Automation of Machine Learning Pipeline Design by an Ontology as an Integrative Meta-Learning Model

Maksim Aliev, Sergey Muravyov
NRU ITMO
St.Petersburg, Russian Federation
imaxaliev@gmail.com, mursmail@gmail.com

***Abstract*—**This article considers the problem of automating the design of machine learning (ML) pipelines. Methods for automating the design of ML pipelines were analyzed. Based on the analysis performed an ontology would be promising for solving the above problem. A method for automating the design of ML pipelines based on ontological engineering was proposed. An ML ontology aimed at constructing pipelines was created. An application for automated construction of pipelines based on the created ontology was developed. The effectiveness of the developed solution has been assessed experimentally, as compared with TPOT, which is one of the state-of-the-art automated pipeline construction tools. The solution presented not only appears to be more efficient in terms of the quality of the result obtained in the minimum required time, but is also comparable to the above tool regardless of the time of running.

I. INTRODUCTION

Machine learning is often described as "a field of study that gives computers the ability to learn without being explicitly programmed" [1]. The demand for functionality provided by the ML techniques is growing fast, and successful solutions based on this approach can be found in an increasing number of fields of science, technology and society in general. Over the past years, automated machine learning (AutoML) solutions have been developed in response to the challenges faced by those who do not have the expertise in machine learning, but who still have to use it occasionally to solve their specific problems [2]. Also, this technology is aimed at eliminating the need to search by brute force for algorithms, their hyperparameters, to name a few.

This article considers the task of automating the design of ML pipelines. In most of the existing approaches, it is considered an optimization problem, which leads to a higher design complexity. On the other hand, there is another approach, called meta-learning, which refers to collecting statistics of the effectiveness of certain algorithms in similar problems [3]. This approach allows reduce overall the pipeline search and optimization complexity, however, it has the following limitation: meta-learning efficiency in ML pipeline design automation depends on the amount of information about similar tasks stored in a particular system. To meet this issue, it seems promising to accumulate data on the effectiveness of pipelines in specific cases extracted by individual systems into a common knowledge base, with further using this base as a meta-learning model for predicting the best possible pipeline for any new tasks.

The solution we present is based on the ontological approach [4]. Ontology means formalization of a certain area of knowledge as a conceptual scheme. It consists of objects - instances of classes of the subject area, relations between them, and other rules, also called axioms, accepted in the field. Axioms are described as subject-object-predicate triplets, based on the RDF (Resource Description Framework) data model. The standard language for describing ontologies is OWL (Web Ontology Language). The choice of the ontological approach for solving the problem of ML pipeline design automation is based on the following features:

1) The inference of a pipeline from an ontology requires almost no computational costs, as each request to the graph database, on average, is executed in a linear time.
2) An ontology is a standardized format that allows data from multiple sources to be integrated into it.
3) The ability to infer implicit information. Thus, it becomes possible to automatically construct a pipeline in specific cases.

Our main goal is reducing the complexity of automated ML pipeline design using an ontology as a meta-learning model. To achieve the above goal we have set the following tasks: (i) to perform the analysis of methods for automation of the ML pipeline design; (ii) to develop a solution for ML pipeline design automation based on the ontological engineering; (iii) to implement it; and finally, (iv) to perform implementation testing and comparison of our results with TPOT (Tree-based Pipeline Optimization Tool), one of the acknowledged AutoML tools [5].

The article has the following structure: the section following the introduction contains the analysis of methods for automation of the ML pipeline design; the third section describes the proposed solution for ML pipeline design automation based on the ontological engineering; the fourth section contains implementation details; and finally, the fifth section deals with the experiments conducted, including app testing and comparing our results with TPOT.

II. Analysis of methods for automation of the pipeline design

The existing approaches have been classified as two categories: those based on optimization and those based on semantic technologies.

A. Tree-based Pipeline Optimization Tool

TPOT solution is essentially the optimization of ML pipelines by using a version of genetic programming (GP), an evolutionary computing technique for automatically constructing computer programs. In terms of implementation, TPOT is a wrapper over scikit-learn, a python ML package [6]. In TPOT, each pipeline operator is linked to the ML algorithm. In order to combine these operators in the ML pipeline, we consider them to be GP primitives, and build GP trees based on them. During the exemplary workflow shown in Fig. 1, two copies of the dataset are modified in a sequential manner by each operator, then merged into one dataset, and finally used to perform classification.

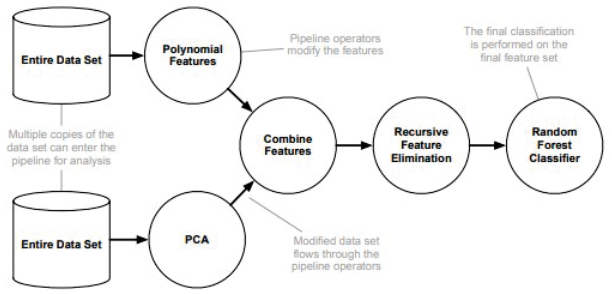

Fig.1. Tree-based machine learning pipeline [1, Fig.2].

The analysis has revealed the following limitations of TPOT solution: (i) a sensible initialization process; (ii) a need to optimize a large population of solutions, which can be slow and costly for certain cases.

B. Tree-based Pipeline Optimization Tool

ML-Plan is a new approach to AutoML based on Hierarchical Task Networks (HTN) [7]. HTN is a well-established intelligent scheduling technique, usually implemented as a heuristic best-first search on the graph, induced by the scheduling technique. The optimization potential of the earlier solutions to setting up HTN-based data mining pipelines has proved to be limited. One of these solutions ranked candidates based on the frequency of RapidMiner usage, while another ran a bottom-up search based on a database of known issues. On the other hand, similarly to Auto-WEKA [8] and auto-sklearn [9], ML-Plan searches for the optimal pipeline by randomly substituting algorithms in each of its phases, down to their completion. The key advantage of ML-Plan over the above-mentioned approaches is that it has a special mechanism to prevent overfitting. Fig. 2 shows an example of ordered hierarchical task networks with 3 assessed pipelines.

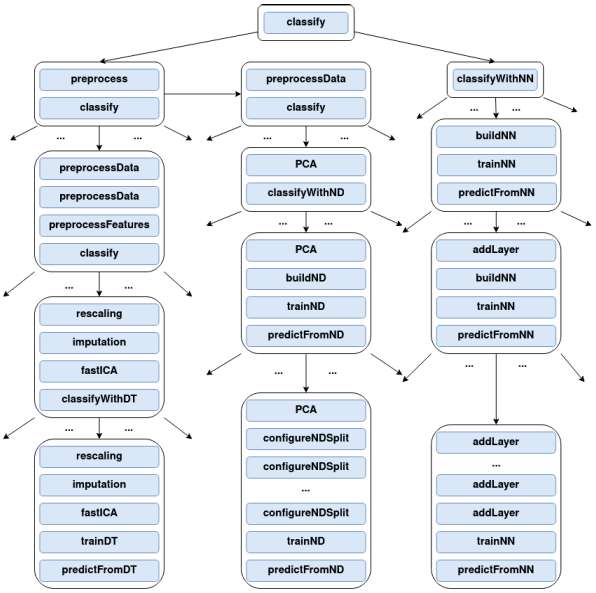

Fig.2 Ordered HTN [7, Fig.2]

The analysis has revealed the following limitations of ML-plan solution: (i) implementation of a predefined preference on learning algorithms, instead of adaptation to the dataset; (ii) the worst case time complexity of a best-first search algorithm is O(NlogN), which will be slow enough when searching among many solutions.

C. AlphaD3M

The Data Driven Discovery of Models (D3M) program was initiated to create an infrastructure for automatic discovery of an ML pipeline [alphad3m]. In the AlphaD3M solution, the pipeline synthesis problem is reduced to a simple single-player game in which the player iteratively builds the pipeline, choosing from a set of actions: insert, delete, or replace parts of the pipeline . The advantage of this approach is that by the end of the process, when the user receives the finished pipeline, the result is fully explainable, including all the actions and decisions that have produced it. Another advantage of this approach is that it leverages recent advances in deep reinforcement learning using self play, specifically the expert iteration algorithm and AlphaZero, by applying a neural network for predicting pipeline performance and action probabilities, along with a Monte-Carlo Tree Search.

The analysis has revealed that AlphaD3M solution has an advantage over previous approaches in terms of running time, which is reduced from hours to minutes. On the other hand, its disadvantages are (i) that it uses a lot of memory while running, and (ii) that its average performance (quality of results) is generally worse than others.

Having analyzed all the methods above, we can conclude that their common disadvantage is that they

require significant computational costs of a certain type to design an efficient pipeline.

D. SemFE

According to [10], development of ML pipelines is a complex and costly process because of the following three challenges since they consume more than 80% the overall time of development. The first is transparency of the results. It includes the fact that quality monitoring, task negotiation and interpretation of results require collaboration between experts from different areas. The asymmetric knowledge backgrounds, including complexity of engineering practices in manufacturing and sophistication of ML algorithms that constrain the transparency of ML results and models, make the communication time consuming and error-prone. The second challenge is data preparation. Data integration from multiple sources is a labor-intensive process that requires multifaceted domain knowledge and plentiful data complications. The third and last challenge is the generalisability of ML models. Each of the developed ML models usually adapts to the dataset and, for this reason, its reuse in a different context requires some effort.

Designed in Bosch, semantically enhanced ML pipeline, SemFE, with feature engineering, addresses these challenges, eliminating knowledge asymmetry, and making data science accessible to non-ML-experts [10]. This solution relies on ontologies for discrete manufacturing monitoring that encapsulate domain and ML knowledge. By processing ontologies through reasoning, SemFE automates the process of creating ML models. In particular, when welding specialists annotate raw welding data with features from a subject ontology, the reasoner obtains feature groups corresponding to them, and then, using the ML ontology, it infers feature processing algorithms and ML algorithms corresponding to the selected feature groups. Fig. A.1 shows a structural scheme of the ML pipeline in a general case.

To summarize, to the best of our knowledge, there are no studies that would treat ontologies as an integrative meta-learning model for constructing ML pipelines.

III. ONTOLOGY-BASED ML PIPELINE DESIGN AUTOMATION METHOD

In this section we describe a proposed method for automation of machine learning pipeline design by an ontology as an accumulative meta-learning model.

A. Base Workflow

To begin with, we define the operating principle of a system that automates the construction of pipelines based on semantic technologies. A scheme of its workflow is presented in Fig. A.2.

At the first stage of working with the software, the user will be required to provide a dataset. The system then extracts the dataset feature names and their corresponding data types. Further, the user is required to select the ML problem to be solved, classification or regression, as well as the target feature. Based on this data, a series of sequential requests to the ontology is carried out to obtain the result -- the implementation of the pipeline. These requests are:

- For pipeline inference;
- for inference of each pipeline step;
- for inference of the implementation of each pipeline step and hyperparameters associated with it.

Finally, the generated code is immediately provided to the user.

B. ML Ontology

To implement the proposed solution, a machine learning ontology based on expert estimation was developed, aimed at constructing pipelines, with the class hierarchy shown in Fig. 3.

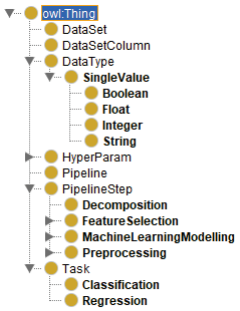

Fig. 3 Ontology class hierarchy

The ontology created has a minimal set of entities required to demonstrate how the solution works. Its structure is scalable enough, so we can expand it.

The process of outputting a pipeline suitable for the dataset is described using the following example.

```
:Classification :hasStoredData :Dataset2
```

Firstly, an instance of the classification task, that has *ObjectProperty hasStoredData*, with the value of the instance of similar dataset, is extracted. The dataset instance is presented below.

```
:DataSet :hasSolution :Pipeline2
```

As seen from the listing, the dataset instance has *ObjectProperty hasSolution*, with the value of the pipeline instance. Next, the structure of the pipeline instance is presented below.

```
:Pipeline :hasStep :SelectKBest, :Linear
SVC
```

The pipeline instance has *ObjectProperty hasStep*, linking it with the algorithm instance used at the current

pipeline step. An annotation order is used here to persist the order of steps. One of the algorithms, *LinearSVC*, is considered as an example.

```
:LinearSVC :hasHyperParam :C_5 , :Dual_F
alse;

      :hasImplementation              "from
sklearn.svm import LinearSVC"
```

It has *ObjectProperty hasHyperParam* that, as the name suggests, is intended to specify its hyperparameters. In this case they are C and dual. The algorithm is linked through *DataProperty hasImplementation* with the corresponding implementation in the Python language.

In general, the process of pipeline inference remains the same for any other ML task.

IV. Implementation

In this section we describe the implementation of the proposed method for an application we developed.

The method is implemented as a client-server web-application. SPARQL was used as the query language based on the RDF model, as it is the W3C's (Word Wide Web Consortium) recommendation for a query language for Semantic Web data.

Back-end part is written on python framework Django. The language choice is based on its high prevalence in the ML field (so the solution can be more competitive) and has the opportunity to execute the resulting pipeline.

Front-end part is written in the Angular framework. The choice is based on its features, including but not limited to component-based architecture, reactivity, typescript usage, HTTP client.

A. Back-end

Firstly, there must be a request to infer data types of features from a dataset. During the execution of the corresponding method, the data set is extracted from the request body, then it is converted to dataframe and the feature data types are inferred based on comparison of the values of its dtypes attribute fields with data types from the ontology.

Further in the course of the workflow, a request must occur to obtain the implementation of the pipeline. Then a graph is created based on the ML ontology in memory, using the rdflib library [rdflib]. Next, to generate a pipeline, a series of requests for data retrieval is made to this graph. Finally, pipeline implementation is passed to the client in the form of json, as with inferred data types.

B. Front-end

The client part consists of one main and two nested components and a single service. First, the *SetUpComponent* is considered. *onDataSetUpload* method accepts a dataset uploaded by the user via *input[type='file']*. Next, the *getFeatureTypes* method of the backend service is called. This method first places the data in the form data structure, then makes a POST request via *http-client* to the API endpoint */infer_feature_dtypes/* to execute the server logic described earlier. The component has methods that handle selection events of the ML problem and the target feature, and also an event of tab switching.

The component that should be presented on the next tab is intended to display the pipeline implementation. During its initialization, it calls *getResults* method of the backend service, passing the task to be solved as a parameter, from where a GET request is made to the */get_pipeline_implementation/* endpoint. This component also implements the following methods: checking for the presence of results and loading a third-party script to highlight the generated syntax.

V. Experiments

This section presents the results of experimenting with the implementation, which includes app testing and estimation of the constructed pipelines.

A. Application testing

First, a manual functional testing of the application is executed. We define the problem to solve and the dataset to learn. On the selected dataset, the multiclass classification problem is solved. After loading the dataset, selecting the task and the target feature, and clicking on the "Get Results" button, the user sees the generated pipeline. Intermediary and final interface states are shown in Fig. 4 and Fig. 5.

We also tested out our app on another data set, this time solving a regression task. The result is shown in Fig.6.

B. Evaluation of results and comparison

We run the pipeline obtained for the classification problem and estimate its quality by cross-validation with the default *scoring* and *cv* parameters, taking mean value from the results given. Next, we calculate the total server response time in the search and pipeline generation workflow. And finally, we run the *TPOTClassifier* algorithm from the *tpot* package with the following parameters: number of generations = 15, size of population per generation = 100, early stop if there is no progress in search = 3. The results of comparison are presented in Fig.7.

Further, by analogy, we run the pipeline obtained for the regression problem, evaluate its quality according to the performance metric we have chosen, and calculate the time spent on its synthesis. Subsequently, we run the *TPOTRegressor* algorithm from the tpot package with all the same parameters as during classification. The results of comparison are presented in Fig. 8.

In both plots, TPOT results are presented as a function, while our results as a dot. The reason is that our solution finds a single pipeline, while TPOT tries to

iteratively optimize solutions given at each next generation until the stop.

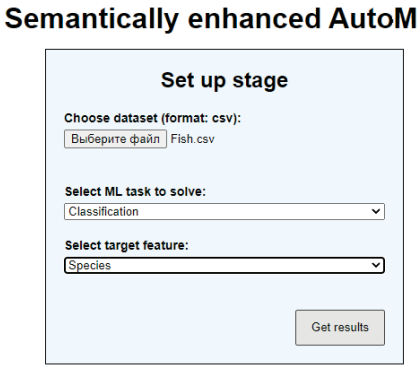

Fig.4 Intermediary interface state (classification task)

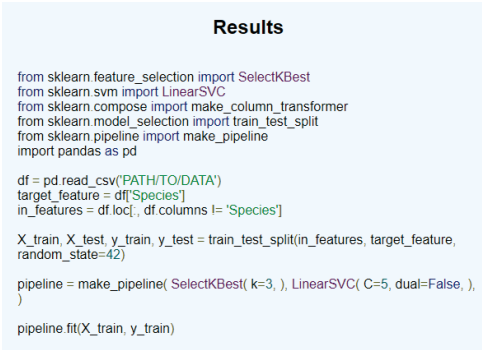

Fig.5 Final interface state (classification task)

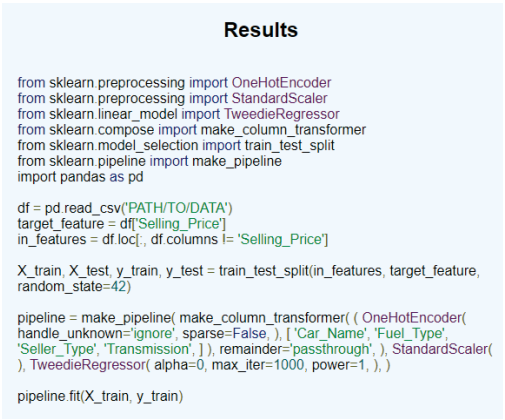

Fig.6 Final interface state (regression task)

Analyzing the results obtained, our solution not only appears to be more effective in terms of the quality of the result given in the minimum time required to obtain it, but is also comparable regardless of the running time. It is also important that the minimum time in our case is a few seconds, while for TPOT it takes about 4 minutes.

VI. Conclusion

The following results were achieved:

1) Methods for automation of ML pipeline design were analyzed. They were classified into two categories: those based on optimization and those based on semantic technologies.
2) A method for automation of the pipeline design based on ontological engineering has been developed.
3) An ML ontology was created, aimed at constructing pipelines.
4) An application has been developed for the automated construction of pipelines based on the created ontology.
5) An experimental evaluation of the efficiency of the developed solution in comparison with TPOT was performed.

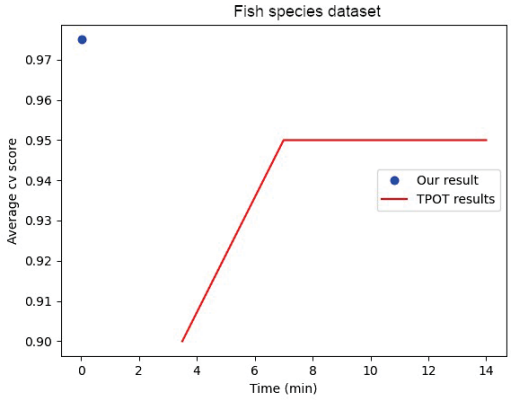

Fig.7 Performance and accuracy comparison: classification task

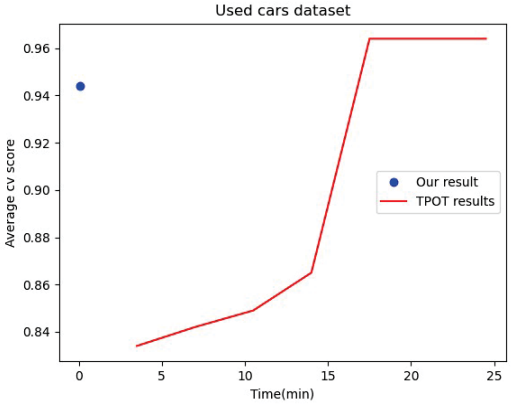

Fig.8 Performance and accuracy comparison: regression task

As a result, our solution appears to be more efficient in terms of operating time and the amount of memory occupied, and also comparable in the quality of the result given. However, our solution can not be competitive with the TPOT at this stage for searching for a pipeline for any possible dataset. On the other hand, from the estimations performed we can conclude that our approach is very prospective for a high-quality solution to the task of ML pipeline design automation, and it will be our next objective to make it as competitive as possible.

In the future we plan to meet the following tasks:

1) Develop persistent graph database for storing metadata about pipelines.
2) Complement the system with the ability of assigning pipelines to data sets via meta-learning.
3) Expand ontology structure and perform experiments on the OpenML-100 data sets.
4) Deploy the project as a publicly available service for automated ML pipeline design.

REFERENCES

[1] R. Olson, N. Bartley, R. Urbanowicz, J. Moore, Evaluation of a tree-based pipeline optimization tool for automating data science, 2016, pp. 485–492. doi: 10.1145/2908812.2908918.

[2] I. Drori, Y. Krishnamurthy, R. Rampin, R. Lourenço, J. Ono, K. Cho, C. Silva, J. Freire, Alphad3m: Machine learning pipeline synthesis, 2018.

[3] J. Vanschoren, Meta learning, Automated Machine Learning, Springer International Publishing, Cham, 2019, pp. 35–61. doi:10.1007/978-3-030-05318-5_2.

[4] P. Hitzler, M. Krötzsch, B. Parsia, P. Patel-Schneider, S. Rudolph, OWL 2 Web Ontology Language Primer, Technical Report, 2012. URL: https://www.w3.org/TR/owl2-primer/.

[5] R. S. Olson, J. H. Moore, Tpot: A tree-based pipeline optimization tool for automating machine learning, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning, Springer International Publishing, Cham, 2019, pp. 151–160. doi:10.1007/978-3-030-05318-5_8.

[6] Scikit-learn package, API docs, Web: https://scikit-learn.org/stable/modules/classes.html.

[7] F. Mohr, M. Wever, E. Hüllermeier, Ml-plan: Automated machine learning via hierarchical planning 107 (2018) 1495–1515. doi:10.1007/s10994-018-5735-z.

[8] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, K. Leyton-Brown, Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning, 2019, pp. 81–95. doi:10.1007/978-3-030-05318-5.

[9] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems, volume 2, 2015, pp. 2755–2763. URL: https://dl.acm.org/doi/10.5555/2969442.2969547.

[10] B. Zhou, Y. Svetashova, T. Pychynski, I.Baimuratov, A. Soylu, E. Kharlamov, Semfe: Facilitating ml pipeline development with semantics, in: CIKM'20: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 3489–3492. doi:10.1145/3340531.3417436.

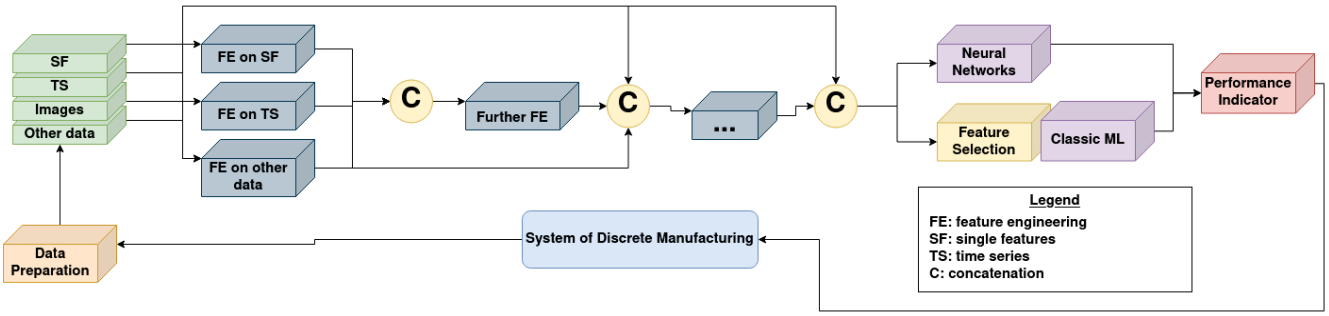

Fig. A.1 SemFE pipeline [11, Fig. 2]

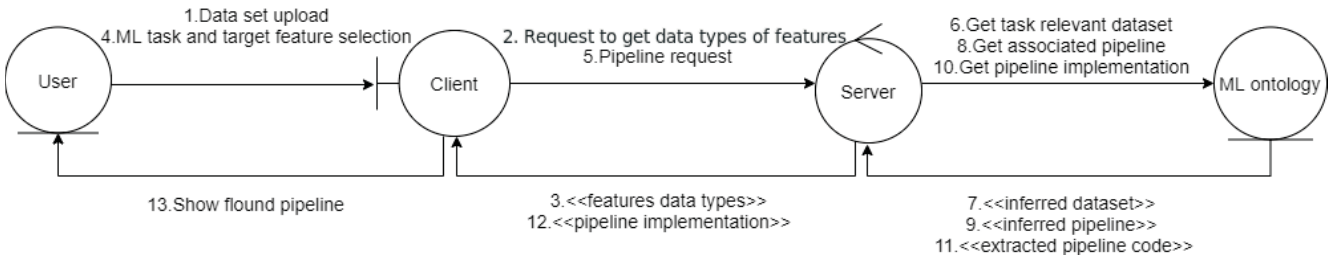

Fig. A.2 Base workflow