SoftwareX 21 (2023) 101311

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

iSee E^3 — The Explanation Experiences Editor

Marta Caro-Martinez, Jesus M. Darias, Belen Diaz-Agudo, Juan A. Recio-Garcia *

Department of Software Engineering and Artificial Intelligence, Instituto de Tecnologías del Conocimiento, Universidad Complutense de Madrid, Spain

ARTICLE INFO

Article history:
Received 20 October 2022
Received in revised form 2 January 2023
Accepted 9 January 2023

Keywords:
Explainable Artificial Intelligence
Machine Learning interpretability
Restful API
Flask
Behaviour trees

ABSTRACT

Existing XAI libraries offer a good number of explanation and visualization methods. We refer to the combination of these XAI methods and their further customization to meet the actual needs of users as explanation strategies. The Explanation Experiences Editor (E^3) is a software tool that allows capturing, editing, and executing explanation experiences through a user-friendly interface based on behaviour trees. To easily integrate third-party XAI methods, E^3 is supported by a restful web service API that *provides* a unified access layer for them. The platform also provides an online repository to store the AI models and normalize their access by explanation methods.

© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00336
Permanent link to Reproducible Capsule	https://editor-dev.isee4xai.com/ https://explainers-dev.isee4xai.com https://models-tf-dev.isee4xai.com https://models-sk-dev.isee4xai.com
Legal Code License	European Union Public License (EUPL)
Code versioning system used	git
Software code languages, tools, and services used	Python, Flask, JavaScript
Compilation requirements, operating environments & dependencies	Python dependencies are defined for each module. Docker containers are provided to ease deployment.
If available Link to developer documentation/manual	Documentation is included in each github repository
Support email for questions	jareciog@fdi.ucm.es

1. Motivation and significance

Interpretability has become a requirement for trusting AI (Artificial Intelligence) models applied to real-world tasks. AI interpretability refers to the degree to which a human can understand the decision or prediction made by an AI model [1]. Interpretability is crucial in AI Machine learning (ML) models where predictions or behaviours are obtained through the execution of numerous operations on large amounts of data. As a result, the system becomes a black box for its users since the obtention process seems offuscated. ML models with high interpretability help to increase trust in their behaviour and improve their performance. Moreover, the right to obtain an explanation of the decision reached by an AI model is now an official regulation of the European Union [2], thus emphasizing the need for interpretable models.

Although some simple ML models are inherently interpretable, there is often a black box nature and lack of transparency associated with the best performing complex AI models like deep neural networks [3]. This problem has triggered a whole new body of work on Explainable Artificial Intelligence (XAI), a field of research that holds great promise for improving the trust and transparency of AI-ML-based systems [4,5]. Concretely, the software artifacts presented in this paper have been developed by the iSee research project,[1] a European consortium that aims to provide an open platform to reuse successful explanation solutions.

As a result of the XAI research efforts, there are several libraries available that offer a good number of explanations and

* Corresponding author.
E-mail address: jareciog@fdi.ucm.es (Juan A. Recio-Garcia).

[1] http://isee4xai.com

https://doi.org/10.1016/j.softx.2023.101311
2352-7110/© 2023 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

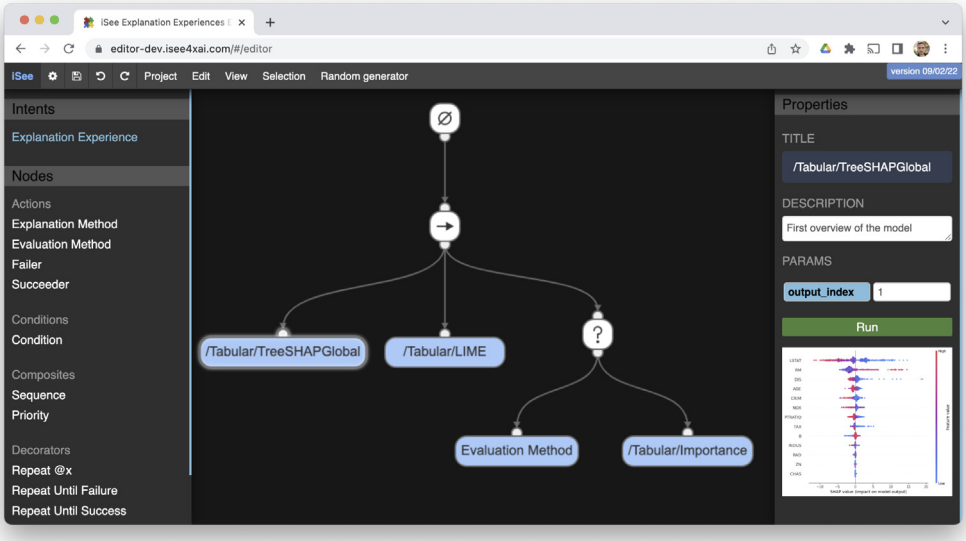

Fig. 1. Snapshot of the E^3 software. Users can use the menu on the left to define the workflow of the explanation experience as a behaviour tree. The right panel shows the details of each node, in this case, the first explanation method has been selected and executed, displaying its output in the bottom-right corner. Live demo available at https://editor-dev.isee4xai.com.

visualization methods. These XAI methods are reusable for different domains and ML models and can be combined and parameterized. However, in most situations, relying on a single explanation method is not enough to provide a complete explanation. Users may require alternative XAI methods depending on their expectations and satisfaction with the previously received explanations. This implies that complete explanation processes should be defined as a workflow containing explanation and visualization methods plus complementary techniques to evaluate user satisfaction. We refer to these workflows as "Explanation Experiences".

Our main goal is to integrate existing XAI methods in the literature, allowing their combination and testing as explanation experiences. However, from a technical point of view, this becomes a challenging task as most XAI methods are scattered across different Python packages. Even well-known multipurpose explainability packages offer only a reduced set of techniques [6] that may not be enough to provide a complete explanation. The solution we propose aims to address these issues by providing an architecture of restful web services that offer common access to different packages of reusable XAI methods for different types of ML models Along with these unified APIs for the access of XAI methods, we present the Explanation Experiences Editor (E^3), a software tool that allows the definition and execution of explanation experiences through a user-friendly interface. While there are several alternatives to define the explanation processes, we propose the use of Behaviour Trees (BTs), as this technique has been broadly applied to represent workflows [7]. This way, the E^3 tool alleviates the task of obtaining model explanations by adding a new abstraction layer between the user and the methods so that no additional code is necessary for the definition, reusing, and testing of explanation experiences.

2. Software description

The E^3 interface is presented in Fig. 1. It allows the definition and execution of behaviour trees representing explanation experiences, where inner nodes are workflow control structures, and leaf nodes represent explanation methods.

Users designing an explanation experience can upload the ML model to be explained and define the workflow of explanation methods to be executed. In our tool, models are stored in the root node of the BT. The control flow is defined using the inner nodes such as sequences and choices, among others. On the other hand, leaf nodes define the configuration of the XAI method to be executed. This way, the editor allows users to specify the configuration parameters for each XAI method for later execution on the selected ML model. There are also evaluation nodes that serve to ask the user for further explanations. Finally, although it is possible to test XAI methods individually, an explanation experience can be executed as a whole to revise its complete behaviour. This tool is supported by several complementary platform service libraries as presented next.

2.1. Software architecture

As illustrated in Fig. 2, there are four different containers in our architecture. The *Explanation Library* provides the XAI methods for the generation of explanations, while the *AI Model Library* contains the methods for the management of the models to be explained. Both of these containers utilize well-known libraries oriented to the development of machine-learning models, such as Scikit-learn [8], TensorFlow [9], Pytorch [10], and so forth. The containers have direct access to a core file storage unit where the models and their additional configuration data are stored. Both libraries provide unified Rest Services APIs to support not only the functionality of the Explanation Experience Editor but also ad-hoc integration with external software. Both APIs have been developed in Python using the Flask [11] Restful microframework.

The *Explanation Library* provides a unified layer to access and execute the explanation methods. These methods are implemented as an API that acts as a wrapper of third-party modules and libraries, following the dependency injection pattern.

To execute these explanation methods our platform also requires an additional module to manage the ML models to be executed and explained. For this task, we have developed the

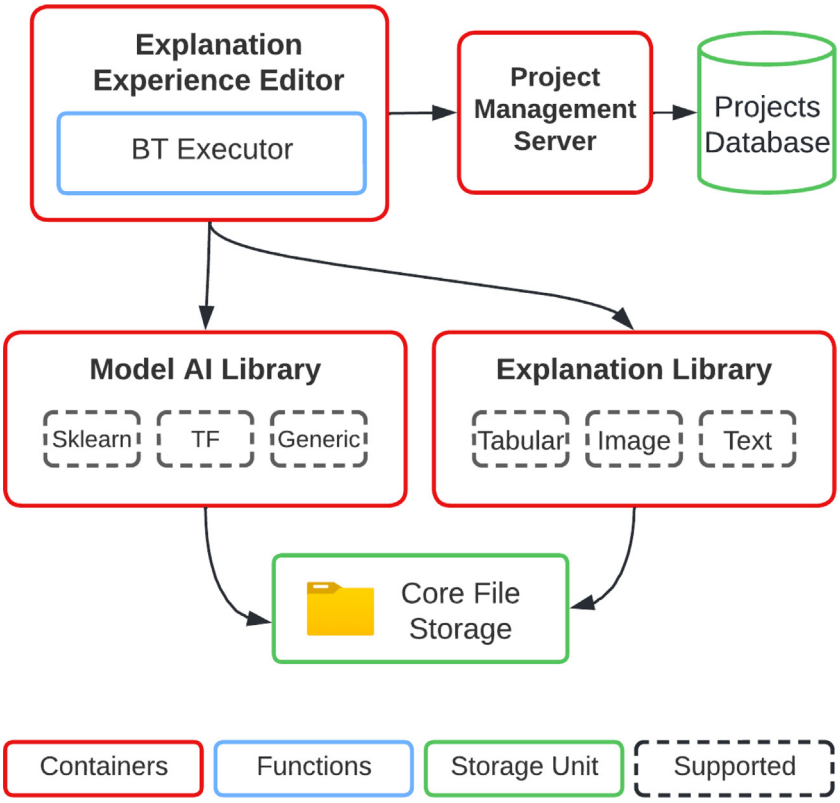

Fig. 2. Software architecture diagram.

AI Model Library, which offers an additional API that allows uploading, updating, and managing the files associated with an ML model, as well as executing them. This module supports models exported from both Scikit-learn and TensorFlow. Thus, the Explanation Library can later access these files to generate explanations.

As for the Explanation Experience Editor, it communicates with the Explanation Library to request the execution of explanation methods. The Editor also requires information from the Model AI module, such as the names or aliases of the existing models. Since the Editor allows users to create and edit multiple BTs in separate projects, it exchanges this data with an additional server dedicated to the management of the projects. This server stores and retrieve all data associated with the saved projects so they can later be imported in the future.

2.2. Software functionalities

The *Explanation Experience Editor* allows creating new projects and managing them. The user can easily create different BTs by dragging and dropping nodes to the building area but it is also possible to randomly generate BTs by specifying certain attributes, such as the minimum number of nodes and the depth of the tree. Additionally, the Editor allows importing and exporting projects, trees, and individual nodes represented as JSON files. One of the main functionalities of the Editor is the BT Executor. While explanation nodes can be executed manually, the BT Executor follows the whole execution flow of a BT and displays the generated explanations according to the specified order to offer a smoother experience to the user. A live demo of this tool is available at: https://editor-dev.isee4xai.com

The Explanation Experience Editor is supported by the functionality provided by both the *AI Model* and the *Explanation* libraries. The API of the *AI Model Library* contains the following methods to manage the AI models[2]:

/upload_model: adds a new model to the core file system or replaces an existing one by using POST or PUT, respectively.

/dataset: uploads a data set to the specified model folder in the core file system when using POST. If using GET, returns the uploaded dataset as a file.

/delete: deletes the model folder associated with the specified model from the core file system.

/info: returns the information related to a previously uploaded model.

/<datatype>/run : executes the prediction function of the model based on the input passed using POST. The supported datatypes are Image, Tabular, and Text.

As for the main functionalities of the *Explanation Library* API[3]:

/Explainers : returns a list with all the available explanation methods when making a GET request.

[2] Source code and documentation: https://github.com/isee4xai/iSeeBackend/tree/dev/AI%20Model%20lib
Testing deployments:
https://models-tf-dev.isee4xai.com
https://models-sk-dev.isee4xai.com.

[3] Source code and documentation: https://github.com/isee4xai/ExplainerLibraries
Testing deployment:
https://explainers-dev.isee4xai.com.

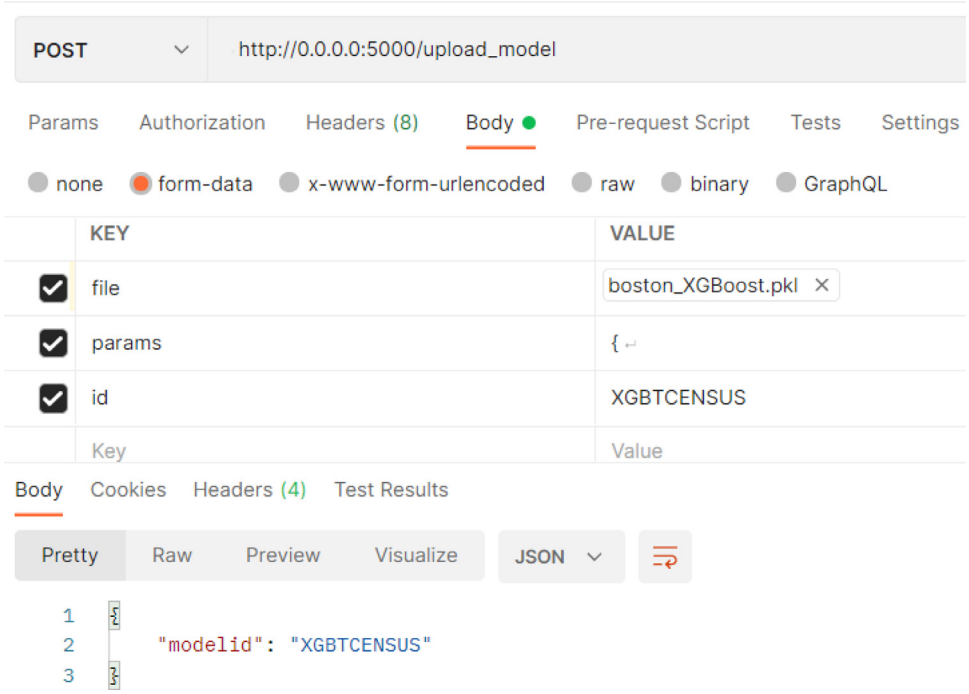

Fig. 3. Illustrative Example. Rest API call to upload a model to the AI Model Library. The uploaded model is identified as *XGBTCENSUS*.

/<*datatype*>/<*explainer_method*> : These are the core resources of the web server, through which is possible to utilize the XAI methods. The supported datatypes are Image, Tabular, and Text. Currently, there are 16 explanation methods available in the API. Making a GET request to a valid resource will return a brief description of the specified method and the required and optional parameters that it receives. Making a POST request providing an identifier for the model and optionally additional parameters will return an explanation in JSON format, as well as an URL to visualize it using the ViewExplanation resource.

/*ViewExplanation*/<*filename*> : returns an image file containing a generated explanation when making a GET request.

3. Illustrative examples

For illustration purposes, we walk through the process of obtaining explanations for an example machine-learning model using the proposed software. The model for this example is a gradient boosting machine based on XGBoost [12]. The purpose of the model is to predict the estimated value of properties based on the Boston Housing Dataset [13]. The whole example is completely detailed in the video tutorial available in the "Help" menu of the E^3 tool.[4]

To explain the underlying behaviour, we use Postman[5] as a tool to reproduce the HTTP requests to the servers.[6]

The first step is to upload the model to the *AI Model Library*. To do this, we make a POST request to the AI Model Server using the **/upload_model** resource, as shown in Fig. 3. It requires several meta-information of the model (nature, feature names, target feature, among others) together with the model dump file exported from either Scikit-learn or Tensorflow. This way, the model is registered in the platform with a unique *id*. In this case, the *id* of the model is *xgbtcensus* as illustrated by 3.

After uploading the model, it is strongly encouraged to upload a pickled file containing the training data when possible. Uploading the training data allows access to a broader range of XAI methods. To do so, send a POST request to the **/dataset** resource indicating the identifier of the model.

Once the model has been uploaded to the AI Model Library, it can be accessed by the Explanation Experience editor as illustrated by Fig. 4. The selection of the model to be explained by the explanation experience being designed can be performed by clicking on the tree's root node. Then a list with all the available models is presented on the right-side panel to allow the user to select the desired one. In our example, we choose the *id* if the model previously uploaded: *xgbtcensus*. Finally, the properties panel of the root node also allows the definition of the query for the model to be explained by local XAI methods.

The definition of the tree is performed by including nodes from the left-side panel. The most relevant nodes are the *Composites* that define the control flow of the tree. Concretely, users can include *Sequence* and *Priority* nodes. The *Sequence* nodes succeed if all of the child nodes also succeed. In contrast, *Priority* nodes succeed if any child nodes succeed. Evaluation methods allow asking the user about his satisfaction with the explanation received until now. This way, the explanation tree being defined in Fig. 4 initially executes the *TreeSHAPGlobal* and *LIME* explanation methods in sequence. Then, the choice node first asks the user if the previous explanations are enough. In case of a negative answer, it executes the following explainer: *Feature Importance*.

Once the explanation experience has been designed it can be exported in JSON format to be executed or deployed externally by calling the Rest Service APIs provided by the *Explanation Library* and the *AI Model Library*. Moreover, and as presented in Fig. 1,

[4] Direct access: https://editor-dev.isee4xai.com/imgs/tutorial.mp4.

[5] Web site: www.postman.com.

[6] A Postman public collection with exhaustive examples of the Rest services API calls to the Explanation Library is available at: https://www.postman.com/collections/196d33e0cad8765cc3f5

Additional user interfaces are being developed for this purpose by the iSee project being out of the scope of this paper.

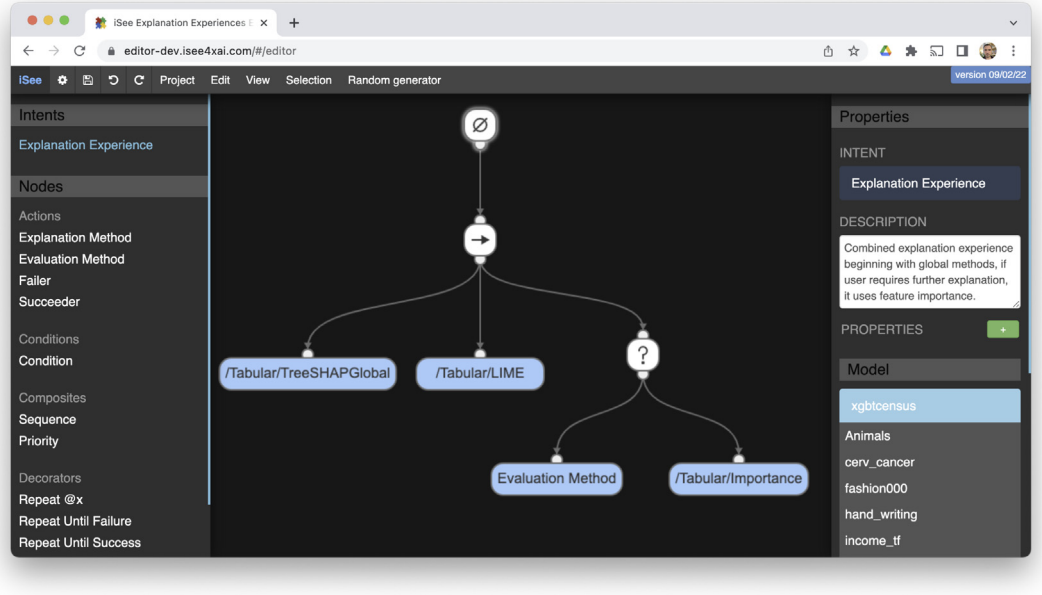

Fig. 4. Illustrative Example (continued). Snapshot of model selection in the E^3 software. This example shows how to select the *XGBTCENSUS* model uploaded to the AI Model Library. Live demo available at https://editor-dev.isee4xai.com.

the execution of an explanation method is very straightforward through the E^3 tool. It only requires selecting the "Explanation Method" node on the right-side panel. It will display the mandatory and optional parameters required by the chosen method. Finally, the execution is performed by clicking on the "Run" button, which calls the Explanation Library and displays the result for visualization below the button.

This user interface hides the details of the manual call to the Rest Service API of the Explainer Library. The underlying process runs as follows:

- Obtaining information about specific methods through the Rest Services API can be achieved by making a GET request to the resource of our choice. The response will contain the necessary information about required and optional parameters.
- To generate an explanation, make a POST request to the same resource passing the identifier of the model, and optionally additional parameters in the *params* attribute as illustrated in Fig. 5 for the TreeSHAP [14] method. The server response is a JSON object which includes the explanation in JSON format and an additional URL to access a visual explanation file.

Concluding our example, the execution of an explanation node such as *TreeSHAP* through a POST request is shown in Fig. 5. In this case, the user is executing the – more complex – local version of the method that also includes the query for the ML model to be explained. It generates a visual chart that is presented to the user as the explanation for that concrete query (presented in Fig. 6). When executing the whole behaviour tree, this explanation will be presented to the user, together with the explanations from the other nodes, following the control flow defined by the composite nodes.

4. Impact

The Explanation Experience Editor and its supporting libraries have been developed under the umbrella of the iSee European research project.[7] Its principal aim is to facilitate the capture, sharing, and re-use of explanation experiences to encourage best practices in explainable AI (XAI). The tools presented in this paper provide an open-access infrastructure that lowers the barrier to experimenting with XAI without committing any software development resources.

In the case of academia, it allows new and established researchers to contribute to the field in a way that gives clarity about the scope and extent of the contributions that their research makes relative to the state-of-the-art captured within the tool. It provides a science-to-technology breakthrough in the form of a means by which models and algorithms can move from the research lab to practical use.

The provision of platform service APIs and the experimental setup of the Explanation Experience Editor is specifically designed to put explainability at the heart of current AI systems with access to an interactive, experimental, and user-centred setup that is trusted by the users.

5. Conclusions

In this work, we presented E^3, the Explanation Experience Editor. This tool allows designing, executing, and reusing explanation experiences that capture successful combinations of explanation methods. It combines a user-friendly interface for designing the explanation experiences with two supporting APIs: the AI model and the Explanation libraries. The former supports the management and execution of ML models, whereas the latter provides unified access to many state-of-the-art XAI methods.

We discussed its architectural design and provided an example illustrating its key functionalities. The software solution we proposed seeks to unify as many XAI techniques as possible by using restful web services APIs with the end goal of bringing explanations one step closer to the users, either researchers or software engineers that work with XAI solutions.

[7] http://isee4xai.com

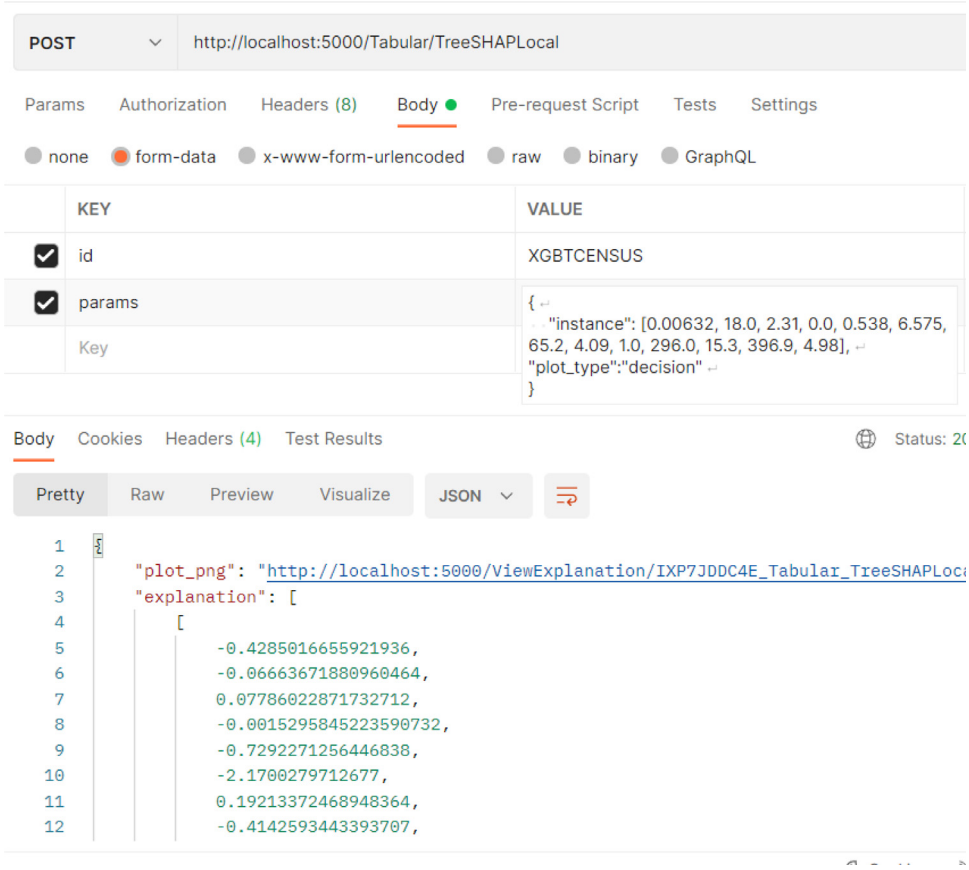

Fig. 5. Rest API call to the Explanation Library for the generation of an explanation using the TreeSHAP explanation method.

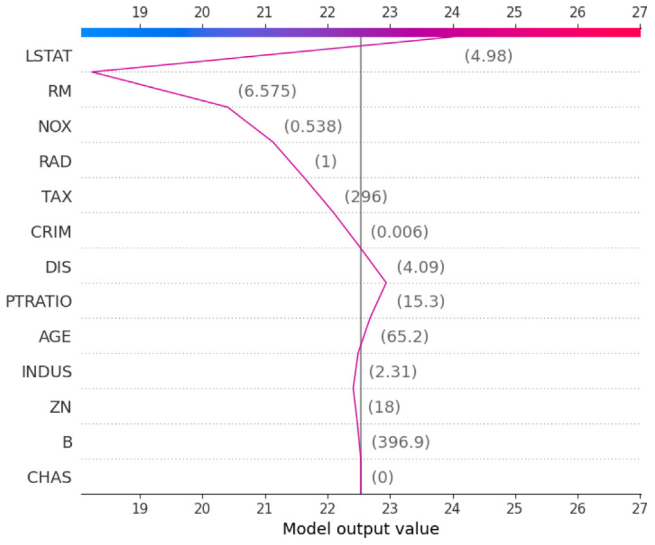

Fig. 6. Illustrative Example (concluded). Resulting explanation generated by the request in Fig. 5 for the TreeSHAP local method that receives the query to the ML model.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This research is a result of the Horizon 2020 Future and Emerging Technologies (FET) programme of the European Union through the iSee project (CHIST-ERA-19-XAI-008, PCI2020-120720-2) funded by MCIN/AEI/10.13039/501100011033 and European Union "NextGenerationEU".

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2023.101311.

References

[1] Molnar C. Interpretable machine learning. second ed.. 2022, URL https://christophm.github.io/interpretable-ml-book.

[2] European Parliament. Artificial intelligence: questions of interpretation and application of international law European Parliament resolution of 20 January 2021 on artificial intelligence: questions of interpretation and application of international law in so far as the EU is affected in the areas of civil and military uses and of state authority outside the scope of criminal justice (2020/2013(INI)). 2021, https://www.europarl.europa.eu/doceo/document/TA-9-2021-0009_EN.pdf.

[3] Hall P, Gill N. An introduction to machine learning interpretability : an applied perspective on fairness, accountability, transparency, and explainable AI. second ed.. Sebastopol: O'Reilly Media, Inc. 2019, URL https://www.oreilly.com/library/view/an-introduction-to/9781492033158/,

[4] Gunning D, Aha D. DARPA's explainable artificial intelligence (XAI) program. AI Mag 2019;40(2):44–58.
[5] Arrieta AB, Díaz-Rodríguez N, Del Ser J, Bennetot A, Tabik S, Barbado A, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Inf Fusion 2020;58:82–115.
[6] Darias JM, Díaz-Agudo B, Recio-García JA. A systematic review on model-agnostic XAI libraries. In: Borck H, Eisenstadt V, Sánchez-Ruiz AA, Floyd M, editors. Workshops proceedings for the 29th international conference on case-based reasoning co-located with the 29th international conference on case-based reasoning (ICCBR 2021), Salamanca (Spain) / Online, September 13-16, 2021. CEUR workshop proceedings, vol. 3017, CEUR-WS.org; 2021, p. 28–39, URL http://ceur-ws.org/Vol-3017/96.pdf.
[7] Iovino M, Scukins E, Styrud J, Ögren P, Smith C. A survey of behavior trees in robotics and AI. Robot Auton Syst 2022;154:104096. http://dx.doi.org/10.1016/j.robot.2022.104096, URL https://www.sciencedirect.com/science/article/pii/S0921889022000513.
[8] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[9] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015, Software available from tensorflow.org. URL https://www.tensorflow.org/.
[10] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché Buc F, Fox E, Garnett R, editors. Advances in neural information processing systems. Vol. 32. Curran Associates, Inc. 2019, p. 8024–35.
[11] Grinberg M. Flask web development: developing web applications with python. O'Reilly Media, Inc; 2018.
[12] Chen T, Guestrin C. XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY, USA: ACM; 2016, p. 785–94. http://dx.doi.org/10.1145/2939672.2939785, URL http://doi.acm.org/10.1145/2939672.2939785.
[13] Chen JM. An introduction to machine learning for panel data. Int Adv Econ Res 2021;27(1):1–16.
[14] Lundberg S, Erion G, Lee S-I. Consistent individualized feature attribution for tree ensembles. 2018.