International Journal of Data Science and Analytics (2023) 16:455–479
https://doi.org/10.1007/s41060-023-00417-5

REGULAR PAPER

AssistML: an approach to manage, recommend and reuse ML solutions

Alejandro Gabriel Villanueva Zacarias[1,2] · Peter Reimann[1,2] · Christian Weber[2] · Bernhard Mitschang[2]

Received: 13 May 2022 / Accepted: 15 June 2023 / Published online: 17 July 2023

Abstract
The adoption of machine learning (ML) in organizations is characterized by the use of multiple ML software components. When building ML systems out of these software components, citizen data scientists face practical requirements which go beyond the known challenges of ML, e. g., data engineering or parameter optimization. They are expected to quickly identify ML system options that strike a suitable trade-off across multiple performance criteria. These options also need to be understandable for non-technical users. Addressing these practical requirements represents a problem for citizen data scientists with limited ML experience. This calls for a concept to help them identify suitable ML software combinations. Related work, e. g., AutoML systems, are not responsive enough or cannot balance different performance criteria. This paper explains how AssistML, a novel concept to recommend ML solutions, i. e., software systems with ML models, can be used as an alternative for predictive use cases. Our concept collects and preprocesses metadata of existing ML solutions to quickly identify the ML solutions that can be reused in a new use case. We implement AssistML and evaluate it with two exemplary use cases. Results show that AssistML can recommend ML solutions in line with users' performance preferences in seconds. Compared to AutoML, AssistML offers citizen data scientists simpler, intuitively explained ML solutions in considerably less time. Moreover, these solutions perform similarly or even better than AutoML models.

Keywords Meta-learning · Machine learning · AutoML · Metadata · Recommender systems

1 Introduction

The lack of experienced data scientists [13] has prompted the emergence of the role of citizen data scientists. Citizen data scientists are practitioners with in-depth understanding of the use case and with basic, but still limited machine learning (ML) knowledge [16]. They are tasked with the development of ML solutions for specific use cases. We use the concept ML solution to refer to the combination and configuration of ML tools and software to perform, e. g., data collection, data preprocessing, model training, and model deployment [36].

Because many options of ML tools and software components exist for an ML solution [27, 35], it is complex for citizen data scientists to determine if a given combination of components is suitable for a new use case. Citizen data scientists face additional practical requirements when developing ML solutions. These practical requirements go beyond the known challenges of ML, e. g., data engineering or parameter optimization, and span over the complete development process. For instance, citizen data scientists are encouraged to reuse components from existing ML solutions to reduce development time and costs. They are then expected to quickly identify ML system options that strike a suitable trade-off across multiple performance criteria. These options also need to be understandable for non-technical users, i. e., recommended ML solutions should explain their suitability for the given use case. Addressing these practical requirements represents a problem for citizen data scientists with limited ML experience. This calls for a method that helps them identify suitable ML software combinations and that thereby meets all these requirements.

✉ Peter Reimann
Peter.Reimann@gsame.uni-stuttgart.de

Alejandro Gabriel Villanueva Zacarias
al.villanueva@gmx.de

Christian Weber
c.ma.weber@gmail.com

Bernhard Mitschang
Bernhard.Mitschang@ipvs.uni-stuttgart.de

1 GSaME, University of Stuttgart, Nobelstraße 12, 70569 Stuttgart, Germany

2 IPVS, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany

Different approaches, i. e., AutoML [32], Meta-learning [32], and Explainable AI [8], have been proposed to tackle the problem of selecting and configuring software components for ML solutions. Yet, they address the above requirements only partially (see Sect. 3). For instance, AutoML is a time-consuming and thus non-responsive process that is known for producing hard-to-interpret black box models [43]. Furthermore, it does not consider multiple performance criteria and trade-offs between them when optimizing ML models.

This paper presents AssistML: a concept to recommend ML solutions for predictive use cases, e. g., for classification or regression. AssistML targets citizen data scientists, enabling them to apply ML solutions without involving more experienced data scientists. It offers an alternative to related approaches such as AutoML and thereby addresses all practical requirements mentioned above. AssistML uses the new use case data and performance preferences provided by citizen data scientists as input. It matches these inputs to metadata about existing ML solutions to facilitate the selection and configuration of ML components that are suitable for the use case data and fulfill the performance preferences. Moreover, AssistML offers intuitive explanations of the recommended ML solutions via a recommendation report.

We implemented and evaluated AssistML based on two exemplary use cases. This evaluation shows that AssistML delivers ML solutions that meet the user's preferences in a considerably fast time. Moreover, the recommended ML solutions clearly show their trade-offs across multiple performance criteria and provide concise, relevant, and interpretable information to the users.

The contents of this paper are an extended version of a previous publication [37]. Concretely, this paper covers the following major extensions:

- We provide further explanations on the metadata model defined by the repository. Furthermore, we discuss how this metadata may be collected with existing tracking platforms, and we introduce a procedure to aggregate and enrich the collected metadata. These details improve the reproducibility of our concept. They enable the integration of metadata from other platforms, so that they can be used by AssistML.
- We also expand our evaluation approach by including two additional evaluation metrics. This confirms the ability of AssistML to provide suitable recommendations and to precisely predict their performance for new use cases across multiple performance metrics.
- We add a completely new step in our evaluation approach that compares the results obtained with AssistML against those obtained from an AutoML system. The comparison is with regard to the obtained performance, the required time to obtain ML models and the complexity of those models. The results for the two use cases demonstrate the utility of AssistML as a more efficient, more transparent alternative to AutoML systems for citizen data scientists.

The remainder of this paper is structured as follows: Sect. 2 presents an application scenario and the results of a literature review to derive requirements for a system recommending ML solutions. With respect to these requirements, the paper discusses related work in Sect. 3. Then, Sect. 4 details the metadata required by our concept, as well as the necessary preprocessing to obtain it. Section 5 builds upon the definitions of the previous section and presents AssistML. Section 6 discusses the evaluation results we obtained with our implementation, as well as its comparison against an AutoML system. Finally, Sect. 7 concludes the paper.

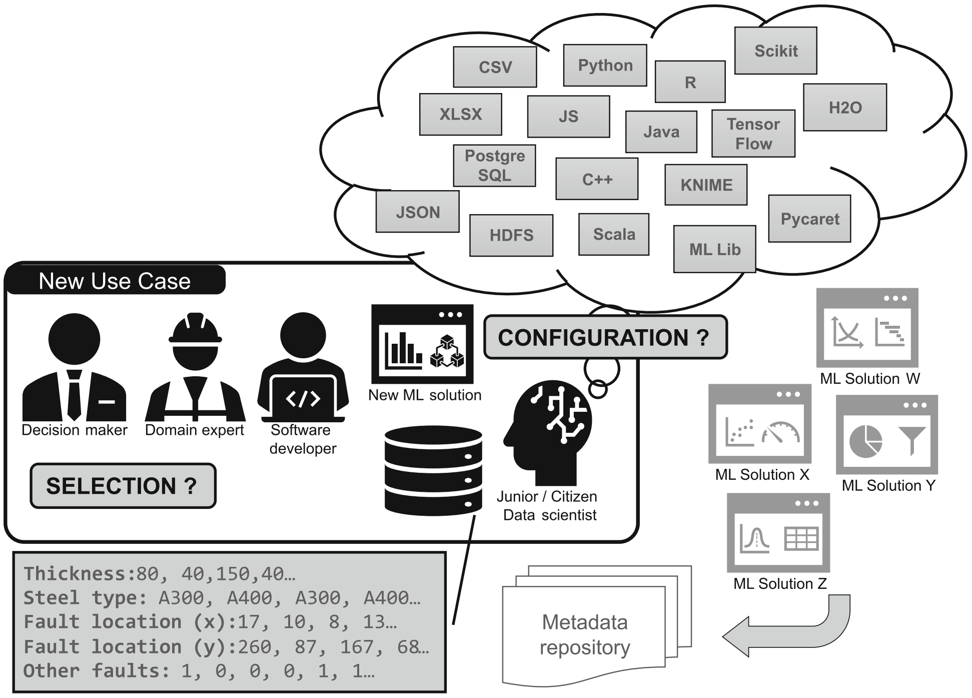
2 Application scenario for ML solution recommendations

The first part in this section describes the exemplary application scenario in which practitioners benefit from recommended ML solutions. The second part discusses the practical requirements for a concept to provide such recommendations.

2.1 Reusing ML solutions for predictive use cases

Figure 1 depicts an exemplary scenario where an ML solution is to be developed for a new use case. A team of practitioners is assigned to this task, e. g., a decision-maker, domain expert, software developer, and citizen data scientist. The team has access to the data of the new use case and to a repository populated with metadata about existing ML solutions, some of which have been developed for similar use cases. Such repositories are part of various ML tools, e. g., MLflow [45], ModelDB [34], MMP [39] or OpenML [33]. So, they are available in many ML projects and organizations.

However, reusing ML solutions from the repository proves to be difficult. The citizen data scientist knows that there exist different languages, libraries, data formats, and deployment models to develop the new ML solution. Yet, besides his or her basic ML knowledge, s/he does not know which of the existing ML solutions were developed for a similar use case and for similar data. Thus, s/he has to inspect each individual ML solution in the repository and determine whether the combination of software components can be reused in the new use case. For example, s/he has to verify that the software components can process the new use case data. S/He also has to validate that the performance of each existing ML solutionmeets the requirements of the new use case. This implies the consideration of performance trade-offs, e. g., preferring ML solutions with a faster prediction speed over others with higher accuracy.



```
Thickness:80, 40,150,40…
Steel type: A300, A400, A300, A400…
Fault location (x):17, 10, 8, 13…
Fault location (y):260, 87, 167, 68…
Other faults: 1, 0, 0, 0, 1, 1…
```

Fig. 1 Application scenario for recommendations of ML solutions

2.2 Practical requirements

The application scenario illustrates the complexity of reusing ML solutions and the human effort involved. To address this problem, citizen data scientists need concepts that recommend suitable ML solutions. This subsection discusses four key practical requirements that a concept for recommending ML solutions must fulfill. These requirements are based on a literature review and on discussions with industry partners.

[R_1] *Reusability* As the number of developed ML solutions in an organization increases, it is more likely that existing ML solutions have configurations similar to the solution needed in the new use case. Citizen data scientists may thus want to reuse those existing ML solutions [23, 39]. This can mean either to reuse the implemented ML solution *as-is* or to consider the combination of ML components and their configurations as *blue-print recommendation* for a new implementation. To address this consideration, a common set of metadata about each ML solution has to be stored in a central metadata repository. This metadata needs to ensure the complete reproducibility of the ML solution [40], so that the recommendation is applicable in new use cases.

[R_2] *Explainability* Recommended ML solutions should be interpretable [2, 8, 11]. The lack of experienced ML experts increases the need for explanations that are especially tailored to non-technical citizen data scientists [2, 41]. Citizen data scientists usually prefer understandable ML solutions with acceptable performance over complex black-box ML solutions with state-of-the-art performance [11, 23]. To address this consideration, meaningful and easy-to-understand explanations have to be generated for each recommended ML solution. Meaningful explanations focus on the effects of ML solution components on prediction results instead of focusing on standard ML metrics [2]. They facilitate citizen data scientists to select a recommendation out of many that match his or her performance preferences [11].

[R_3] *Responsiveness* The process to generate recommendations should be responsive [2, 4, 13, 45]. ML solution-development is an iterative process. Thereby, citizen data scientists test different hypotheses to assess whether an ML solution fulfills the use case needs [4, 13]. A recommendation system hence should suggest several ML software combinations to citizen data scientists, so that they may compare the solutions and select the most suitable one [45]. Throughout this iterative search, citizen data scientists' preferences develop too, as they narrow down suitable ML solutions. However, the numbers of iterations and of ML solutions citizen data scientists may assess are constrained by the resources allocated to the use case [13]. In order to reduce time and costs for the search of ML solutions, it is important to provide recommendations in a responsive manner.

[R_4] *Multi-criteria trade-offs* In practical use cases, the suitability of an ML solution depends on multiple factors instead of a single ML quality metric, e. g., only accuracy or RMSE [38]. For example, a very accurate ML solution may not be useful if it takes too long to make predictions or if it requires a lot of data and complex data preprocessing [7, 19]. In that case, a less accurate, but faster ML solution that requires less data can be a better recommendation [2, 4, 11].

Table 1 Assessment of related approaches

	AutoML	Meta-learning	Explainable AI
[R_1] Reusability	○	●	○
[R_2] Explainability	○	○	◐
[R_3] Responsiveness	○	○	◐
[R_4] Multi-criteria trade-offs	○	◐	○

○ Not fulfilled ◐ Partially fulfilled ● Fulfilled

Similarly, excessive computing infrastructure investments or many software dependencies may render a complex ML solution unsuitable for the new use case. It is important to generate recommendations with consideration of multiple criteria and objectives [20]. Furthermore, citizen data scientists have to be provided with useful information that enable them to identify and compare the trade-offs between individual recommended solutions.

3 Related work

This section discusses three related approaches for ML solutiondevelopment: AutoML systems, Meta-Learning, and Explainable AI. The related work assesses whether each approach fulfills the practical requirements introduced in Sect. 2.2. Table 1 shows a summary of the analysis.

3.1 AutoML systems

Automated machine learning (AutoML) systems generate an optimized supervised learning model or pipeline [12, 43]. The final ML model is optimized to, e. g., minimize generalization error or maximize accuracy. Open source examples of AutoML systems include Auto-sklearn [12] or TPOT [22]. For each new use case, AutoML systems search for high-performing ML models in a predetermined configuration space, consisting of learning algorithms, (hyper-)parameters and basic feature engineering [43]. So, AutoML systems are not based on existing ML solutions or on metadata describing these solutions in a repository. Therefore, they do not fulfill requirement R_1.

AutoML systems also do not fulfill R_2. They tend to produce complex ML models that citizen data scientists usually fail to interpret. They even prefer complex to simpler models if the complex models are only marginally better in the considered optimization objective [43]. Furthermore, AutoML systems do not offer adequate explanations to citizen data scientists.

AutoML systems are not responsive at all (R_3), as they are known to be resource- and time-intensive [43]. Even if the system is constrained by a time budget, the runtime to obtain a single ML model can take up to 60 min [12]. Citizen data scientists are usually not willing to wait this long time, i. e., they want to assess different configurations of ML solutions in a faster pace.

Fulfilling R_4 requires the consideration of multiple criteria. AutoML systems do not support this by design, since optimization strategies are executed to improve the value of one single evaluation metric, e. g., cross-validation loss [12] or area under the operator curve [14]. So, it is not possible to consider any trade-offs between several optimization criteria with AutoML systems.

3.2 Meta-learning

Meta-learning approaches can be used to recommend ML models or their settings for a new task and data set [24]. Metadata about the configuration, used training data, and observed performance of already existing ML models form the basis for meta-learning approaches [32]. Several platforms have been proposed to collect these metadata [33, 39, 45]. These platforms collect metafeatures about the data sets, the learning task, e. g., classification, the used (hyper-)parameters, and data preprocessing techniques. In addition, they store results of evaluation metrics. Meta-learning platforms thus fulfill R_1.

For meta-learning approaches to fulfill R_2, they must explain why an ML model suits the new data set. Explanations have to avoid the use of ML metrics that non-expert citizen data scientists cannot understand. Most meta-learning approaches base their selection on complex metrics, e. g., covariances of text feature pairs [26] or data complexity measures [6]. Using these metrics, each approach estimates the similarity between the new data set and the data sets used for existing ML models. However, it is hard for citizen data scientists to understand these complex metrics and to reproduce how they determine the similarity of data sets. Thus, R_2 remains unfulfilled.

Regarding requirement R_3, meta-learning approaches are expected to find various suitable ML models in a short time. Yet, meta-learning approaches do not adequately meet this requirement, as they are limited to a specific type of algorithm [6, 26, 31]. For example, the approach of Raina et al. [26] is designed exclusively for text classification with logistic regression. Similarly, the approach of Biondi and

Prati [6] is designed only for support vector machines. To efficiently use such meta-learning approaches independently of the underlying ML algorithm, a citizen data scientist has to invest additional time. To this end, s/he needs to execute and compare different meta-learning approaches in terms of their predictive quality and performance. As this is error-prone and time-consuming, R_3 is not fulfilled.

Meta-learning approaches evaluate performance towards a single performance metric at a time, usually accuracy [6] or classification error [26]. Data similarity can only be considered indirectly in the selection of suitable ML models [6]. Thus, meta-learning approaches may not consider more than one performance metric and one similarity metric at a time. They only partially fulfill the consideration of multiple criteria and trade-offs between them, as required by R_4.

3.3 Explainable AI

Explainability approaches can either describe the general functioning of an ML model [8], i. e., give *global* explanations, or the reasoning behind an individual prediction, i. e., give *local* explanations. Since ML solution development is guided by the overall behavior of the solution, the discussion focuses on global explanations. Here, model-agnostic global explainers can be used with different supervised learning algorithms [8]. These kinds of explainers, however, only use the data set of the ML model they are applied to. They do not establish comparisons with other data sets or with other ML models. Thus, R_1 is not satisfied.

Regarding the explainability requirement (R_2), model-agnostic, global explainability approaches provide two types of explanations. The first type are visualizations [1, 15], e. g., partial dependency plots. These approaches plot the effect of one or multiple data features on the model's performance, e. g., on accuracy. Most plots are only able to display one or two data features at a time, rendering them impractical for ML models with many data features. The plots also display metrics such as obscurity [1] in unfamiliar formats, e. g., in various diverging and falling curves. These plots are difficult to interpret for non-experts. The second type of explanations are relevance metrics. These metrics quantify the impact that the presence or absence of a data feature has on the model. Example metrics are attribute interactions [18] or attribute weights [30]. Similar to the plots, the interpretation of these metrics is difficult for non-experts. In both cases, the interpretation of ML models depends on the use of explainability concepts, which have to be interpreted as well. Citizen data scientists however usually do not completely understand these explainability concepts. Thus, explainability approaches only partially fulfill R_2.

To fulfill the responsiveness requirement, explainability approaches must provide explanations in short execution times. However, approaches such as *leave-one-out* [8] or Explainer global [30] need to iterate multiple times over the data features. This is time-consuming when a big amount of features needs to be handled. Therefore, explainability approaches partially fulfill R_3.

Finally, none of the approaches allows the consideration of multiple criteria when offering explanations. Explanations are offered for one performance aspect at a time. For instance, GFA plots obscurity versus accuracy [1]. In ASTRID, attribute interactions are computed with respect to accuracy [18]. These approaches do not allow the consideration of trade-offs, e. g., between accuracy and training time. Thus, R_4 is not fulfilled.

4 AssistML metadata repository

AssistML provides its recommendations based on previously developed ML solutions from a repository. The repository contains source code, data sets, and metadata from previous development projects. Source code comprises all scripts and configuration files needed to execute the ML solution. Data sets refer to the employed training and test data samples. Both kinds of resources are defined and created during development projects, i. e., their structure and contents are not restricted by the repository. They are stored for later reuse with their characteristics made visible via the metadata of the solutions.

This section discusses the nature of the metadata required by AssistML in three parts. The first part specifies the metadata fields required to document ML solutions for AssistML. The second part discusses the methods to collect this metadata, e. g., with existing tracking platforms. The third part describes how this metadata is enriched in preparation to their use by AssistML.

4.1 Metadata model

In this section, we describe the metadata fields that AssistML requires to recommend ML solutions. Let m_i be the set of metadata describing the i-th ML solution in the metadata repository M:

$$m_i = \{id_i, u_i, d_i, s_i, p_i\} \qquad (1)$$

This ML solution receives an identification code id_i. This code describes the ML solution in three parts: (1) the learning algorithm with an abbreviation (e. g., "DTR" for decision trees), (2) the use case (e. g., "faultdetection"), and (3) a sequence number to distinguish between several solutions with the same combination of learning algorithm and use case. This for instance results in the code "*DTR-faultdetection-001*". Moreover, each instance m_i includes

Table 2 Exemplary metafeatures by feature type

Feature type	Example metafeatures
Numerical	Number of outliers, number of missing values
Categorical	Number of categories, imbalance ratio (count of the most frequent category over the count of the least frequent category)
Datetime	Minimum and maximum deltas between chronological timestamps, frequencies of months and days of the week
Unstructured text	Relative vocabulary size, Shannon's entropy [3]. A *bag-of-words* representation [28] with minimal text preprocessing is required (word-tokenization, lowercasing, stopword and punctuation removal)
Image	Image size (width, height), number of (color) channels (RGB, depth information), average number of objects/regions of interest per image
Time series	Uni-variate versus multi-variate, time series length(s), number of channels, average number of outliers, type of elements/behavioral attribute (e. g., numerical, categorical, datetime, images), type of contextual attribute (e. g., datetime, equidistant time points)

four metadata subsets u_i, d_i, s_i, p_i that are defined as follows:

$$u_i = \{taskType_i, taskOutput_i, deployment_i\} \tag{2}$$

$$d_i = \{allMF_i, singleMF_i, preprocessing_i, featsUsed_i\} \tag{3}$$

$$s_i = \{language_i, platform_i, algorithm_i, hParams_i, settLabels_i\} \tag{4}$$

$$p_i = \{metrics_i, metricsLabels_i, margins_i\} \tag{5}$$

The *use case set* $\mathbf{u_i}$ firstly describes the *type of analytics task* the ML solution m_i performs. This This refers to supervised learning tasks, e. g., binary or multi-class classification as well as regression [29]. The *task output* indicates the kind of result the ML solution produces, e. g., whether it delivers single predictions or class probabilities for a new observation. Finally, the *type of deployment* describes the options of how the solution may be used in production environments. Examples can be deploying the ML solution in a cluster or as a stand-alone program on a single host.

In parameter $allMF_i$, the *data subset* $\mathbf{d_i}$ contains a list of descriptive metrics, i. e., metafeatures [32] about the complete data set for which an ML solutionis developed. Different data sets can be compared on the basis of these metafeatures. They describe the original data set prior to the data preprocessing required by the ML solution. Examples include the total number of instances, the number of data features, and percentages of the shares of specific feature types in the data. In AssistML, each data feature can belong to one of the following feature types: numerical, categorical, datetime, unstructured text, image data, or time series data. These feature types are common in various application domains, e. g., manufacturing, human resources, logistics or finance.

AssistML computes metafeatures for these feature types as a data set is either used to look for recommendations or is added to the repository. $singleMF_i$ is a vector of length j that describes each data feature j with these additional metafeatures specific to its feature type. Table 2 gives examples of the metafeatures that can be computed for each feature type. For instance, numerical features may be further described by the number of outliers or the number of missing feature values, while categorical features may have metafeatures for the number of valid categories or for an imbalance ratio. Note that the list in Table 2 is not exhaustive and thus metafeatures can be added or removed for each feature type.

Metadata in $preprocessing_i$ describe the used data preprocessing techniques. This is subdivided into a separate description for each feature type in d_i, as shown in Eq. 6 with $numData_i$ for the numerical feature type, $catData_i$ for the categorical type, $timData_i$ for the type datetime, $unsData_i$ for unstructured text, $imgData_i$ for image data, and $tsData_i$ for time series data. For each feature type, two lists indicate the sequence of techniques applied for the ML solutionto be able to read the data ($featEncoding_i$) and to select features ($featSelection_i$) (see Eq. 7 for the numerical feature type). An example content of $featEncoding$ is a list with the two values $[MeanImputation, StandardScaler]$. This indicates to first impute missing values with the mean value and then to scale all feature values into a range from 0 to 1. Afterwards, $featSelection$ may contain the list $[PCA\_Preserve80\%Variance, ExpertRemoval]$. The first value indicates to first perform a principal component analysis (PCA) on the numeric features and to preserve the features that contribute 80% of the variance. The second value, i. e., *ExpertRemoval*, is a domain-specific annotation, which indicates the decision of a domain expert to remove additional numerical features. Other feature types have similar metadata. The list $featsUsed_i$ (see Eq. 3) indicates which data features from the data set d_i are finally used by the ML solution.

$$\begin{aligned} preprocessing_i =& \{numData_i, catData_i, \\ & timData_i, unsData_i, \\ & imgData_i, tsData_i\} \end{aligned} \tag{6}$$

$$\begin{aligned} numData_i =& \{featEncoding_i, \\ & featSelection_i\} \end{aligned} \tag{7}$$

The *technical settings* s_i describe the configuration and parameters needed to reproduce the ML solution. This metadata set includes lists with the programming language(s) ($language_i$), as well as the software platform(s) and libraries used with their specific version number ($platform_i$). Furthermore, $algorithm_i$ specifies the ML algorithm implementation used by the ML solution, e. g., sklearn.naive_bayes. GaussianNB. Custom hyperparameter values are stored in the list $hParams_i$. The following list, $settLabels_i$, is an aggregation of different fields in this metadata subset, which we thus explain in Sect. 4.3.

The *performance set* p_i contains performance values and explanations of the ML solution. This set contains three groups of metadata. The first group, $metrics_i$, contains values of performance metrics obtained while testing the ML solutionon a test dataset. For instance, $metrics_i$ can capture the accuracy, precision, recall, and training time values for a classifier. The values in $metrics_i$ are scaled to a range [0,1], going from worst or slowest performance to best or fastest performance. The second group, $metricsLabels_i$, is an aggregation of the $metrics_i$, which we again explain in more detail in Sect. 4.3.

The third group, $margins_i$, includes margin heuristics for all data features. A margin heuristic for a particular feature estimates how useful variations in the values of a data feature are for an ML algorithm to perform its task. Based on the heuristic value, AssistML indicates citizen data scientists which data features to consider more or less useful to develop a similar ML solution (see Sect. 5.4). For instance, assume a margin heuristic where a high margin value, i. e., close to one, shows that a data feature has very different values in true positive and false positive samples. This big difference indicates that the data feature helps improve the classification precision. A low margin value, i. e., close to zero, instead shows that true positive and false positive samples may not be differentiated by the considered feature. So, a feature with a low value of this margin heuristic is unsuitable for the ML solution.

In case of binary classification, the margin of a data feature estimates the differences in feature values between correctly classified and incorrectly classified observations. The margin heuristics are computed with n-sized samples between several combinations of true or false predictions, i. e., between true positives (TP) and false positives (FP), between true negatives (TN) and false negatives (FN), between TPs and FNs, and finally between TNs and FPs. These kinds of margin heuristics can also be used for multi-class classification tasks. In that case, the margin values for individual classes need to be micro- or macro-averaged [44]. Therefore, we focus the discussion of margin formulas for the binary classification case.

Depending on the feature type, the margin heuristic is computed differently. In the following, we explain how to compute the margin heuristic to distinguish between TP and FP samples. For numeric data and datetime deltas, a margin heuristic describing the ability of a feature to distinguish between TP and FP samples is defined as follows:

$$margin^{Pnum} = \frac{\sum_1^n |\overrightarrow{TPfeat} - \overrightarrow{FPfeat}|}{\mu(\mu(TPfeat), \mu(FPfeat))} \tag{8}$$

where $\overrightarrow{TPfeat}$ are the sorted data feature values in the true positives sample, $\overrightarrow{FPfeat}$ are the sorted data feature values in the false positives sample and $\mu(TPfeat)$ and $\mu(FPfeat)$ are the average values of these samples.

Figure 2 illustrates the meaning of the margin heuristic for a numeric feature. The blue and red dots show the variations of feature values for observations sorted by their feature value in the TP and FP samples, respectively. The distance between a pair of blue and red dots shows how well the TP and FP samples can be distinguished only with this data feature. Conversely, the closer the dots are, the less easy it is to distinguish the TP and FP samples using this data feature. The margin heuristic in Eq. 8 summarizes this distance in a single value.

Margin heuristics for categorical and unstructured text data are as follows:

$$margin^{Pcat} = \sum \left| \frac{|levels(TPfeat)|}{|TPfeat|} - \frac{|levels(FPfeat)|}{|FPfeat|} \right| \tag{9}$$

where $|\ TPfeat\ |$ is the number of instances in the true positives sample and $|\ FPfeat\ |$ the number of instances in the false positives sample. $|\ levels(TPfeat)\ |$ is the count of each distinct feature value in the true positives sample, e. g., how many times the value *medium* appears in the feature *size* in the TP sample. $|\ levels(FPfeat)\ |$ is correspondingly the count of each distinct feature value in the false positives sample.

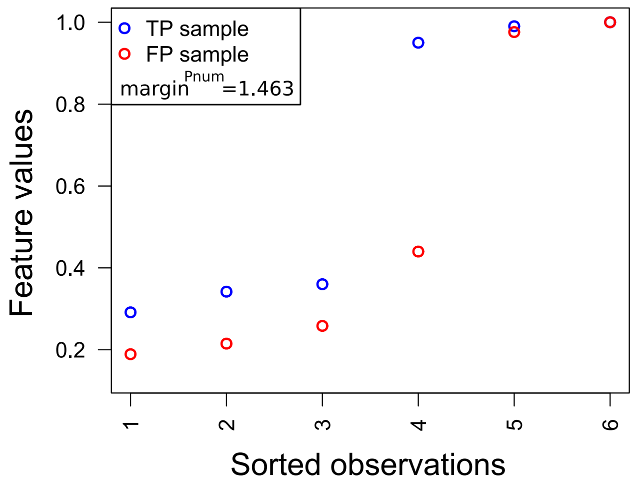


Fig. 2 Value difference of a numerical feature between TP and FP samples

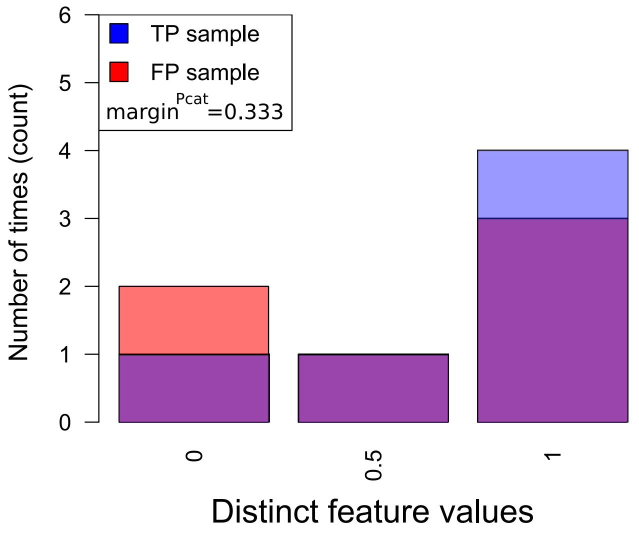


Fig. 3 Histograms of the values of a categorical feature in the TP and FP samples. The purple color shows how much both histograms of TP and FP samples overlap

Figure 3 illustrates the meaning of the margin heuristic for a categorical feature. The blue and red histograms show the number of times (count) that the distinct feature values "0" (small value), "0.5" (medium value), and "1" (high value) appear in the TP and FP sample, respectively. The TP sample has more observations with the value "1" and fewer observations with the value "0" than the FP sample. Observations with the value "0.5" occur with the same frequency among TP and FP samples. The purple color shows how much the two histograms of the TP sample and the FP sample overlap. The smaller this overlap, the easier it is to distinguish TP and FP observations with this data feature. Conversely, the higher this overlap between both histograms, the less suitable is the given feature to distinguish between TP and FP samples. The margin heuristic summarizes the distribution difference between the two histograms of TP and FP samples in a single value.

4.2 Metadata extraction

Tracking platforms, e. g., MLflow Tracking [9, 45], ModelDB [34] and OpenML [5, 33], generate metadata about an ML solution during the solution's development. Metadata at this stage consists of values and characteristics that describe the ML solution in exhaustive detail. For instance, these platforms can collect metadata about the data set, the training settings, or the hyperparameters of the learning algorithm. This is achieved by storing data from logs, code, or manual annotations with minimal preprocessing or interpretation. Additional data can be obtained from the ML solutiondocumentation, e. g., from test cases or dependency documentations.

The platforms store this metadata in different formats, e. g., as local files,[1] in a relational database,[2] or in a centralized tracking server .[3] They organize metadata into semantically consistent groups. For instance, MLflow organizes the basic metadata fields into two types of key-value pairs, i. e., parameters and metrics. MLflow then groups the set of parameters and metrics used to train and validate an ML model into a run. Finally, several runs are grouped into an experiment.

However, each platform leaves the end user to decide which specific metadata to collect. This limits the possibility of comparing ML solutions developed by different teams, because the individual teams may decide to store different kinds of metadata. This even holds if the teams use the same tracking platform. For example, one team can decide to track only the F1 score, while others may decide to collect precision and recall values separately. MLflow attempts to handle this problem by autologging all hyperparameters and performance metrics generated by default by model training functions of popular ML libraries [9]. However, autologging does not cover other important information about an ML solution. For instance, end users still need to decide which meta features about the data sets or which information about used preprocessing techniques or learning algorithms the platforms have to document.

In AssistML, the metadata model prescribes which fields to collect, regardless of the software libraries, data types or

[1] https://mlflow.org/docs/latest/tracking.html.

[2] https://docs.verta.ai/verta/experiment-management/experiment-management_concepts.

[3] https://docs.openml.org/OpenML_definition/.

learning algorithms used. This releases citizen data scientists from deciding which metadata to collect. Furthermore, the repository promotes metadata standardization, i. e., all ML solutions are finally described by the same set of metadata fields. This guarantees the comparability of ML solutions with each other.

In principle, existing tracking platforms can be used to collect the metadata fields in AssistML's model introduced in Sect. 4.1. For instance, MLflow Tracking can collect fields from the subsets u_i, d_i and s_i as MLflow parameters and performance metrics in subset p_i as MLflow metrics. This is possible via a series of manual log annotations at different steps of the development process. Software dependencies are then stored separately in a YAML file. While this ensures that all metadata is present, MLflow does not offer further means to organize the metadata into the subsets and the structure prescribed by the AssistML metadata model.

A potential solution is to use ETL tools or other data integration solutions to integrate the metadata collected by MLflow into AssistML's model. The key-value pairs in MLflow can be given standard, unique, and fully qualified names to identify their position in AssistML's metadata model. Examples can be the key for the algorithm implementation (*m.s.algorithm*) or the key for the accuracy value (*m.p.metrics.accuracy*). These qualified names can then be used by schema matching techniques [25] to query MLflow data sources and thus to obtain the metadata fields needed to complete AssistML's model.

Furthermore, AssistML abstracts from the implementation-specific limitations of individual tracking platforms and generates instead JSON documents via code annotations. This document structure allows metadata to be organized into key-value pairs, lists and objects, with which any semantic structure provided by common tracking platforms can be reproduced. Conversely, this document structure also provides the schema that ETL tools require to be able to integrate data from common tracking platforms into AssistML's repository. AssistML collects the metadata for each subset as the corresponding development step takes place. This means that subsets d_i, s_i and p_i get their data via code annotations during the data preprocessing, model training, and model validation steps. In the case of the subset u_i, which is to be specified before and during development, AssistML extracts the information from project documentation about the ML solutiondevelopment, e. g., in software specifications.

4.3 Metadata enrichment

The raw metadata collected via AssistML's metadata model is rich in details. This raw metadata is needed to ensure that any interpretation of this metadata remains transparent and traceable and that documented ML solutions are reproducible. However, it makes it difficult for citizen data scientists to compare ML solutions directly via the raw metadata fields. This mainly concerns the technical settings in subset s_i and performance metrics in p_i because they contain many low-level details. AssistML therefore aggregates these metadata fields and enriches the metadata with the aggregates. This facilitates the task of citizen data scientists to analyze and compare ML solutions. Specifically, we now detail how the previously introduced fields $settLabels_i$ and $metricsLabels_i$ aggregate data from their respective sets s_i and p_i.

Aggregating the training settings s_i The documentation of training settings in set s_i requires to be extensive to ensure reproducibility. The software dependencies in $m.s.platform_i$ and the custom hyperparameters in $m.s.hParams_i$ need to list exact version numbers and values to be able to reproduce the ML solution. However, for citizen data scientists willing to estimate the complexity of deploying an ML solution, the total number of dependencies to satisfy or the total number of customer hyperparameters to set may be more useful than their specific values.

Therefore, we aggregate the contents of the fields $m.s.platform_i$ and $m.s.hParams_i$ and store the aggregated information in a field $m.s.settLabels_i$. For every ML solution, the repository counts the number of software dependencies and the number of hyperparameters. We then use the counts of all ML solutions to group the solutions into bins or categories that meaningfully divide the whole set of solutions in the repository. For example, assume the ML solutions in the repository have between 0 and 50 custom hyperparameter values. Then, they can be divided into three groups: i) a group for ML solutions with 0 to 4 custom hyperparameter values (low), ii) a group for ML solutions with 5 to 15 custom hyperparameter values (medium), and iii) a group with ML solutions with more than 15 custom hyperparameter values.

The goal is to have a categorization of the ML solutions that is meaningful to the citizen data scientist. This means that there is no generally predefined number of bins or predefined thresholds to categorize ML solutions into these bins. Instead, the characteristics of the ML solutions in the repository determine the criteria to categorize this metadata depending on the nature of all ML solutions currently stored in the repository. For instance, assume that all solutions in the repository are either simple classifiers with at most 30 custom hyperparameters or complex deep learning architectures with at least 200 hyperparameters. Here, a more meaningful grouping can consist of two categories only. Otherwise, if ML solutions are only developed with a small number of strictly controlled technology stacks, there is usually no need to categorize the number of software dependencies at all, as many solutions may already share the same value.

Note that this categorization is an approximation to the actual complexity of the solutions' training settings. It assumes that each hyperparameter is equally difficult to set

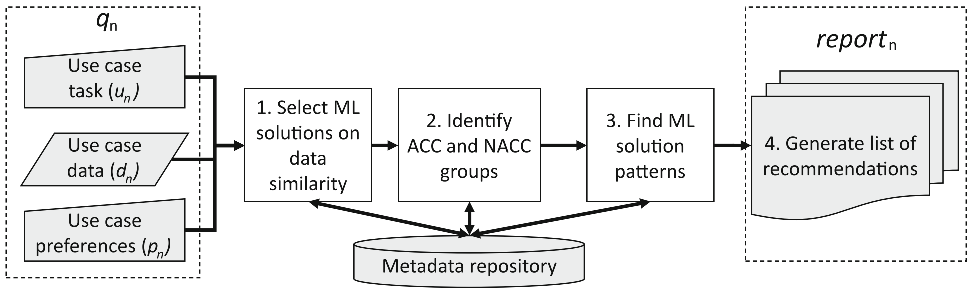

q_n
Use case task (u_n)
Use case data (d_n)
Use case preferences (p_n)
1. Select ML solutions on data similarity
2. Identify ACC and NACC groups
3. Find ML solution patterns
4. Generate list of recommendations
$report_n$
Metadata repository

Fig. 4 Overview of the steps in AssistML

and equally important to the training process. Nevertheless, it also makes very different ML solutions, e. g., with five completely different hyperparameters, comparable with one another. This way, this kind of metadata enrichment facilitates the discovery of ML solution patterns (see Sect. 5.3).

Aggregating the performance metrics p_i The performance values measured in $m.p.metrics_i$ usually have unique numerical values. For instance, two ML solutions may have accuracy values of 0.963821 and 0.959846 and are thus considered different when only looking at these raw values. Nevertheless, the difference of 0.003975%-points has little significance in practice. From this practical point of view, these two solutions can therefore be seen as comparably equal in accuracy.

Hence, we aggregate performance values in $m.p.metrics_i$ and store the aggregates in a new metadata field $m.p.metricsLabels_i$. This subset contains a performance label for each metric based on the values in $m.p.metrics_i$. The performance labels intuitively indicate non-experts how good a solution is in comparison to others. Performance labels are letter grades, going from A+ (best values) to E (worst values). Grades A to E denote five groups, each covering 20% of the values of that performance metric among ML solutions in the repository. The grade A+ is reserved for outlier solutions with the highest performance values in the repository. For fairness, these labels only compare ML solutions developed for the same task and data set. To determine the threshold values for each grade, AssistML computes five quantiles. The ML solutions with performance values within each quantile are assigned the performance label corresponding to that quantile. ML solutions with upper outlier performance values receive instead the A+ label. For each considered performance metric, one aggregated performance label is assigned to the ML solutions.

5 AssistML: step-wise recommendation approach

This section explains the four main steps that constitute AssistML (see Fig. 4). They recommend a short list of ML solutions for a new use case n based on metadata from the repository M. AssistML requires three inputs in form of a query q_n (see Eq. 10) to obtain a list of recommended ML solutions in $report_n$. Thereby, query q_n requires only the absolute minimum information, making the query easier to specify for citizen data scientists.

The first input in q_n is a description of the desired use case task u_n. The task may be described, e. g., as binary classification with a single prediction as output. The second input is an annotated extract of the new use case data d_n. Citizen data scientists use annotations to indicate the feature types of each feature contained in the data, i. e., whether the feature contains numerical, categorical, unstructured text, datetime, image, or time series data. As third input, citizen data scientists set preferences they have about the ML solution's performance (p_n). These preferences include ranges for performance metrics that define acceptable values. For instance, a classification task can have range limits for accuracy, precision, recall, and training time. A range of 0.25 in a metric means that only ML solutions with values in the top 25% of all relevant solutions in the repository, i. e., for similar task and data set, are considered acceptable.

$$q_n = \{u_n, d_n, p_n\} \tag{10}$$
$$u_n = \{taskType_n, taskOutput_n\} \tag{11}$$
$$d_n = \{data_n, featTypes_n\} \tag{12}$$
$$p_n = \{ranges_n\} \tag{13}$$

5.1 Step 1: select ML solutions on data similarity

This step selects ML solutions from repository M based on the similarity of their data set and task to the data set and task of the new use case in query q_n. Only the solutions developed for the most similar data sets and tasks, i. e., m_s, are passed to the following steps. The goal is to ensure that the recommended ML solutions were developed for a use case that is as similar as possible.

Algorithm 1 gives an overview of the first step. It requires access to the repository M, and the subsets u_n and d_n of the user query. The first task is to analyze the provided data. The sample data $data_n$ and its feature type annotations $featTypes_n$ are used to compute summary metafeatures $allMF_n$ and individual metafeatures $singleMF_n$ for all fea-

Algorithm 1 Select solutions on data similarity

```
Require: $M, u_n, d_n$
Ensure: $m_s, w_t$
1: $\{allMF_n, singleMF_n\} \leftarrow$
   $analyzeQueryData(d_n.data, d_n.featTypes)$
2: function SELECT($u_n, d_n$)
3:     $m_{s0} = \{m_i \in M | m_i.taskType = u_n.taskType$ and
       $m_i.taskOutput = u_n.taskOutput\}$
4:     if $m_{s0} = \{\}$ then
5:         terminate ASSISTML
6:     else
7:         $\{m_s, w_t\} \leftarrow$CHECKSIMILARITY($m_{s0}$,
           $d_n.allMF, d_n.singleMF$)
8:         return $\{m_s, w_t\}$
9:     end if
10: end function
```

tures of the new use case data. These metafeatures make it possible to determine the similarity between the new data set d_n and existing data sets d_i in M.

Data set similarity is expressed in four levels. Each similarity level is defined by criteria regarding the use case task and data set. Firstly, the base similarity or m_{s0} (see Line 3) describes ML solutions developed for the same type of task and same type of output. For example, m_{s0} can refer to all solutions in M for binary classification ($taskType$) and producing single predictions ($taskOutput$). If no solutions in M fulfill the criteria, the whole recommendation process is terminated (Line 5). This is indicative of a completely new use case, for which none of the solutions in M can be reused. In that situation, other development approaches should be followed to create a new solution.

If solutions with similarity level 0 are available in M, the function $check\ Similarity()$ applies filter criteria to determine which solutions fulfill levels 1 to 3. The filter criteria for each level are given in Eqs. 14,15,16. The similarity levels are checked consecutively, so that $m_{s3} \subseteq m_{s2} \subseteq m_{s1} \subseteq m_{s0}$.

$$m_{s1} = \{m_{s1} \in m_{s0} \mid d_i.allMF.featTypes = d_n.allMF.featTypes\} \quad (14)$$

$$m_{s2} = \{m_{s2} \in m_{s1} \mid ratios(d_i.allMF.featTypes) \simeq ratios(d_n.allMF.featTypes)\} \quad (15)$$

$$m_{s3} = \{m_{s3} \in m_{s2} \mid d_i.singleMF \simeq d_n.singleMF\} \quad (16)$$

The filter for m_{s1} removes solutions whose associated data sets d_i do not have the feature types of the new data set d_n. For a new data set with, e. g., numerical data, this filter removes ML solutions not trained on numerical data at all. Conversely, ML solutions trained with at least one numerical feature have similarity level 1. The filter for m_{s2} keeps only those solutions whose associated data sets d_i have feature type ratios similar to the feature type ratios of the new data set d_n. Ratios are considered similar if they are within one decile of the feature type ratios of the new data set. This means, e. g., removing ML solutions not trained on data sets with 95% $\pm$ 5%-points of numerical features. The filter for m_{s3} compares the metafeatures in $singleMF$. Again, the solutions in m_{s3} must have $singleMF$ values within one decile of the $singleMF$ values of the new data set. This means that we, e. g., compare the percentage of outliers of numeric features in the new data set d_n to the percentage of outliers of the numeric features in associated data sets d_i in M. If the percentage of outliers of all numeric features is within +- 5%, e. g., [0.2, 0.0, 0.11] and [0.18, 0.05, 0.07], the solution's similarity is level 3.

Step 1 passes over the ML solutions trained on the most similar data sets (m_s in Line 8), i.e., having the highest similarity level. If m_s have a similarity level lower than 3, the function $checkSimilarity()$ adds *distrust points* and warnings to w_t. This is the basis of a distrust score that estimates the suitability of the recommendations for the use case. It is inspired by the ML test score of Breck et al. [7]. Formally, $w_t = \{distrustPoints, warnings(t)\}$, i.e., a scalar value $distrustPoints$ and a list of short explanations $warnings(t)$ describing a problematic condition t. AssistMLadds distrust points for each problematic condition t that occurs at any step. During this first step, $w_t.distrustPoints$ can receive up to 3 distrust points depending on the highest similarity level found:

- Similarity level 0 awards three distrust points and adds the explanation *"Dataset similarity level 0. Only the type of task and output match. Distrust Pts increased by 3"*.
- Similarity level 1 awards two distrust points and adds the explanation *"Dataset similarity level 1. Datasets have shared feature types, but different ratios of the types. Distrust Pts increased by 2"*.
- Similarity level 2 awards one distrust point and adds the explanation *"Dataset similarity level 2. Datasets have similar ratios of feature types, but different metafeature values. Distrust Pts increased by 1"*.
- Finally, AssistML awards no distrust points if the highest similarity level 3 is achieved. In that case, the explanation simply confirms the similarity to the user: *"Dataset similarity level 3. Datasets have features with similar metafeature values. No Distrust Pts added"*.

5.2 Step 2: identify (nearly) acceptable ML solutions

The second step divides ML solutions from the previous step based on their performance into two groups: m_{ACC} (ACCeptable) and m_{NACC} (Nearly ACCeptable). An ML solutionbelongs to m_{ACC} if its performance values for each relevant metric fall in the percentile range specified by the user in $p_n.ranges$. For instance, a range of 0.2 for accuracy

Algorithm 2 Identify [nearly] acceptable ML solutions

```
Require: $m_s$, $p_n.ranges$
Ensure: $m_{ACC}, m_{NACC}, w_t$
1: function CLUSTER($m_s$, $ranges$)
2:     $cls \leftarrow$ DBSCAN($m_s$)
3:     $mts \leftarrow m_s.p.metrics$
4:     $qlim \leftarrow p_n.ranges$
5:     for $j = 1$ to $[cls]$ do
6:         $accl = max(mts(i)) - qlim(i) * (max(mts(i)) - min(mts(i)))$
7:         accFit(i) $= \frac{|\{m_s|m_s \in \text{cls(j)} \wedge \text{mts(i)} \geq \text{accl}\}|}{|\{m_s|m_s \in \text{cls(i)}\}|}$
8:         $naccl = max(mts(i)) - 2qlim(i) * (max(mts(i)) - min(mts(i)))$
9:         $naccFit(i) = \frac{|\{m_s|m_s \in cls(j) \wedge mts(i) < accl \wedge mts(i) >= naccl\}|}{|\{m_s|m_s \in cls(i)\}|}$
10:        if $(naccFit(i) \geq 51\%)$ then
11:            $m_{ACC} \leftarrow cls(i)$
12:        else[$(naccFit(i) \geq 51\%)$]
13:            $m_{NACC} \leftarrow cls(i)$
14:        end if
15:    end for
16:    if $(m_{ACC} = \{\})$ then
17:        terminate ASSISTML
18:    else
19:        $w_t \leftarrow$ TESTFITS($m_{ACC}$,$m_{NACC}$)
20:        return $m_{ACC}, m_{NACC}, w_t$
21:    end if
22: end function
```

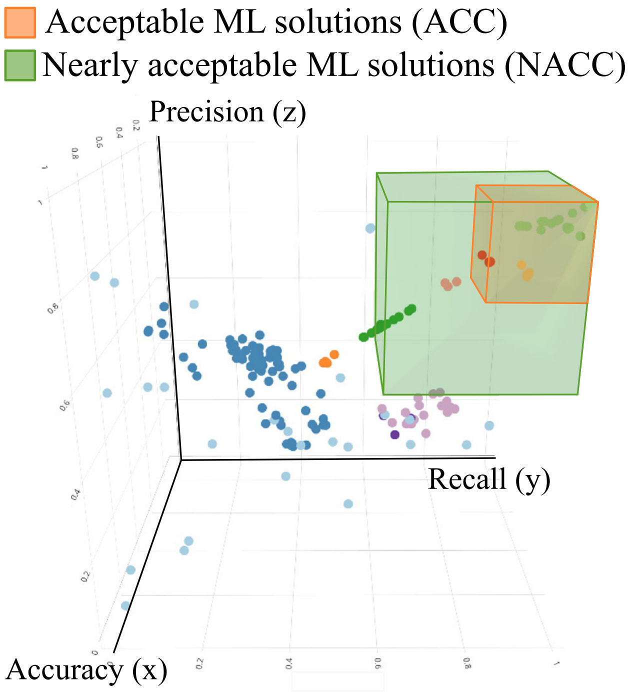


Fig. 5 Visualization of ACC and NACC groups where each dot represents an ML solution

means that only the ML solutions of m_s which are among the upper 20% regarding accuracy are acceptable. For example, if the accuracy values of solutions in m_s vary between 0.7 and 0.92, ML solutions with an accuracy of at least 0.87 are considered acceptable. This corresponds to 20% of the value range subtracted from the maximum value, i. e., substracting 20% of 0.22 from 0.92, or $0.92 - 0.044 = 87.6$.

An ML solution belongs to m_{NACC} if its performance values are lower than those of solutions in m_{ACC}. Nevertheless, the performance values of m_{NACC} solutions may only be lower than the threshold for m_{ACC} solutions by the amount defined by $p_n.ranges$. In the example with a 0.2 accuracy range, ML solutions in m_s belong to m_{NACC} if their accuracy values are lower than 0.87, but greater than 0.82 (given that $0.92 - 0.088 = 83.2$). ML solutions in m_{NACC} serve as comparison for ML solutions in m_{ACC}.

The first task of Algorithm 2 (Line 2) uses the DBSCAN algorithm [10] to cluster the solutions in m_s based on their performance values in $m_s.p.metrics$. The need for clustering derives from the fact that companies typically store hundreds of ML solutions in their repositories (R_1). The consequence is that it becomes inefficient to check and compare the performance of every single ML solution. Clustering analysis reduces the number of necessary checks by grouping similarly performing ML solutions. For instance, Fig. 5 illustrates more than 200 ML solutions developed for different use cases. It arranges these solutions as colored dots in a three dimensional space of the performance metrics accuracy, precision, and recall. The color indicates one of the 16 clusters to which the ML solutions belong, as determined by a clustering algorithm. The plot shows that ML solutions from different use cases can have similar performance trade-offs, resulting in clearly separated clusters. The identification of ACC and NACC solutions is thereby facilitated, as we only need to compare 16 clusters instead of 200 individual ML solutions.

The DBSCAN algorithm is appropriate for this task for two reasons. First, it removes the need for citizen data scientists to predefine the number of clusters, thus removing a potential source of bias. Second, DBSCAN's minimum distance parameter guarantees that all ML solutions in a cluster have the same performance trade-off across all considered performance metrics. We set this distance parameter to the epsilon radius with a value of 0.05. Furthermore, we parameterize the minimum number of points (*minPoints*) DBSCAN requires to form a dense region to three points.

The second task (Lines 7 and 9) computes the fit of each cluster i in the ACC and in the NACC group. The fit of a cluster is the proportion of its ML solutions that are inside the acceptable ($accFit$) or nearly acceptable ($naccFit$) performance ranges. The third task (Lines 11 and 13) uses $accFit$ and $naccFit$ to assign the current cluster of ML solutions to either m_{ACC} or m_{NACC}. A value of 51% is used in both assignments to indicate a simple majority in the cluster.

Once all clusters have been processed, AssistML is interrupted if no cluster of ML solutions can be assigned to m_{ACC}.

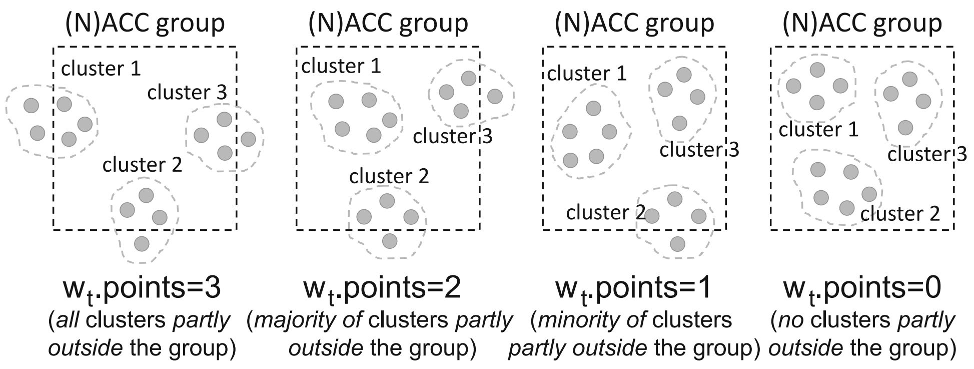

(N)ACC group
cluster 1
cluster 2
cluster 3
w_t.points=3
(*all* clusters *partly outside* the group)
w_t.points=2
(*majority of* clusters *partly outside* the group)
w_t.points=1
(*minority of* clusters *partly outside* the group)
w_t.points=0
(*no* clusters *partly outside* the group)

Fig. 6 Assignment of points to the distrust score after clustering

This means that no ML solution in m_s meets the performance preferences of the new use case. In such a situation, it is better to develop a new ML solution from scratch with senior ML experts.

Otherwise, Algorithm 2 delivers the ML solutions in m_{ACC} and m_{NACC} along with w_t. Beforehand, function *testFits()* adds distrust points and warnings to w_t depending on how well clusters fit in the groups m_{ACC} or m_{NACC}. The criteria to award distrust points are illustrated in Fig. 6. If for each cluster, at least one ML solution is outside its m_{ACC} or m_{NACC} group, three distrust points are awarded. If the simple majority of clusters have at least one solution outside a group, two distrust points are awarded. If this only holds for the simple minority of clusters, one distrust point is awarded. In each of these cases, a corresponding explanation is added to $w_t.warnings$ to describe the situation in each group. No distrust points are given if all ML solutions of all clusters are fully inside their group.

Note that there may be few use cases, where the DBSCAN algorithm identifies single ML solutions to be outliers regarding the considered performance metrics. If this is the case, we assign all identified outlier ML solutions to a dedicated noise group. This noise group is not returned to citizen data scientists by Algorithm 2, but stored in a separate file in our prototypical implementation. The reason is that we want to show such a noise group to more experienced data scientists to allow for an in-depth analysis of the outlier ML solutions. On the one hand, outlier solutions may be a sign to populate AssistML's repository with additional ML solutions, so that finally more clear clusters of solutions may be found around the previous outliers. On the other hand, if the repository is already populated with a reasonable number of solutions, an outlier ML solution may also indicate a particularly good but also a bad fit between the outlier's configuration or hyperparameters and the characteristics of the new use case data. This however has to be verified by more experienced data scientists.

Algorithm 3 Find ML solution patterns

```
Require: $m_{ACC}, m_{NACC}$
Ensure: $rules_m$
1: function PATTERNS($m_{ACC}, m_{NACC}$)
2:     settings $\leftarrow m_{ACC}.s, m_{NACC}.s$
3:     $metricsLabels \leftarrow m_{ACC}.p.metricsLabels,$
       $m_{NACC}.p.metricsLabels$
4:     $rules_m \leftarrow$ FPGROWTH($settings,$
       $metricsLabels, minConf, minSupport$)
5:     $rules_m \leftarrow$ REMOVEDUPLICATES($rules_m$)
6:     $rules_m \leftarrow \{rules_m :$
       $confidence(rules_m) < 1$
       $\& \ leverage(rules_m) > 0$
       $\& \ lift(rules_m) > 1\}$
7:     return $rules_m$
8: end function
```

5.3 Step 3: find ML solution patterns

The third step searches for patterns in the metadata of the ACC and $NACC$ solutions. For this purpose, Algorithm 3 builds association rules with the settings and performance metadata of m_{ACC} and m_{NACC} solutions. Association rules describe patterns of the kind: *"IF training time label is D (antecedent), THEN ML solution has neural networks algorithm and number of custom hyperparameters is 5-10 (consequent)"*. Each association rule of this kind indicates the common occurrence, not causality, of the antecedent and the consequent among relevant ML solutions.

The first task of this step generates frequent item sets and rules using the FP-Growth algorithm [17] (Line 4 in Algorithm 3). Its input is metadata describing the configuration $settings$ and the performance labels $metricsLabels$ of both the acceptable and nearly acceptable ML solutions. The step combines metadata from both m_{ACC} and m_{NACC} to capture patterns across both groups, since these patterns can be decisive for the solution's performance.

The step configures the FP-Growth algorithm by means of two parameters to prompt the creation of as many association rules as possible. The minimum confidence $minConf$, which measures a rule's reliability, is set to 0.7. The minimum support $minSupport$, which sets the threshold to consider a rule frequent, is set to 0.25. Depending on the ML solutions contained in the metadata repository, it may be necessary to readjust these values. If the repository contains ML solutions with many different configurations and a wide range of performance values, $minConf$ may need to be set at a higher value to avoid the identification of conflicting patterns. On the contrary, ML solutions with very similar configurations and a limited range of performance values may require a higher value for $minSupport$ to avoid the identification of irrelevant patterns. The result of Line 4 is a list of $rules_m$, where each rule has confidence, leverage and lift values.

The following task (Line 5) removes duplicate rules with the same items and the same values of confidence, leverage and lift. This is because the two rules express the same co-occurrence pattern. The next task (Line 6) removes low-quality rules from $rules_m$. Here, the step removes rules with a confidence value of 1 or with leverage of 0 or with lift of 1. These metric values indicate that rules are trivial or that the antecedent and consequent are statistically independent.

5.4 Step 4: generate list of recommendations

The final step generates a report containing a list of k ML solutions for both m_{ACC} and m_{NACC}. The m_{ACC} and m_{NACC} examples indicate a citizen data scientist the pros, cons, and effects of using different ML solution configurations. Hence, s/he can make informed decisions for the new use case. The lists are stratified by the type of learning algorithm to ensure variety in the recommendations. The report additionally includes the original query q_n and $w_t.warnings$ with explanations of problematic situations collected during the previous steps. Furthermore, it contains the $distrustScore_n$ to estimate how applicable a recommendation is. This distrust score is computed based on the sum of points in $w_t.distrustPoints$ accumulated over t identified problematic conditions as shown in Eq. 17

$$distrustScore_n = \frac{\sum w_t.distrustPoints}{t} \tag{17}$$

A dedicated recommendation report describes each selected ML solution in $m_{ACC}(k)$ and $m_{NACC}(k)$ individually. Table 3 shows a simplified version of this report for a sample recommendation. The report describes the ML solution's performance and configuration in a simple and intuitive way for citizen data scientists. An overall score summarizes the different performance values in $p.metrics$ (see line 2). For example, the overall score can be computed on accuracy, precision, recall, and training time. This score metric is the average of these four normalized metrics, scaled to a range from 0 (worst) to 1 (best). Also, the performance labels in $p.metricsLabels$ further describe the solution's performance for each considered metric individually (see lines 3–5).

A global explanation of the ML solution's output based on $p.margins$ completes the description of the solution's performance (see lines 6–7). This global explanation exemplifies the ML solution's behavior regarding certain data features. Data features with low margin values, e. g., lesser than 0.05, are considered unsuitable for the ML solution's task. Data features with high margin values, e. g., greater than 0.3, are considered suitable.

The report also describes the solution's configuration with metadata from m_{ACC} and m_{NACC}. Firstly, the report includes the preprocessing used on each feature type (see lines 8–9). Secondly, any pattern from $rules_m$ that contains an element of the solution's configuration, e. g., the same implementation library, is added to the *ML Solution Patterns* section of the report (see lines 10–11). Data about the software and hardware resources needed to deploy the ML solution, the used programming language and algorithm implementation library, as well as the number of custom hyperparameters complete the report in the *Deployment Description* (see lines 12–16).

The generated reports are ranked using the overall score value. If there are ties, the individual performance metrics rank further involved ML solutions. A sequence deduced from the performance ranges in p_n of user query q_n determines the order in which to use the performance metrics for this ranking. To this end, metrics with narrower ranges are assumed more important than metrics with wider ranges.

6 Prototype and evaluation

The first part in this section describes the AssistML prototype. The second part explains our approach to evaluate the functionality of AssistML on the basis of two use cases. The third part discusses three types of evaluation results obtained with the prototype and evaluation approach, i. e., the recommendation reports, the performance of recommended and reused ML solutions, and the prototype's execution times. The fourth part discusses in detail the comparison of AssistML recommendations with ML models trained by an AutoML system. The last part in this section assesses how AssistML fulfills the requirements introduced in Sect. 2.2. The source code, evaluation data, and a short demonstration video of the AssistML prototype is available on GitHub.[4]

6.1 Prototypical implementation

The AssistML prototype consists of six modules developed in R and Python. Figure 7 depicts the module architecture of the prototype. Arrows indicate the data and logic flow between the modules. Module 1 provides a graphical

[4] https://github.com/al-villanueva/assistml

Table 3 Sample ML solution recommendation report

1	RFR_kick_011	
2	Overall score: 0.9914	
3	Performance Labels	
4	Accuracy: A	Precision: A
5	Recall: A	Training Time: C
6	**Output Explanation**	
7	Feature-3 is suitable for the task. Feature-6 is unsuitable for the task.	
8	**Data Preprocessing**	
9	Categorical data is read via One-Hot Encoding (. . .)	
10	**ML Solution Patterns**	
11	IF ML solution has [random forests algorithm] and [number of custom parameters is 5 - 15] then [recall label is A] and [library used is sklearn].	
12	**Deployment Description**	
13	Deployed on single_host with 2 cores with 2.6 GHZ	
14	Language	python v. 3.6.0
15	Implementation	sklearn.ensemble. ExtraTreeClassifier v. 0.22.2
16	Nr. Dependencies: 6	Nr. Parameters: 14

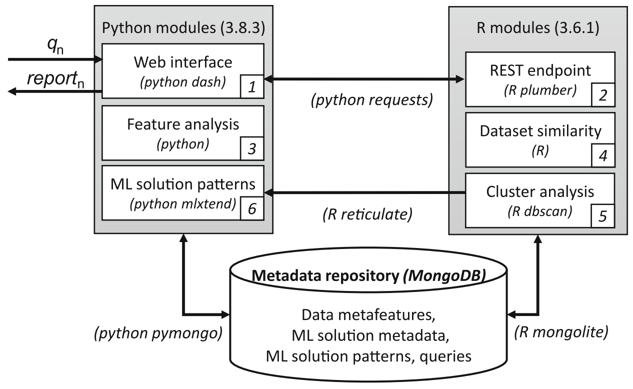


Fig. 7 Module architecture of the Assist ML prototype. Used libraries are shown in parentheses

web interface for the citizen data scientist to input a new query q_n. Module 2 implements a REST endpoint to receive query data from the web interface or from programmatic calls. This module also orchestrates the execution of the remaining modules to carry out the recommendation concept. Module 3 computes metafeatures for data sets provided within queries. This module thus renders new data sets comparable to the data sets contained in the metadata repository. It stores the metafeatures in the metadata repository using the pymongo[5] library.

Modules 4, 5 and 6 implement steps 1, 2, and 3 of AssistML. These modules obtain their input data from the REST endpoint and the metadata repository. Modules 4 and 5 communicate with the metadata repository using the mongolite [6] library, while module 6 accesses the repository via pymongo. At the end, module 2 collects all results and generates the recommendation report $report_n$. This report is then shown to the user via the web interface (module 1). We implemented the metadata repository as document collections in MongoDB. This allows the modules to use JSON documents as data exchange format.

Moreover, we populate the metadata repository with 228 previously developed ML solutions. These solutions represent a diverse selection of solution configurations in order to reflect the most important application scenarios in companies. Table 4 gives specific details about the ML solutions in the metadata repository. We developed solutions for different data sets with various feature types, i. e., categorical, numeric, datetime and unstructured text. The description of use case data in Table 4 includes the respectively used feature types in parentheses. If multiple feature types are included, they are listed from most frequent to least frequent in the data set. The use cases cover ML tasks for both binary and multi-class classification. We thereby trained predictive models using various learning algorithms from different libraries and programming languages.

We collected several performance metrics for each ML solutionusing 5-fold cross-validation. These include accuracy, precision, recall, training time, execution time for a single prediction, the F1 score and the confusion matrix.

5 https://pymongo.readthedocs.io/en/stable/index.html

6 https://jeroen.github.io/mongolite/

Table 4 Metadata repository contents

Element (count)	Description
Data sets (5)	kick auction (categorical, numeric)
	bank marketing (categorical, numeric)
	human activity recognition (numeric)
	gasdrift (numeric)
	amazon fine food reviews (numeric, unstructured text, categorical, datetime)
ML tasks (2)	Binary classification, multi-class classification
ML algorithms (9)	Decision trees, random forests, neural networks, logistic regression, naive Bayes, K nearest neighbors, gradient boosting machines, support vector machines, general linear model
Languages (2)	Python, R
Libraries (4)	scikit-learn, RWeka, pycaret, H2O
ML solutions (228)	Individually developed ML solution instances

Further information regarding the system requirements and necessary installation steps to run the AssistML prototype is provided in the GitHub repository.

6.2 Evaluation approach

The prototypical implementation enables the evaluation of AssistML with a multi-step approach. Figure 8 shows the four steps of our evaluation as well as the results they produce. The following paragraphs explain the components of our evaluation approach, i. e., the use cases, the evaluation settings and the evaluation steps.

The evaluation approach considers two new use cases, which are unknown to the metadata repository. The first and main use case deals with the task of fault detection during *steel plates* production. It is based on a public data set for binary classification.[7] As basis for comparison, we additionally use a complementary use case based on the public data set *adult*.[8] It describes a predictive task to determine if an adult has an annual income over 50 000 dollars.

The first step in the evaluation approach issues queries q_n for each use case using two evaluation settings. These settings represent two levels of performance preferences. For simplicity, the same preferences are set across all considered performance metrics, i. e., accuracy, precision, recall, and training time. Settings *q-steel-10* and *q-adult-10* demand ML solutions to have the top 10% values in all four metrics to be considered acceptable. Settings *q-steel-20* and *q-adult-20* describe less restrictive demands, i. e., ML solutions with the top 20% performance values are considered acceptable. In practice, switching from the top 10% to top 20% setting can be the result of dealing with a performance trade-off. In that

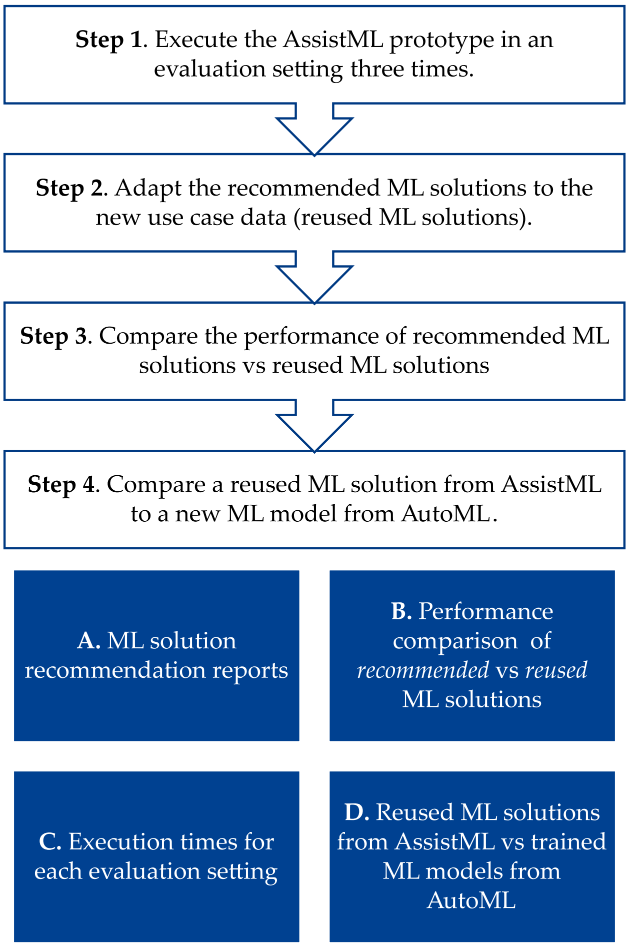

Step 1. Execute the AssistML prototype in an evaluation setting three times.
Step 2. Adapt the recommended ML solutions to the new use case data (reused ML solutions).
Step 3. Compare the performance of recommended ML solutions vs reused ML solutions
Step 4. Compare a reused ML solution from AssistML to a new ML model from AutoML.
A. ML solution recommendation reports
B. Performance comparison of *recommended* vs *reused* ML solutions
C. Execution times for each evaluation setting
D. Reused ML solutions from AssistML vs trained ML models from AutoML

Fig. 8 Steps of the evaluation approach (above in white) and its results (below in blue)

[7] https://www.openml.org/d/1504.

[8] https://www.openml.org/d/1590.

Listing 1 Elements of the query q_n for the evaluation setting *q-steel-10*.

```
taskType: "binary_classification",
taskOutput: "single_prediction",
data: "steel-plates-fault.csv",
accuracy_range: 0.10,
precision_range: 0.10,
recall_range: 0.10,
trtime_range: 0.10,
```

Listing 2 Elements of the query q_n for the evaluation setting *q-adult-20*.

```
taskType: "binary_classification",
taskOutput: "single_prediction",
data: "adult_1590.csv",
accuracy_range: 0.20,
precision_range: 0.20,
recall_range: 0.20,
trtime_range: 0.20,
```

case, the citizen data scientist may look for ML solutions with high values on a critical performance metric at the expense of others. We executed each evaluation setting three times following a randomized plan in order to track the average execution times to generate recommendations per evaluation setting.

As mentioned in Sect. 5, AssistML requires citizen data scientists to specify only the absolute minimum information in a query q_n. Listing 1 shows the elements of an example query to be specified for the evaluation setting *q-steel-10*. Here, citizen data scientists search for ML solutions supporting a binary classification (*taskType* at line 1) with a single prediction as output (*taskOutput* at line 2). In addition, they specify the data set as a reference to a CSV-based file (line 3). In lines 4–7, the performance preferences of the use case *q-steel-10* are set. The values 0.10 indicate that only those ML solutions are to be considered acceptable that are among the top 10% in both accuracy, precision, recall, and training time.

To illustrate the major differences of the queries q_n among the evaluation settings, Listing 2 shows the query for the setting *q-adult-20*. The type and output of the ML task are the same as for all other settings. The query for setting *q-adult-20* differs from that of setting *q-steel-10* firstly in the reference to the data for which a ML solution is to be found (line 3). In addition, the performance preferences in lines 4 to 7 are set to the values 0.20 in order to return only those ML solutions that are among the top 20% of all metrics.

Note that our prototypical implementation offers a graphical interface to ease the task to specify a query. Furthermore, diagnosis and parsing functions that run in the background reduce the effort to input all necessary information. For instance, one of these background functions parses the specified file to extract important information about the feature types that are contained in this file. More information about this graphical interface and about the background functions may be found on our GitHub repository.[9]

For each of the two use cases and for each evaluation setting, AssistML delivers a list of n recommended ML solutions. Step 2 of the evaluation approach (see Fig. 8) adapts each recommended ML solution on the new use case data. Thereby, the source code and configuration settings of the recommended solution are reused, i.e., applied directly to the *steel plates* or *adult* data. Citizen data scientists only need to replace the names of the data features of the old data set with those of the new one in the source code inputs. The source code covers the whole ML pipeline, e.g., the sampling strategy, seed values, hyperparameters, algorithm implementation and software dependencies. We call the results of these adaptions *reused ML solutions*. Step 3 collects metadata to compare the performance of recommended ML solutions and reused ML solutions.

The last step of the evaluation approach uses an AutoML system to generate a new baseline ML solution for both use cases. This step compares a new ML model produced by AutoML against a reused ML solution obtained with AssistML. This allows the discussion of the advantages and disadvantages of both AutoML and AssistML. Here, we used *H2O AutoML* 3.32.1.6 with default configurations. The execution of H2O AutoML was triggered from an R script in R 3.6.3 on Windows 10. H2O AutoML trains and cross-validates ML models with different learning algorithms [21]. These include: gradient boosting machines, general linear models, random forests, and neural networks. It also trains stacked ensembles with combinations of these algorithms. The trained models are then ranked based on a single performance metric, which varies depending on the type of learning task. In our evaluation approach, the AutoML system ranks models using the area under the curve (AUC) metric.

6.3 Evaluation results

Overall, the evaluation approach delivers the following results: (a) ML solution recommendation reports for both use cases, (b) the performance comparison between *recommended* and *reused ML solutions*, (c) the execution times of the AssistML prototype for each of the four evaluation settings, and (d) the comparison of AssistML recommendations with the ML models trained with an AutoML system.

[9] https://github.com/al-villanueva/assistml.

The following paragraphs discuss the first three results. The last result is discussed in detail in Sect. 6.4. For simplicity, the discussion of specific results focuses on representative examples. The corresponding paragraph indicates this at the beginning. The complete list of recommendations generated for all four evaluation settings is available in our GitHub repository.[10]

ML solution recommendation report The report shown in Table 3 in Sect. 5.4 recommends the ML solution RFR_kick_011 in the q-steel-10 setting. The *Performance Labels* intuitively show the performance of this recommended ML solution. They give users a sense of how the solution compares to other solutions for the same use case. For instance, the labels in lines 4 and 5 show that the recommended ML solution achieves overall good prediction performance at the cost of a longer training time.

Recommendation reports also indicate the ML solution strengths and weaknesses w. r. t. data features. Citizen data scientists can compare suitable or unsuitable data features of existing ML solutions (see line 7 *Output Explanation*) to the data features in their new use case. As a result, they can select only those that resemble the suitable ones. *Data Preprocessing* information is given in line 9. In this example, one-hot-encoding is applied on categorical data. The information in lines 7 and 9, along with the source code in the repository, reduce the effort and time needed to prepare the new use case data.

ML Solution Patterns offer global explanations (see Sect. 3.3) of the ML solution. They provide citizen data scientists with relevant relationships on configuration and performance. For instance, the ML solution pattern in line 11 indicates that solutions with the Random Forest algorithm as well as 5 to 15 custom parameters tend to have a good recall (A label) and use the sklearn library. Citizen data scientists can restrict any adaptations they do on the ML solution to those that are indicated by these patterns. For instance, the pattern in line 11 helps them to avoid to tune more than the 15 parameters. This again reduces the time and effort they need for any adaptations. The report ends with *Deployment Requirements* (lines 13–16) to let the citizen data scientist decide if the solution can be deployed in the new use case.

Performance comparison between recommended and reused ML solutions The following discussion focuses on evaluation settings q-steel-10 and q-adult-10. Nevertheless, similar observations can be made for the other evaluation settings.

Each evaluation setting includes the comparison of the performance values of each recommended ML solution against the values of its corresponding reused ML solution. The recommended ML solution m_{rec} is one built for a previous use case, i. e., contained in the metadata repository, and described in a recommendation report, e. g., Table 3. The reused ML solution m_{reu} is one that adapts the source code of the recommended solution to use it with the new use case data, i. e., with the *steel plates* or *adult* data set.

We compare each pair of m_{rec} and m_{reu} in an evaluation setting using the absolute error for the three considered performance metrics, namely accuracy absolute error ($AccAE$), precision absolute error ($PreAE$), and recall absolute error ($RecAE$). For instance, Eq. 18 shows the formula for $AccAE$.

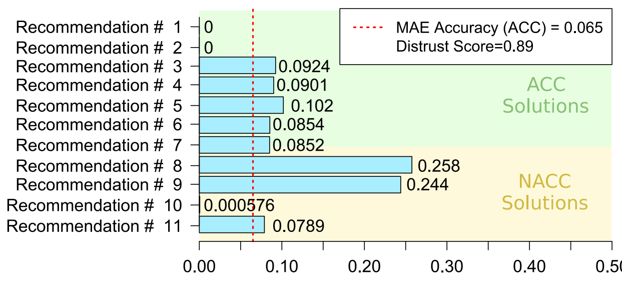
$$AccAE = \mid m.acc_{rec} - m.acc_{reu} \mid \tag{18}$$

where $m.acc_{rec}$ and $m.acc_{reu}$ are the accuracy values of m_{rec} and m_{reu}, respectively. We obtain the value for $m.acc_{rec}$ from the ML solution metadata in the repository. We obtain the value for $m.acc_{reu}$ by adapting the source code of the recommended ML solution m_{rec}, so that it can process the new data. Then, we evaluate the accuracy of the resulting reused ML solution m_{reu} by applying it to the new data. We conduct the same procedure with similar equations for precision and recall.

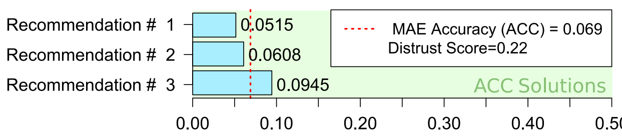
Note that these absolute errors do not evaluate the individual reused ML solutions. Instead, they assess the ability of AssistML to make recommendations with precise predictions of their performance. In this context, the performance values of a recommended solution m_{rec}, e. g., $m.acc_{rec}$, constitute a prediction of using the combination and configuration of ML components in m_{rec} on the new use case data. The performance of m_{reu}, e. g., $m.acc_{reu}$, is the true value of using that combination and configuration of ML components on the new data. The absolute error hence quantifies the variation between the expected or predicted performance of an recommended ML solution and the actually observed performance in the reused ML solution. As a consequence, smaller values of $AccAE$ $PreAE$ and $RecAE$ indicate that the performance prediction made by the recommendation m_{rec} is indeed close to the actual performance obtained once the recommended solution is reused and adapted to m_{reu}. Then, citizen scientists can be more confident that they get the performance that AssistML promises them with the recommendation m_{rec}.

Figure 9a shows the values $AccAE$ for recommendations in the setting q-steel-10, from the highest ranked ACC recommendation (Recommendation # 1) to the lowest ranked NACC recommendation (Recommendation # 11). Figure 9b shows $AccAE$ for ACC recommendations in the q-adult-10 setting. In this setting, the metadata repository does not contain ML solutions whose performance values lie in the NACC group. Therefore it only shows solutions from the ACC group. In general, ACC recommendations lead to smaller, more consistent absolute errors than NACC recommendations. This confirms the suitability of ACC solutions

[10] https://github.com/al-villanueva/assistml.

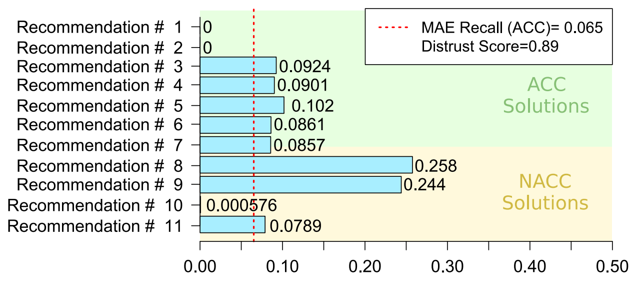


(a) q-steel-10 setting.

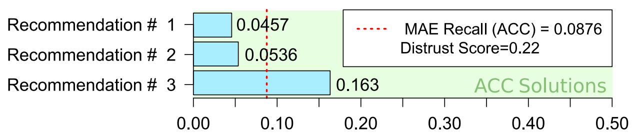


(b) q-adult-10 setting

Fig. 9 Accuracy absolute error ($AccAE$) per recommendation in the q-steel-10 and q-adult-10 settings

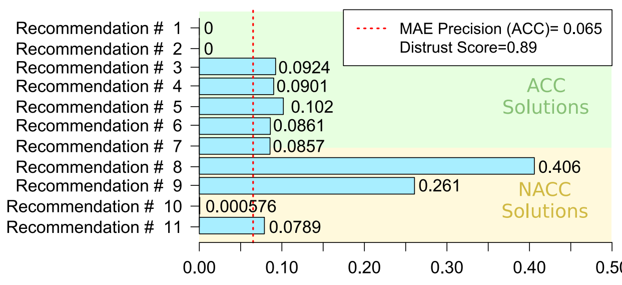


(a) q-steel-10 setting.

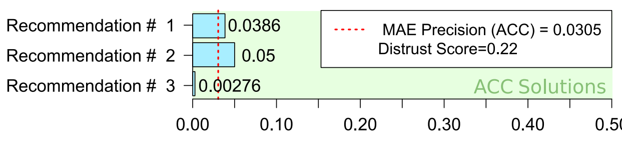


(b) q-adult-10 setting

Fig. 11 Recall absolute error ($RecAE$) per recommendation in the q-steel-10 and q-adult-10 settings



(a) q-steel-10 setting.



(b) q-adult-10 setting

Fig. 10 Precision absolute error ($PreAE$) per recommendation in the q-steel-10 and q-adult-10 settings

as recommendations for a new use case. Nevertheless, NACC recommendations can be contrasted with ACC recommendations to analyze the small differences that contribute to the good performance of ACC recommendations.

The mean absolute error of accuracy (MAE Accuracy), shown as a dotted red line, summarizes the comparison for the ACC recommendations. Note that MAE Accuracy is higher in setting q-steel-10, where the data set similarity is lower (level 1). AssistML warns the user about this low data set similarity with a high distrust score of 0,89. In setting q-adult-10, where the data set similarity is higher (level 2), the MAE and the distrust score provided by AssistML decrease. This behavior of the values confirms the ability of the distrust score to inform users about the error to expect when adapting the recommended ML solutions.

The values of $PreAE$ (see Fig. 10) and $RecAE$ (Fig. 11) behave similarly to the values of $AccAE$ for both use cases. This indicates that AssistML is able to make recommendations and predict their performance for the new use case data across all considered performance metrics.

Recommendations #1 and #2 for the evaluation setting q-steel-10 represent the most accurate cases, i. e., where the recommended and reused ML solutions achieve the same performance across all metrics. This means that citizen data scientists get exactly the performance that AssistML-promises with the recommended ML solutions when they apply these recommendations to their new use case data. A detailed analysis of the recommendations' metadata indicates that the training settings in these recommended ML solutions include grid search tuning for the hyperparameters. The reused ML solutions do not employ the final hyperparameter values, but they reuse the grid search training settings and carry out hyperparameter tuning for the new use case data again. This way, the reused ML solutions can better adapt to the nature of the new data set.

Recommendation #8 for the evaluation setting q-steel-10 represents an exceptional case, because the absolute error is between 25 and 40 % in each performance metric. A detailed analysis reveals that this NACC recommendation #8 employs the same learning algorithm as ACC recommendation #2, i. e., support vector machines. However, they have different configurations and training settings, e. g., different implementation libraries, data sampling ratios, and library dependencies. This highlights the importance of having both ACC and NACC recommendations. NACC recommendations serve as counter examples with which citizen data scientists can better understand why ACC recommendations perform well.

Table 5 Execution times of the AssistML prototype. Values in seconds (s)

Evaluation setting	Mean (s)	Median (s)	Standard Dev. (s)
q-steel-10	30.906	26.822	8.376
q-steel-20	12.292	12.225	0.762
q-adult-10	41.102	41.058	1.292
q-adult-20	41.545	41.410	0.792

Also note that the MAE for only ACC recommendations, shown as the dotted red lines, is relatively small in both evaluation settings. In both use cases, it is between 3 and 8.7% across all performance metrics. This is acceptable for citizen data scientists. It constitutes a negligible price to pay, since they can save huge efforts by avoiding the time-consuming implementation of completely new ML solutions. One alternative for citizen data scientists is to tune the hyperparameters and other settings to improve the performance in the new use case. However, this additional effort may only lead to slight performance increases of about 1 to 2%-points for any of the metrics. Such an elaborate parameter tuning is usually not a practical option for citizen data scientists in light of such small performance increases. Another alternative is to use an AutoML system to obtain an optimized ML model without any user involvement. This however may imply several sacrifices, e. g., the simplicity of the model or the explanations about the model's behavior. This alternative is further discussed in Sect. 6.4.

Execution times to generate recommendations. The results obtained confirm the advantages of saving efforts and time during the development of ML solutions. The prototype delivers recommendation lists for both use cases in a matter of seconds. Table 5 shows the average and median execution times collected after carrying out a randomized execution plan three times for all four evaluation settings. The execution time for any evaluation setting never surpasses 45 s, with standard deviation remaining also low. In some cases, it is even below 15 s.

Note that these measurements concern the execution of single experiments. In a productive environment, the demands for this assistant service may fluctuate, with demand peaks and bottoms. The prototypical implementation is thus designed as a stateless web service that can be scaled and replicated as necessary.

6.4 Detailed comparison to AutoML

AutoML represents an alternative for citizen data scientists to develop ML models. We applied H2O's AutoML [21] to both the *steel plates* and *adult* data sets. H2O's AutoML uses default settings to reflect the interest of the citizen data scientist to minimize input.

In the following, we compare AutoML results with the reused ML solutions recommended by AssistML for the most demanding evaluation settings, i. e., q-steel-10 and q-adult-10. The comparison focuses on the advantages and disadvantages of using each development approach for the citizen data scientist. The comparison includes the execution time, the number of recommended ML models, the performance values of the models, their complexity, and the amount of information each approach provides to explain ML solutions.

Comparison on the steel plates data Table 6 summarizes the output of both approaches for the *steel plates* data. From it, it is clear that AutoML takes much longer than AssistML to deliver results. AutoML takes 41 min to deliver a trained ML model, whereas AssistML takes only 30.9 s to provide recommendations. Nevertheless, AssistML still requires to adapt the source code of the recommended ML solution to the new use case data. This *adaptation time* varies depending on the user's skills and the complexity of the solution. Thus, the adaptation time is indicated in Table 6, but not quantified. Yet, AssistML provides information to facilitate this adaptation via the recommendation report. Furthermore, it provides access to the original source code in the repository that may be applied to the new use case data by mainly adjusting the names of data features. Afterwards, the ML model in the ML solutionstill needs to be trained. For the recommendations in evaluation setting q-steel-10, this takes negligible time.

Besides the highest ranked ML model, AutoML also gives a *leaderboard* of all the models it trained. The intention is to offer a set of options to the citizen data scientists. Yet, the number of additional models may become too big to analyze and compare comprehensively. AssistML takes this into consideration and caps the length of the list of recommendations to a sample, to avoid overwhelming citizen data scientists with too many options. In the q-steel-10 setting, H20's AutoML delivers a huge list of 60 ML models, while AssistML limits its recommendations to a reasonable amount of eleven ML solutions.

Regarding the observed performance, we use accuracy to compare models in AutoML's leaderboard to the reused solutions based on ACC recommendations from AssistML. This metric is delivered by both approaches and is likely to be the first metric that citizen data scientist use to assess the quality of an ML solution. Overall, AutoML delivers less performing models than AssistML for the *steel plates* use case. Even the AutoML model with the highest accuracy (79.68%) is below the reused ML solution with the lowest accuracy (85.33%). Besides this, the lowest accuracy of AutoML models is 50%, which is achieved by 13 different Deep Learning classifiers. This is probably a sign of unsuitable hyperparameters leading to overfitting. They are still included in the leaderboard, because they are part of the set of models that AutoML trains.

Moreover, the range of accuracy values of the AutoML models (29,68%-points) is twice as big as that of AssistML reused ACC solutions (14,67%-points). This can also be related to several underfitted models that are included in AutoML's leaderboard. Furthermore, this leaderboard of AutoML comprises many more ML models, which also are more diverse regarding their accuracy values. Contrary to this, AssistML's concept of limiting recommended solutions to the ACC solutions contributes to reduce the range in accuracy values that reused solutions may have. This results in fewer, only relevant ML solutions being presented to the user. The reused ML solutions are then less diverse regarding accuracy. Furthermore, they have a less probability to be over- / underfitted, as AssistML ensures that their hyperparameters have been tried and tested in several similar use cases.

Regarding the complexity of the resulting ML models, Table 6 compares the highest ranked AutoML model to the first recommendation from AssistML. AutoML produces an stacked ensemble consisting of five models based on four different learning algorithms. This is in line with AutoML's goal to find ways to further optimize performance, regardless of the increased complexity resulting ML models may have. Instead, AssistML recommends a single classifier based on a single learning algorithm, i. e., a Gaussian naive Bayes classifier. This represents a substantial difference in efforts for a citizen data scientist wishing to understand the functioning of either result. For AutoML, the citizen data scientist must understand the logic of four different learning algorithms, plus the logic behind stacking [42] to combine their individual predictions. For AssistML, s/he only needs to understand one learning algorithm.

Finally, the citizen data scientist obtains different amounts of information about the recommendations from each approach. AutoML provides six different performance metrics and their confusion matrices to describe each ML model in the leaderboard. The performance metrics include area under the curve (AUC), mean square error (MSE), root-mean-square error (RMSE), Logloss, area under the precision and recall curve (AUCPR) and error/accuracy. In contrast, AssistML describes the ML solution with much more information in the more intuitive recommendation report. In addition to real values of performance metrics, this recommendation report offers performance labels. These are more appropriate for citizen data scientist to compare different ML solutions and to analyze the trade-offs between several metrics. Furthermore, the report provides more easy-to-understand explanations of a solution's functioning, e. g., regarding the suitability of certain features or data preprocessing techniques as well as ML solutionpatterns. This information essentially ease the task of citizen data scientists to understand a recommended ML solutionand finally to apply it on the new use case data.

The results in this use case illustrate the disadvantages of using AutoML compared to AssistML. For the sake of avoiding user intervention, AutoML incurs in much longer execution times. It trains a predefined set of models with default hyperparameters, which leads to overfitted models. In the end, AutoML delivers many less performing and more complex ML models for citizen data scientists to choose. Moreover, AutoML systems only dispose of a few expert ML metrics to understand the functioning of the models. Here, AssistML provides citizen data scientists with more helpful and even less complex information in a recommendation report.

Comparison on the adult data Table 7 summarizes the outputs of both approaches for the *adult* use case. The execution times in this use case are similar to those in the *steel plates* use case. Citizen data scientists need to wait over 40 min to obtain ML models from AutoML, but only 41 s to obtain recommendations from AssistML.

Concerning the number of ML models or ACC recommendations, AutoML still provides users with a similarly high number of ML models, whereas AssistML can even narrow down its number of ACC recommendations. This is in part due to the contents of the repository, but also thanks to the concept of comparing ACC with NACC solutions. This concept lets AssistML determine if a solution is relevant to be included in the recommendation list, and if not, to omit it.

Regarding the observed performance, the comparison shows this time that both approaches achieve similar accu-

Table 6 AutoML versus AssistML for the evaluation setting *q-steel-10*

Criterium	AutoML	AssistML
Execution time	41 min	30.9 s + adaption time + 0.3 to 0.7 s training time
Number of models /recommendations	60	11
Performance (accuracy)	50–79.68%	85.33–100%
Highest ranked model/ solution	Stacked ensemble with 5 models: 1 deep learning, 2 random forests, 1 gradient boosting machines, 1 general linear model	Gaussian naive Bayes classifier
Available information	6 performance metrics and confusion matrix	ML solution recommendation report

Table 7 AutoML vs AssistML for the evaluation setting *q-adult-10*

Criterium	AutoML	AssistML
Execution time	41.76 min	41.1 s + adaption time + 0.45 to 13.5 s training time
Number of models /recommendations	56	3
Performance (accuracy)	80.03–83.60%	75.64–84.90%
Highest ranked model/ solution	Stacked ensemble with 54 models: 26 deep learning, 2 random forests, 25 gradient boosting machines, 1 general linear model	Random forest classifier
Available information	6 performance metrics and confusion matrix	ML solution recommendation report

racy values with their best ML model or reused solution, with a marginal difference of 1.3%-points still in favor of AssistML. This further shows that the AutoML's optimization approach to model training again does not result in better performance. Yet, it is important to note that the value range of AutoML models is narrower (3.57%-points) than that of AssistML ACC recommendations (9.26%-points).

However, AutoML achieves comparable performance in this use case at the expense of significantly more complex ML models. The complexity of AutoML's highest ranked ML model is even much higher than that in the steel plates setting. AutoML's highest ranked ML model in the adult setting consists of an stacked ensemble of 54 individual models based on four learning algorithms. Furthermore, AutoML again only delivers complex ML metrics to explain this model. Understanding the functioning of such a complex model is likely to require the intervention of expert data scientists. On the other hand, AssistML recommends a Random Forest classifier, along with the more intuitive explanations in the recommendation report. This makes it possible for citizen data scientists to understand this ML solution without involving expert data scientists.

The results in the *adult* use case further illustrate the advantages of using AssistML. AssistML can deliver a compact list of ML solution recommendations in less than 42 s. Furthermore, AssistML reused solutions achieve performance on par with that of AutoML models, but with a significantly less complex structure. Even then, AssistML provides more comprehensive and intuitive information about the performance and configuration of its recommendations. This reduces the effort of adapting existing source code, while it also facilitates understanding of its functioning. Ultimately, this increases the probability of citizen data scientists to achieve good performance intuitively and quickly.

6.5 Assessment

In this section, we discuss whether the characteristics of AssistML fulfill the four practical requirements that citizen data scientists face to develop ML solutions (see Sect. 2.2).

Regarding [R_1], the concept is expected to reuse information from existing ML solutions. AssistML fulfills this requirement with the use of a metadata repository containing previously developed ML solutions. The repository contains source code, training and test data to reproduce the results reported for each ML solution. Moreover, the metadata, summarized in the recommendation report, describe in detail the adaptations that are necessary to apply the recommended ML solutions to the new data sets. These resources significantly speed up the development of new solutions.

The concept fulfills [R_2], as it provides recommendation reports (see Table 3) that offer global explanations of the ML solutions, i. e., of their overall behavior and composition. The reports describe the solution's performance and configuration in an intuitive manner. They thereby avoid the use of expert metrics and enable the comparison of good and bad configurations via comprehensible performance labels. In addition, they provide patterns to explain the ML solutions' functioning w. r. t. certain data features and data preprocessing techniques. In this regard, they offer citizen data scientists more intuitive explanations than approaches of AutoML, Meta-Learning, or XAI to understand and select ML solutions.

Regarding [R_3], the concept is expected to provide recommendations in a responsive manner. AssistML has very short execution times when generating recommendations and thus fulfills this requirement. In addition, citizen data scientists only need to specify a minimum amount of information to issue queries q_n. These two factors allow citizen data scientists to assess the feasibility of multiple ML solutions efficiently. They can iterate on the recommendation process quickly and thereby, e. g., change their performance preferences. In this respect, AssistML offers citizen data scientists more flexibility than AutoML to explore different implementation alternatives.

AssistML fulfills [R_4], as citizen data scientists may state preferences for multiple user-defined performance criteria in a query q_n. AssistML then considers all criteria simultaneously and offers intuitive performance labels for each of them. These labels clearly indicate the trade-offs an ML solutionimplies across different criteria. Citizen data scientists may use this to intuitively compare the recommended ML solutionrelative to others.

7 Summary and future work

This paper presents AssistML, a concept to recommend ML solutions. This concept is tailored to the practical requirements of practitioners with less ML knowledge, e. g., citizen data scientists. AssistML analyzes metadata of existing ML solutions stored in a repository to recommend suitable matches for the user query based on data similarity and performance evaluations. Citizen data scientists can adapt and reuse the recommended ML solutions with reduced effort and time thanks to the source code and metadata available in the repository. Furthermore, AssistML offers intuitive explanations of the recommended ML solutions, e. g., by identifying frequent patterns among data features. We also provide a prototypical implementation of the concept and its evaluation with two use cases. The AssistML prototype is responsive, as it provides recommendations within 12 to at most 42 s. In addition, the finally implemented ML solutions show a good performance that is very close to the performance of the recommendations. Compared to AutoML, AssistML offers citizen data scientists simpler, intuitively explained ML solutions in considerably less time. Moreover, these solutions perform as well as or even better than AutoML models.

Overall, AssistML provides very promising results, which can serve as basis for a new development approach for ML solutions in organizations. Possible future work directions include the analysis of the queries issued to AssistML to guide the expansion of the metadata repository. For instance, frequently used performance criteria can be turned into preference patterns. Moreover, our evaluation of AssistML discussed in Sect. 6 is currently restricted to data sets with categorical, numeric, datetime, and textual feature types. Future work may hence populate the metadata repository with ML solutions that are tailored to other features types and more high-dimensional data sets, such as time series data or image data. Usually, ML solutions that are tailored to such data characteristics are neural networks, e. g., CNNs or RNNs. This way, we may evaluate whether AssistML is able to compare different neural network structures contained in the repository and to find the best network structure for a given high-dimensional data set.

Author Contributions AGVZ wrote the main manuscript text, prepared the figures, and designed the prototypical implementation and carried out the evaluation. PR co-wrote and reviewed the main manuscript text and provided feedback on the figures. CW and BM reviewed the manuscript and provided feedback on the figures.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was supported by the Landesministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg (Ministry of Science, Research and the Arts of the State of Baden-Wurttemberg) within the Nachhaltigkeitsförderung (sustainability support) of the projects of the Exzellenzinitiative II. The authors also thank the German Federal Ministry of Education and Research (BMBF) for financial support in the Software Campus initiative.

Data Availability All data, materials, and software code used for our prototypical implementation and for our evaluation are available on GitHub (https://github.com/al-villanueva/assistml).

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Adler, P., et al.: Auditing black-box models for indirect influence. Knowl. Inf. Syst. **54**(1), 95–122 (2018). https://doi.org/10.1007/s10115-017-1116-3
2. Baier, L., et al.: challenges in the deployment and operation of machine learning in practice. In: Proceedings of the 27th European Conference on Information Systems (2019)
3. Bank, M., et al.: Textual characteristics for language engineering. In: Proceedings of the 8th International Conference on Language Resources and Evaluation, pp. 515–519 (2012)
4. Bernardi, L., Mavridis, T., Estevez, P.: 150 Successful machine learning models: 6 lessons learned at Booking.com. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1743–1751 (2019). https://doi.org/10.1145/3292500.3330744
5. Bilalli, B., Abelló Gamazo, A., Aluja Banet, T.: On the predictive power of meta-features in OpenML. Int. J. Appl. Math. Comput. Sci. **27**(4), 697–712 (2017). https://doi.org/10.1515/amcs-2017-0048
6. Biondi, G.O., Prati, R.C.: Setting parameters for support vector machines using transfer learning. J. Intell. Robot. Syst. **80**(1), 295–311 (2015)
7. Breck, E., et al.: The ML test score: a rubric for ML production readiness and technical debt reduction. In: Proceedings of the 2017 IEEE International Conference on Big Data, pp. 1123–1132 (2017). https://doi.org/10.1109/BigData.2017.8258038
8. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. J. Artif. Intell. Res. **70**, 245–317 (2021). https://doi.org/10.1613/jair.1.12228
9. Chen, A., Chow, A., Davidson, A., DCunha, A., Ghodsi, A., Hong, S.A., Konwinski, A., Mewald, C., Murching, S., Nykodym, T., et al.: Developments in MLflow: a system to accelerate the machine learning lifecycle. In: Proceedings of the 4th International Workshop on Data Management for End-to-End Machine Learning (2020)

10. Ester, M., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pp. 226–231 (1996)
11. Ethayarajh, K., Jurafsky, D.: Utility is in the eye of the user: a critique of NLP leaderboards. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pp. 4846–4853 (2020)
12. Feurer, M., et al.: Auto-Sklearn 2.0: the next generation (2020). arXiv:2007.04074
13. Flaounas, I.N.: Beyond the technical challenges for deploying machine learning solutions in a software company. In: Proceedings of the ICML Workshop on Human in the Loop Machine Learning (2017)
14. Gijsbers, P., et al.: An open source AutoML benchmark. In: Proceedings of the 6th ICML Workshop on Automated Machine Learning (2019)
15. Goldstein, A., et al.: Peeking inside the black box: visualizing statistical learning with plots of individual conditional expectation. J. Comput. Graph. Stat. **24**(1), 44–65 (2015)
16. Gröger, C.: Building an Industry 4.0 analytics platform. Datenbank-Spektrum **18**(1), 5–14 (2018). https://doi.org/10.1007/s13222-018-0273-1
17. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. **29**(2), 1–12 (2000). https://doi.org/10.1145/335191.335372
18. Henelius, A., et al.: Interpreting classifiers through attribute interactions in datasets. In: Proceedings of the 2nd ICML Workshop on Human Interpretability in Machine Learning (2017)
19. Hirsch, V., Reimann, P., Kirn, O., Mitschang, B.: Analytical approach to support fault diagnosis and quality control in end-of-line testing. Proced. CIRP **72**, 1333–1338 (2018). https://doi.org/10.1016/j.procir.2018.03.024
20. Hirsch, V., Reimann, P., Mitschang, B.: Incorporating economic aspects into recommendation ranking to reduce failure costs. Proced. CIRP **93**, 747–752 (2020). https://doi.org/10.1016/j.procir.2020.03.026
21. LeDell, E., Poirier, S.: H2O AutoML: scalable automatic machine learning. In: Proceedings of the 7th ICML Workshop on Automated Machine Learning (AutoML) (2020). https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf
22. Olson, R.S., et al.: Automating biomedical data science through tree-based pipeline optimization. In: Proceedings of the European Conference on the Applications of Evolutionary Computation, pp. 123–137 (2016)
23. Paleyes, A., Urma, R.G., Lawrence, N.D.: Challenges in Deploying Machine Learning: a Survey of Case Studies. NeurIPS Workshop on ML Retrospectives, Surveys & MetaAnalyses (2020)
24. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
25. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. Int. J. Very Larg. Data Bases (VLDB J.) **10**(4), 334–350 (2001). https://doi.org/10.1007/s007780100057
26. Raina, R., Ng, A.Y., Koller, D.: Constructing informative priors using transfer learning. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 713–720 (2006)
27. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F., Dennison, D.: Hidden technical debt in machine learning systems. Adv. Neural Inf. Process. Syst. **28**, 2503–2511 (2015)
28. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. **34**(1), 1–47 (2002). https://doi.org/10.1145/505282.505283
29. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. Inf. Process. Manag. **45**(4), 427–437 (2009). https://doi.org/10.1016/j.ipm.2009.03.002
30. Subianto, M., Siebes, A.: Understanding discrete classifiers with a case study in gene prediction. In: Proceedings of the 7th IEEE International Conference on Data Mining, pp. 661–666 (2007)
31. Van Rijn, J.N., Hutter, F.: Hyperparameter importance across datasets. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2367–2376 (2018). https://doi.org/10.1145/3219819.3220058
32. Vanschoren, J.: Meta-learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Automated Machine Learning—Methods, Systems, Challenges, pp. 35–61. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5_2
33. Vanschoren, J., et al.: OpenML: networked science in machine learning. SIGKDD Explor. Newsl. **15**(2), 49–60 (2013). https://doi.org/10.1145/2641190.2641198
34. Vartak, M., Subramanyam, H., Lee, W.E., Viswanathan, S., Husnoo, S., Madden, S., Zaharia, M.: ModelDB: a system for machine learning model management. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics (2016)
35. Villanueva Zacarias, A.G., Ghabri, R., Reimann, P.: AD4ML: axiomatic design to specify machine learning solutions for manufacturing. In: Proceedings of the 24th International Conference on Information Reuse and Integration for Data Science (IRI), pp. 148–155. IEEE (2020). https://doi.org/10.1109/IRI49571.2020.00029
36. Villanueva Zacarias, A.G., Reimann, P., Mitschang, B.: A framework to guide the selection and configuration of machine-learning-based data analytics solutions in manufacturing. Proced. CIRP **72**, 153–158 (2018). https://doi.org/10.1016/j.procir.2018.03.215
37. Villanueva Zacarias, A.G., Weber, C., Reimann, P., Mitschang, B.: AssistML: a concept to recommend ML solutions for predictive use cases. In: Proceedings of the 8th International Conference on Data Science and Advanced Analytics (DSAA) (2021). https://doi.org/10.1109/DSAA53316.2021.9564168
38. Wagstaff, K.L.: Machine Learning that Matters. In: Preceedings of the 29th International Conference on Machine Learning, pp. 1851–1856 (2012)
39. Weber, C., Hirmer, P., Reimann, P.: A model management platform for industry 4.0—enabling management of machine learning models in manufacturing environments. In: Proceedings of the 23rd International Conference on Business Information Systems (BIS), pp. 403–417 (2020). https://doi.org/10.1007/978-3-030-53337-3_30
40. Weber, C., Hirmer, P., Reimann, P., Schwarz, H.: A new process model for the comprehensive management of machine learning models. In: Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS), pp. 415–422. SCITEPRESS, Heraklion, Kreta, Griechenland (2019). https://doi.org/10.5220/0007725304150422
41. Wilhelm, Y., Schreier, U., Reimann, P., Mitschang, B., Ziekow, H.: Data science approaches to quality control in manufacturing: a review of problems, challenges and architecture. In: Proceedings of the 14th Symposium on Service-Oriented Computing (SummerSOC), Communications in Computer and Information Science (CCIS), pp. 45–65. Springer-Verlag (2020). https://doi.org/10.1007/978-3-030-64846-6_4
42. Wolpert, D.H.: Stacked generalization. Neural Netw. **5**(2), 241–259 (1992)
43. Xin, D., et al.: Whither AutoML? Understanding the role of automation in machine learning workflows. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp. 8–13 (2021)

44. Yang, Y.: An evaluation of statistical approaches to text categorization. Inf. Retr. (1999). https://doi.org/10.1023/A:1009982220290
45. Zaharia, M., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. **41**(4), 39–45 (2018)