

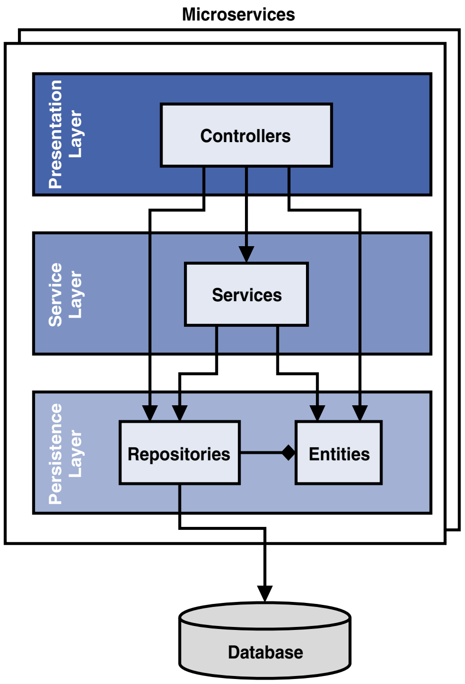


Fig. 7. Layered architecture of microservices.

requesting and sending data via the RESTful APIs because of its popularity, ease of use and interpretability.

J.M.-Service. This service is responsible for the creation and submission of jobs to be executed by an available low level ML execution framework (e.g. Apache Spark) on the available runtime environment (e.g. a cluster or single computer). Therefore, it interfaces with the persistence and processing layer below which encapsulates the specification of a certain runtime environment.

The J.M.-Service not only allows to execute ML tasks but also tracks and monitors the status of the running tasks. Moreover, it reads the execution results stored by the executing framework somewhere in the runtime environment (e.g. in an execution directory of the task on e.g. a file system) and sends them to the D.M.-Service for storage in a database, so that the execution statistics and results can be later visualized in the UI. The J.M.-Service provides an abstract job execution and monitoring interface to the web application UI through its RESTful APIs. This completely decouples the UI from the specification of the runtime environment. The main functionalities of J.M.-Service REST-APIs are described by the following URL patters:

1. **/jobs:** a GET request on this URL is used a list of spark jobs.
2. **/jobs:** a POST request on this URL is used to create of a spark job and its corresponding processing directory in HDFS.
3. **/jobs/id:** a GET request on this URL is used to retrieve a spark job for a specific id.
4. **/jobs/id:** a DELETE request on this URL is used to delete a spark job for a specific id with its corresponding processing directory in HDFS.
5. **/jobConfigurations:** a GET request on this URL is used to show a list of spark configurations.
6. **/jobConfigurations:** a POST request on this URL is used to create spark configuration.
7. **/jobConfigurations/id:** a DELETE request on this URL is used to delete a specific spark configurations for specific id.
8. **/jobSetup:** a POST request on this URL is used to copy the packaged jars and pre-trained saved machine learning models into HDFS.
9. **/submitJob/id:** a POST request on this URL is used to submit a spark job.

D.M.-Service. This service is responsible for the storage and preparation of required inputs to execute a job on the runtime environment, namely storing and providing datasets, models containing (pre-trained) algorithms and hyper-parameters, to name a few. The D.M.-Service uses its own database to store the required data as well as all results produced from performing ML tasks. On the one hand, the UI applications interact with this service to upload, manage and retrieve data, model information as well as configurations. Also the J.M.-Service interacts with the D.M.-Service to retrieve information about datasets, models and configurations, copy models from the database to the execution environment of a task and to push result information back to the D.M.-Service. The D.M-Service then stores all information about the execution of a task and the results in its own database, so that these information can be later used for the visualization of the results and the overall performance of the ML jobs as already shown in Fig. 6a.

The main functionalities of the D.M.-Service REST-APIs are described in the following URL patters:

1. **/algorithms:** a GET request on this URL is used to retrieve a list of the available machine learning algorithms.
2. **/algorithms/id:** a GET request on this URL is used to retrieve a specific machine learning algorithm.
3. **/categories:** a GET request on this URL is used to retrieve a list of the available machine learning categories, for example classification, regression, clustering, to name a few.
4. **/dataSets:** a GET request on this URL is used to show available datasets
5. **/dataSets:** a POST request on this URL is used to create meta data of a dataset.
6. **/dataSets/id:** a GET request on this URL is used to retrieve the metadata of a specific dataset.

7. **/dataSets/id/data:** a POST request on this URL is used to upload a local data file into HDFS and upload the dataset's reference.
8. **/dataSets/id/descriptiveStatistics:** a POST request on this URL is used to prepare model for calculating the descriptive statistics for a specific dataset.
9. **/mlModels:** a GET request on this URL is used to retrieve a list of pre-trained machine learning models.
10. **/mlModels/id:** a GET request on this URL is used to retrieve metadata of a specific machine learning model.
11. **/mlModels/id:** a DELETE request on this URL is used to delete a specific pre-trained machine learning model.
12. **/mlModelPredictions/id:** a GET request on this URL is used to retrieve the prediction file for a specific machine learning model.
13. **/mlPipelines:** a GET request on this URL is used to retrieve a list of machine learning execution pipelines.
14. **/mlPipelines/id:** a GET request on this URL is used to get the meta data for a specific machine learning pipeline.

Persistence and Processing Layer: It hides the low level details of the runtime environment from the implementation of the services. The services use generic functions implemented in this layer to interface with the job runtime directory in HDFS and the database infrastructure installed on the runtime as well as performing dedicated tasks on the runtime environment for instrumenting installed ML frameworks to e.g. perform job execution. For each ML runtime environment, the persistence and processing layer will contain an adapter which maps model and execution details to the specific framework (see Sect. 5 for further discussion on issues related to the prototype and interfacing to the Apache Spark runtime environment).

Typically, all information related to the execution of a certain job is collected in a job runtime directory on a file system of the runtime platform. Thus, the persistence and processing layer contains functionalities for creating such directories depending on the execution framework. More generally, all data items managed by the D.M.-Service are stored in a database infrastructure which is defined by an abstract object-like interface. This interface can be implemented in the runtime infrastructure by using different database technologies as shown in Sect. 5.

5 Evaluation

So far the concept and architecture of the proposed microservice-based framework is discussed. In this section, two aspects of the experimental performance evaluation will be detailed. On the one hand, the effect of caching RDDs in Apache Spark is analyzed by comparing the execution time of training and testing the benchmark evaluation models in case of memory caching and without memory caching of the input time series datasets. On the other hand, the execution time and framework overhead for evaluating the efficiency of the framework are measured, highlighting the advantage of storing and retrieving ML models and discovering the threshold, at which the use of the proposed framework is recommended for better performing machine learning tasks in Big Data environments. Before presenting the obtained results, first the execution workflow is explained. Then the experimental setup and the related configurations are presented.

5.1 Execution Workflow

In the present work, the well-known Apache Spark framework installed on a Big Data computing cluster using an Apache Hadoop software stack as runtime engine for executing ML jobs is used. ML execution environments typically use a job runtime directory in a file system for storing all information needed for job execution (e.g. for storing models to executed, algorithm configurations and results). On a Big Data cluster based on the Apache Hadoop, HDFS is typically used as distributed file system and the runtime directory for a job can be accessed by all computing nodes of the cluster using the HDFS interfaces. Therefore, for implementing the persistence and processing layer on the cluster, HDFS and a postqreSQL database are utilized to store the required input and the output produced from performing ML tasks. The postqreSQL database system is used as an object-relational database to store all information managed by the D.M.-Service, e.g. ML categories, ML algorithms, hyperparameters, pre-trained models, jar files, references of datasets stored in HDFS, pre-trained model pipelines and untrained model pipelines.

HDFS is also utilized to store datasets and the output of successful jobs executed in Apache Spark before being read by the J.M.-Service. The dataset storage on HDFS allows it to have "Big Data" as input, i.e. datasets which are extremely large. To achieve the goal of storing pre-trained ML models in the form of binary objects, the Large Object feature of PostgreSQL is used. This feature uses the Large Object Manager Interface which stores only a reference named oid in the database table pointing to the actual object stored in the system table pg largeobject. This method breaks the binary data into chunks and allows storing objects of up to 2 GB within the database. However, another format such as Predictive Model Markup Language (PMML) will be considered in the future.

Figure 8 shows the the basic methodological workflow for task execution as it is implemented in the prototype for submitting jobs to the Apache Spark runtime. For each new job, the persistence and processing layer generates on behalf of the J.M-Service a Universally Unique Identifier (UUID) as jobID which will be sent back to the D.M.-Service. The usage of a UUID guaranties the uniqueness of the id, making it suitable to use in a distributed environment, such as a Big Data environment.

Corresponding to each jobID, a temporary job runtime directory with the UUID as a name is created in HDFS by the J.M.-Service, which uses the File

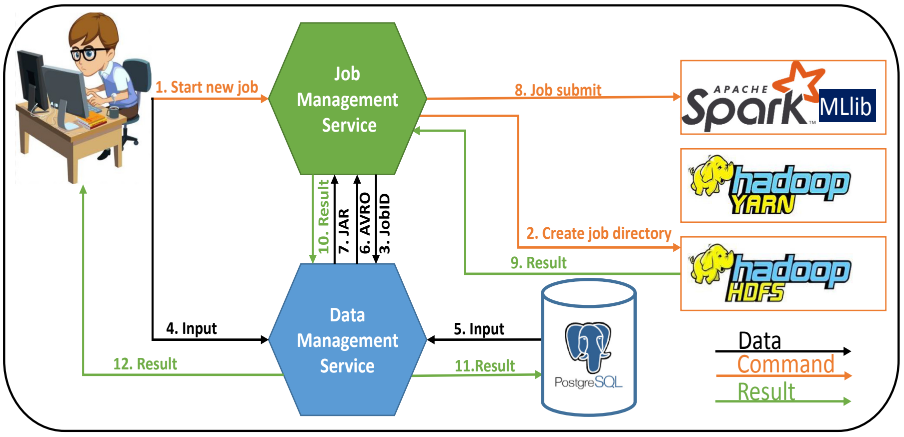


Fig. 8. Execution workflow.

System (FS) shell instruction of HDFS[8] to achieve that. Then, the J.M-Service then calls the D.M.-Service to fetch the necessary artifacts (e.g. model, runtime configuration) from the database and pass it to the J.M-Service as an Apache Spark AVRO file. After that, the J.M-Service places the AVRO file in the persistence and processing layer in the job runtime directory.

The decision for utilizing AVRO was made, because AVRO uses a schema which decouples the solution from the implementation including error prevention. An AVRO file contains the received jobID and the chosen cluster configurations. However, if no cluster configurations are chosen in the UI, the default one will be fetched from the database and used in this task. Besides cluster configurations, algorithm hyperparameters and metadata related to the execution of algorithms, namely the name of application main class are included in the AVRO file for execution. The name of the application main class is required by Apache Spark to find the main code entry point for executing the task. While all datasets are stored in the HDFS, path references pointing to the files are stored in the database of the D.M.-Service. Once the user chooses a dataset, the path reference of the dataset in HDFS is fetched from the database and included in the AVRO file. After that, the D.M.-Service fetches the corresponding jar file from the database and sends it to the J.M.-Service. At this point, all required information to perform the task is passed to the J.M.-Service which creates a spark-submit job and sends it for execution to Apache Spark.

As a result of executing e.g. a task performing forecasting on a time series dataset, the forecasting results, forecasting performance and the forecasting model in the form of a binary object are located in the temporary job runtime directory of the task. After executing the job, all of these results are stored in the temporary job runtime directory and read afterwards by the J.M.-Service to be passed to the D.M.-Service. The D.M.-Service receives the results and stores

[8] https://hadoop.apache.org/docs/stable/.

them in the form of a pipeline in the database to be retrieved later. Simultaneously, the D.M.-Service sends the results to the UI to be rendered and visualized for the user.

5.2 Experimental Setup and Configurations

The aforementioned microservice architecture is implemented using Java and tested while running on a local workstation which is a MacBook with a 2.7 GHz Intel Core i5 processor and 8 GB of RAM. Both microservices are implemented as standalone Spring Boot applications which are configured to run on different HTTP ports, namely 8090 and 8080. To run our web application, the embedded Apache Tomcat server from Spring Boot is utilized.

For our evaluation and to investigate the effectiveness of our framework, local execution context and cluster execution context have been configured. In the local context, Spark (v. 2.3.0) on top of Hadoop (v. 2.7.6) as state-of-the-arts technology to perform machine learning tasks is installed on the aforementioned workstation, where the executors and drivers run in a single JVM. In the cluster context, we utilize a powerful Big Data stack, in which Apache Spark is fit on top of Yet Another Resource Negotiator (YARN) as a resource manager and Hadoop Distributed File System (HDFS) as a primary data storage. The Big Data stack is deployed on a cluster of 3 logical machine nodes. Each of them has 32 cores and 80.52 GB RAM. The nodes are connected to each other by a LAN with 10 GBit/s bandwidth.

Table 3. Default and custom configurations used in cluster context.

Default	Custom
Drivers.cores = 1	Drivers.cores = 1
Driver.memory = 1 GB	Driver.memory = 1 GB
Executors.cores = 2	Executors.cores = 2
Executors.memory = 1 GB	Executors.memory = 70 GB
Executors.instances = 1	Executors.instances = 3

In the cluster context, we distinguish two configuration setups, namely default and custom as presented in Table 3. Random Forest (RF), Multiple Linear Regression(MLR), Gradient Boosted Trees (GBTs) and Decision Tree (DT) are used as base classifiers to build the data-driven forecasting models. MLlib, which is a Spark's scalable ML library is employed to build the models. To train and test the forecasting models, a simulated energy multivariate time series dataset is used. MLR is a widely used supervised algorithm which assumes a linear relationship between one or multiple independent input variables and a dependent output variable [44]. Table 4 presents the default values of the MLR hyperparameters.

Table 4. Default hyperparameters of MLR algorithm in MLlib.

Hyperparameter	Description	Default
maxIter	Maximum number of iterations	100
regParam	Regularization/Shrinkage parameter	0.0

DT algorithm [44] is a supervised algorithm, often chosen for its interpretability. It has the ability to capture the non-linear structures in data, unlike MLR. A DT is essentially a binary tree which recursively partitions the input space and consists of internal nodes and leaves (i.e. terminal nodes). It is constructed starting from the root and its nodes are split down based on the largest decrease in impurity. For classification trees, the impurity is often measured with the Gini impurity or entropy. However, for regression trees, where the target is continuous, the impurity is based on variance reduction. Table 5 presents the default values of the DT hyperparameters. RF algorithm [44] builds a forest of multiple DTs that are independently trained. Whereas, single DTs are often said to overfit, the RF algorithm does not overfit because of the Law of Large Numbers [7]. Also, randomness is applied to the training process of RF by utilizing random feature subsets for node splitting. Since, each DT is trained separately, multiple trees can be trained in parallel. For the final prediction, the individual votes of all trees are combined. Table 6 presents the default values of the RF hyperparameters.

Table 5. Default hyperparameters of DT algorithm in MLlib.

Hyperparameter	Description	Default
maxBins	Maximum number of bins for split decision and discretization of continuous features	32
maxDepth	Number of trees in the forest	5
minInstancesPerNode	Minimum number of trees (training instances) in children must have by splitting	1

Table 6. Default hyperparameters of RF algorithm in MLlib.

Hyperparameter	Description	Default
maxDepth	Maximum depth of individual trees in the forest	5
numTree	Number of trees in the forest	20

In contrast to RF which trains the trees independently, GBTs algorithm [44] employs the Boosting technique training one tree at a time. Successively, to correct the errors made by previous trees, a DT is fitted on the residuals of the previous tree, instead of a fraction of the original data. The final prediction is based on a weighted majority vote. Table 7 presents the default values of the GBTs hyperparameters.

Table 7. Default hyperparameters of GBTs algorithm in MLlib.

Hyperparameter	Description	Default
maxDepth	Maximum depth of the individual trees	5
maxIter	Maximum number of iterations	20
stepSize	Controls the contribution/weight of each tree	0.1
subsamplingRate	Training data proportion used for learning each tree	1.0

Tuning hyperparameters is an important step of the Machine Learning Pipeline (MLP), since they can not only significantly influence the forecasting performance of a model, which is not our focus in the present work, but also the processing time.

Table 8. ML algorithms hyperparameters after tuning.

ML algorithm	Hyperparameters
Multiple Linear Regression (MLR)	Max iterations (ntree) = 20 Regularization parameter = 0.5
Decision Tree (DT)	Max bin = 5 Max depth = 5 Min instance split = 1
Gradient Boosted Trees (GBTs)	Max depth = 5 Number of trees = 20 Step size = 0.1 Sampling rate = 1.0
Random Forest (RF)	Max depth = 5 Number of trees (ntree) = 20

Based on the main property of our microservice-based framework in facilitating training and testing ML models in Big Data environments, an efficient hyperparameter tuning is performed for the aforementioned ML algorithms to ensure that the time measurements are taken for a best case scenario of the aforementioned algorithms. The results are depicted in Table 8.

As mentioned before, one of the main advantages of the proposed framework is to store pre-trained models in order to use them later in production. Thus, for evaluation, two execution contexts are determined, namely the untrained model pipeline and pre-trained model pipeline. In the first one, as its name implies, the user follows the general methodology to perform a ML task, in which the model is trained from scratch and afterwards tested. In the second one, the user selects a pre-trained model from the database and uses it to perform or test a ML task with a new dataset without the need for building a new model. In the present article, the main goals of evaluation are discovering the effect of caching in Apache Spark, the advantage of storing ML models and reusing them, measuring the framework overhead and determining the thresholds for efficiently performing ML tasks on the cluster. To this aim, time measurements need to be precisely defined. As time measurements, we defined T_{total} and T_{fo} according to Eq. 1 and 2 respectively.

$$T_{total} = T_{exe} + T_{fo} \tag{1}$$

where:

- T_{exe}: is the execution time required by Apache Spark to perform a ML task in context of pre-trained pipelines or untrained pipelines.
- T_{fo}: is the framework overhead.

$$T_{fo} = T_{co} + T_{dbo} \tag{2}$$

where:

- T_{co}: describes the communication overhead between microservices and inside the Big Data infrastructure.
- T_{dbo}: describes the overhead for storing and retrieving required data from the database.

5.3 Experimental Results and Analysis

In the following, the evaluation results are discussed. As the focus is on the execution time and the framework overhead raised while performing ML tasks, the accuracy of forecasting will not be taken into account.

Effect of Caching in Apache Spark. Resilient Distributed Datasets (RDDs) are the basic data structure of Apache Spark developed as a fault-tolerant immutable collection of objects which can be computed on different nodes of the cluster[9]. Caching RDDs in Apache Spark is a widely used mechanism for speeding up the running applications. This is especially helpful, when running iterative machine learning applications, where the data is accessed repeatedly. If RDD is not cached, nor checkpointed, it is re-evaluated again each time an action is invoked on that RDD. The training time is measured as the time it takes to fit the model on the training data. The prediction time is similarly computed for applying the resulted model on testing data. Since Spark utilizes lazy evaluation for data transformations, meaning an operation is not executed until an action is called on the data, the prediction time has to be measured in combination with performing an action. The main advantages of the lazy evaluation mechanism in Apache Spark are:

[9] https://spark.apache.org/docs/latest/rdd-programming-guide.html.

– Increased manageability of RDDs because the source code of our machine learning algorithms is organized into smaller operations which in turns reduces the number of passes on data by grouping the operations.
– More efficient computation time and an increased speed, as only the necessary values are computed saving the communication round-trip time between the drivers and clusters.
– Better optimization of operations on data by reducing the number of queries.

Table 9. Mean computation time for training and testing different algorithms in the cases of caching and no caching of input data.

Machine learning algorithms	Training time (s)		Prediction time (s)	
	No caching	Caching	No caching	Caching
Multiple Linear Regression (MLR)	16.07	3.57	3.92	0.87
Decision Tree (DT)	15.88	3.21	3.41	0.86
Gradient-boosted trees (GBTs)	37.04	21.74	8.61	1.77
Random Forest (RF)	23.11	12.48	5.75	1.12

Table 9 shows how caching of the input time series datasets affects the performance of the implemented algorithms, using their default hyperparameters and default cluster configurations. For calculating these values, the experiments are repeated three times. Afterwards, the mean values are calculated as final performance indicator. Obviously, the need for caching will be larger in the case of large datasets, as more operations are required and larger amount of data are loaded and accessed repeatedly, therefore and to precisely discover the effect of caching, the models are trained and tested on a small dataset size i.e. 4 MB. As shown in this table, combining lazy evaluation with caching reduces the training and prediction time of all algorithms by approximately 75%.

Advantage of Storing and Retrieving ML Models. The main ML task used for this part of evaluation is to perform short-term energy generation forecasting using MLR, RF, DT and GBTs data-driven models on simulated energy multivariate time series dataset. The algorithm hyperparameter configurations shown in Table 8 are used. For better utilization and exploitation of the available abilities of the underlying Big Data cluster, the custom configurations shown in Table 3 are used. A feature space consisting of 5 features, namely temperature, humidity, cloud coverage, hour and day is used to build the forecasting models. A dataset of 4 GB size is used for training and testing ML models, where 80% of the input time series dataset are used as a training set and 20% as testing set. For each ML algorithm, the experiment is repeated three times. Afterwards, the mean values are calculated as final performance indicator. Figure 9 shows the T_{total} required by the framework to perform the aforementioned task in case of pre-trained and untrained model pipeline.

In general, the total time T_{total} is strongly related to the complexity of ML models. As this complexity increases, T_{total} required to perform the task will dramatically increases. The base classifier of both RF and GBTs algorithms is the DT algorithm. Consequently, the complexity of RF and GBTs models will be higher than the complexity of the DT model. As seen in Fig. 9, RF and GBTs introduced higher T_{total} than DT and MLR algorithms.

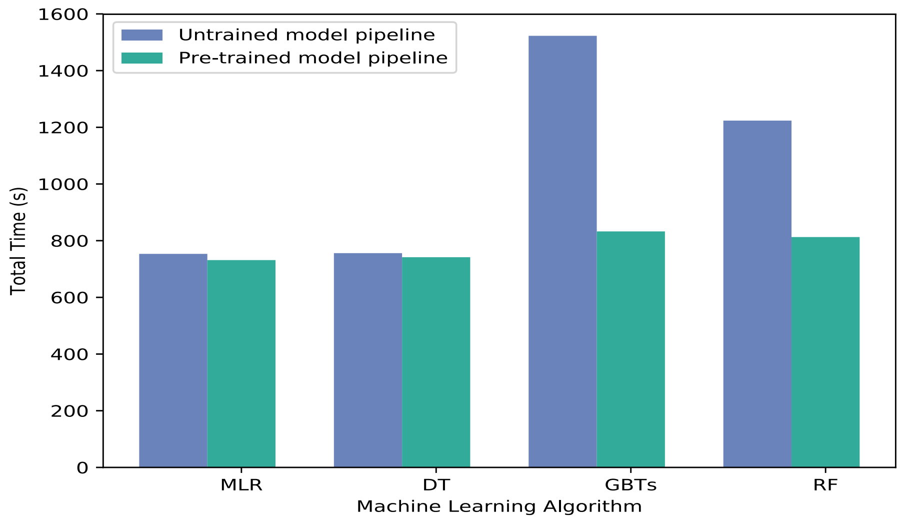


Fig. 9. T_{total} required for training and testing models (untrained model pipeline) and for testing (pre-trained model pipeline) on simulated energy multivariate time series dataset with size 4 GB.

Both GBTs and RF are algorithms for learning ensembles of trees, but the training processes are different. While GBTs algorithm trains one tree at a time, RF algorithm can train multiple trees in parallel. This can be seen clearly in Fig. 9, in which GBTs show higher T_{total} than RF. In our experiments, both MLR and DT algorithm introduce lower T_{total} compared to RF and GBTs. The efficiency of storing ML model can clearly be seen in case of complex ML models, namely GBTs and RF models, and will rise with growing complexity of the model. As the complexity of model increases, the time needed to perform the same task with each new dataset will dramatically increase and the benefit of using pre-trained models will also increase. E.g., by performing forecasting, we gain a time of 690 and 411 s in case of GBTs and RF respectively. In contrast to that, only small time will be gained in case of retrieving and reusing simpler models such as MLR and DT as seen in Fig. 9.

As a result, the recommendation of storing ML models and reusing them in testing is higher in case of complex models than for simpler ones. This experimental study gives an evidence for the importance of storing and retrieving ML models as a major property in our framework. However, the experiments are performed only with a dataset of 4 GB size. As this size increases, the complexity of the ML models will increase too, paving the road to save and gain more time for performing ML tasks with new datasets based on pre-trained models without the need for training these models.

Framework Overhead. To evaluate the framework overhead, MLR models for short-term energy generation forecasting are used. The algorithm hyperparameter configurations shown in Table 8 besides the custom cluster configurations are used in this group of experiments. The evaluation instruments the untrained model pipeline, in which the training and the testing steps of ML models are required. The goal of the study is to evaluate the effect of input dataset size on framework performance in the form of framework overhead defined in Eq. 2. For this evaluation, the size of the input datasets is upscaled to 64 GB, as bigger datasets typically expose more load on the framework infrastructure.

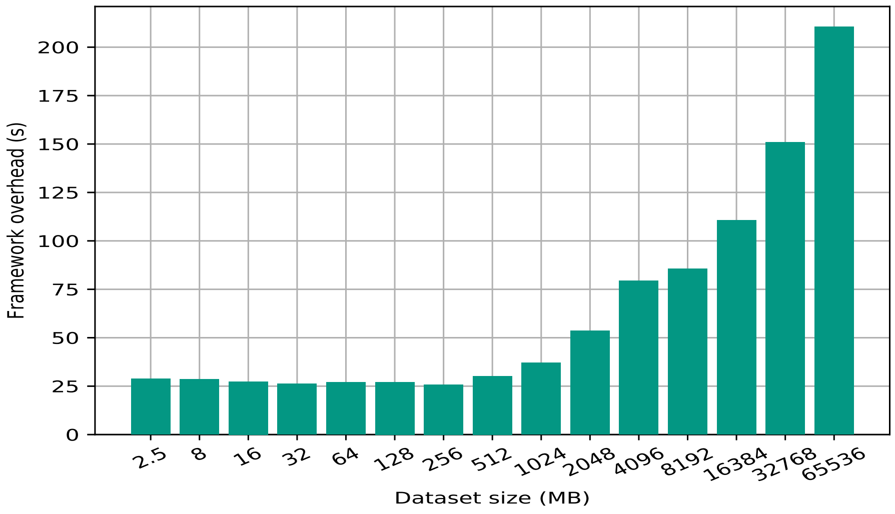


Fig. 10. Effect of input datasets size used for training and testing MLR models on the framework overhead.

As defined in Eq. 2, the framework overhead encompasses communication overhead and database overhead. The obtained results depicted in Fig. 10 show that the proposed framework introduces an approximately constant communication overhead averaging at around 26 s for datasets with sizes up to 512 MB. The framework overhead starts to increase for a size of input datasets larger than 512 MB. The reason behind this is the additional overhead raised inside the Big Data environment for resource scheduling, coordination and network communications in the cluster. Precisely, an increasing size of the input dataset naturally leads to an increased overhead due to data replication, disk I/O and the

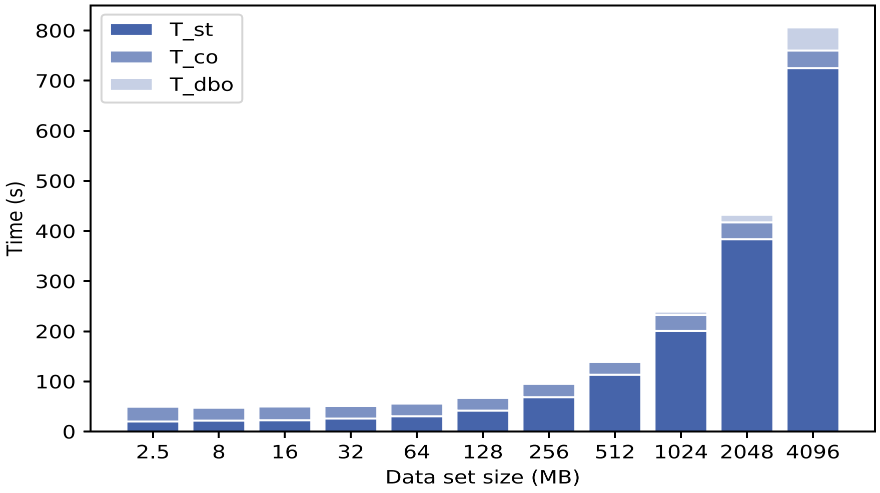


Fig. 11. Effect of input datasets size used for training and testing MLR models on the framework overhead (detailed overview).

serialization of data inside the execution environment of the cluster. A detailed increasing in overhead can be seen also in Fig. 11.

Despite this increment, the introduced framework overhead is still low compared to the execution time spent in performing a ML task as shown in Table 10. For example, the portion of framework overhead is 210,47 s in the worst case, namely for 65 GB input multivariate time series datasets. Consequently, our evaluation demonstrates, that it maintains high performance ML processing with low framework overhead to facilitate and solve ML tasks in Big Data environments, where the user gains great benefits from reusing pre-trained models.

Cluster Utilization Threshold. This section discusses the question "when to use the proposed framework for performing Ml tasks more efficiently on a cluster?". Clearly, the dataset size has an essential effect on the complexity of machine learning models and therefore on runtime performance. As the size of the dataset used for training and testing machine learning models grows, the complexity of model will increase which dramatically affects the total execution time in our microservice-based framework. While MLR forecasting models have the lower complexity, the RF forecasting models represent the higher complex models in our evaluation study. Moreover, DT forecasting model has higher and lower complexity from LR and GBT respectively as seen in Fig. 9.

The algorithm hyperparameter configurations shown in Table 8 are used. The input dataset size is changed between 2.5 MB and 4 GB in the experiments for investigating the effect of dataset size on the framework overhead and execution

Table 10. Execution time in Apache Spark vs. framework overhead for MLR models.

DataSet size (MB)	Execution time (s)	Framework overhead (s)
4	19,99	28, 78
8	21,79	28,53
16	22,6	27,41
32	25,88	26,21
64	30,54	27,05
128	41,7	27,13
256	68,54	25,82
512	113,25	30,2
1024	200,66	37,06
2048	383,75	53,54
4096	724,98	79,37
8192	1016,48	85,65
16384	4724,98	110,66
32768	6383,75	150,88
65536	11804,36	210,47

time. The total time T_{total} is compared to the time required for performing the same task in local and cluster context. The ratio of local time and cluster time is defined as *abs_threshold* in Eq. 3.

$$abs\_threshold = \frac{T_{local}}{T_{total}} \tag{3}$$

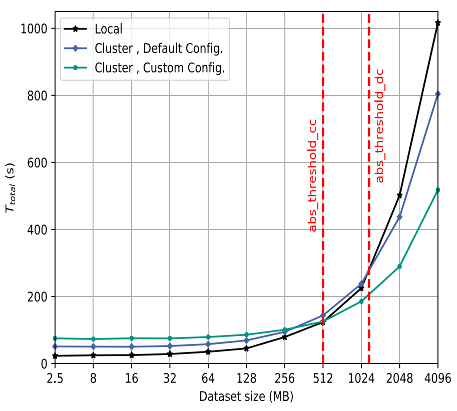
where:

– T_{local}: encompasses the total time required to locally execute a machine learning task.

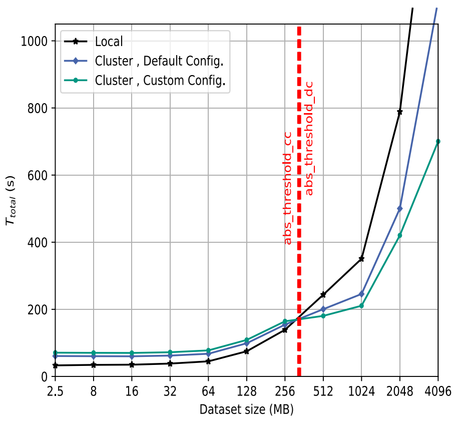
The main idea behind defining *abs_threshold* is to find the dataset size for which the total time in local context exceeds the total time required by the framework to execute tasks in cluster context. From this point, it is highly recommended to use a cluster. Precisely, to effectively perform machine learning tasks, this ratio should be greater than 1.

Performing machine learning tasks in cluster context introduces additional overhead. The main reason behind this interest lies in the time cost for resource scheduling, coordination and network communications in the cluster. Figure 12 shows the mean total time in local and cluster modes, including default and custom configurations, using various dataset sizes. It can be observed that enlarging the dataset size from 2.5 MB to 64 MB has no significant effect on both T_{local} and T_{total}.

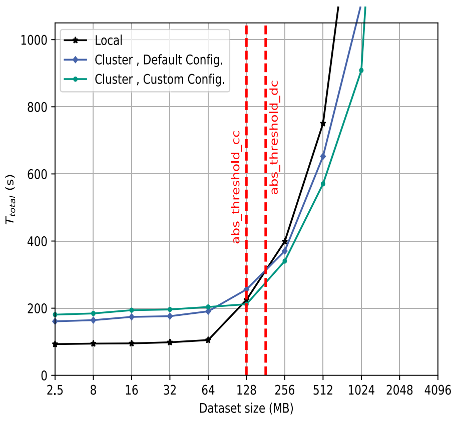
As seen in Fig. 12a and for data less or equal to 256 MB, T_{total} in both cluster modes is larger than T_{local} in local mode which can be explained by



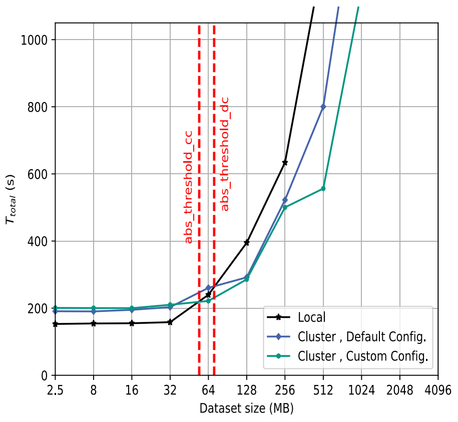
(a) Multiple Linear Regression (MLR).



(b) Decision Tree (DT).



(c) Gradient-boosted trees (GBTs).



(d) Random Forest (RF).

Fig. 12. Mean T_{total} in case of local and cluster (default, custom) configurations mode to determine the cluster utilization *abs_threshold* for MLR, DT, RF and GBTs algorithms.

the added overhead for processing the application on the cluster. Thus, running Spark applications locally for these dataset sizes is more efficient. For a dataset size less than 256 MB, T_{total} with custom configurations is larger than T_{total} with default configurations, since two additional nodes are used in these configurations where each of them introduces an overhead. As expected, when the dataset size grows larger, utilizing a cluster becomes more desirable which is shown by the intersection points highlighted by the two red lines, where these points depend on the configurations. As the *abs_threshold_cc* (cc: custom configurations) is found at a dataset size of 512 MB making the custom configurations the most efficient beyond that point, the *abs_threshold_dc* (dc: default configurations) lies at a dataset size of >= 1024 MB. Consequently, the computing power of the cluster can be seen and the time consumed locally to perform a task will exceed the time required to perform the same task on cluster. Therefore, it is recommended here to use the cluster.

Comparing Figs. 12a, 12b, 12c and 12d, we conducted that as the complexity of machine learning model increases, the *abs_threshold* will be early arrived. The reason is that the complex models need more calculation costs. As a result, the performance in cluster context will earlier outperform the performance in local context because of the power of the underlying deployed Big Data cluster. Concerning RF model which represents the highest complex model in our benchmark evaluation, the *abs_threshold* will be arrived for input dataset in size of about 64 MB. In contrast to that, lower complexity models such as DT models introduced *abs_threshold* for 300 MB.

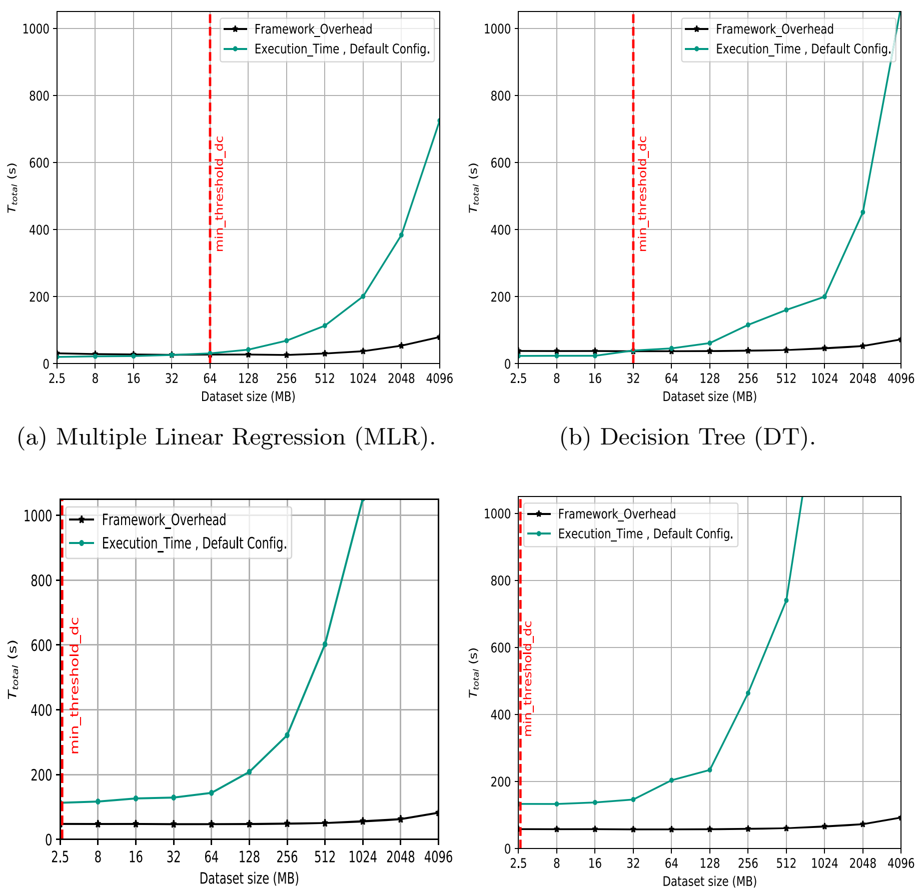
As mentioned before, there is an inherent overhead in the framework arising from e.g. database communication and the use of the cluster. The smaller this overhead is compared to Spark's execution time, the more efficient the framework is. To gain insight into how the efficiency of the framework varies as the dataset grows, a new threshold, referred as *min_threshold*, is defined and formulated in Eq. 4:

$$min\_threshold = \frac{T_{exe}}{T_{fo}} \tag{4}$$

This threshold is defined based on the fact that for an efficient execution of a task, the overhead time should not exceed the time required for the execution. Consequently, to effectively perform machine learning tasks, this threshold should be greater than 1. The main difference between *min_threshold* and *abs_threshold* lies in the context, in which they are calculated. While *abs_threshold* compares the total time required to perform a machine learning task in local and cluster context, the other one is calculated only in cluster context comparing the framework overhead with the execution for different dataset size. As a result, we discovered the point at which it is recommended to use our framework in cluster context.

This group of experiments are conducted using default cluster configurations summarized in Table 3 and also repeated three time for more robust results. The obtained mean results, presented in Fig. 13 show that *min_threshold* has the same behavior of *abs_threshold*. It is evident that for very small dataset sizes *min_threshold* is less than 1 since more time is spent on T_{fo} than T_{exe}. The *min_threshold* comes closest to 1 at the size of 64 MB and 32 MB for MLR and DT respectively as seen in Figs. 13a and 13b. Beyond this point T_{exe} starts to exceed T_{fo} which implies that for larger dataset sizes it is recommended to use the framework in cluster context. Precisely, the gap between T_{fo} and T_{exe} increases proportionally to the dataset size, since T_{exe} is strongly dependent on it. As the complexity of the model increases, the *min_threshold* will be shifted to meet smaller dataset size i.e. 2.5 MB as seen in Figs. 13c and 13d. Combining the results of *abs_threshold* and *min_threshold*, it is recommended to perform

ML tasks using the proposed microservice-based framework if both of these thresholds are greater than 1.



(a) Multiple Linear Regression (MLR).

(b) Decision Tree (DT).

(c) Gradient-boosted trees (GBTs).

(d) Random Forest (RF).

Fig. 13. Mean T_{total} in case of local and cluster (custom) configurations mode to determine the cluster utilization *min_threshold* for MLR, DT, RF and GBTs algorithms.

6 Conclusion and Future Works

The present paper introduces a new highly scalable generic microservice-based framework to ease and streamline the performing of ML tasks in Big Data environments. This framework provides a user-friendly UI built on top of a service layer that eases the usage of ML frameworks on Big Data infrastructure and hides cluster details from the user. Despite the ability of training, testing, managing, storing and retrieving of machine learning models in the context of Big Data, the framework provides functionalities for uploading, exploring and visualizing datasets using state-of-the-arts web technologies. Moreover, ML model selection and management in form of storing pipelines are supported. Each pipeline corresponds to a specific ML task, in which ML algorithm, dataset, hyperparameters and performance results are stored in an integrated package providing the user the ability for deeper comparison and better selection of ML models. To reduce the difficulty as well as the complexity of performing tasks in Big Data environments, cluster configurations can be easily tuned and adjusted in the UI.

In a comprehensive evaluation study, the advantage of storing and retrieving ML models is demonstrated. The results also show that the caching of RDDs in Apache Spark plays an essential role in saving the execution time required for performing the task on the cluster. Moreover, by measuring the framework overhead and comparing it to the model calculation time, it could be demonstrated that the proposed framework introduces an acceptable low overhead relative to an increasing size of an input dataset. For efficient utilization of the proposed framework, certain thresholds are defined to determine the dataset size, in which it is highly recommended to use clusters in favor to single computers for performing a given ML task.

The proposed framework is an ongoing work for developing an even more interactive and intelligent framework for fully automating, managing, deploying, monitoring, organizing, and documenting of ML tasks.

Future work will discover the effect of caching in the case of using the best hyperparameters that optimize the performance of the ML algorithms. We will also extend the functionalities of the framework to cover automated preprocessing, model selection and hyperparameter tuning leveraging the advantage of meta learning. Classification, clustering and a wide range of ML application scenarios will be taken into account. Deep Learning as a pluggable engine will be integrated in the persistence and storage layer to support performing Deep Learning tasks. In-depth user feedback assessment by a large number of users, in particular, non-expert users will be collected and analyzed too. For tenancy and secure management of ML tasks, user authentication and authorization issues will be also taken into account.

References

1. Vernon, V.: Implementing Domain-Driven Design, p. 612. Addision-Wesley, Upper Saddle River (2013)
2. Fielding, R.T.: Architectural Styles and the Design of Network-Based Software Architectures. AAI9980887. University of California, Irvine (2000)
3. Nielsen, J.: 10 usability heuristics for user interface design. Nielsen Norman Group **1**, 1 (1995)
4. Sebastiani, F.: Machine learning in automated texT categorization. ACM Comput. Surv. (CSUR) **34**(1), 1–47 (2002)
5. Padmanabhan, J., Johnson Premkumar, M.J.: Machine learning in automatic speech recognition: a survey. IETE Tech. Rev. **32**, 1–12 (2015)

6. Kononenko, I.: Machine learning for medical diagnosis: history, state of the art and perspective. Artif. Intell. Med. **23**(1), 89–109 (2001)
7. Voyant, C., et al.: Machine learning methods for solar radiation forecasting: a review. Renew. Energy **105**, 569–582 (2017)
8. Jurado, S., Nebot, A., Mugica, F., Avellana, N.: Hybrid methodologies for electricity load forecasting: entropy-based feature selection with machine learning and soft computing techniques. Energy **86**, 276–291 (2015)
9. Gandomi, A., Haider, M.: Beyond the hype: Big Data concepts, methods and analytics. Int. J. Inf. Manag. **35**(2), 137–144 (2015)
10. Karun, A.K., Chitharanjan, K.: A review on Hadoop-HDFS infrastructure extensions. In: 2013 IEEE Conference on Information and Communication Technologies, pp. 132–137. IEEE (2013)
11. Nadareishvili, I., Mitra, R., McLarty, M., Amundsen, M.: Microservice Architecture: Aligning Principles, Practices and Culture. O'Reilly Media Inc. (2016)
12. Vartak, M., et al.: Model DB: a system for machine learning model management. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics, p. 14. ACM (2016)
13. Johanson, A., Flogel, S., Dullo, C., Hasselbring, W.: OceanTEA: exploring ocean-derived climate data using microservices (2016)
14. Brewer, R.S., Johnson, P.M.: WattDepot: an open source software ecosystem for enterprise-scale energy data collection, storage, analysis and visualization. In: 2010 First IEEE International Conference on Smart Grid Communications. 2010 1st IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 91–95, Gaithersburg, MD, USA. IEEE (2010)
15. Shrestha, C.: A web based user interface for machine learning analysis of health and education data (2016)
16. Schelter, S., Böse, J.-H., Kirschnick, J., Klein, T., Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments (2017)
17. Obe, R.O., Hsu, L.S.: PostgreSQL: Up and Running: a Practical Guide to the Advanced Open Source Database. O'Reilly Media Inc. (2017)
18. Meng, X., et al.: MLlib: machine learning in Apache Spark. J. Mach. Learn. Res. **17**(1), 1235–1241 (2016)
19. Zaharia, M., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. **41**(4), 39–45 (2018)
20. Chan, S., Stone, T., Szeto, K.P., Chan, K.H.: Predictionio: a distributed machine learning server for practical software development. In: Proceedings of the 22nd ACM International Conference on Information and Knowledge Management, pp. 2493–2496. ACM (2013)
21. TensorFlow Serving. https://www.tensorflow.org/serving. Accessed 4 Feb 2020
22. kubeflow. https://www.kubeflow.org/. Accessed 4 Feb 2020
23. Candel, A., Parmar, V., LeDell, E., Arora, A.: Deep Learning with H2O. H2O. AI Inc. (2016)
24. Borthakur, D.: The Hadoop distributed file system: architecture and design. In: Hadoop Project Website, vol. 11, p. 21.0 (2007)
25. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA, pp. 1–10. IEEE, May 2010
26. Vavilapalli, V.K., et al.: Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing - SOCC 2013. The 4th Annual Symposium, pp. 1–16. ACM Press, Santa Clara (2013)

27. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
28. Microservices. https://martinfowler.com/articles/microservices.html. Accessed 18 Feb 2020
29. Newman, S.: Building Microservices: Designing Fine-Grained Systems, 1st edn. O'Reilly Media, Beijing (2015)
30. Coughlin, K., Piette, M., Goldman, C., Kiliccote, S.: Estimating demand response load impacts: evaluation of base line load models for non-residential buildings in California. Technical report, Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA, USA (2008)
31. Khotanzad, A., Afkhami-Rohani, R., Lu, T.L., Abaye, A., Davis, M., Maratukulam, D.J.: ANNSTLF-a neural-network based electric load forecasting system. IEEE Trans. Neural Netw. **8**(4), 835–846 (1997)
32. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software, p. 529. Addison-Wesley, Boston (2004)
33. Shoeb, A.H., Guttag, J.V.: Application of machine learning to epileptic seizure detection. In: ICML (2010)
34. Shahoud, S., Gunnarsdottir, S., Khalloof, H., Duepmeier, C., Hagenmeyer, V.: Facilitating and managing machine learning and data analysis tasks in Big Data environments using web and microservice technologies. In: Proceedings of the 11th International Conference on Management of Digital EcoSystems, pp. 80–87 (2019)
35. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann (2016)
36. Aman, S., Simmhan, Y., Prasanna, V.K.: Improving energy use forecast for campus micro-grids using indirect indicators. In: 2011 IEEE 11th International Conference on Data Mining Workshops. IEEE, pp. 389–397 (2011)
37. Hong, T., Gui, M., Baran, M., Willis, H.: Modeling and forecasting hourly electric load by multiple linear regression with interactions. In: IEEE PES General Meeting. IEEE, pp. 1–8 (2010)
38. Metaxiotis, K., Kagiannas, A., Askounis, D., Psarras, J.: Artificial intelligence in short term electric load forecasting. Energy Convers. Manag. **44**(9), 1525–1534 (2003)
39. Mori, H., Takahashi, A.: Hybrid intelligent method of relevant vector machine and regression tree for probabilistic load forecasting. In: 2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies, pp. 1–8. IEEE (2011)
40. Cui, C., Wu, T., Hu, M., Weir, J.D., Li, X.: Short-term building energy model recommendation system: a meta-learning approach. Appl. Energy **172**(2016), 251–263 (2016)
41. Mitchell, T.M.: Machine Learning. McGraw-Hill Series in Computer Science, 414 pp. McGraw-Hill, New York (1997)
42. Cruz, J.A., Wishart, D.S.: Applications of machine learning in cancer prediction and prognosis. Cancer Inform. **2**, 59–77 (2006)
43. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
44. Machine Learning Library (MLlib) Guide. https://spark.apache.org/docs/latest/ml-guide.html. Accessed 19 Feb 2020
45. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and unsupervised discretization of continuous features. In: Proceedings of the Twelfth International Conference on Machine Learning, vol. 12, pp. 194–202 (1995)
46. Hahne, F., Huber, W., Gentleman, R., Falcon, S.: Bioconductor Case Studies. Springer, New York (2010). https://doi.org/10.1007/978-0-387-77240-0

47. Chapelle, O., Scholkopf, B., Zien, A.: Semi-supervised learning. IEEE Trans. Neural Netw. **20**(3), 542–542 (2009). (Chapelle, O. et al. (eds.) (2006)) (bibbook reviews)
48. Kaelbling, L., Littman, M., Moore, A.: Reinforcement learning: a survey. J. Artif. Intell. Res. **4**, 237–285 (1996)
49. Mikowski, M., Powell, J.: Single Page Web Applications: JavaScript End-to-End. Manning Publications Co. (2013)
50. Kuan, J.: Learning Highcharts. Packt Publishing Ltd. (2012)