Feature Engineering Pipeline Optimisation in AutoML Workflow using Large Language Models

Illarion Iov, Nikolay Nikitin

Outline

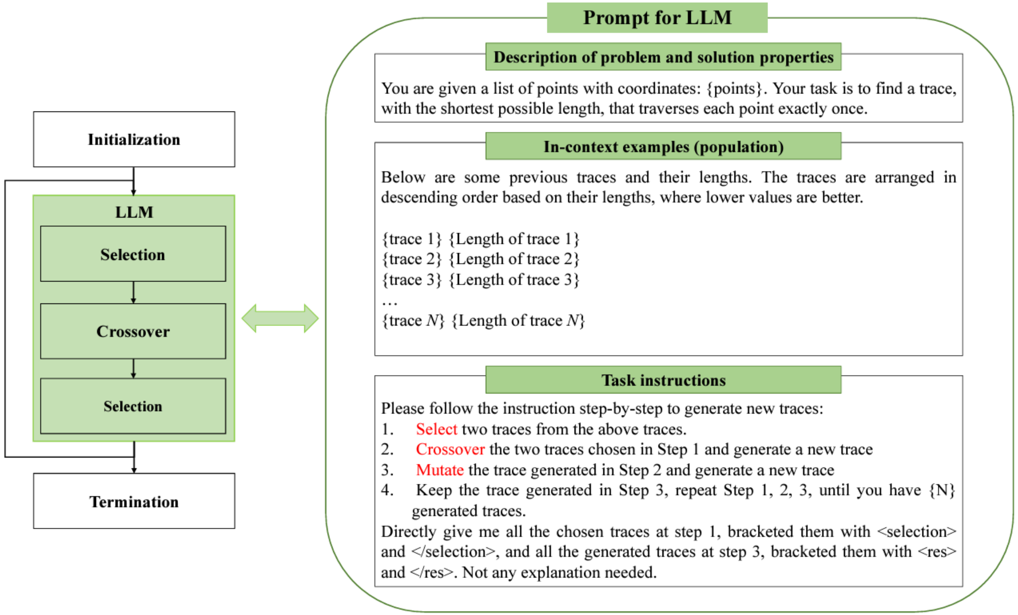
- Introduction
- Feature generation pipeline
- Proposed approach
- Optimisation methods
- Results discussion
- Further research

Introduction

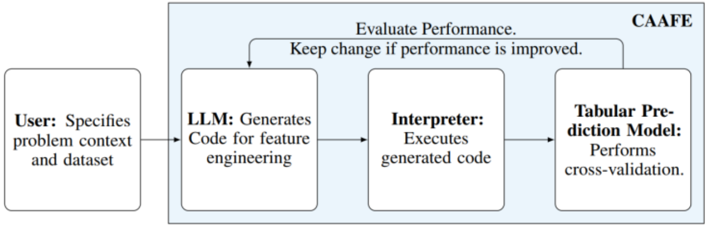
- Feature Engineering is one of the most time consuming steps in machine learning workflow
- FE is only partially implemented in AutoML solutions. It also lacks domain knowledge
- Large high-dimensional search space hinders performance of Automatic Feature Engineering tools
- Only tabular data is considered (including time-series)

Large Language Models are capable of domain knowledge injection and pattern recognition for optimisation problems. May improve Feature Engineering performance of AutoML tools

Large Language Models



Pattern recognition: general black-box optimisation, evolutionary optimisation, prompt enhancement



Domain knowledge: code generation for Feature Engineering

Feature Engineering Pipeline

Operation name	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 new 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
LabelEncoding	Label encoding of categorical features. In-place operation
OneHotEncoding	One hot encoding of categorical features

Atomic data operations used for feature engineering

Pipeline example (features taken from the titanic dataset)

Initial idea: DAG of atomic data operations

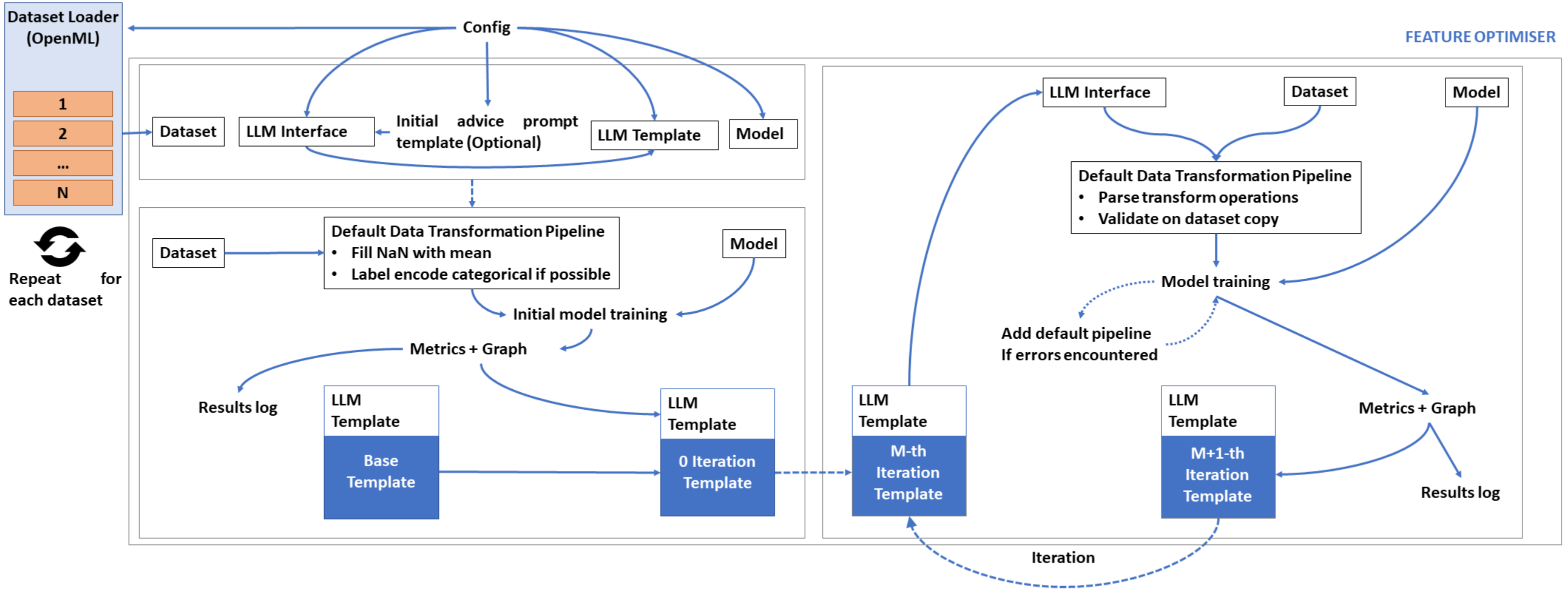
OperationName(Inp1, Inp2 -> Out)

- Operations only depend on order
- LLM proposals rarely use the output on next steps

Final pipeline format: operation sequence of predefined atomic operations.

General optimization scheme

Feature optimiser generates the pipeline for a single dataset. For each dataset, the initial prompt is constructed from the predefined pipelines and the initial LLM call. Then the optimisation cycle starts



Prompt structure

Prompt/pipeline	Average 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

List of evaluated pipeline structures and the corresponding scores on an example dataset

LLM prompt is composed from the following parts. For each, the source is added in brackets

1) Task description (config)
2) Pipeline format (config)
3) List of available operations (config)
4) Dataset description (dataset source)
5) Dataset meta-features (dataset source)
6) Initial advise (LLM generated)
7) Previous evaluations (model training)
8) Instruction (config)

Configuration file is the same for all the tasks. Dataset features does not change while the dataset-specific initial advise is generated anew on each optimization run. Previous evaluations are updated on each iteration

Optimisation methods

Random Search (Baseline). Sample generation: get number of operations from range, for each, specify type from the atomic ones, and a random number of input features as a random choice

LLM optimisation. Get response from initial prompt, parse and convert into the pipeline. Fit if necessary, evaluate score and update the prompt with the pipeline and its score. Next response is generated considering the new sample

Population optimization. Same as the LLM optimization, for each request, a population of pipelines is generated. Each is evaluated, and only the best one is included to the prompt.

Multistep approach. Extension for others. Before optimisation, the initial advise is requested from the LLM. Response is expected to contain the data insights and direct advises on the operations. Method results in more exploitative optimization.

Datasets

Dataset	OpenML ID	Description
Titanic	40945	Multiple data types, many open examples on feature engineering
credit-g	31	Simple numeric and nominal features, financial domain
blood-transfusion-service-center	1464	Small number of numeric features, healthcare domain
steel-plates-fault	1504	Large number of numeric features, high default score of 0.98, engineering domain
monks-problems-2	334	Small set of nominal features, test dataset for ML algorithms
tic-tac-toe	50	Dataset with simple analytical solution, nominal features only

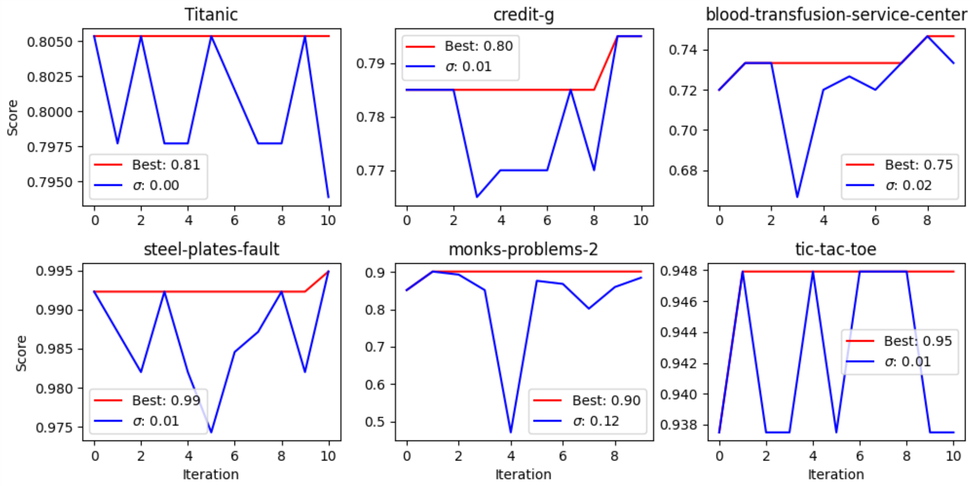
Datasets used for method evaluation. For each, some characteristics are listed

OpenML datasets contain:

- Feature and target description
- Description
- Meta features
- Collection date

Both domain-specific and general optimisation problems are included to the evaluation set. Only tabular data, all general data types are covered.

Results



Optimisation results example for a single-proposal mode

Dataset	Random Search		Single		Population		Single multi-step		Population multi-step	
	Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
Titanic	0.798	**0.973**	0.8	0.805	**0.912**	**0.973**	0.797	0.805	0.805	0.805
credit-g	0.779	0.8	0.779	0.795	**0.813**	**0.82**	0.78	0.79	0.789	0.795
blood-transfusion-service-center	0.723	0.753	0.723	0.746	**0.755**	**0.773**	0.734	0.753	0.746	0.76
steel-plates-fault	0.987	0.994	0.986	0.994	**0.998**	**1.0**	0.97	0.992	0.995	**1.0**
monks-problems-2	0.724	0.9	0.825	0.9	**0.95**	**0.966**	0.794	0.909	0.895	0.9
tic-tac-toe	0.902	0.958	0.942	0.947	0.937	0.937	**0.954**	**0.994**	0.949	0.958

Optimisation results for different approaches

- Population-based approach overperforms other methods while having comparable time cost
- Having the same total number of samples, population-based random search and multi-step population approaches did not achieve the same results
- Multi-step optimisation reduces proposal diversity and usually does not improve the score
- Black-box optimisation capability of LLM is the major score improvement factor. Domain knowledge might require code generation

Discussion

Datasets:

- Non-informative description for some datasets
- Low number of data samples for advises
- Meta-features from OpenML do not contain all necessary information

Pipeline

- Low number of implemented operations
- Custom output name unavailable, non-trivial names for operations with undetermined number of outputs (e.g. PCA). DAG solves the problem
- Text encoding should be improved. Operation splitter in response may affect the result. Line break results in lower quality pipelines. Token “->” may also affect the output quality

Optimisation

- Pipelines may be better if the evolutionary scheme is applied

LLM requests

- Many features were not utilized, e.g. function calls, system information, data embeddings

Pipeline

Pipeline format matters:

Line break as operation separator results in operations losing connection to neighbors. Token “->” is often used for “question”-”answer” in non-instruct models.

Solutions: turn into DAG or choose other sequence encoding method.

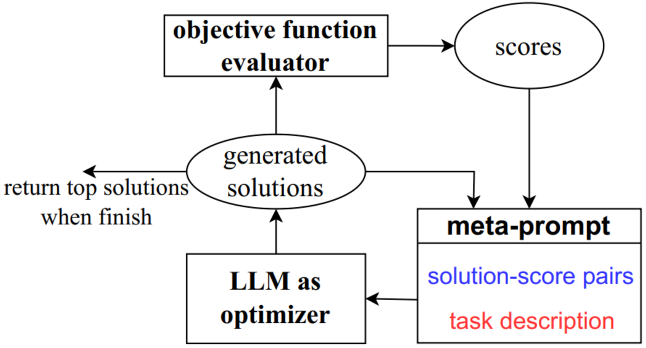
Example of DAG optimisation: initial assumptions for FEDOT framework. Starting the genetic algorithm from the optimized fitted solution with best result among other proposals.

LLM evolutionary optimisation

ITMO

Multistep approach advances: **meta-prompt for optimisation** (OPRO). Challenge: some composite prompt parts are generated for each dataset. Such a method suffers from all coordinate descent disadvantages

LLM-driven evolutionary algorithm applies the usual steps, i.e. selection, crossover, mutation, separately, while generating the LLM response



Prompt optimisation framework OPRO

Algorithm 1: LLM-driven EA (LMEA)

Input: The optimization problem T, maximum number of generations G, population size N;

Output: the best found solution s^*

$P \leftarrow$ randomly initialize N solutions to T;

$g = 1$;

while $g \leq G$ **do**

- *prompt* $\leftarrow$ construct prompt based on T and *pop*;
- $P' \leftarrow$ instruct LLM with *prompt* to generate N offspring solutions;
- $P \leftarrow$ the top N solutions among $P \cup P'$;
- Self-adapt the temperature of LLM if necessary;
- $g \leftarrow g + 1$;

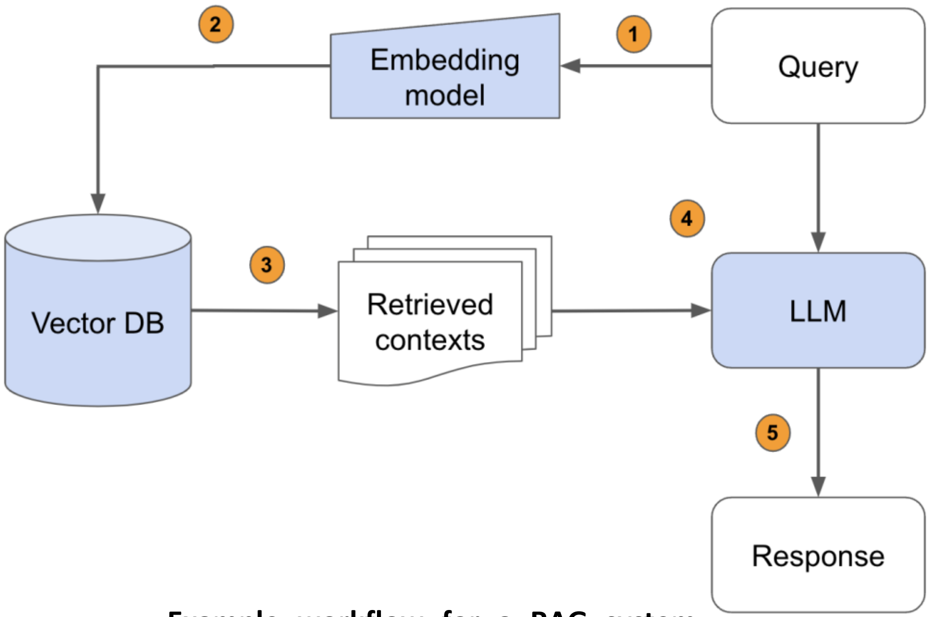
end

$s^* \leftarrow$ the best solution in P;

return s^*

LLM-driven evolutionary algorithm. Most operations are already implemented

LLM RAG



Example workflow for a RAG system with an agent over of existing data

Retrieval Augmented Generation (RAG) adds some new information to the prompt based on the user-defined functions.

Allows one to add more context to the prompt by using only the embedded data.

Example workflow

0) Separate documents from a knowledge base are embedded and saved to the vector DB (e.g. Redis)

1) Embedding model applied to initial query
2) Vector DB is searched for the best-matching document
3) New prompt is created with the retrieved information included
4) Prompt is passed to the LLM
5) Informed answer is generated

LLM RAG Application

1. Dataset information injection. Non-trivial document splitting, similar embeddings
2. LLM as a data analysis expert: based on the initial advice. First response – list of required data analysis operations. Each document in vectorDB: function description with the code or existing result. Further optimisation includes the evaluation results
3. LLM as a feature engineering expert. Get the most fitting data operations based on the advice. Next LLM request: apply the code to existing dataset. Does not depend on the pipeline format

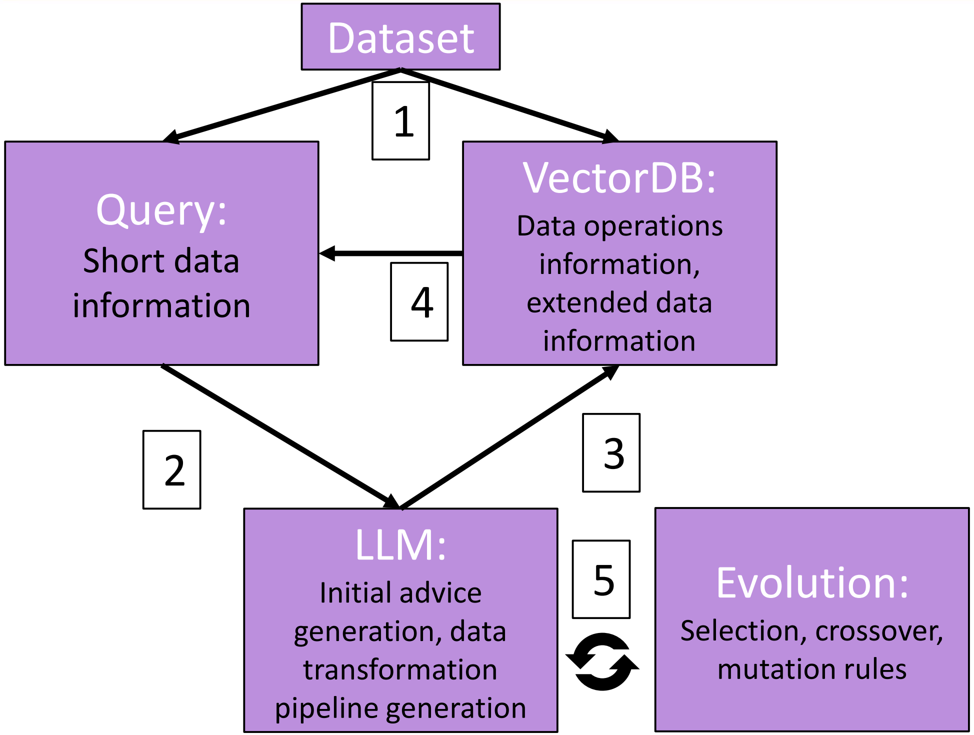
Further research

- Improve the pipeline format and update the operations set
- Optimise the prompt by using OPRO. Check how composite prompt optimisation is different from the original method (lower time and recourses for optimisation)
- Use self-hosted LLM for further fine-tuning and RAG implementation
- Implement existing evolutionary optimisation method for DAG pipeline for Feature Generation
- Implement RAG to insert more data into the prompt

Other directions worthy of mentinoning:

- Consider using code generation for creating separate atomic operations
- Generalise the approach by solving other tasks (time-series feature generation and FEDOT assumption have already been tried)

General Scheme of Feature Engineering Algorithm



1. Initial query generation, database initialization
2. LLM initial advise generation
3. Information retrieval from database using the embedding
4. Query update
5. Pipeline improvement based on evolutionary optimization. On each step, the pipeline is modified by custom rules

THANK YOU
FOR YOUR TIME!

ITsMOre than a UNIVERSITY

Your contact info