A Horizontal Tuning Framework for Machine Learning Algorithms Using a Microservice-based Architecture

Simona-Vasilica OPREA[1], Adela BÂRA[1], Gabriela DOBRIȚA (ENE)[1,2], Dragoș-Cătălin BARBU[1,2]*

[1] Bucharest University of Economic Studies, Department of Economic Informatics and Cybernetics, 6 Piața Romană, Bucharest, 010374, Romania
simona.oprea@csie.ase.ro, bara.adela@ie.ase.ro

[2] Bucharest University of Economic Studies, Doctoral School of Economic Informatics, 11 Tache Ionescu Street, Bucharest, 010352, Romania
gabrielaene02@gmail.com, barbu.dragos@gmail.com (**Corresponding author*)

Abstract: Usually, data collected through surveys or by means of sensors is prone to errors and inaccuracies, such as missing data and outliers. Such datasets consist of numerical and string variables, with a high variety of values. Emerging issues, for instance, missing or categorical data lead to errors in running most of the machine learning algorithms. Data analysis and pre-processing are usually more substantial and time-consuming than the implementation of the machine algorithms. Nevertheless, the obtained results are significantly influenced by the way missing data or outliers are approached. This paper presents various methods for coping with null and extreme values. Furthermore, it highlights the significance of encoding and scaling the analysed data and their impact on the performance of the machine learning algorithms. Thus, this paper proposes a methodology for a Missing, Outliers, Encoding & Scaling (MOES) horizontal tuning framework using microservices as applications for data processing in order to obtain the best combination of the employed methods. For exemplification purposes, a real data set from the banking sector is used. Furthermore, the proposed methodology was tested using a second real data set from the utilities sector and the results also showed that both the AUC (Area under the Curve) and execution time were better than in the case of employing the PyCaret Python library.

Keywords: Microservices, Data pre-processing, Machine learning, Horizontal framework.

1. Introduction

Machine learning (ML) algorithms gain more significance as they enhance the decision-making processes and help us understand trends using predictions (Lungu et al., 2016) and consumers' behaviour. Numerous data sets are studied with ML algorithms aiming to foresee clusters (Huang et al., 2020), classify records (Oprea & Bâra, 2021), detect anomalies (Himeur et al., 2021) or improve estimator error (Garcia Rodriguez et al., 2021) in various fields. Usually, the pre-processing stage is performed to fix issues such as missing data and non-numeric variables that impede ML algorithms to run properly. Recently, more attention has been given to the hyperparameters of the algorithms to improve their performance (Schmied et al., 2021) or even to trends such as hyperautomation (Lasso Rodriguez et al. 2020). Moreover, an inquiry into machine learning-based automatic configuration tuning services on real-world database management systems is provided in (Van Aken et al., 2021). As AutoML optimizes the ML pipelines and hyperparameters, it also provides a search at the architectural level. Auto-PyTorch is showcased in (Zimmer, Lindauer & Hutter, 2021), which combines the two functions to allow fully automated deep learning and proposes a benchmark for learning curves for deep neural networks. Therefore, data pre-processing is a stage that is carried out once and sometimes with simple methods, whereas tuning is predominantly focusing on the ML algorithms` optimizers, their combinations (Khan et al., 2020) and cross-validation stages. In this way, a gap regarding data pre-processing was identified, as it is omitted or tackled at a superficial level in most scientific papers.

The concept for this framework is based on the premise that each step in the development of a ML model can be developed in an individual capsule application which will be referred to as a component. Each component comes with a pre-defined role, but it can be called at any time by the user. The result obtained after each called step will be stored at the component level so that it can be used later by methods from other components. Thus, the proposed solution must bring together a series of independent functionalities, specific to data analysis and the implementation of ML algorithms. The complexity of the problem of such a framework can be solved by decomposing it into simple services, dealing with a single aspect, easy to maintain and extend.

For developing the technical solution, the process started from a basic concept typical of the classic software design methods, domain-based architectures, according to which the application is

https://doi.org/10.24846/v32i3y202303 ICI Bucharest © Copyright 2012-2023. All rights reserved

built around the business domain that the respective software must serve. It is a fundamental notion for microservice-based architecture because it helps to identify the purpose of each individual service, thus being the boundary of the service's scope of activity (Auer et al, 2021). It should not be forgotten that each built model must be applicable to a singular context and not intersect with other areas of the application. This contextualization ensures the logical and structural unity of the components and leads to the avoidance of confusion. The contexts will be mapped onto a map that will outline the application as a whole (Hannousse & Yahiouche, 2021). Unlike a Service-Oriented Architecture (SOA) or layered architecture (Hustad & Olsen, 2021; Haghgoo et al., 2021) a specific context also involves the inclusion of data model and data, all specific to that context (Lenarduzzi et al., 2020). Microservices outline functionality of the process logic and do not exclusively refer to layers with a technical role, laid horizontally, from user interaction to the database (Mazzara et al., 2021). Their most important feature must be independent operation. A microservice is considered loosely coupled if it can be changed without changing other areas of the application. It must be built in such a way that it has a unitary structure, that is, it has a single, well-defined purpose (Di Francesco, Lago & Malavolta, 2019; Hamzehloui, Sahibuddin, & Ashabi, 2019).

The remainder of this paper is structured as follows. Section 2 briefly presents similar scientific research. Section 3 proposes an original framework using microservices as applications for tuning the data pre-processing methods and evaluating their impact on the results of a classification problem. For simulation purposes, in Section 4 a simple ML classification algorithm - linear regression, was considered as tuning several algorithms is out of the scope of this paper. The analysed data set refers to the clients of a bank, their credits and the probability of contracting a new credit. Details regarding clients, such as age, sex, profession, fidelity, prescoring and previous credits depiction are included in the data set. The conclusions of this research are presented in Section 5.

2. Literature Survey

Microservices can be deployed to run using virtual machines or containers. Containers are preferred for the proposed particular solutions because they are portable, offer modularity, and require far fewer resources than virtual machines, not requiring the embedded operating system to function (Soldani, Tamburri & Van Den Heuvel, 2018). This approach ensures efficient scaling, and existing container orchestration mechanisms such as Kubernetes help manage container clusters on single systems (Shi et al., 2021; Bernstein, 2014). An important decision is related to how the components communicate. In the case of synchronous communication, a request is initialized by a component to the server on which another component is running, during which the request remains blocked until the operation is completed. On the other hand, in the case of asynchronous communication, the caller does not wait for the results to be received to initiate another operation. Considering the possibility of long waiting times for running algorithms for example, asynchronous implementation is preferable, because it is not necessary to keep the connection open between components for long time intervals. A hybrid model of these two approaches lends itself well to the architectural concept of the application. Therefore, APIs will be implemented that are based on the request-response model, but also on event handling (Cinque, Corte & Pecchia, 2022).

Effective and successful use of ML techniques in data analysis requires considerable effort on the part of programmers, but especially on the part of end users, for whom effective analysis brings value. Expertise in the field of application of the business sector is a necessary factor for achieving performance. Most of the time, the experience of the programmers does not cover the technical notions necessary to understand the context in which the algorithm is applied, and the importance of the obtained results cannot be fully assimilated (Abdullah, Iqbal, & Erradi, 2019). So constant interaction between ML developers and end users is necessary, but this involves increased development time and effort on both sides. Therefore, the proposed solution consists in granularizing the involved processes, by implementing specific functionalities and calling them as needed, in the desired order, using data sets which were previously altered by other methods. Thus, the analysis process can be more efficient, and to some extent automated, by storing the obtained results and sending them to other methods, without repeating the implementation steps. In this way, a dynamic analysis environment is outlined, through which a rapid testing of multiple analysis variations can be achieved, from preprocessing to the selection of the best model. Once the data has been entered

into the pre-processing system, it is desired to validate and verify the data in a scalable way to run the algorithms, and even change or adjust the implementation method for various models by adjusting certain parameters or even the training data and checking the accuracy of the results (Baškarada, Nguyen & Koronios, 2020). All these steps can be automated, but human intervention cannot be completely excluded.

Several data pre-processing stages, such as cleaning, transforming, reduction, scaling, partitioning, augmentation and transfer learning are surveyed by Fan et al. (2021) focussing mainly on building operational data and its usage. The impact of the data pre-processing is emphasised by Zhu & Gao (2016) for classification purposes. Moreover, a survey of the data pre-processing methods for IoT data is performed by Jane & Arockiam (2021) focusing on its usage and impact on the performance indicators. The necessity to implement pre-processing methods is clarified and some criteria for implementation are provided, which underlines the influence of the pre-processing on the performance of the classification algorithms.

Furthermore, influence of data pre-processing on neural network performance is studied in (Zhou & Ooka, 2021). Without data pre-processing, the gradient descent algorithm can fail to reduce errors during the training process. Kakkar et al. (2018) combined data pre-processing methods with imputation techniques for software defect prediction. The effect of pre-processing of the numeric features on the performance of the classification algorithms is also investigated (Alshdaifat et al., 2021).

Several normalization methods and techniques for missing values are evaluated for various datasets implementing a couple of classification algorithms. It was concluded that the impact of the pre-processing techniques depends on the classification algorithm. Žliobaite & Gabrys (2014) investigated the problem of adaptive pre-processing. Three scenarios were described using computational examples. They found that combining adaptive pre-processing with adaptive online prediction is a good approach that could be the starting point in building future data pre-processing frameworks to enhance the prediction accuracy. A combined framework based on data pre-processing, neural networks and a multi-tracker optimizer for wind speed prediction is proposed in (Wang et al., 2020). The results proved that the framework achieves a higher forecast accuracy and support for wind power dispatch.

3. Methods and Materials

The implementation of the ML algorithms usually implies several stages, such as: Exploratory Data Analysis (EDA), pre-processing, processing, tuning the hyperparameters of the ML algorithms and evaluation of the results. They can be organized in a pipeline that consists in a sequence of end-to-end activities. First, it is essential to understand the data and potential issues that could arise while running the ML algorithms. For instance, the missing data issue has to be identified and approached as only few algorithms are able to handle it. Most of ML algorithms fail to run and end in error when they encounter null values. Furthermore, categorical data columns are not usually processed by ML algorithms, thus encoding them is mandatory.

Pre-processing the data mainly consists of several stages, such as: treating the missing and extreme values, encoding and scaling. These tasks are probably the most time-consuming for data scientists because there are numerous methods that could lead to the best results, and it is difficult to guess the right combination. Processing the data is the next step in ML algorithm implementation containing feature engineering, extraction and selection, splitting, training and testing the data sets. Tuning the hyperparameters of the ML algorithms to improve the involved models is a well-known practice that data sciences often experience, but this paper proposes a method to tune the pre-processing stages that are incorporated into a flexible horizontal tuning framework. It represents a combination of the main pre-processing stages to enhance the performance of the ML algorithms. Let us assume that the available methods for missing values are numbered from *M.0* to *M.n*, where *M.0* is the no missing value method. Considering Python as the software development platform, *M.1* could be *fillna* with mean method, *M.2* could be *fillna* with *bfill* or *ffill* method or a combination of the two, *M.3* could be interpolation (*interpolate*) and *M.4* could be multiple imputation with chained equations or mice method and so on. The extreme value methods are numbered from *O.0* to *O.m*, where *O.0* is the no outlier treatment

ICI Bucharest © Copyright 2012-2023. All rights reserved

and so on. In Figure 1, the MOES (missing, outliers, encoding and scaling) horizontal tuning framework is depicted. Depending on EDA, pre-processing activities are carried out, but instead of just considering one method for each activity for missing values (for instance *fillna* with zero), multiple methods are considered in combination with other methods for outliers, encoding and scaling. Thus, all combinations are tested for one or more performance indicators that are characteristic for a ML algorithm. This enables one to obtain the best combination of the data pre-processing methods that will generate the pre-processed data set that will be subject to feature selection and training.

The purpose of the proposed framework is to offer an efficient and optimal mechanism to achieve the necessary combination of human interaction at a minimum necessary level with the automatic process of data analysis and machine learning.

3.1 Data Insights

The user will be presented with information about the uploaded data set depending on the button chosen. The file is passed to the constructor of the Pandas package *DataFrame* object, and information such as the number of records, column names, and data type for each column is displayed by retrieving the results of the *pandas.DataFrame.info()* function call as it is illustrated in Figure 2.

3.2 Data Insights Application Using User Input

The user may prepare the data set for analysis, and the information is taken through forms in which the user fills in information according to the chosen process. If it is desired to remove columns from the data set that are not relevant for the analysis, the names of the columns must

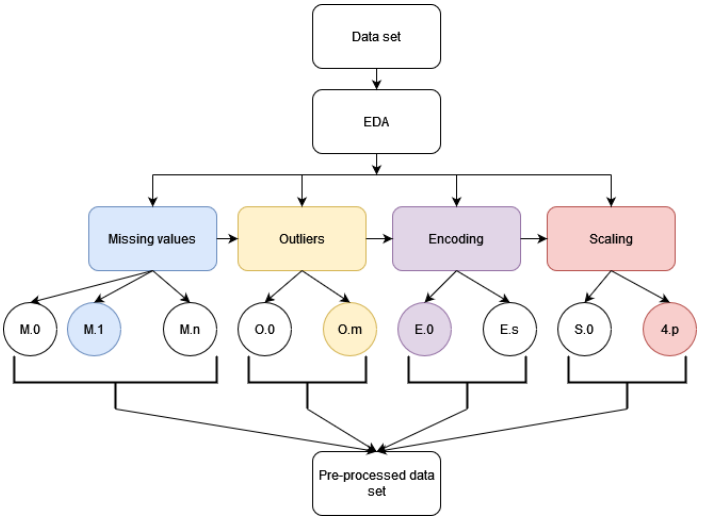


Figure 1. MOES horizontal tuning framework

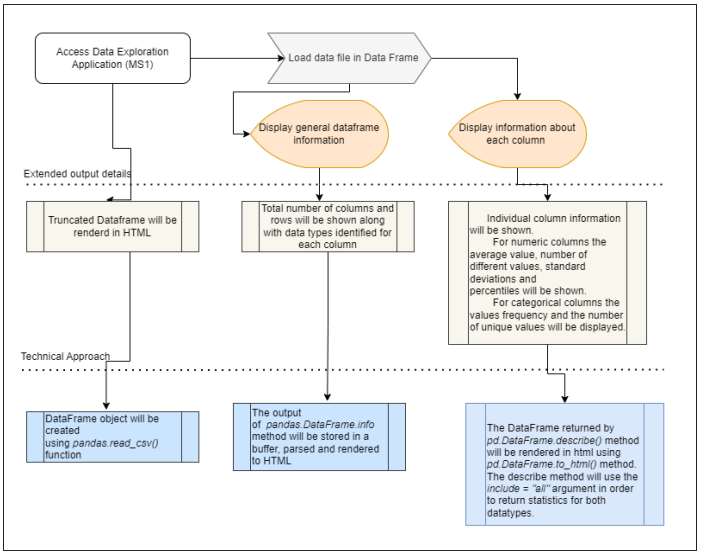


Figure 2. Getting data insights

be specified, and they will be deleted by calling the *pandas.DataFrame.drop()* function. Pandas package allows selecting rows and/or columns using indexing. The index therefore represents the address where a certain piece of information can be found. After the dataset is loaded, an integer index is assigned to each row and each column, starting from 0. If a specific column is desired to be used for indexing, the user provides the name of that column, and the loaded dataset will be altered by the *pd.DataFrame.set_index()* function. If the data set has to be sorted, the names of the columns that will establish the order of the records as well as the sorting mechanism must be mentioned. It is also necessary for the user to specify the names of the columns that hold calendar data for their conversion, which is done through the *pd.DataFrame.to_datetime()* function as it is shown in Figure 3. Character string or numeric data are automatically identified by Pandas.

3.3 Application for Processing and Cleaning Data

The user can choose whether to delete duplicate information, which is done by the *pandas.DataFrame.drop_duplicates()* method. The names of numeric or character string columns that hold categorical variables can be also specified, so that they can be processed efficiently later. The conversion is done through the *pandas.DataFrame.astype()* method, with the "category" parameter. Furthermore, an important step in data preprocessing is data encoding (as it can be seen in Figure 4), given that many ML algorithms can only make use of numerical data. The framework

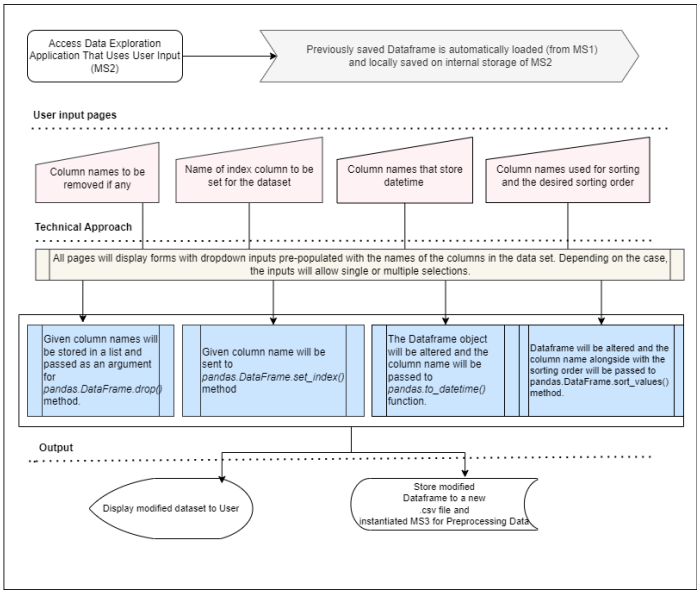


Figure 3. Data insights application using user input

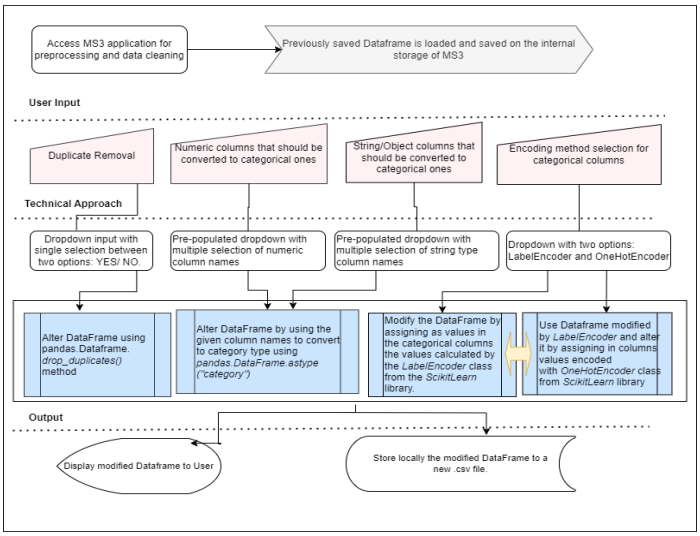


Figure 4. Application for preprocessing and cleaning data

ICI Bucharest © Copyright 2012-2023. All rights reserved

implements two encoding variants, using methods available on the *SciKit-Learn* library: *LabelEncoder* and *OneHotEncoder*, with the mention that the second one needs the first one to be applied. So, the user can choose the desired method, and all the columns that hold category or character string data are transformed. The results are displayed to the user and the data set is stored and can be retrieved by the next service.

3.4 Application for Handling Missing Data

The user can choose the desired option for replacing the missing values, this step being optional. The user fills in the information through dropdown-type inputs from where the user chooses the names of the columns for which operations are performed to complete the missing elements. For each chosen column, the user can choose one of the following methods: deleting the respective rows or column (implemented using the *pandas.DataFrame.dropna()* method), replacing the values with a value provided by the user (implemented using the *pandas.DataFrame.fillna()*, which receives the respective value as a parameter) or replacing the missing values with the mean or median of the values of the respective column calculated with the *pandas.DataFrame.mean()* and *pandas.DataFrame.median()* methods. In addition to these variants, the missing values can be replaced with estimated values. The prediction algorithm uses linear regression as it is illustrated in Figure 5. Another method is to impute values, and it is implemented by using the *SimpleImputer()* model from the *datawig* library. The results are displayed, and the new data set is stored for later use. The advantage of such an implementation approach consists in the fact that the user can return to this service, choosing different methods of handling missing values to re-estimate the involved models.

3.5 Application for Data Standardization

Another particularly important step in data analysis is standardization and/or normalization. The performance of the implemented models is highly dependent on the quality of the employed data, and this step ensures smoothing and scaling of the data. Depending on the method chosen by the user, this service will implement two methods available in the *SciKitLearn* library: *MinMaxScaler()* and *StandardScaler()* as it can be seen in Figure 6. The normalized data set will be saved.

3.6 Application for Outlier Treatment

The *ScikitLearn* library, through the ensemble package, provides the implementation of the *Isolation Forest* algorithm, which only needs the parameter that provides the accepted margin of

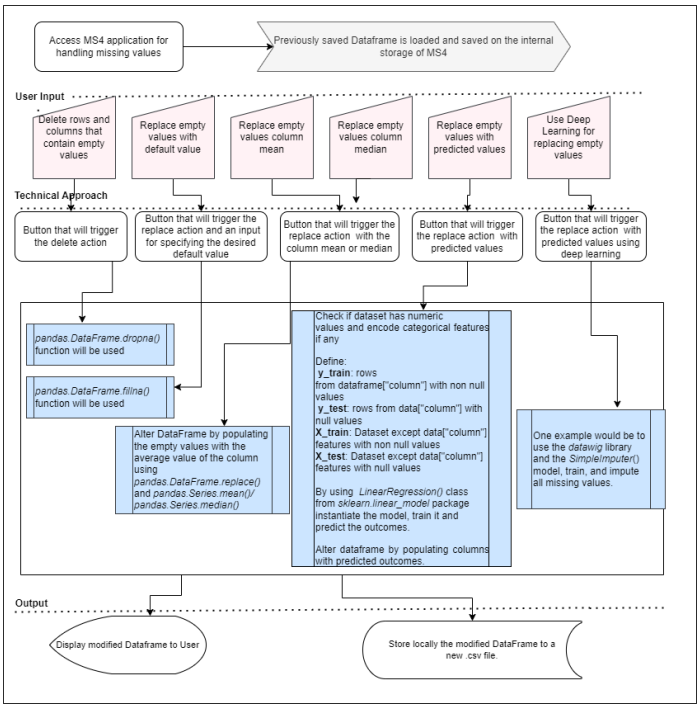


Figure 5. Application for handling missing data

allowed extreme values. The application trains the algorithm on the entire data set, predicts the rows identified as outliers, and alters the data set by removing those rows. Similarly, the Local Outlier Factor algorithm, available in the same library, can be implemented. Furthermore, the algorithm using *Support Vector Machine* (SVM) can be used to classify values as extreme or not. The algorithm can be called through the *OneClassSVM*() class from the *sklearn.ensemble* package.

The interquartile range can also be used to identify such values for each user-specified column, and the implementation of this variant is shown in Figure 7. The altered data set is stored at the level of this service.

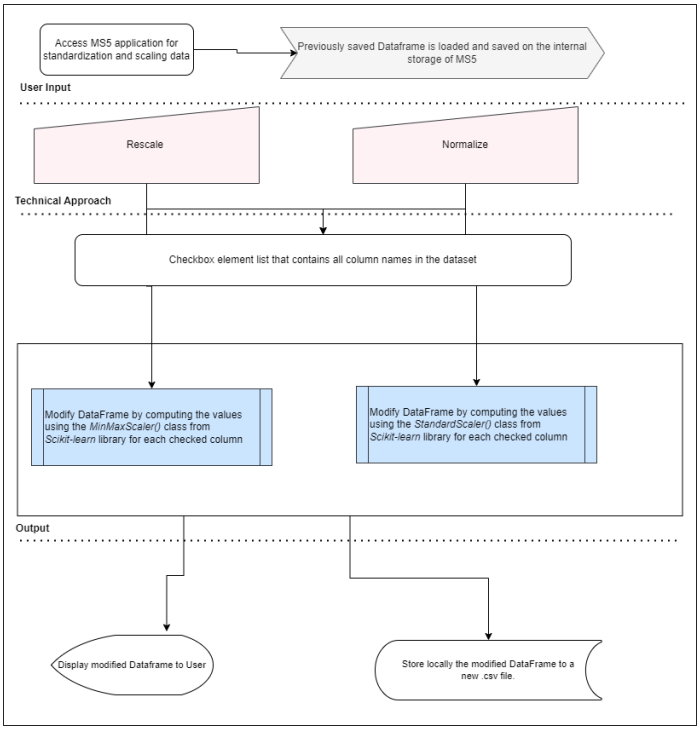


Figure 6. Application for data standardization

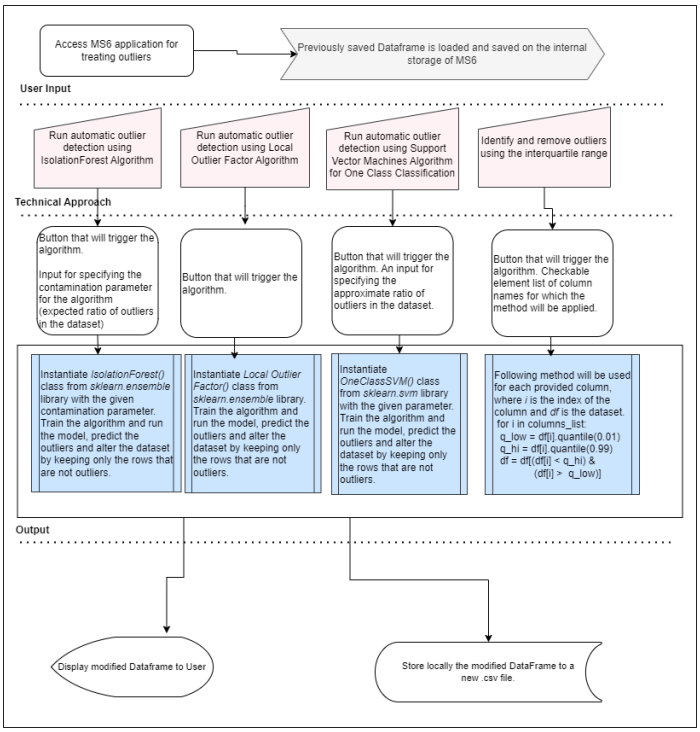


Figure 7. Application for outlier treatment

ICI Bucharest © Copyright 2012-2023. All rights reserved

3.7 Application for Data Filtering and Grouping

The user can filter the data set by specifying the columns and values by which the set will be filtered. Filtering can be approached in three ways: filtering numeric values, string or calendar date columns, or retrieving records within certain ranges determined by row or column indexes. Data grouping will be done by specifying grouping columns and choosing grouping functions from those provided by the application. The grouping will be done using the two functions with the corresponding parameters: *pandas.DataFrame.aggreagte()* and *pandas.DataFrame.groupby()* as it is shown in Figure 8. A similar solution for tuning the pre-processing of data does not exist. However, libraries (such as PyCaret from Python) represent an end-to-end pipeline tool that automates the entire process from data analytics to assessment of the results. Nevertheless, PyCaret insists on the automation of the process and the implementation of multiple ML algorithms. In the pre-processing stage, PyCaret handles the data set and it identifies missing values, outliers, categorical features, data imbalance, etc. and provides a solution to pre-process the data set and prepare it for training. For the missing values, the options are very simple, the null values are replaced by mean, median or zero. Mean is the default option. The other available options are median and zero.

In comparison with the proposed method, PyCaret offers a simple imputation method and does not check the efficacy of combining the solutions for missing values, outliers, encoding and scaling. If the target column is imbalanced, PyCaret imposes SMOTE to balance the data, but this is not always worthy. Another drawback that can be encountered with raw data sets is that it is not easily processed by PyCaret. For instance, columns with an unknown format will not be automatically processed. They previously require elimination or additional processing.

A brief comparison between missing, outliers, encoding and scaling (MOES) and PyCaret is described in Table 1.

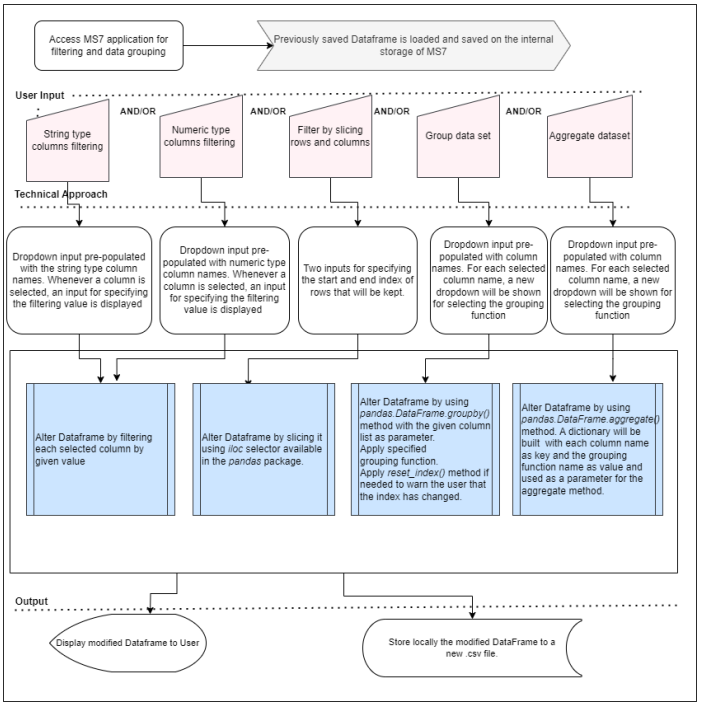


Figure 8. Application for data filtering and grouping

Table 1. Comparison between MOES and PyCaret

Criteria	MOES	PyCaret
Flexible	Yes	No
Architecture	Horizontal	Vertical
Pipeline	Yes	Yes
Advantages	Better results Less time-consuming	Lower involvement of the researcher Lower-level coding
Disadvantages	More coding and involvement of the data scientist	Not capable to handle all data sets (trouble) Lower performance indicators

4. Results

The analyzed data set consists of 20 columns and 18,239 rows with details about the clients of a commercial bank, such as client identifier, name, profession, sex, age, marital status, currency, annual income, annual income in RON, account, credit, date, credit category, credit description, prescoring, savings, fidelity score, requested amount, threshold amount, and contracting probability of a new credit. Out of them 12 columns are numeric, and 8 columns are non-numeric, thus the data set has a heterogenous structure.

Figure 9 describes the missing values as a heatmap (left side) and in percentages (right side) for the analyzed data set. 4,784 (26.23%) and 4,391 (24,07%) out of the 18,239 total values are missing from the prescoring and fidelity column, respectively. Without treating the missing values, the ML algorithm (logistic regression) fails to provide results. In addition, most ML algorithms end in error when they encounter null values.

In this section, several simulations were described. First, for missing values, four methods were implemented with MOES, and the results were compared. The rest of the pre-processing stages were not modified. The settings for the other three pre-processing stages are: the outliers were not approached, *LabelEncoder* and *StandardScaler* were considered for encoding the categorical columns and scaling. The four methods for missing values are: *fillna* with *mean* method, *fillna* with *bfill*, *interpolate* and *mice*.

Initially, PyCaret failed to process the data set as the Date column has an unknown format. The column was eliminated from the raw data set. With MOES, the same column was converted to datetime and then the month and year were extracted, and Date column was finally eliminated. Therefore, PyCaret fails to work properly with problematic data sets and data science has to remove or previously transform them.

As PyCaret performs numerous steps, it requires more time to run, and the results are not always the best ones. For instance, in Table 2, the Area Under the Curve (AUC) performance indicator and the execution time are compared for MOES with four different missing values methods and for PyCaret. Although PyCaret requires only a few lines of code, the execution time for only one basic ML algorithm (linear regression) is much longer in comparison with the execution time for MOES.

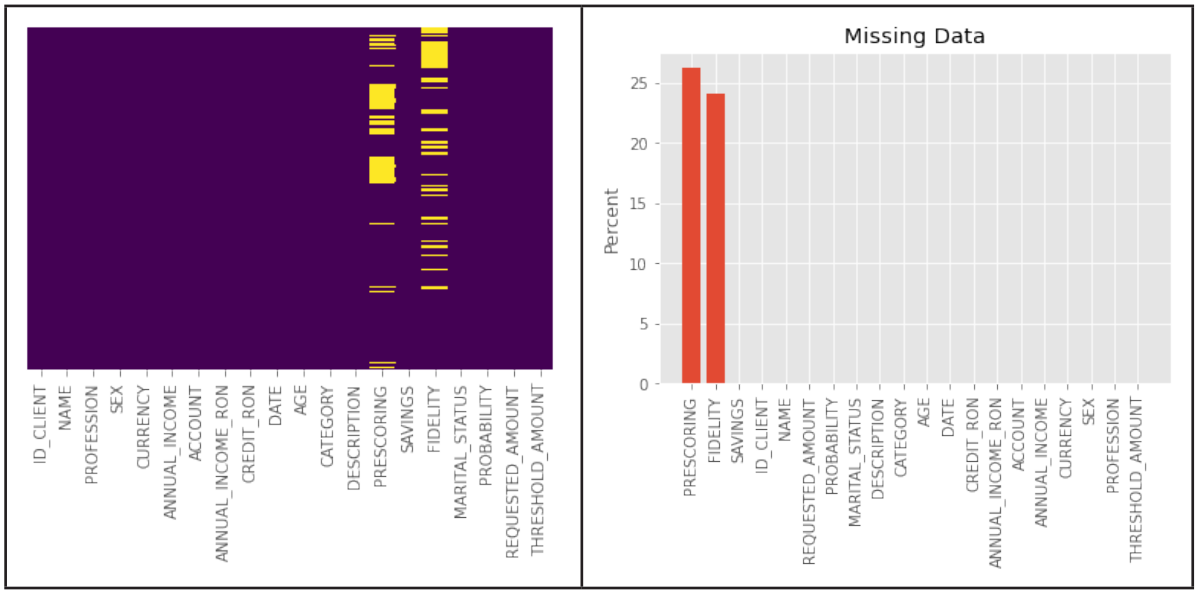
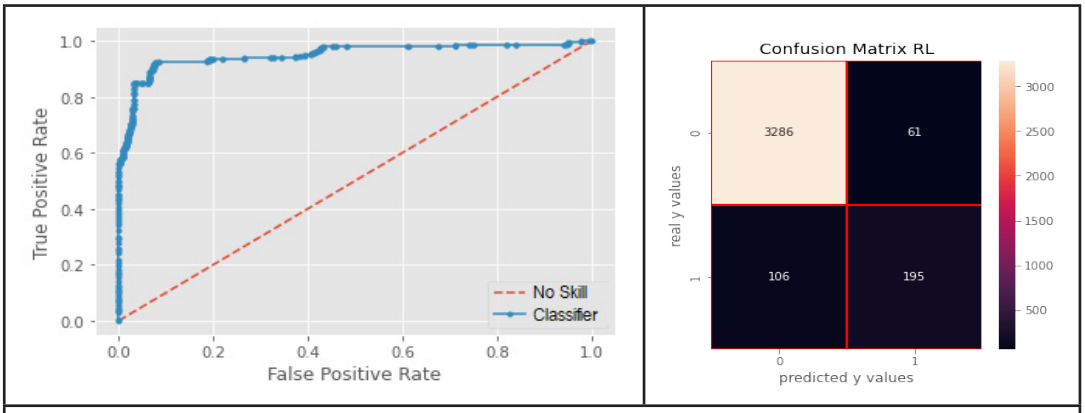


Figure 9. Missing values for the analyzed data set

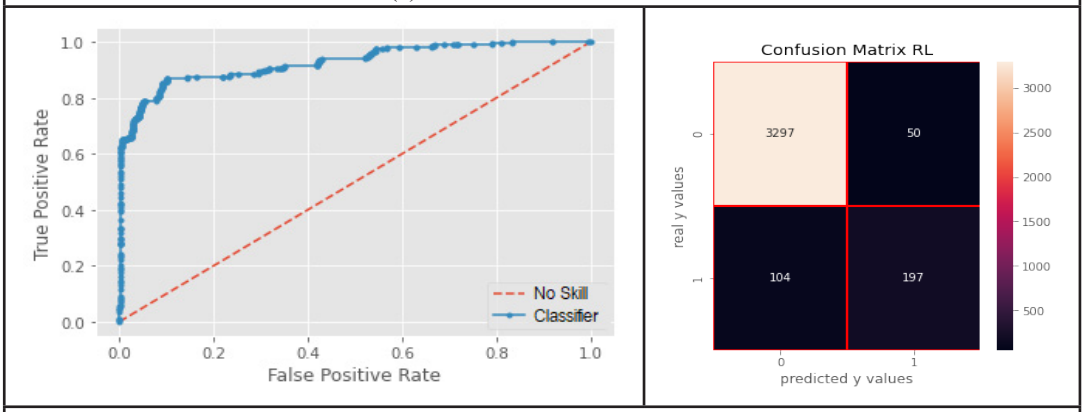
Table 2. AUC and execution time for MOES and PyCaret

	Fillna mean	Fillna bfill	Interpolate	Mice	PyCaret
AUC	0.952	0.928	0.934	0.999	0.986
Execution time	0m 26s	0m 26s	0m 26s	0m 41s	8m 15s

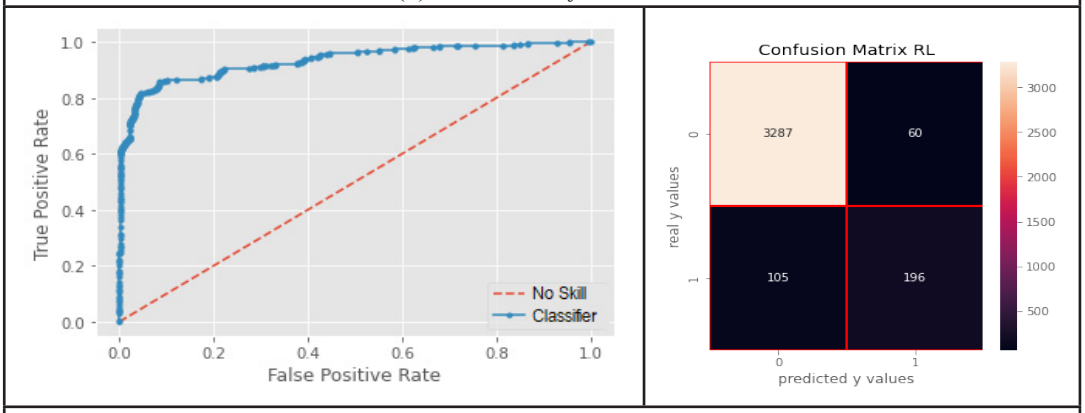
ICI Bucharest © Copyright 2012-2023. All rights reserved



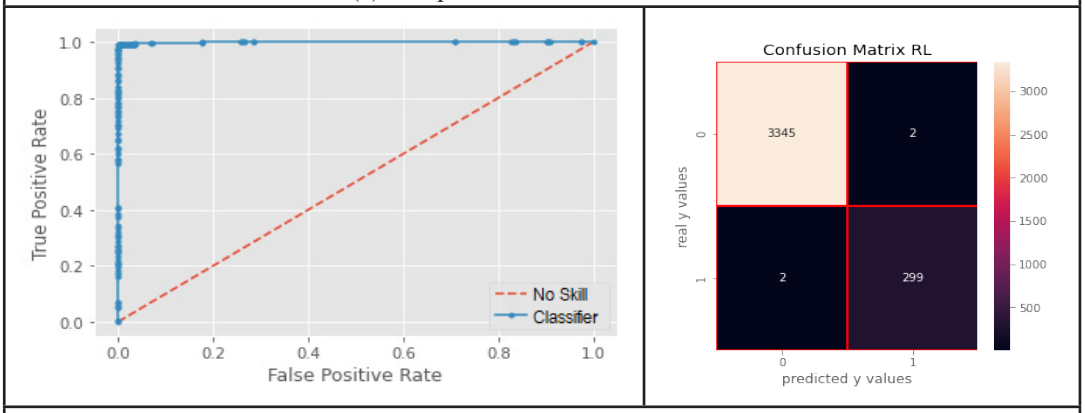
(a) *Fillna* with *mean* method



(b) *Fillna* with *bfill* method



(c) Interpolation with linear method



(d) Multiple imputation with chained equations

Figure 10. ROC-AUC curves and confusion matrices for the four methods for missing values

In Figure 10, the ROC-AUC curves and confusion matrices are presented for MOES, whereas in Figure 11, similar graphs are shown for PyCaret Python library.

The training data set includes 3,347 clients that will not contract a new credit and 301 clients that will contract a new credit from the bank. In the scenario using *fillna* with *mean* method (Figure 10(a)), 61 of the 3,347 clients are mistakenly classified as clients that would contract a new credit. Moreover, 106 of the 301 clients are also mistakenly classified as clients that would not contract a new credit. Similar results are obtained in the second and third scenario using *fillna* with *bfill* method and interpolation (Figure 10(b), (c)). However, the best results are obtained with *mice* method, only 2 of the 3,347 and 2 of the 301 clients are mistakenly classified (Figure 10(d)), which is a very good performance for a model using a classifier. Furthermore, AUC score is almost perfect (0.99).

As it is illustrated in Figure 11, PyCaret split the initial data set into 4,030 clients with no intention to opt for a new credit contract and 348 clients that will contract a new credit. The area under the curve is good (0.99), but the clients who were mistakenly classified are more numerous compared with the case when MOES was implemented.

Thus, the horizontal tuning framework provided the best combination of the pre-processing stages: *mice* for missing values, no method for outliers, *LabelEncoder* for encoding and *StandardScaler* for scaling data.

For outliers, two methods where implemented: *zscore* and elimination of the values that were under or above certain quantiles (for instance: 0.01 and 0.99), but the best results were obtained when no method for outliers was implemented as outliers are acceptable for ML algorithms.

However, for encoding the categorical columns, two methods were used: *LabelEncoder* and *OneHotEncoder.* Similarly to the case of missing values, encoding the categorical columns is mandatory. The first method, *LabelEncoder*, outperformed *OneHoteEncoder*.

For scaling purposes, two methods were implemented, namely *RobustScaler* and *MinMaxScaler*, but *StandardScaler* slightly outperformed them.

5. Conclusion

This paper proposed a horizontal tuning framework using microservices as applications for data pre-processing. It consists in combining methods for missing values, outlier encoding and scaling. For replicability purposes, detailed diagrams of the analyzed applications were provided.

For the tested data set, the horizontal tuning framework provided the best combination of the pre-processing stages in the following configuration: mice method for missing values, no method for outliers, *LabelEncoder* for encoding and *StandardScaler* for scaling data.

Although there is no similar framework, the results obtained with the proposed framework were compared with those obtained with the PyCaret Python library and the conclusion was that the proposed framework provided better results in terms of execution time and area under the curve performance indicator. One thing that should be noticed is that one method could

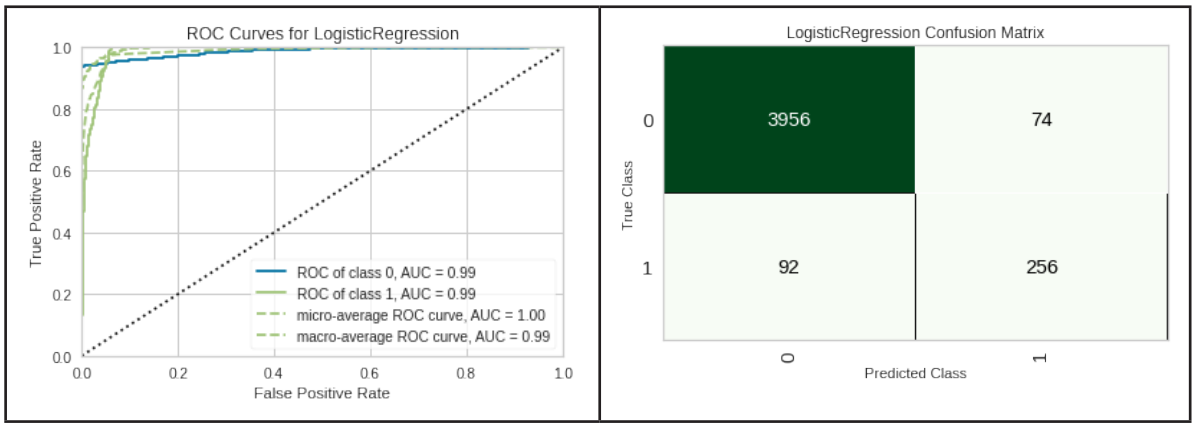


Figure 11. ROC-AUC curves and confusion matrix with PyCaret library

ICI Bucharest © Copyright 2012-2023. All rights reserved

perform best for a certain data set, that is why the horizontal tuning framework using microservices as applications was proposed, namely to combine the pre-processing stages and obtain the best combination for a certain data set.

Furthermore, the proposed method and PyCaret were also applied for another data set that consisted of electricity data for Tunisia used for identifying non-technical losses and found out that the proposed method outperformed PyCaret for both AUC and execution time.

Acknowledgements

This work was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS-UEFISCDI, project number PN-III-P4-PCE-2021-0334, within PNCDI III.

REFERENCES

Abdullah, M., Iqbal, W. & Erradi, A. (2019) Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software*. 151, 243-257. doi: 10.1016/j.jss.2019.02.031.

Alshdaifat, E., Alshdaifat, D., Alsarhan, A., Hussein, F. & El-Salhi, S. (2021) The effect of preprocessing techniques, applied to numeric features, on classification algorithms' performance. *Data*. 6(2): 11. doi: 10.3390/data6020011.

Auer, F., Lenarduzzi, V., Felderer, M. & Taibi, D. (2021) From monolithic systems to Microservices: An assessment framework. *Information and Software Technology*. 137: 106600. doi: 10.1016/j.infsof.2021.106600.

Baškarada, S., Nguyen, V. & Koronios, A. (2020) Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems*. 60(5), 428-436. doi: 10.1080/08874417.2018.1520056.

Bernstein, D. (2014) Containers and cloud: From LXC to docker to kubernetes. *IEEE Cloud Computing*. 1(3), 81-84. doi: 10.1109/MCC.2014.51.

Cinque, M., Corte, R. D. & Pecchia, A. (2022) Microservices Monitoring with Event Logs and Black Box Execution Tracing. *IEEE Transactions on Services Computing*. 15(1), 294-307. doi: 10.1109/TSC.2019.2940009.

Di Francesco, P., Lago, P. & Malavolta, I. (2019) Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*. 150, 77-97. doi: 10.1016/j.jss.2019.01.001.

Fan, C., Chen, M., Wang, X., Wang, J. & Huang, B. (2021) A Review on Data Preprocessing Techniques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data. *Frontiers in Energy Research*. 9: 652801. doi: 10.3389/fenrg.2021.652801.

Garcia Rodriguez, M.J., Rodriguez Montequin, V., Aranguren Ubierna, A., Santana Hermida, R., Sierra Araujo, B. & Zelaia Jauregi, A. (2021) Award Price Estimator for Public Procurement Auctions Using Machine Learning Algorithms: Case Study with Tenders from Spain. Studies in Informatics and Control. 30(4), 67-76. doi:10.24846/v30i4y202106.

Haghgoo, M., Dognini, A., Storek, T., Plamanescu, R., Rahe, U., Gheorghe, S., Albu, M., Monti, A. & Müller, D. (2021) A cloud-based service-oriented architecture to unlock smart energy services. *Energy Informatics*. 4(1). doi: 10.1186/s42162-021-00143-x.

Hamzehloui, M. S., Sahibuddin, S. & Ashabi, A. (2019) A study on the most prominent areas of research in microservices. *International Journal of Machine Learning and Computing*. 9(2), 242-247. doi: 10.18178/ijmlc.2019.9.2.793.

Hannousse, A. & Yahiouche, S. (2021) Securing microservices and microservice architectures: A systematic mapping study. *Computer Science Review*. 41, 100415. doi: 10.1016/j.cosrev.2021.100415.

Himeur, Y., Ghanem, K., Alsalemi, A., Bensaali, F. & Amira, A. (2021) Artificial intelligence based anomaly detection of energy consumption in buildings: A review, current trends and new perspectives. *Applied Energy*. 287(3): 116601. doi:. 10.1016/j.apenergy.2021.116601.

Huang, D., Wang, C.D., Wu, J.S., Lai, J.H. & Kwoh, C. K. (2020) Ultra-scalable spectral clustering and ensemble clustering. *IEEE Transactions on Knowledge and Data Engineering*. 32(6), 1212-1226. doi: 10.1109/TKDE.2019.2903410.

Hustad, E. & Olsen, D. H. (2021) Creating a sustainable digital infrastructure: The role of service-oriented architecture. *Procedia Computer Science*. 181, 597-604. doi: 10.1016/j.procs.2021.01.210.

Jane, V. & Arockiam, L. (2021) Survey on IoT Data Preprocessing. *Turkish Journal of Computer and Mathematics Education*. 12(9), 238-244.

Kakkar, M., Jain, S., Bansal, A. & Grover, P. S. (2018) Combining data preprocessing methods with imputation techniques for software defect prediction. *International Journal of Open Source Software and Processes*. 9(1), 1-19. doi: 10.4018/IJOSSP.2018010101.

Khan, Z., Gul, A., Perperoglou, A., Miftahuddin, M., Mahmoud, O., Adler, W. & Lausen, B. (2020) Ensemble of optimal trees, random forest and random projection ensemble classification. *Advances in Data Analysis and Classification*. 14, 97-116. doi: 10.1007/s11634-019-00364-9.

Lasso-Rodriguez, G., Winkler, K. (2020) Hyperautomation to fulfil jobs rather than executing tasks: the BPM manager robot vs human case, Romanian Journal of Information Technology and Automatic Control [Revista Română de Informatică și Automatică]. 30(3), 7-22. doi:10.33436/v30i3y202001.

Lenarduzzi, V., Lomio, F., Saarimäki, N. & Taibi, D. (2020). Does migrating a monolithic system to microservices decrease the technical debt?. *Journal of Systems and Software*. 169: 110710. doi: 10.1016/j.jss.2020.110710.

Lungu, I., Bâra, A., Căruțașu, G. Pîrjan, A. & Oprea, S.V. (2016) Prediction intelligent system in the field of renewable energies through neural networks. *Economic Computation and Economic Cybernetics Studies and Research*. 50(1), 85-102. WOS: 000372478800005

Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S.T. & Dustdar, S. (2021) Microservices: Migration of a Mission Critical System. *IEEE Transactions on Services Computing*. 14(5), 1464-1477. doi: 10.1109/TSC.2018.2889087.

Oprea, S.-V. & Bâra, A. (2021) Machine learning classification algorithms and anomaly detection in conventional meters and Tunisian electricity consumption large datasets. *Computers & Electrical Engineering*. 94(C): 107329. doi: 10.1016/j.compeleceng.2021.107329.

Schmied, T., Didona, D., Döring, A., Parnell, T. & Ioannou, N. (2021) Towards a General Framework for ML-based Self-tuning Databases. In: *Proceedings of the 1st Workshop on Machine Learning and Systems, EuroMLSys 2021, 26 April, 2021, Online, United Kingdom*. New York, NY, United States, Association for Computing Machinery. pp. 24-30.

Shi, Z., Jiang, C., Jiang, L. & Liu, X. (2021) HPKS: High Performance Kubernetes Scheduling for Dynamic Blockchain Workloads in Cloud Computing. In: *IEEE 14th International Conference on Cloud Computing, CLOUD, 5-10 September, 2021, Chicago, USA*. IEEE. pp. 456-466.

Soldani, J., Tamburri, D. A. & Van Den Heuvel, W. J. (2018) The pains and gains of microservices: A Systematic grey literature review. *Journal of Systems and Software*. 146: 215-232. doi: 10.1016/j.jss.2018.09.082.

Van Aken, D., Yang, D., Brillard, S., Fiorino, A., Zhang, B., Bilien, C. & Pavlo, A. (2021) An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. In: *Proceedings of the VLDB Endowment*. 14(7). pp. 1241-1253. doi: 10.14778/3450980.3450992.

Wang, J., Wang, Y., Li, Z., Li, H. & Yang, H. (2020) A combined framework based on data preprocessing, neural networks and multi-tracker optimizer for wind speed prediction. *Sustainable Energy Technologies and Assessments*. 40: 100757. doi: 10.1016/j.seta.2020.100757.

Zhou, Q. & Ooka, R. (2021) Influence of data preprocessing on neural network performance for reproducing CFD simulations of non-isothermal indoor airflow distribution. *Energy and Buildings*. 230: 110525. doi: 10.1016/j.enbuild.2020.110525.

Zhu, C. & Gao, D. (2016) Influence of data preprocessing. *Journal of Computing Science and Engineering.* 10(2), 51-57. doi: 10.5626/JCSE.2016.10.2.51.

Zimmer, L., Lindauer, M. & Hutter, F. (2021) Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9), 3079-3090. https://doi: 10.1109/TPAMI.2021.3067763.

Žliobaite, I. & Gabrys, B. (2014) Adaptive preprocessing for streaming data. *IEEE Transactions on Knowledge and Data Engineering*. 26(2), 309-321. doi: 10.1109/TKDE.2012.147.

ICI Bucharest © Copyright 2012-2023. All rights reserved