Записки научных
семинаров ПОМИ
Том 540, 2024 г.

I. L. Iov, N. O. Nikitin

FEATURE ENGINEERING PIPELINE OPTIMISATION IN AUTOML WORKFLOW USING LARGE LANGUAGE MODELS

Abstract. One important way to achieve more efficient automated machine learning is to involve meta-optimisation for all stages of the pipeline design. In this work, we aim to use large language models for feature engineering steps as both optimisers and domain-knowledge experts. We encode the feature engineering pipeline in natural language as a sequence of atomic operations. Black-box optimisation is implemented by requesting a feature engineering pipeline from the LLM using a prompt consisting of predefined instructions, dataset description, and previously evaluated pipelines. To increase the time efficiency and stability of optimisation, we implement a population-based algorithm to produce a set of pipelines with each LLM response instead of a single one. Multi-step optimisation is attempted to provide the LLM with additional domain knowledge. To analyse the performance of the proposed approach, we conduct a set of experiments on the open datasets. Random search has been chosen as a baseline for the optimisation task. We find that while straightforward results obtained with the gpt-3.5-turbo model are close to the baseline with the same time cost, population-based pipeline generation outperforms the baseline and other approaches. Our results confirm that the proposed approach can increase the overall performance of machine learning models with the same time cost for optimisation and fewer tokens needed to obtain the result.

§1. Introduction

Automated machine learning (AutoML) is a rapidly advancing field [1], as it allows one to apply machine learning techniques with no expertise, which is required for classical ML pipeline implementation. Selecting the correct features and models presents a considerable challenge, often demanding numerous iterations to reach the optimal solution. Conventional

Key words and phrases: AutoML, large language models, feature engineering, black-box optimisation.

This research was carried out within the state assignment of the Ministry of Science and Higher Education of the Russian Federation, project no. FSER-2024-0004.

AutoML approaches rely on domain knowledge and meta-features of the data to achieve the goal. Meta-learning attempts to replace expert knowledge with higher-level meta-models, designed to choose the models and parameters in automated machine learning (AutoML) more efficiently. Foundational language models encapsulate vast domain knowledge and can be used as experts for meta-learning.

In this work, large language models are used for feature engineering, which is not only the most time-consuming part of an ML system's design [9], but also requires a specialist to have domain knowledge and skills in machine learning methods. Feature engineering workflows are encoded as operation sequences, which may both be optimised using conventional methods and LLMs as optimisers. The random search optimisation method is applied as a simple baseline for a set of open datasets. First, a general optimisation strategy is applied by iteratively requesting the LLM to generate the feature engineering operations sequence based on the dataset information and the previous iterations logs. A population-based approach is proposed to generate a population of pipelines from each prompt to improve the exploitation capability and reduce the time and token cost of the optimisation. Finally, multi-step optimisation is implemented for both strategies to provide the LLM with more domain knowledge and presumably get a more effective solution.

§2. Related works

2.1. AutoML and MetaLearning. The AutoML field has been extensively studied in recent years, as it is able to apply machine learning methods to various domains even if the specialist does not have expert knowledge of ML methods. In other cases, it can also fully automate ML problem-solving when it is challenging to involve a human expert, which is common for industrial applications. Another important application of AutoML is the baseline generation to be further improved by an expert, thus reducing the time of searching for the final solution.

An AutoML pipeline can be roughly divided into four steps [62]:

- the data preparation step includes collecting data and forming a dataset,
- the feature engineering step includes creating new features from existing ones, reducing dimensionality, and otherwise extracting the most informative data representation with the least number of features,

- the model generation step may be implemented in many ways, yet it always results in a trained model as a candidate for the new best solution;
- finally, in the model evaluation step model scores are evaluated and either a new learning cycle starts or the final model is determined based on the evaluation result and the model selection method.

2.1.1. *Data Preparation.* Many methods have been developed for each step; no existing solution covers all of them. Data preparation can include results from the web [2], dataset balancing [3], and GANs used for synthetic data generation. Knowledge-based systems such as [12] can be used to clean data, while others view data cleaning as a boosting [4] or hyperparameter tuning problem [5]. Data augmentation is widely used, especially in computer vision with methods like contrast, shift, image blending and mixup, and specific packages are designed to help with such operations, including torchvision [6], ImageAug [7] and Albumentations [8].

2.1.2. *Feature Engineering.* Feature engineering is the most important step of the ML system design pipeline, as this step sets up the highest quality achievable for the entire system by defining feature predictability. It is also the most time-consuming step in the traditional pipeline [9] on average. Feature extraction methods aim to reduce the feature dimensionality, feature construction expands the dataset with new features and feature selection drops the least important features.

Feature selection may be performed by any optimisation method from random search to complete search, while simulated annealing and genetic algorithm are the most popular. For these methods, either some data can be used as a scoring criterion (like the variance or correlation coefficient), or some model evaluation result is used. Feature construction is the most demanding of human expertise, as new features have to be more representative than initial ones, which usually requires them to be based on some domain knowledge. Common data transformations include standardization, normalization and statistical operations for numeric features, yet the possibility of exploring the whole operator space is doubtful. Feature extraction reduces dimensionality using well-known methods like PCA [10] or LDA [11] or with more advanced ones including ICA [17] or unsupervised autoencoder-tree-based feature extraction.

2.1.3. *AutoML.* Various AutoML methods and frameworks are being developed, and no existing solution can fully replace humans at each step of

the ML workflow. A group of frameworks can be distinguished, which only support fixed pipelines, choosing the best solution from the set of candidates [13–15] or simply using the predefined single pipeline [16]. These fixed pipelines focus mostly on model generation, while only a few implement feature engineering techniques. Other solutions [18, 19] feature composite pipelines allowing for more variability, which is even more important in feature engineering tasks. Various methods are being used for pipeline generation. The simplest one is the Grid Search, which applies to fixed pipelines. Among others are evolutionary algorithm [20], sequential model-based optimisation for pipeline configuration [13] etc.

AutoML solutions also differ in the data types they can process, as many [13, 19, 22] are only able to process tabular datasets. Other frameworks [15, 23] use textual information to generate pipelines, and others [14, 24] can process images as features. Some solutions are designed for specific data types, such as time series [25]. Using LLMs, both tabular and textual information can be encoded naturally, so further work is focused on such data.

2.1.4. *Feature Engineering in AutoML.* Automated feature engineering is implemented as a part many general AutoML solutions [20, 50, 51] while others are specifically designed for automated feature engineering [21, 33–36].

One of the main complexities of feature engineering is that many possible ways exist to process the data to obtain new features. While many features can be acquired from a predefined set of operators implemented by different frameworks, many possibly useful features are specific to the domain and require meta-knowledge to be extracted. Even if the desired feature exists among many predefined ones, it may take a long time for an optimisation algorithm to converge and capture that feature. At the same time, it can be easily acquired with domain knowledge being incorporated into the system.

Most general AutoFE solutions follow the same strategy of sequential application of three feature engineering steps: Feature Extraction, Feature Synthesis, and Feature Selection, thus expanding the initial dataset with new features and then pruning the least meaningful ones.

Some solutions only address feature extraction, among which are the usual PCA [10], ICA [37], LDA [11], LLE [38]. A more advanced approach implies using a set of predefined analytical functions to process the data. Among the ways to choose the best set of features are solutions operating

with meta-features of any kind. METABU [39] extracts new useful features as linear combinations of a large number of predefined meta-features. Authors of [41] use a dynamic dataset clustering algorithm based on the Markov process for a set of benchmark models to perform meta-feature extraction.

Feature Synthesis employs transformation operations, feature aggregation, and mixing to produce new meaningful features based usually on the statistical properties of the data. Different solutions for tabular data include [42, 44, 53]. The method presented in [46, 53] uses all operations simultaneously to produce a large set of new features and then uses classification models to choose the most impactful features. This method requires more memory and may produce too many features to process [28].

On the contrary, other approaches [44, 45] increase the search space by an incremental feature addition from smaller search subspaces. Such approaches increase the computational complexity of feature engineering, which may also be inappropriate for certain tasks, including NAS. In addition to existing methods, it is possible to estimate the operator effectiveness using a neural network before applying any. It is implemented in various ways in [45–47].

Feature selection minimises the dataset size to achieve the following goals: learning stability, as meaningless features may harm overall performance, model size and efficiency due to fewer parameters, and data explanation, as feature importance may provide additional insight for the data. Defining a set of features to drop is a combinatorial problem that can be addressed in various ways. Some solutions evaluate a score for each feature that is directly associated with the importance of the feature, and then drop the least meaningful features [45, 48]. Other methods, such as AutoLoss [49] and AutoDropout [52], implement feature dropout strategies to get the most relevant feature set, and thus optimum dropout patterns and hyperparameters are learned.

2.2. Large language models. Large language models (LLMs) are text foundation models, trained to complete the next text token on large amounts of unlabelled text data, and tuned further for specific tasks such as answering questions, code generation, or instruction following.

2.2.1. *LLMs for Black-Box optimisation.* LLMs can be used for black-box optimisation tasks. A large amount of meta-knowledge allows for better

convergence at the first steps; however, convergence slows down when suboptimal solutions are reached, which is common for most optimisation methods. Exploration/exploitation balance can be controlled by the temperature settings of an LLM with high temperature leading to more random and explorative steps and vice versa. Pattern recognition ability and extensive domain knowledge make LLMs advantageous for data wrangling tasks such as blank filling and error search. Recently, Neural Architecture Search (NAS), another important branch of AutoML, has received more attention, as LLMs [26, 27] have proven to be able to further push the main challenge of this field, reducing time complexity. However, it is notably unclear whether the promising results in NAS are coming from the LLMs' abilities or data leaks, as many high-quality solutions for benchmark datasets are available on the Web and thus could be used for LLM training (this issue can be partially solved by using new datasets with publishing date succeeding the date of LLM dataset gathering date, which is usually known for the most popular LLMs).

It has been shown that large language models are capable of solving various pattern recognition tasks. The authors of [55] use LLMs as general sequence modellers to solve spatial pattern tasks, complete sequences, and improve return-conditioned policies such as CartPole stability optimisation by discovering the oscillations behaviour. Zero-shot capabilities make it possible to use LLMs as general black-box optimisers with natural language optimisation tasks being the main application as it allows one to directly optimise the text as a feature without encoding it into numeric features and vice versa. Both traditional numeric optimisation and natural language optimisation are shown in [30]. The authors also propose using optimisation by PROmpting (OPRO) to optimise LLM prompts maximizing LLM's performance.

2.2.2. *LLMs for Feature Engineering.* Feature generation with LLM has been addressed by the authors of the CAAFEE framework [28], who attempted to automatically generate and execute the code for the new features using LLM. The results show that it is possible to achieve the goal, but only when the best available model (GPT-4 [29]) is used.

However, the authors did not perform iterative feature generation and used LLMs as zero-shot predictors. It can be assumed that the results can be improved by also incorporating LLM optimisation abilities [30] into the pipeline, which makes it possible to achieve the same result with

less powerful LLMs such as GPT-3.5 or open source LLAMA 2 [31] or Mixtral [32], both of which are approaching the quality of GPT-3.5.

§3. Proposed approach

Large language models can be utilized at the feature engineering step both as optimisers and as domain knowledge experts. The latter was implemented in [28] as a code generation for feature generation. This method allows one to possibly obtain the most suitable and domain-specific features. Applying an iterative optimisation process to such an approach is complicated by many code errors emerging on one side and nonchanging proposals on the other side. Prompt optimisation was implemented in [30] by making a prompt containing both the task description and some of the best results achieved during optimisation so far. Using whole prompts as individuals is costly concerning token number and computation time. One way to get fewer errors, use fewer tokens in prompts, and decrease computation efficiency is to encode the feature generation pipeline in a short natural language description.

The following optimisation approach is proposed, with each element described in more detail in the following sections. Feature optimisation instance consists of a dataset, an LLM interface, and an LLM template. LLM template contains the information on the dataset, instructions, and possible options for the feature generation pipeline. If the initial advise option is enabled, the first LLM query updates the template with the data insight to improve future proposals. On the initial iteration, a default feature generation pipeline is used to train the ML model, which type is defined in the configuration, and evaluate metrics to add to the template. The next iterations start from an LLM request for a pipeline, pipeline evaluation and a template update. The high-level scheme of the proposed approach is shown in Fig.1.

3.1. Feature Generation Pipeline Evaluation. Feature engineering pipeline can be presented as a sequence of atomic data operations such as scaling, imputation, etc. Having a set of such atomic operations, one can define a pipeline by assigning input feature names for each operation. This format allows one to insert the pipelines into LLM prompts for further optimisation and also parse and apply the LLM response. The typical operation sequence looks as follows: *FillNaMean(Feature1)* $\rightarrow$ *Std(Feature2,Feature3)* $\rightarrow$ *PCA()*

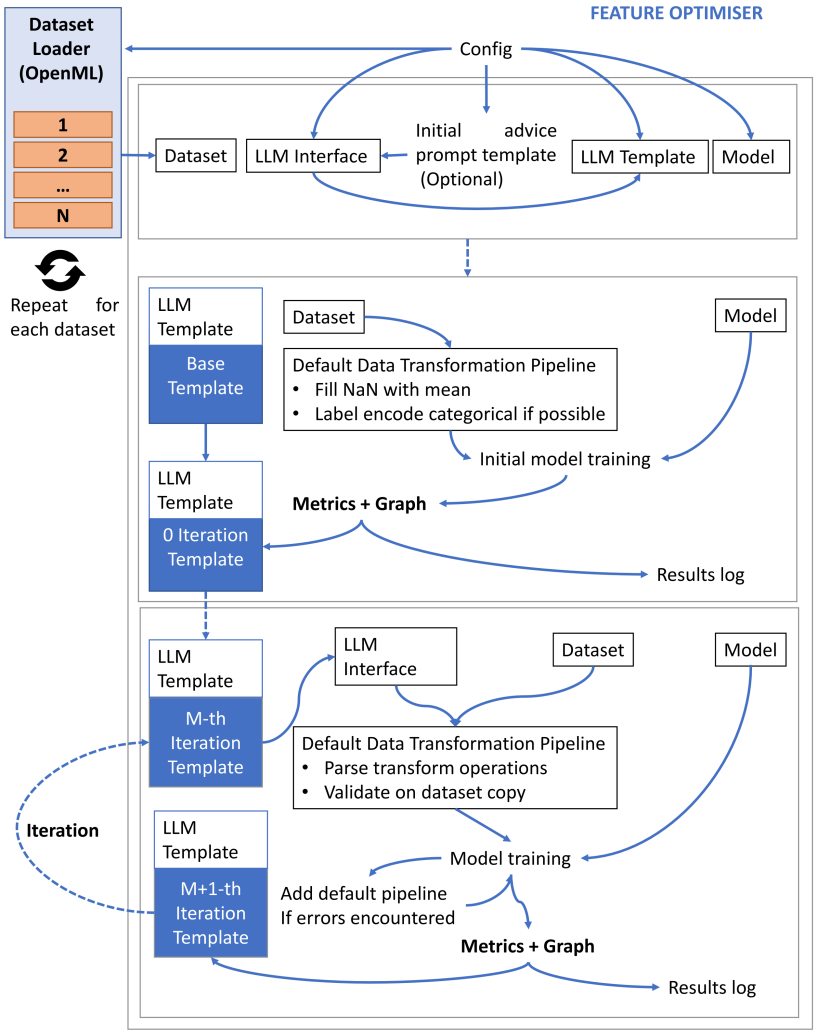

Figure 1. General scheme for the proposed approach. The initialization step, the first optimisation iteration without LLM proposal, and the main optimisation cycle are shown in blocks.

Figure 2. Data operations pipeline example (features taken from the Titanic dataset).

The presented pipeline, if parsed and applied, imputes absent features for Feature1 by mean values, performs standard scaling to both Feature2 and Feature3 and then applies PCA transformation to all columns. Other operation splitters may be used instead of "→" and it was noticed that they may affect performance, e.g. line breaks result in more errors, especially because of previous operations in sequence. The attention mechanism is likely to lose the connection between the subsequent operations, which is also confirmed by the fact that pipelines split by line breaks tend to be repeated without change on each optimisation iteration. All proposed pipelines are stored both as text and as a graph during the optimisation process. The graph example is shown in Fig. 2.

The following set of atomic data operations has been chosen for pipeline generation. For any operation, it is possible to use the result of previous operations as an input.

3.2. Feature Generation Pipeline Optimisation. The LLM prompt structure is crucial to get the feature generation pipeline appropriate to the domain and meta-features of the dataset. Along with the optimisation instructions, it can also contain the dataset description with some domain knowledge, meta-features and possibly some clues on the operations which might be a good solution. OpenML Dataset Sharing Platform [56] provides a wide variety of datasets with descriptions and meta-features. Python API was used to download the data and form the LLM prompt from the dataset description.

To make evaluating different prompts more efficient, an LLM template was formed from named paragraphs, each taken from the config, directly from the dataset description, or from the previous model training cycle. The following paragraph order was chosen for further prompt optimisation:

(1) Task description (from config)
(2) Pipeline format description (from config)
(3) List of available operations (from config)

Table 1. Atomic data operations used for the feature search algorithm.

Operation	Description
Add	Add any number of input columns to form a new column
Sub	Subtract two input columns to form a new column
Mul	Multiply any number of input columns to form a new column
Div	Divide two column values to form a new column
Pca	Create columns pca_0, pca_1 ... from PCA on input columns
FillnaMean	Fill missing values with mean inplace
FillnaMedian	Fill missing values with median inplace
Std	Inplace Standard scaling of input columns
Minmax	Inplace MinMax scaling of input columns
Drop	Drop input columns in place
Binning	Binning of numerical features. In-place operation
Label	Label encoding of categorical features. In-place operation
OneHot	One hot encoding of categorical features

(4) Dataset description (from dataset source)
(5) Previous evaluations (from model training)
(6) Instruction (from config)

Other structure versions and their performance will be discussed in Section 4. This prompt setup allows for paragraph swapping, insertion and removal using the prompt structure on each iteration. Furthermore, it can be used for automatic prompt generation using OPRO [30] for a specific paragraph, as the prompt cannot be optimised as a whole. A sample LLM prompt is shown in Appendix A.

All completions are being created with the gpt-3.5-turbo model, which, according to [28], may improve the overall result of an AutoML solution by providing new features to the dataset. Each LLM response is parsed into atomic data operations from the initial list. As it may contain errors, the data operation pipeline is validated on the copy of the dataset. If any operation results in an error or takes too much time or memory to perform (e.g. one-hot encoding on a numeric feature), it is dropped from the pipeline.

For the first iteration, the default pipeline is built by filling all the absent values in numerical features by the mean value of the column, encoding categorical features with less than 10 unique values by label, and dropping

the rest. Train and test data splits are created for each dataset, and each pipeline is first applied to the train data, so scaling, PCA and other operations do not result in data leaks between train and test datasets. Then the model can be trained, the type of which is defined in the problem configuration. CatBoost [59], SVM Classifier [61], and Random Forest Classifier [60] classification models are available for use; All the experiments shown use Random Forest Classifier. Accuracy metrics and pipeline graphs are stored in the evaluation log and written to the LLM prompt. The new prompt is then used to obtain the pipeline proposal from the LLM, which is then parsed and used to initialise the new data operations pipeline. Another model training results are added to the log and prompt after evaluation, thus forming an optimisation cycle. The optimisation scheme is shown in Figure 1.

3.3. Prompt optimisation. Initial prompt structure may not be optimal for the task, and a manual prompt search has been performed. Different prompt structures can provide an estimate of the impact of dataset description and meta-features on the proposal quality. As the LLM serves as both a domain-knowledge expert and a black-box optimiser, these roles may be unbalanced. While the optimisation capability was already demonstrated in [30], the importance of domain knowledge is uncertain. It is significant for the LLM model used, as the authors of [28] have shown that only the GPT-4 model can use the information from the dataset to provide meaningful features. In contrast, other models do not always perform with the same success.

The following prompt versions were tested:

(1) Default structure
(2) Prompt without dataset description
(3) Prompt with direct instructions on the data operations
(4) Prompt with advice on the data qualities
(5) Prompt with added meta-features

For the final prompt, all the individual paragraphs which are predefined, were optimised manually. The Titanic dataset[1] was used for prompt optimisation as it contains numeric, categorical and natural language data, requires data imputation and scaling. There are also many examples of feature engineering for this dataset, which makes it possible to compare LLM proposals with a variety of human-made ones. Titanic is a binary

[1]Available at `https://api.openml.org/d/40704`

Table 2. Manual prompt search results. The random forest model scores for the Titanic dataset are shown. A total of 5 pipelines were obtained for each prompt design, except for fixed pipelines.

Prompt/pipeline	**Avg score**	**Max score**
Default pipeline	–	0.72
Manual pipeline	–	0.82
Manual pipeline with data leak	–	0.97
Default prompt	0.77	0.79
Prompt without dataset description	0.77	0.8
Prompt with direct instructions	0.97	0.98
Prompt with advises	0.8	0.8
Prompt with meta-features	0.81	0.81
Prompt without initial default evaluation	0.76	0.78

classification dataset, it is also notable for containing the "boat" feature, which is present usually for the samples with the same target value and is absent for the rest. This allows us to get a higher score compared to the default pipeline. All proposals from LLM have been compared with the manually built pipeline with an acceptable score as a baseline. Each prompt has been used for pipeline construction 5 times, and the results are presented in Table 2.

The data leak capture results suggest that a full dataset description may indeed be used for the chosen optimisation method to improve pipeline quality. It can also be confirmed by the poor result of the dataset without any dataset description. Meta-features, on the other hand, do not make any significant difference, while taking a lot of token space in the prompt, making it more costly time- and resource-wise. No results are shown for prompts with other paragraphs deleted. If any of the task description, operation set, or instruction paragraphs are absent, the LLM output is usually poorly formatted and cannot be parsed, as it includes some additional advice, code for feature generation, etc. For that reason, the default prompt structure with a modified individual paragraph is used for optimisation. To obtain better results, the prompt optimisation OPRO [30] has been proposed, and it has been shown that it outperforms the human-mad prompts in most tasks. To improve the efficiency of optimisation, one of

the possible next steps might be to prioritise the optimisation of the fixed prompt paragraphs. Then, the usual grid search method may be used [57].

3.4. Random search baseline. Table 2 shows that even a single request to LLM results in a pipeline close to the one built manually. However, it is necessary to confirm that further optimisation is efficient enough. The baseline proposed is the random search method. On each iteration, the LLM response is mocked with a random pipeline string; The following pipeline parameters are chosen at random: total number of nodes, type of each operation, number of columns affected by each operation and column names. While such a method results in many processing errors, an acceptable length of pipeline can be achieved nonetheless by changing the search hyperparameters. A timeout of 2 min has been added to the data transformation and model training steps to avoid large datasets which may occur if the one-hot encoding is being applied to noncategorical features. Despite the error rate, random search is still much more time-efficient than any other method relying on the LLM requests. For that reason, any proposed method should be at least as effective as a random pipeline search concerning time cost.

3.5. Population-based optimisation. The initial approach implies passing the prompt to the LLM on each iteration, getting the pipeline proposal, and adding it to the next prompt. The prompts share most of the information, being only different in the previous evaluations section. Such an approach has several issues.

- Computational inefficiency; LLM has to process the data anew on each iteration.
- Data transformation and model training take less than a minute usually, thus making token quota exhaust rapidly, due to the large number of requests.
- API limit on the number of requests in a minute is easily broken. The optimisation process has to be paused after each iteration to not exceed the limit. This further decreases the time efficiency.
- Self-loops appear sometimes when the pipeline is proposed twice on the subsequent iterations. It is then repeated on all the next iterations with no respect to the score. Even the specific prompt instructions on the self-repetition do not help to change the trend.
- On the other hand, some pipelines naturally contain errors and either do not improve the result, or even do not allow one to evaluate

the score. Preventing such cases would likely lead to the previous problem being more frequent.

We propose the following solution to handle the above issues: the prompt template is changed, so the LLM proposes a population of 10 various pipeline samples instead of a single one. All pipelines are being processed; the model training result is evaluated. The pipeline with the best score is then added to the prompt as a result of a whole iteration, and the next population is generated. This approach requires more time to transform the data and train the models and less time to get the response from the LLM. With each population containing 10 samples, the balance between LLM and algorithm time cost is maintained, thus reducing overall computation time. It was also observed that the no-repetition instruction is more strictly followed in response, producing unique results at the cost of a larger portion of unsuitable ones.

3.6. Multi-step LLM prompting. It has been shown in [58] that many tasks can be handled more efficiently by LLM if taken step by step. Feature generation usually involves getting information on the data distribution (e.g. mean, skewness, number of samples), share of missing values, etc. Building a data transformation pipeline can be challenging without this knowledge, even when handled by a human. A possible way for the LLM to improve the quality of the proposed pipelines is to have some data insight inside the prompt. It is known that direct instructions inside a prompt may drastically improve the results. However, these results were previously obtained manually. We propose a multi-step approach that operates as follows: before the optimisation process, a single request to the LLM is made, only once for each dataset. The prompt is structured as follows:

(1) experiment description;
(2) dataset description;
(3) dataset meta-features;
(4) instruction.

Prompt optimisation has been performed using the same method as pipeline generation. A sample prompt for the Titanic dataset is given in Appendix B. LLM response is expected to provide the data insights and possible advice on the data transforms. However, the whole set of operations was not included to reduce the prompt token number, making it less likely to get such proposals. The general optimisation process is performed after the dataset advice has been generated and added to the prompt.

Table 3. OpenML datasets used for performance analysis, characteristics and reasoning for pick.

Dataset	Description
Titanic	Multiple data types, many open examples on feature engineering
credit-g	Simple numeric and nominal features, financial domain
blood-transfusion-service-center	Small number of numeric features, healthcare domain
steel-plates-fault	Large number of numeric features, high default score of 0.98, engineering domain
monks-problems-2	Small set of nominal features, test dataset forML algorithms
tic-tac-toe	Dataset with simple analytical solution, nominal features only

§4. Results

A total of 6 OpenML datasets were used to evaluate each method proposed. Descriptions and specific characteristics are shown in Table 3. Datasets listed are also used for feature generation in [28]. Classification tasks were chosen for evaluation, but the method applies to other problems too. For each dataset, accuracy is evaluated. Different metrics can be selected for optimisation.

The baseline result is gained from the random search optimisation. Other optimisation strategies are also available. such as tabu search, simulated annealing, genetic algorithms etc. However, the search space is small enough for a random search to provide meaningful results. While other methods are expected to overperform the random search in both the convergence rate and final score, random search is considered an acceptable baseline for the proof of concept. For each dataset, 25 iterations were run, with 5 forming an irrelevant pipeline on average. For the rest of the pipelines, the model training score is recorded together with the best result achieved. For each optimisation step, the initial iteration score is used if the current pipeline cannot be evaluated, as it can be replaced with the default pipeline afterwards.

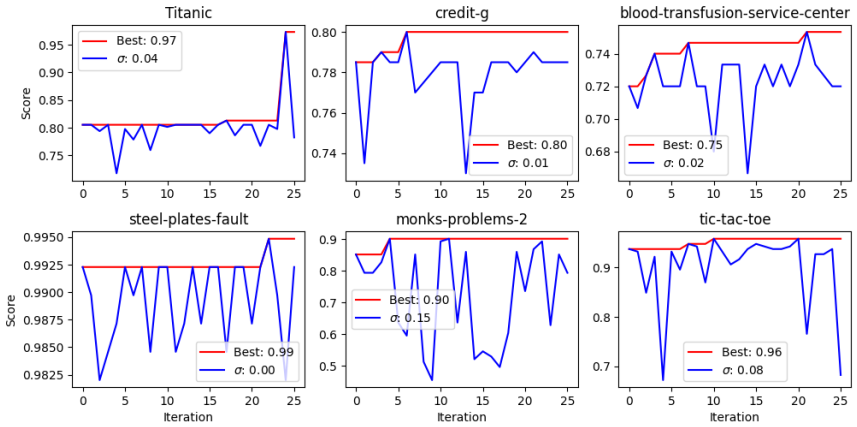

Figure 3. Optimisation results for the Random Search data transformation pipeline optimisation. For each dataset, the current result for the iteration is shown in blue and the top result among the previous iterations is shown in red. Best – the best score during optimisation, σ – score standard deviation.

The results are shown in Fig. 3. For all datasets, we find some improvement during the optimisation process. While some datasets, such as tic-tac-toe, do not benefit much from random search, the score may substantially increase for others, such as Titanic. The rise from 0.8 to 0.97 score means that the mentioned 'boat' feature transformation has been proposed. This transformation is reached at the 10th iteration on average, meaning that other methods featuring LLM pipelines have to achieve at least the same result during the smaller number of iterations. It is worth mentioning that the standard deviation of the score is relatively high for random search due to the random origin of the pipeline, and also there are no gradual emerges as can be expected.

4.1. Single-proposal approach. The default optimisation scenario is the single proposal method with the standard prompt defined in Section 3.3. To match the time spent on the optimisation, a total of 10 optimisation iterations is used. The optimisation results are shown in Fig. 4.

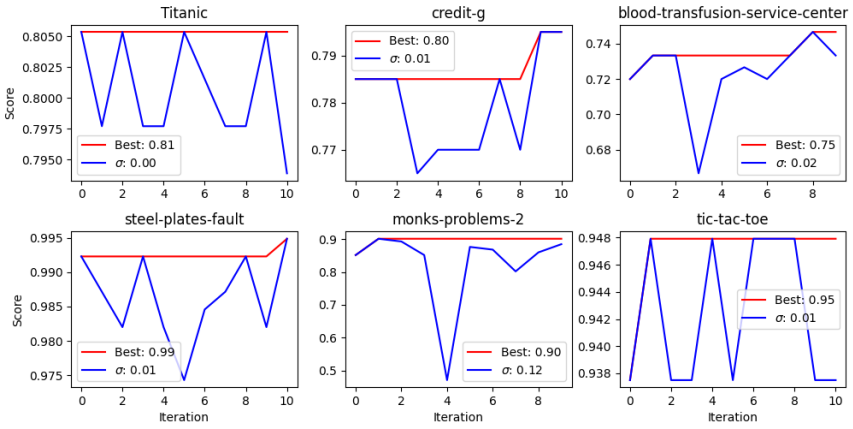

Figure 4. Optimisation results for the single-proposal data transformation pipeline optimisation using an LLM response. For each dataset, the current result for the iteration is shown in blue and the top result among the previous iterations is shown in red. Best – the best score during optimisation, σ – score standard deviation.

The standard method does not provide any notable increase in performance: for all datasets it does not outperform the baseline. The standard deviation has decreased, and the score does not change over iterations, meaning that the proposed pipelines change insignificantly. Sample pipelines for the Titanic dataset can be found in Appendix C. Although the initial structure varies and no errors occur during data processing, several operations are omitted as they are irrelevant to the dataset. Most meaningful operations are dropped for all datasets, leaving only the default pipeline for model training and not improving the score. Some experiments also end up with the same operations proposed on each iteration.

4.2. Population-based approach. The population approach allows the optimiser not only to evaluate more pipelines using the same time but also the LLM response is more diverse if multiple pipelines are generated at the same time. As the most time-consuming operation is the LLM call, mostly due to API limitations, the same number of iterations are used during optimisation. The results are presented in Fig. 5.

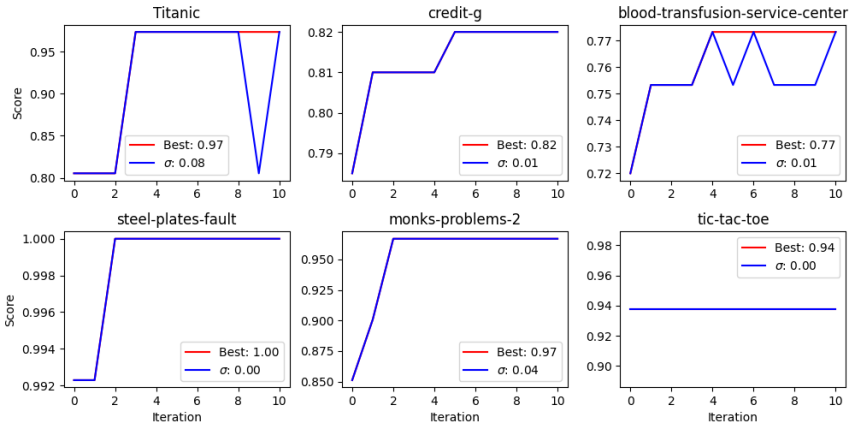

Figure 5. Optimisation results for the population-proposal data transformation pipeline optimisation using an LLM response. For each dataset, the current result for the iteration is shown in blue and the top result among the previous iterations is shown in red. Best – the best score during optimisation, σ – score standard deviation.

The highest performing result is usually achieved on the first iterations. Almost no deviation is present in the scores, as only the best-performing pipeline is recorded. It is worth mentioning also that the best and current result graphs are almost the same for each dataset, meaning that the previous top evaluation is also present at least once in the population. At the same time, even on the last optimisation iterations, the populations provided are diverse enough to support further exploration.

4.3. Multi-step optimisation. The data operations advice is generated for each dataset as described in Section 3.6 to further improve the optimisation efficiency. The results for single-proposal optimisation are shown in Fig. 6 and Table 4.

There is no substantial improvement in single-proposal cases compared to previous methods. To further analyze the method performance, for all methods the top and average scores have been evaluated. The results are shown in Table 5.

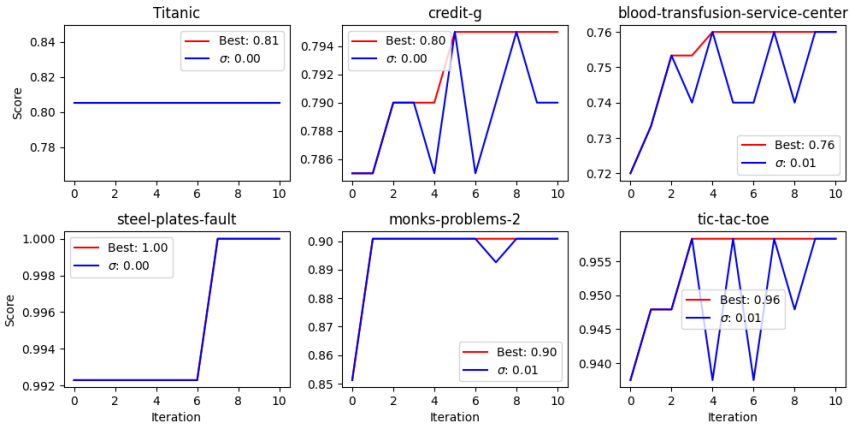

Figure 6. Optimisation results for the multi-step single-proposal data transformation pipeline optimisation using an LLM response. For each dataset, the current result for the iteration is shown in blue and the top result among the previous iterations is shown in red. Best – the best score during optimisation, σ – score standard deviation.

Table 4. Results for single-proposal evaluation.

Dataset	**Random Search**		**Single**		**Population**	
	Avg	Max	Avg	Max	Avg	Max
Titanic	0.798	**0.973**	0.8	0.805	**0.912**	**0.973**
credit-g	0.779	0.8	0.779	0.795	**0.813**	**0.82**
blood-transfusion-service-center	0.723	0.753	0.723	0.746	**0.755**	**0.773**
steel-plates-fault	0.987	0.994	0.986	0.994	**0.998**	**1.0**
monks-problems-2	0.724	0.9	0.825	0.9	**0.95**	**0.966**
tic-tac-toe	0.902	0.958	0.942	0.947	0.937	0.937

The population-based approach demonstrates the top performance among all others for all datasets except *the tic-tac-toe*. A notable increase in the average and maximum score is obtained for the *Titanic*, *steel-plates-fault* and *monks-problems-2* datasets.

Table 5. Optimisation results for different methods for each dataset.

	Single multi-step		**Population multi-step**	
	Avg	Max	Avg	Max
Titanic	0.797	0.805	0.805	0.805
credit-g	0.78	0.79	0.789	0.795
blood-transfusion-service-center	0.734	0.753	0.746	0.76
steel-plates-fault	0.97	0.992	0.995	**1.0**
monks-problems-2	0.794	0.909	0.895	0.9
tic-tac-toe	**0.954**	**0.994**	0.949	0.958

The population-based method does not provide any notable impact, as the initial dataset advice does not change and limits the explorative capabilities of the model. For single proposal cases, the effect of multi-step optimisation may result in better initial pipelines, as the total number of pipelines evaluated is fewer than the number evaluated in the population method. If being compared, the multi-step approach for single-proposal mode has at least the same performance in most cases and outperforms the default method in 3 cases. As the prompt may be optimised further to provide better data insight, multi-step optimisation may be more effective for single-proposal cases.

§5. Conclusions and Discussions

The impact of different prompt parts has been estimated by manual pipeline evaluation for each completion. The results lead to the conclusion that black-box optimisation is the main source of improvement, while the data description and even the direct instructions are less effective or may even hinder the optimisation process.

The single-proposal optimisation does not notably outperform the random search baseline due to poor pipeline variance and many errors in the pipelines proposed. Simultaneously, the population of pipelines produced at the same time results in more diverse pipelines proposed, assuring exploration and only the best pipeline is recorded and passed to the next iteration, making the next pipelines less diverse, achieving exploitation on later steps.

Multi-step optimisation uses the same dataset insight and pipeline advice on each iteration, allowing the single-proposal method to find more optimal solutions. Population-based method, at the same time, struggles with exploration as the initial advice limits the pipeline diversity. We conclude that in the current stage, it is more efficient to use the multi-step optimisation if the pipeline evaluation is time-consuming, and a good initial assumption is required. Otherwise, population generation is generally more promising. It may also be assumed that some other dataset advice format may result in acceptable results for the population-based method. Using OPRO for both parts of the optimisation is, for now, viewed as the most simple way to increase the feature engineering impact on the final score.

Code and Data Availability. The code and data for all implemented algorithms and experiments are available at the repository `https://github.com/AngrySigma/LAAFE`

Appendix A. Default prompt example

```
You should perform a feature engineering for the provided dataset to
improve the performance of the model.
The output is an operation pipeline written as in PIPELINE EXAMPLE:
operation1(df_column)->operation2(df_column_1,
df_column_2)->operationN()
Empty brackets mean that operation is applied to all columns of the
dataset. Please, don't use spaces between operations and inputs.
Name operations exactly as they are listed in initial message. Do
not add any other information to the output.
Add: Add two input columns together to a new column "add_{number of
previous add operations + 1}"
Mul: Multiply two input columns together to a new column
"mul_{number of previous mul operations + 1}"
FillnaMean: Fill missing values with mean inplace
Drop: Drop input columns inplace
LabelEncoding: Label encoding of categorical features. Inplace
operation
Dataset description will be provided further.
START DATASET DESCRIPTION
Dataset name: Titanic
The original Titanic dataset, describing the survival status of
individual passengers on the Titanic. The titanic data does not
```

```
contain information from the crew, but it does contain actual ages
of half of the passengers. The principal source for data about
Titanic passengers is the Encyclopedia Titanica. The datasets used
here were begun by a variety of researchers.
One of the original sources is Eaton and Haas (1994) Titanic:
Triumph and Tragedy, Patrick Stephens Ltd, which includes a
passenger list created by many researchers and edited by Michael
A. Findlay.
 The variables on our extracted dataset are pclass, survived,
name, age, embarked, home.dest, room, ticket, boat, and sex.
pclass refers to passenger class (1st, 2nd, 3rd), and is a proxy
for socio-economic class. Age is in years, and some infants had
fractional values. The titanic2 data frame has no missing data and
includes records for the crew, but age is dichotomized at adult
vs. child. These data were obtained from Robert Dawson, Saint
Mary's University, E-mail. The variables are pclass, age, sex,
survived. These data frames are useful for demonstrating many of
the functions in Hmisc as well as demonstrating binary logistic
regression analysis using the Design library. For more details and
references, see Simonoff, Jeffrey S (1997): The "unusual episode"
and a second statistics course. J Statistics Education, Vol. 5 No.
1.
Data columns: {0: [0 - pclass (numeric)], 1: [1 - survived
(nominal)], 2: [2 - name (string)], 3: [3 - sex (nominal)], 4: [4 -
age (numeric)], 5: [5 - sibsp (numeric)], 6: [6 - parch (numeric)],
7: [7 - ticket (string)], 8: [8 - fare (numeric)], 9: [9 - cabin
(string)], 10: [10 - embarked (nominal)], 11: [11 - boat (string)],
12: [12 - body (numeric)], 13: [13 - home.dest (string)]}
Data example:
pclass ... home.dest
0 1 ... St Louis, MO
1 1 ... Montreal, PQ / Chesterville, ON
2 1 ... Montreal, PQ / Chesterville, ON
3 1 ... Montreal, PQ / Chesterville, ON
4 1 ... Montreal, PQ / Chesterville, ON

[5 rows x 13 columns]
Target:
0 1
1 1
2 0
3 0
```

```
4 0
Name: survived, dtype: category
Categories (2, object): ['0' < '1']
END DATASET DESCRIPTION
Previous pipeline evaluations and corresponding metrics:
Initial evaluation: 'accuracy': 0.8053435114503816,
Pipeline: FillnaMean(pclass)->Drop(name)->LabelEncoding(sex)
->FillnaMean(age)->FillnaMean(sibsp)->FillnaMean(parch)
->Drop(ticket)
Iteration 1: 0.8053435114503816, Pipeline:
Add(parent.child,fare)->Mul(sibsp,parch)
Based on the information provided, please choose the operations you
want to use in your pipeline and write them in the output format.
Operation inputs have to match the columns of the dataset. Please,
do not propose the pipelines already met in history.
```

Appendix B. Dataset advise prompt

```
 You have a dataset with the following description. The task is
feature engineering for tabular datasets to improve the performance
of the machine learning model.
START DATASET DESCRIPTION
Dataset name: Titanic
The original Titanic dataset, describing the survival status of
individual passengers on the Titanic. The titanic data does not
contain information from the crew, but it does contain actual ages
of half of the passengers. The principal source for data about
Titanic passengers is the Encyclopedia Titanica. The datasets used
here were begun by a variety of researchers.
One of the original sources is Eaton & Haas (1994) Titanic: Triumph
and Tragedy, Patrick Stephens Ltd, which includes a passenger list
created by many researchers and edited by Michael A. Findlay.

The variables on our extracted dataset are pclass, survived,
name, age, embarked, home.dest, room, ticket, boat, and sex.
pclass refers to passenger class (1st, 2nd, 3rd), and is a proxy
for socio-economic class. Age is in years, and some infants had
fractional values. The titanic2 data frame has no missing data and
includes records for the crew, but age is dichotomized at adult
vs. child. These data were obtained from Robert Dawson, Saint
Mary's University, E-mail. The variables are pclass, age, sex,
survived. These data frames are useful for demonstrating many of
```

```
the functions in Hmisc as well as demonstrating binary logistic
regression analysis using the Design library.
For more details and references, see Simonoff, Jeffrey S (1997):
The "unusual episode" and a second statistics course. J Statistics
Education, Vol. 5 No. 1.
Data columns: 0: [0 - pclass (numeric)], 1: [1 - survived
(nominal)], 2: [2 - name (string)], 3: [3 - sex (nominal)], 4: [4 -
age (numeric)], 5: [5 - sibsp (numeric)], 6: [6 - parch (numeric)],
7: [7 - ticket (string)], 8: [8 - fare (numeric)], 9: [9 - cabin
(string)], 10: [10 - embarked (nominal)], 11: [11 - boat (string)],
12: [12 - body (numeric)], 13: [13 - home.dest (string)]
Data example:
pclass ... home.dest
0 1 ... St Louis, MO
1 1 ... Montreal, PQ / Chesterville, ON
2 1 ... Montreal, PQ / Chesterville, ON
3 1 ... Montreal, PQ / Chesterville, ON
4 1 ... Montreal, PQ / Chesterville, ON

[5 rows x 13 columns]
Target:
0 1
1 1
2 0
3 0
4 0
Name: survived, dtype: category
Categories (2, object): ['0' < '1']
END DATASET DESCRIPTION
Dataset qualities:
'AutoCorrelation': 0.6062691131498471,
'ClassEntropy': 0.959422170862815,
'Dimensionality': 0.0106951871657754,
'EquivalentNumberOfAtts': 8.338046296490324,
'MajorityClassPercentage': 61.80290297937356,
'MajorityClassSize': 809.0,
'MaxAttributeEntropy': 1.1534325740767195,
'MaxKurtosisOfNumericAtts': 27.027986349441676,
'MaxMeansOfNumericAtts': 160.8099173553719,
'MaxMutualInformation': 0.20550487272008,
'MaxNominalAttDistinctValues': 3.0,
'MaxSkewnessOfNumericAtts': 4.367709134122926,
```

'MaxStdDevOfNumericAtts': 97.6969219960031,
'MeanAttributeEntropy': 1.0463713372687837,
'MeanKurtosisOfNumericAtts': 11.031689239559626,
'MeanMeansOfNumericAtts': 37.86088226053372,
'MeanMutualInformation': 0.11506558452028,
'MeanNoiseToSignalRatio': 8.093695057746512,
'MeanNominalAttDistinctValues': 2.3333333333333335,
'MeanSkewnessOfNumericAtts': 1.9636285454084217,
'MeanStdDevOfNumericAtts': 27.769024103336104,
'MinAttributeEntropy': 0.9393101004608482,
'MinKurtosisOfNumericAtts': -1.3150788243435427,
'MinMeansOfNumericAtts': 0.38502673796791437,
'MinMutualInformation': 0.02462629632048,
'MinNominalAttDistinctValues': 2.0,
'MinSkewnessOfNumericAtts': -0.5986471102803975,
'MinStdDevOfNumericAtts': 0.837836018970125,
'MinorityClassPercentage': 38.19709702062643,
'MinorityClassSize': 500.0,
'NumberOfBinaryFeatures': 2.0,
'NumberOfClasses': 2.0,
'NumberOfFeatures': 14.0,
'NumberOfInstances': 1309.0,
'NumberOfInstancesWithMissingValues': 1309.0,
'NumberOfMissingValues': 3855.0,
'NumberOfNumericFeatures': 6.0,
'NumberOfSymbolicFeatures': 3.0,
'PercentageOfBinaryFeatures': 14.285714285714285,
'PercentageOfInstancesWithMissingValues': 100.0,
'PercentageOfMissingValues': 21.035687002073555,
'PercentageOfNumericFeatures': 42.857142857142854,
'PercentageOfSymbolicFeatures': 21.428571428571427,
'Quartile1AttributeEntropy': 0.9393101004608482,
'Quartile1KurtosisOfNumericAtts': -1.269309018679956,
'Quartile1MeansOfNumericAtts': 0.47039724980901376,
'Quartile1MutualInformation': 0.02462629632048,
'Quartile1SkewnessOfNumericAtts': -0.0808576659180329,
'Quartile1StdDevOfNumericAtts': 0.8586292112546657,
'Quartile2AttributeEntropy': 1.0463713372687837,
'Quartile2KurtosisOfNumericAtts': 10.095100729172625,
'Quartile2MeansOfNumericAtts': 16.08800805071377,
'Quartile2MutualInformation': 0.11506558452028,
'Quartile2SkewnessOfNumericAtts': 2.0383750450744884,

```
'Quartile2StdDevOfNumericAtts': 7.727579045259853,
'Quartile3AttributeEntropy': 1.1534325740767195,
'Quartile3KurtosisOfNumericAtts': 22.91280574039027,
'Quartile3MeansOfNumericAtts': 65.17408879985216,
'Quartile3MutualInformation': 0.20550487272008,
'Quartile3SkewnessOfNumericAtts': 3.9750925407229336,
'Quartile3StdDevOfNumericAtts': 63.24323167838142,
'StdvNominalAttDistinctValues': 0.5773502691896258
Now you should perform the first step. Please, provide the data
insightsand possible meta-features to get a better data processing
pipeline. You can also directly advise on the operations. Please,
only give brief textual advice, avoid using code or anything else.
You should fit in the limit of 500 characters.
```

Appendix C. Sample data transformation proposals for Titanic dataset

```
 1) Add(age,sibsp)->Add(parch,fare)->Drop(room)
->LabelEncoding (embarked)->LabelEncoding(sex)
2) Add(parch,fare)->Drop(room)->LabelEncoding(embarked)
->LabelEncoding(sex)->FillnaMean(body)
3) LabelEncoding(sex)->LabelEncoding(embarked)->Add(age,sibsp)
->Add(parch,fare)
4) LabelEncoding(embarked)->OneHotEncoding(sex)
5) Add(parch,fare)->LabelEncoding(embarked)->LabelEncoding(sex)
->FillnaMean(body)
6) Add(sibsp,parch)->Add(parch,fare)->LabelEncoding(embarked)
->LabelEncoding(sex)->FillnaMean(body)
7) Add(age,sibsp)->Add(parch,fare)->LabelEncoding(embarked)
->LabelEncoding(sex)->FillnaMean(body)->Minmax()
8) OneHotEncoding(sex)->FillnaMean(body)->Minmax()
9) Pca(pclass,age,sibsp)->Pca(pclass,age,parch,fare)
10) Drop(body)->Pca(pclass,age,sibsp)
```

Pipelines after omitting the irrelevant operations from:

```
 1) Add(age,sibsp)->Add(parch,fare)->LabelEncoding(embarked)
->LabelEncoding(sex)
2) Add(parch,fare)->LabelEncoding(embarked)->LabelEncoding(sex)
->FillnaMean(body)
3) LabelEncoding(sex)->LabelEncoding(embarked)->Add(age,sibsp)
->Add(parch,fare)
4) LabelEncoding(embarked)->OneHotEncoding(sex)
```

```
5) Add(parch,fare)->LabelEncoding(embarked)->LabelEncoding(sex)
->FillnaMean(body)
6) Add(sibsp,parch)->Add(parch,fare)->LabelEncoding(embarked)
->LabelEncoding(sex)->FillnaMean(body)
7) Add(age,sibsp)->Add(parch,fare)->LabelEncoding(embarked)
->LabelEncoding(sex)->FillnaMean(body)
8) OneHotEncoding(sex)->FillnaMean(body)
9) -
10) Drop(body)
```

References

1. A. Doke and M. Gaikwad, *Survey on Automated Machine Learning (AutoML) and Meta Learning*, in: 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT) (2021), pp. 1–5.
2. B. Collins, J. Deng, K. Li, and L. Fei-Fei, *Towards Scalable Dataset Construction: An Active Learning Approach*, in: Proc. Computer Vision–ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Part I (2008), pp. 86–98.
3. N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, *SMOTE: Synthetic Minority Over-Sampling Technique*. — J. Artif. Intell. Res. **16** (2002), 321–357.
4. S. Krishnan, M. Franklin, K. Goldberg, and E. Wu, *BoostClean: Automated Error Detection and Repair for Machine Learning*, ArXiv preprint arXiv:1711.01299 (2017).
5. S. Krishnan and E. Wu, *AlphaClean: Automatic Generation of Data Cleaning Pipelines*, ArXiv preprint arXiv:1904.11827 (2019).
6. S. Marcel and Y. Rodriguez, *Torchvision: The Machine-Vision Package of Torch*, in: Proceedings of the 18th ACM International Conference on Multimedia (2010), pp. 1485–1488.
7. A. Jung, *ImgAug Documentation*, readthedocs.io, Jun. 25, 2019 (2019).
8. A. Buslaev, V. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. Kalinin, *Albumentations: Fast and Flexible Image Augmentations*. — Inf. **11**, 125 (2020).
9. M. Munson, *A Study on the Importance of and Time Spent on Different Modeling Steps*. — ACM SIGKDD Explor. Newsl. **13** (2012), 65–71.
10. A. Maćkiewicz and W. Ratajczak, *Principal Components Analysis (PCA)*. — Comput. Geosci. **19** (1993), 303–342.
11. P. Xanthopoulos, P. Pardalos, and T. Trafalis, *Linear Discriminant Analysis*, in: Robust Data Mining (2013), pp. 27–33.
12. X. Chu, J. Morcos, I. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, *KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing*, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (2015), pp. 1247–1261.

13. M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, *Auto-sklearn 2.0: Hands-Free AutoML via Meta-Learning.* — J. Mach. Learn. Res. **23** (2022), 11936–11996.
14. N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, *AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data*, ArXiv preprint arXiv:2003.06505 (2020).
15. A. Vakhrushev, A. Ryzhkov, M. Savchenko, D. Simakov, R. Damdinov, and A. Tuzhilin, *LightAutoML: AutoML Solution for a Large Financial Services Ecosystem*, ArXiv preprint arXiv:2109.01528 (2021).
16. R. Hyndman and Y. Khandakar, *Automatic Time Series Forecasting: The Forecast Package for R.* — J. Stat. Softw. **27** (2008), 1–22.
17. H. Li, S. Yu, and J. Prìncipe, *Deep Deterministic Independent Component Analysis for Hyperspectral Unmixing*, in: ICASSP 2022—IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (2022), pp. 3878–3882.
18. I. Polonskaia, N. Nikitin, I. Revin, P. Vychuzhanin, and A. Kalyuzhnaya, *Multi-Objective Evolutionary Design of Composite Data-Driven Models*, in: 2021 IEEE Congress on Evolutionary Computation (CEC) (2021), pp. 926–933.
19. R. Olson and J. Moore, *TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning*, in: Workshop on Automatic Machine Learning (2016), pp. 66–74.
20. N. Nikitin, P. Vychuzhanin, M. Sarafanov, I. Polonskaia, I. Revin, I. Barabanova, G. Maximov, A. Kalyuzhnaya, and A. Boukhanovsky, *Automated Evolutionary Approach for the Design of Composite Machine Learning Pipelines.* — Future Gener. Comput. Syst. (2021).
21. H. Song, *AutoFE: Efficient and Robust Automated Feature Engineering*, Massachusetts Institute of Technology, 2018.
22. T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni, *ATM: A Distributed, Collaborative, Scalable System for Automated Machine Learning*, in: 2017 IEEE International Conference on Big Data, Boston, MA, USA (2017), pp. 151–162.
23. E. LeDell and S. Poirier, *H2O AutoML: Scalable Automatic Machine Learning*, in: 7th ICML Workshop on Automated Machine Learning (AutoML) (2020).
24. Microsoft, *Neural Network Intelligence*, 2021. Available at: `https://github.com/microsoft/nni`.
25. C. Wang, X. Chen, C. Wu, and H. Wang, *AutoTS: Automatic Time Series Forecasting Model Design Based on Two-Stage Pruning*, ArXiv preprint arXiv:2203.14169 (2022).
26. M. Nasir, S. Earle, J. Togelius, S. James, and C. Cleghorn, *LLMatic: Neural Architecture Search via Large Language Models and Quality-Diversity Optimization*, ArXiv preprint arXiv:2306.01102 (2023).
27. H. Dong, Y. Gao, H. Wang, H. Yang, and P. Zhang, *Heterogeneous Graph Neural Architecture Search with GPT-4*, ArXiv preprint arXiv:2312.08680 (2023).
28. N. Hollmann, S. Müller, and F. Hutter, *CAAFE: Combining Large Language Models with Tabular Predictors for Semi-Automated Data Science*, in: 1st Workshop on the Synergy of Scientific and Machine Learning Modeling @ ICML 2023 (2023).

29. Achiam, J. et al., *GPT-4 Technical Report*, ArXiv preprint arXiv:2303.08774 (2023).
30. C. Yang, X. Wang, Y. Lu, H. Liu, Q. Le, D. Zhou, and X. Chen, *Large Language Models as Optimizers*, ArXiv preprint arXiv:2309.03409 (2023).
31. H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, and Others, *Llama 2: Open Foundation and Fine-Tuned Chat Models*, ArXiv preprint arXiv:2307.09288 (2023).
32. A. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. Chaplot, D. Casas, E. Hanna, F. Bressand, and Others, *Mixtral of Experts*, ArXiv preprint arXiv:2401.04088 (2024).
33. G. Zhu, S. Jiang, X. Guo, C. Yuan, and Y. Huang, *Evolutionary Automated Feature Engineering*, in: Pacific Rim International Conference on Artificial Intelligence (2022), pp. 574–586.
34. B. Hilprecht, C. Hammacher, E. Reis, M. Abdelaal, and C. Binnig, *DiffML: End-to-End Differentiable ML Pipelines*, in: Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning (2023), pp. 1–7.
35. R. Bonidia, A. Santos, B. Almeida, P. Stadler, U. Rocha, D. Sanches, and A. Carvalho, *BioAutoML: Automated Feature Engineering and Meta-Learning to Predict Noncoding RNAs in Bacteria*. — Brief. Bioinform. **23**, bbac218 (2022).
36. G. Zhu, Z. Xu, C. Yuan, and Y. Huang, *DIFER: Differentiable Automated Feature Engineering*, in: International Conference on Automated Machine Learning (2022), pp. 1–17.
37. J. Stone, *Independent Component Analysis: A Tutorial Introduction*, MIT Press, 2004.
38. L. Saul and S. Roweis, *An Introduction to Locally Linear Embedding*, Unpublished. Available at: `http://www.cs.toronto.edu/~roweis/lle/publications.html`.
39. H. Rakotoarison, L. Milijaona, A. Rasoanaivo, M. Sebag, and M. Schoenauer, *Learning Meta-Features for AutoML*, in: ICLR 2022—International Conference on Learning Representations (Spotlight) (2022).
40. V. Lopes, A. Gaspar, L. Alexandre, and J. Cordeiro, *An AutoML-Based Approach to Multimodal Image Sentiment Analysis*, in: 2021 International Joint Conference on Neural Networks (IJCNN) (2021), pp. 1–9.
41. C. Xue, J. Yan, R. Yan, S. Chu, Y. Hu, and Y. Lin, *Transferable AutoML by Model Sharing over Grouped Datasets*, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2019), pp. 9002–9011.
42. S. Chang, C. Wang, and C. Wang, *Automated Feature Engineering for Fraud Prediction in Online Credit Loan Services*, in: 2022 13th Asian Control Conference (ASCC) (2022), pp. 738–743.
43. A. Fatima, F. Khan, A. Raza, and A. Kamran, *Automated Feature Synthesis from Relational Database for Data Science Related Problems*, in: 2018 International Conference on Frontiers of Information Technology (FIT) (2018), pp. 71–75.
44. U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy, *Cognito: Automated Feature Engineering for Supervised Learning*, in: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW) (2016), pp. 1304–1307.

45. G. Katz, E. Shin, and D. Song, *ExploreKit: Automatic Feature Generation and Selection*, in: 2016 IEEE 16th International Conference on Data Mining (ICDM) (2016), pp. 979–984.
46. U. Khurana, H. Samulowitz, and D. Turaga, *Feature Engineering for Predictive Modeling Using Reinforcement Learning*, in: Proceedings of the AAAI Conference on Artificial Intelligence **32** (2018).
47. F. Nargesian, H. Samulowitz, U. Khurana, E. Khalil, and D. Turaga, *Learning Feature Engineering for Classification*, in: IJCAI (2017), pp. 2529–2535.
48. G. Borboudakis and I. Tsamardinos, *Extending Greedy Feature Selection Algorithms to Multiple Solutions.* — Data Min. Knowl. Discov. **35** (2021), 1393–1434.
49. H. Li, T. Fu, J. Dai, H. Li, G. Huang, and X. Zhu, *AutoLoss-Zero: Searching Loss Functions from Scratch for Generic Tasks*, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2022), pp. 1009–1018.
50. C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, *Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms*, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2013), pp. 847–855.
51. H. Jin, Q. Song, and X. Hu, *Auto-Keras: An Efficient Neural Architecture Search System*, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019), pp. 1946–1956.
52. V. Dodballapur, R. Calisa, Y. Song, and W. Cai, *Automatic Dropout for Deep Neural Networks*, in: Neural Information Processing: 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23–27, 2020, Proceedings, Part III (2020), pp. 185–196.
53. F. Horn, R. Pack, and M. Rieger, *The AutoFeat Python Library for Automated Feature Engineering and Selection*, in: Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I (2020), pp. 111–120.
54. L. Dhamo, F. Carulli, P. Nickl, K. Wegner, V. Hodoroaba, C. Würth, S. Brovelli, and U. Resch-Genger, *Efficient Luminescent Solar Concentrators Based on Environmentally Friendly Cd-Free Ternary AIS/ZnS Quantum Dots.* — Adv. Opt. Mater. **9** (2021), 2100587.
55. S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. Arenas, K. Rao, D. Sadigh, and A. Zeng, *Large Language Models as General Pattern Machines* (2023).
56. M. Feurer, J. van Rijn, A. Kadra, P. Gijsbers, N. Mallik, S. Ravi, A. Mueller, J. Vanschoren, and F. Hutter, *OpenML-Python: An Extensible Python API for OpenML*, ArXiv preprint arXiv:1911.02490 (2020).
57. S. LaValle, M. Branicky, and S. Lindemann, *On the Relationship Between Classical Grid Search and Probabilistic Roadmaps.* — Int. J. Robot. Res. **23** (2004), 673–692.
58. J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models* (2023).
59. L. Prokhorenkova, G. Gusev, A. Vorobev, A. Dorogush, and A. Gulin, *CatBoost: Unbiased Boosting with Categorical Features* (2019).

60. T. Ho, *Random Decision Forests*, in: Proceedings of the 3rd International Conference on Document Analysis and Recognition (1995), pp. 278–282.
61. C. Cortes and V. Vapnik, *Support-Vector Networks*. — Mach. Learn. **20** (1995), 273–297.
62. X. He, K. Zhao, and X. Chu, *AutoML: A Survey of the State-of-the-Art*. — Knowl. Based Syst. **212** (2021), 106622.

ITMO University, St. Petersburg, Russia
E-mail: illariov1809@gmail.com

ITMO University, St. Petersburg, Russia
E-mail: nicl.nno@gmail.com

Поступило 15 ноября 2024 г.