Ecole Normale Supérieure de Lyon

Genetic Programming applied to Automatic Workflow Composition in context of Time Series Forecasting

Vincent Gardies

Supervisor: **Carlos García-Martínez**

May-July 2024

Department of Computer Science
University of Córdoba

Córdoba, Spain

Contents

1 Introduction

1.1 Context and Background

The rapid advancements in machine learning have paved the way for Automated Machine Learning (AutoML), which aims to democratize the use of machine learning by automating the design, optimization, and deployment of machine learning models. One of the key challenges in AutoML is the Automatic Workflow Composition (AWC) problem : the goal for the automated design of machine learning pipelines is to handle a variety of tasks without requiring extensive human intervention. These pipelines typically include steps such as data preprocessing, feature selection, model selection, and hyperparameter optimization.

EvoFlow is a system designed to address the AWC problem by using Genetic Programming (GP) within a grammar-guided framework. Initially, EvoFlow was developed to effectively generate and optimize workflows, so that they easily could classify data based on a set of predefined classes. However, classification tasks represent only a subset of the potential applications in machine learning, with forecasting tasks—where the goal is to predict future values based on historical data—being equally important in various domains such as finance, economics, and weather prediction.

Given the growing importance of forecasting in real-world applications, EvoFlow has to improve to be able to handle these tasks.

1.2 Objective of the Internship

The primary objective of this internship was to adapt EvoFlow to accommodate forecasting workflows, thereby extending its applicability and enhancing its utility in the AutoML domain. This adaptation involves : modifying the underlying Genetic Programming framework and the context-free grammar (CFG) used by EvoFlow to generate workflows, ensuring that it can effectively handle the unique challenges posed by forecasting tasks, such as time series data dependencies and temporal feature extraction.

1.3 Structure of the Report

This paper will unfold as follows : first, a literature review, covering key concepts in AutoML, Genetic Programming, and forecasting tasks ; then a detailed overview of EvoFlow's initial design and limitations ; following this, the methodology will be discussed, as to outline the approach taken to adapt EvoFlow for forecasting, including changes to the G3P grammar and the development process ; the results and analysis section will evaluate the performance of our adaptation of EvoFlow ; finally, an analysis of the implications of this work on the broader AWC problem, on the limitations of the current adaptation, and suggestions for future research will take place.

2 Litterature Review

2.1 Auto ML and Automatic Workflow Composition

2.1.1 AutoML Overview

Automated Machine Learning (AutoML) refers to the automation of the end-to-end process which applies machine learning to real-world problems. AutoML tools aim to make machine learning accessible to non-experts by automating tasks such as data preprocessing, feature engineering, model selection, hyperparameter tuning, and model evaluation. The ultimate goal is to generate a machine learning pipeline that can be deployed with minimal human intervention while still achieving high performance.

2.1.2 Automatic Workflow Composition (AWC)

Within the broader scope of AutoML, the Automatic Workflow Composition (AWC) problem focuses on the automatic construction of machine learning workflows or pipelines. These workflows typically consist of a sequence of operations including : data transformations ; feature extraction ; and model training. AWC aims to identify the optimal combination and sequence of these operations to maximize the performance of the machine learning model.

AWC is a particularly challenging aspect of AutoML because the search space for possible workflows is vast and often non-linear. This problem further complicates because of the dependencies between different workflow components, where the choice of one operation can significantly impact the effectiveness of subsequent operations. As a result, AWC requires sophisticated optimization techniques, capable of navigating this complex search space.

2.2 Genetic Programming and G3P

2.2.1 Genetic Programming (GP)

Genetic Programming (GP) is a type of evolutionary algorithm that automatically makes programs or symbolic expressions evolving to solve a given problem [1]. Inspired by the principles of natural selection, GP operates on a population of candidate programs, which are represented as tree structures. Each node in the tree represents an operation or function, while the leaves represent inputs or variables.

GP uses genetic operators such as crossover (recombination of subtrees between parent programs) and mutation (random modification of nodes or subtrees) to explore the search space. The fitness of each program is evaluated based on its ability to solve the problem, and the most successful programs are selected to form the next generation. This process continues iteratively until an optimal or near-optimal program is found.

GP is particularly well-suited for problems where the solution can be represented as a sequence of operations or functions, making it a natural fit for AWC

tasks. By making workflows evolve in a similar manner to which programs evolve, GP can automatically discover effective machine learning pipelines..

2.2.2 Grammar-Guided Genetic Programming (G3P)

Grammar-Guided Genetic Programming (G3P) is an extension of standard GP that incorporates formal grammars to guide the evolution of programs. A grammar is a set of rules that defines the allowable structures and operations in the programs being evolved. In G3P, these rules are typically expressed in a context free grammar (CFG), which provides a flexible, yet constrained, way to generate syntactically correct programs.

G3P offers several advantages over standard GP. By enforcing grammatical constraints, G3P can prevent the generation of invalid or nonsensical programs, thereby reducing the search space, and improving the efficiency of the evolutionary process. Additionally, the use of grammars allows for more domain-specific knowledge to be encoded into the evolution process, leading to more effective and specialized solutions.

In the context of Evoflow, the G3P approach is used to evolve machine learning workflows that are both syntactically correct and effective for the task at hand. The grammar defines the set of allowable operations and their possible combinations, ensuring that the evolved workflows adhere to the principles of machine learning and data processing.

2.3 Forecasting

2.3.1 Overview of Forecasting Tasks

Forecasting means predicting future values based on historical data, distinguishing it from classification tasks that assign discrete labels. Forecasting is crucial in various domains, including financial market predictions, demand forecasting, weather prediction, and inventory management. The primary challenge in forecasting is modeling temporal dependencies, such as trends, seasonality, and autocorrelation, which are essential to obtain accurate predictions.

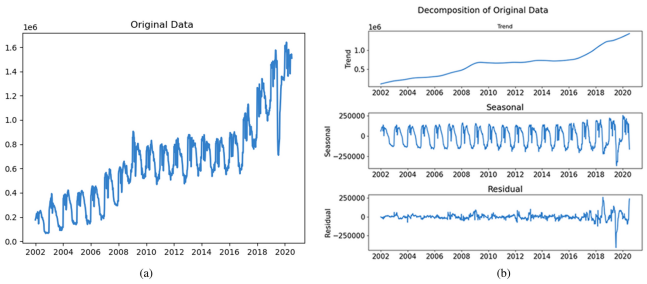

Figure 2: Decomposition of a time series (a) into Trend, Seasonal and Residual time series (b)

2.3.2 Machine Learning in Forecasting

In machine learning, forecasting often involves transforming time series data into a format suitable for model training. This process includes creating lagged features and aggregating exogenous variables, which help models capture temporal dynamics and external influences.

Lagged Features Lagged features are past values of the time series that are used as predictors for future values. For instance, to predict the value at time t, lagged features might include values from time $t-1$, $t-2$, and so on. These features enable the model to learn from historical patterns and trends.

Exogenous Variables Exogenous variables are external factors influencing the time series ; but those factors are not part of the primary data series, even though they are linked. Examples include : eco nomic indicators ; weather conditions ; or promotional activities. Incorporating these variables into the forecasting model allows it to account for external influences that may impact future values.

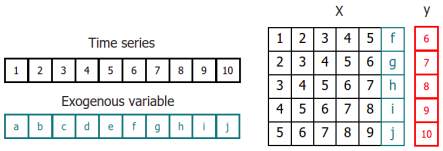

Figure 3: Data preparation for Forecasting (from [2]).

2.3.3 Forecasting Methods

Recursive Forecasting Recursive forecasting, also known as iterative forecasting, generates forecasts sequentially. The process involves : making an initial forecast ; updating the model with actual observations, as they become available ; and using the updated model to predict further into the future. This method is particularly useful for handling time series data with complex dependencies.

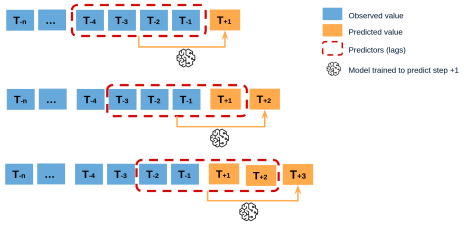

Figure 4: Recursive Multi-Step Forecasting (from [2]).

Direct Forecasting Direct forecasting involves generating forecasts for all future time steps in a single pass. This method does not rely on previous forecasts but instead predicts future values directly from historical data. Direct forecasting is often used in models where future predictions are made without iterative updates.

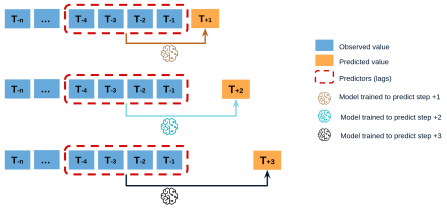

Figure 5: Direct Multi-Step Forecasting (from [2]).

2.3.4 Forecasting Techniques

Several forecasting techniques are employed to address different types of time series data. Here, we discuss key methods, including their advantages and limitations:

- **Time Series Analysis:**
  - **ARIMA (AutoRegressive Integrated Moving Average):** ARIMA models are widely used for forecasting univariate time series data. They capture autocorrelations and trends by combining autoregressive and moving average components with differencing to achieve stationarity.
    - *Pros:* Effective for capturing and forecasting trends and seasonality in stationary time series data; provides a structured framework for model selection and diagnostics.
    - *Cons:* Assumes linearity and stationarity, which may not be suitable for all time series; model selection and parameter tuning can be complex.
- **Machine Learning Approaches:**
  - **Regression Trees and Random Forests:** These techniques use decision trees or ensembles of trees to predict future values based on historical data. They are particularly useful for capturing non-linear relationships and interactions.
    - *Pros:* Handles non-linear relationships well; robust to overfitting with ensemble methods like Random Forests; suitable for large datasets.
    - *Cons:* Simple decision trees can overfit the training data; may require substantial computational resources for training.

- **Support Vector Machines (SVM):** SVMs can be adapted for regression tasks (SVR) by finding a hyperplane that fits the data within a margin of tolerance. They are effective in high-dimensional spaces and can model both linear and non-linear relationships.
  - *Pros:* Effective for complex and high-dimensional data; flexible kernel functions for various patterns.
  - *Cons:* Computationally intensive; requires careful parameter tuning and can be difficult to interpret.
- **Neural Networks:** Deep learning models, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), are designed to capture intricate patterns and dependencies in time series data.
  - *Pros:* Can model complex and long-range dependencies; adaptable to various types of data and forecasting tasks.
  - *Cons:* Requires large amounts of data and computational resources; models can be difficult to interpret and prone to overfitting without proper regularization.

2.4 Recent Innovations in AutoML for Forecasting

2.4.1 AutoGluon-TS

AutoGluon-TS [3] is an extension of the AutoGluon framework, designed specifically for time series forecasting. It leverages AutoML to automate various stages of the forecasting pipeline, including model selection, hyperparameter tuning, and ensemble learning.

- **Automatic Model Selection:** AutoGluon-TS automates the selection of models from a diverse set of forecasting algorithms, including traditional statistical models and modern machine learning approaches. This ensures that the most appropriate models are chosen based on the characteristics of the time series data.
- **Ensemble Learning:** It employs ensemble methods to combine predictions from multiple models, improving accuracy and robustness. The ensemble approach helps in capturing different aspects of the time series data, leading to more reliable forecasts.
- **Hyperparameter Optimization:** AutoGluon-TS utilizes automated hyperparameter tuning to optimize model performance. It systematically explores various configurations to find the best set of hyperparameters for each model.

2.4.2 BOAT

BOAT (Bayesian Optimization for Automated Time Series) [4] is another advanced AutoML framework tailored for time series forecasting. BOAT focuses

on optimizing forecasting models through Bayesian optimization techniques, enhancing the efficiency of model training and selection.

- **Bayesian Optimization:** BOAT uses Bayesian optimization to iteratively select and refine models based on their performance. This probabilistic approach helps in efficiently exploring the hyperparameter space and finding optimal configurations with fewer iterations.
- **Model Adaptation:** BOAT can adapt existing models to better fit specific forecasting tasks by optimizing model components and hyperparameters. This adaptability ensures that the models are well-suited to the unique characteristics of each time series.
- **Scalability:** The framework is designed to handle large-scale forecasting problems by efficiently managing computational resources and scaling up to accommodate complex datasets.

2.4.3 Comparison and Specificity

- **Approach:** While AutoGluon-TS emphasizes automated model selection and ensemble learning to aggregate the strengths of various models, BOAT focuses on Bayesian optimization to fine-tune models and hyperparameters. AutoGluon-TS offers a broader range of models and is well-suited for diverse forecasting scenarios, whereas BOAT excels in optimizing existing models and handling large-scale problems.
- **Hyperparameter Tuning:** AutoGluon-TS utilizes systematic hyperparameter tuning across multiple models, ensuring comprehensive optimization. In contrast, BOAT's Bayesian optimization provides a more targeted and efficient search for optimal hyperparameters, often requiring fewer evaluations.
- **Ensemble Methods:** AutoGluon-TS integrates ensemble learning to enhance model robustness and accuracy, combining predictions from different models. BOAT, however, primarily focuses on optimizing individual models and does not emphasize ensemble approaches.

3 EvoFlow: A Grammar-Based Evolutionary Approach for Automated Workflow Composition

Automated Workflow Composition (AWC) is a crucial aspect of Automated Machine Learning (AutoML), focusing on automating the selection and sequencing of machine learning algorithms and data preprocessing steps. This section introduces *EvoFlow* [5], a novel approach that leverages grammar-based genetic programming (G3P) to enhance the flexibility and efficiency of AWC tasks. GitHub Repository: evoflow_awc

3.1 Purpose

EvoFlow is designed to overcome the limitations of traditional AWC methods, which often rely on rigid, predefined workflow templates. By utilizing a context-free grammar (CFG) to guide the evolutionary process, EvoFlow can generate a wide variety of workflow structures that are tailored to the specific data and task at hand. This flexibility is crucial for adapting to the diverse requirements of different machine learning problems.

3.2 Key Mechanisms of EvoFlow

EvoFlow operates through several distinct mechanisms that contribute to its effectiveness.

Grammar-Guided Workflow Generation: EvoFlow is centered around the use of a context-free grammar (CFG) that defines the permissible operations and their combinations within a workflow. This grammar serves as a blueprint for producing syntactically correct workflows, ensuring that each candidate solution meets the requirements of valid machine learning pipelines. The CFG encodes domain knowledge, including possible preprocessing steps, feature engineering methods, model types, and evaluation metrics, thereby guiding the evolutionary process towards viable solutions.

Population Initialization: EvoFlow begins by initializing a population of workflows, each represented as a tree structure where nodes correspond to specific operations or functions, and leaves correspond to data inputs or hyperparameters. The initialization process is designed to explore a diverse set of possible workflows, covering a broad range of potential solutions to the AWC problem.

Fitness Evaluation: The fitness of each workflow in the population is evaluated based on its performance on a given task. This evaluation typically involves training the workflow on a subset of the data and assessing its predictive accuracy, generalization capability, and computational efficiency. EvoFlow may also incorporate ensemble techniques, where the diversity of predictions across workflows is considered an additional fitness criterion, enhancing the robustness of the final solution.

Genetic Operators: EvoFlow employs specialized genetic operators tailored to the AWC problem. The crossover operator exchanges subtrees between two parent workflows, effectively combining different parts of their structures. This operator is guided by the CFG to ensure that the resulting offspring are syntactically valid workflows. Crossover enables the exploration of new workflow combinations that may inherit the strengths of both parent workflows. Additionally, the mutation operator introduces random modifications to a workflow by altering nodes, changing hyperparameters, or replacing subtrees. This operator is also constrained by the CFG, preventing the generation of invalid workflows. Mutation helps maintain diversity within the population and allows EvoFlow to explore new areas of the search space.

Selection and Archiving: After each generation, EvoFlow selects the best-

performing workflows based on their fitness scores. Additionally, an archive of diverse workflows is maintained, ensuring that the population does not converge prematurely on a single solution. This diversity mechanism is crucial for building robust ensembles, as it encourages the retention of workflows that offer complementary strengths and weaknesses.

Termination and Workflow Selection: The evolutionary process continues until a predefined number of generations have been completed or a time budget has been exhausted. The final output of EvoFlow is either a single optimal workflow or an ensemble of workflows selected from the archive, each contributing to the overall predictive performance of the model.

3.3 Conclusion

EvoFlow represents a significant advancement in the field of Automated Workflow Composition. By leveraging the power of Grammar-Guided Genetic Programming and incorporating mechanisms to ensure diversity and optimization, EvoFlow provides a flexible and powerful tool for automating the design of machine learning workflows. Its ability to adapt to different tasks and datasets makes it a valuable addition to the AutoML toolkit.

4 Methodology

4.1 Approach to Adapting EvoFlow

The strategy to adapt EvoFlow for forecasting tasks was designed to leverage its existing strengths in workflow generation while extending its capabilities to handle the specific challenges posed by time series data. The adaptation process involved several key steps, including the modification of the Grammar-Guided Genetic Programming (G3P) framework, the development of new software components, and the evaluation of the adapted system on forecasting tasks.

4.2 Modifying the G3P Grammar

The G3P grammar used in EvoFlow was originally tailored for classification tasks, encoding domain knowledge related to data preprocessing, feature engineering, and model selection. To support forecasting tasks, the grammar needed to be extended to include operations specific to time series data, such as temporal feature extraction, and forecasting model. The modifications ensured that the grammar could generate workflows capable of handling the sequential nature of time series data, while still adhering to the constraints required for valid machine learning pipelines.

The core of the grammar, which is quite simple, is the following:

⟨*workflow*⟩ ::= ⟨*forecastingBranch*⟩

⟨*forecastingBranch*⟩ ::= ⟨*forecaster*⟩
| ⟨*preprocessingBranch*⟩ ; ⟨*forecaster*⟩

⟨*preprocessingBranch*⟩ ::= ⟨*preprocess*⟩
| ⟨*preprocessingBranch*⟩ ; ⟨*preprocess*⟩

But it then complexifies, because for example some forecasters are using regressors or even forecasters:

⟨*forecaster*⟩ ::= (Forecaster) ; (Forecaster Hyper Parameters)
| ⟨*regressor*⟩ ; (Forecaster) ; (Forecaster Hyper Parameters)
| ⟨*forecaster*⟩ ; (Forecaster) ; (Forecaster Hyper Parameters)

⟨*preprocess*⟩ ::= (Preprocessor) ; (Preproccesor Hyper Parmaters)
| ⟨*preprocess*⟩ ; (Preprocessor) ; (Preproccesor Hyper Parmaters)

⟨*regressor*⟩ ::= (Regressor) ; (Regressor Hyper Parameters)
| ⟨*regressor*⟩ ; (Regressor) ; (Regressor Hyper Parameters)

4.2.1 Research into Python Libraries for Forecasting

During the adaptation process, extensive research was conducted to identify suitable Python libraries that could be integrated into the EvoFlow framework to facilitate the creation of forecasting pipelines. Three prominent libraries were selected: `scikit-learn` (`sklearn`), `skforecast`, and `sktime`.

Scikit-learn: [6]
`scikit-learn` is a widely-used Python library that provides simple and efficient tools for data mining and data analysis, built on NumPy, SciPy, and matplotlib. In the original version of EvoFlow, `scikit-learn` played a critical role in enabling the creation of various machine learning models, including classification and regression tasks. Given its robust implementation of algorithms and ease of use, `scikit-learn` continues to be an integral part of EvoFlow's architecture in the new version. It serves as the foundation upon which more specialized forecasting libraries like `skforecast` and `sktime` are built.

Skforecast: [7]
`skforecast` is a Python library designed for time series forecasting, which simplifies the use of machine learning models such as linear regression, decision trees, and even complex models like XGBoost for forecasting tasks. It

enables users to easily implement and evaluate forecasting models, leveraging the familiar `scikit-learn` API. The library includes utilities for creating lag features, handling rolling windows, and tuning hyperparameters, making it an ideal choice for integrating into the EvoFlow framework.

Sktime: [8]
`sktime` is another Python library focused on unified interfaces and tools for time series analysis, including forecasting, classification, and regression. It offers a wide range of time series algorithms and workflows, providing a consistent interface for implementing, testing, and comparing different models. `sktime` supports various tasks from basic forecasting to complex multivariate time series problems, which aligns with the goals of extending EvoFlow's capabilities to cover a broader range of forecasting scenarios.

The integration of these libraries into the G3P grammar allowed EvoFlow to generate and optimize workflows that include state-of-the-art forecasting models and techniques. These libraries also facilitated the incorporation of advanced time series features and preprocessing steps, further enhancing the adaptability and effectiveness of the workflows generated by EvoFlow. The continued use of `scikit-learn`, alongside the new additions of `skforecast` and `sktime`, ensures that EvoFlow remains a powerful and versatile tool for both traditional machine learning tasks and advanced time series forecasting.

4.2.2 Implementation

In practice, the grammar is an Extensible Markup Language file (.xml).

Terminal terms are defined, with there hyper parameters in the following way:

Definition of Prophet, a Terminal Term

```
<terminal name="Prophet" type="forecaster" code="sktime.forecasting.fbprophet.Prophet">
    <hparam name="pht::growth" type="categorical" values="linear;logistic"/>
    <hparam name="pht::growth_floor" type="uni_float" lower="0" upper="0.1" default="0" strong_default="True"/>
    <hparam name="pht::growth_cap" type="uni_float" lower="1" upper="100"/>
    <hparam name="pht::n_changepoints" type="uni_int" lower="1" upper="50" default="25"/>
    <hparam name="pht::changepoint_range" type="uni_float" lower="0.7" upper="0.95"/>
    <hparam name="pht::seasonality_mode" type="categorical" values="additive;multiplicative"/>
    <hparam name="pht::seasonality_prior_scale" type="uni_float" lower="0" upper="20"/>
    <hparam name="pht::holidays_prior_scale" type="uni_float" lower="0" upper="20"/>
    <hparam name="pht::changepoint_prior_scale" type="uni_float" lower="0" upper="0.5"/>
    <hparam name="pht::mcmc_samples" type="uni_int" lower="1" upper="100" default="0" strong_default="True"/>
    <hparam name="pht::alpha" type="uni_float" lower="0" upper="0.2"/>
    <hparam name="pht::uncertainty_samples" type="uni_int" lower="0" upper="2000" default="False"
    ↪  strong_default="True"/>
</terminal>
```

This term has a name, a type and a code which is the library where the class is defined. Moreover, twelve hyperparameters are defined: *growth*, *growth_floor*, *growth_cap*, *n_changepoints*, *changepoint_range*, *seasonality_mode*, *seasonality_prior_scale*, *holidays_prior_scale*, *changepoint_prior_scale*, *mcmc_samples*, *alpha*, *uncertainty_samples*. Each of these hyperparameters has in its name the prefix *pht::* referring to *Prophet* the term they come from. Besides, they all have

their own type and some precision like default value, boundaries, distribution accordingly to their type.

Whereas **Non Terminal** terms are defined this way:

Definition of Prophet, a Terminal Term

```
<non-terminal name="regressor">
    <production-rule>RandomForest;RandomForest_hp</production-rule>
    <production-rule>LinearRegression;LinearRegression_hp</production-rule>
    <production-rule>RidgeRegression;RidgeRegression_hp</production-rule>
    <production-rule>LassoRegression;LassoRegression_hp</production-rule>
    <production-rule>ElasticNetRegression;ElasticNetRegression_hp</production-rule>
    <production-rule>SVR;SVR_hp</production-rule>
    <production-rule>DecisionTreeRegressor;DecisionTreeRegressor_hp</production-rule>
    <production-rule>GradientBoostingRegressor;GradientBoostingRegressor_hp</production-rule>
    <production-rule>regressor;AdaBoostRegressor;AdaBoostRegressor_hp</production-rule>
    <production-rule>KNeighborsRegressor;KNeighborsRegressor_hp</production-rule>
    <production-rule>BayesianRidge;BayesianRidge_hp</production-rule>
    <production-rule>GaussianProcessRegressor;GaussianProcessRegressor_hp</production-rule>
    <production-rule>HuberRegressor;HuberRegressor_hp</production-rule>
</non-terminal>
```

As we can see in the example, every production rule gives a new term coupled with its hyperparameters term. Besides we can see that some production rules include a regressor, meaning that a recursive derivation is possible.

4.2.3 Content of the grammar

There are 3 types of component introduced in the grammar [9]: *preprocessors*, *forecasters* and *regressors*

The **preprocessors** are: (from sklearn by default)

- Scalers:
  - *Normalizer*
  - *Min-Max Scaler*
  - *Robust Scaler*
- Encoder:
  - *One Hot Encoder*
- Transformers:
  - *Time Series Differentiator* (from skforecast)
  - *Power Transformer*
- Selectors:
  - *Variance Treshold*
  - *Generic Univariate Select*
  - *Random Selector* (self made)
- Imputers:
  - *Simple Imputer*
  - *kNN Imputer*
- Featurer:
  - *Polynomial Features*

The **regressors** are: (from sklearn by default)

- *Random Forest*
- *Linear Regression*
- *Ridge Regression*
- *Lasso Regression*
- *Elastic Net Regression*
- *SVR*
- *Decision Tree Regressor*
- *Gradient Boosting Regressor*
- *Ada Boost Regressor*
- *K Neighbors Regressor*
- *Bayesian Ridge*
- *Gaussian Process Regressor*
- *Huber Regressor*

The **forecasters** are: (from skforecast by default)

- *Auto Regressive Forecaster*
- *Auto Regressive Direct Forecaster*
- *Recurrent Neural Network*
- *Prophet* (from sktime)
- *SARIMAX*
- *Equivalent Date Forecaster*
- *Decomposers* (using statsmodel [10])

4.3 Software Development and Integration

The development process involved updating the EvoFlow codebase to incorporate the changes in the G3P grammar and to handle the new types of workflows generated for forecasting tasks. This required significant coding effort to implement new operators, data structures, and evaluation metrics suited for forecasting. The integration phase ensured that these new components were compatible with the existing EvoFlow system, maintaining its ability to effectively solve the Automatic Workflow Composition (AWC) problem while expanding its applicability.

4.3.1 Parametrization of Hyperparameters

An important enhancement in the EvoFlow codebase involved the parametrization of hyperparameters, allowing for greater flexibility and control over the workflow configuration. This section outlines the new features introduced for hyperparameter management:

- **Default Values with High Probability:** To streamline the configuration process, a mechanism was introduced to specify default values for hyperparameters with high probability. This feature ensures that commonly used default settings are applied automatically unless explicitly overridden. This approach not only simplifies the setup for standard scenarios but also helps maintain consistency and reliability across different experiments.

- **Union of Types for Hyperparameters:** The system now supports a union of types for hyperparameters, allowing multiple types to be associated with a single hyperparameter. This flexibility enables users to define hyperparameters that can accept a range of data types, such as integers, floats, or strings. For example, a hyperparameter could be specified to accept either an integer or a float, catering to different requirements without needing separate configurations.
- **Introduction of Tuples:** In addition to supporting traditional data types, the system now accommodates new types, including tuples. Tuples offer a structured way to group multiple values under a single hyperparameter. This feature is particularly useful for hyperparameters that require a combination of values, such as ranges or sets of parameters. Moreover, this type is necessary to parametrize SARIMAX's model.

These advancements in hyperparameter parametrization enhance the system's flexibility, allowing users to tailor configurations more precisely and adapt to various scenarios in forecasting tasks. The integration of these features ensures that EvoFlow remains robust and versatile in addressing diverse Automatic Workflow Composition (AWC) challenges.

4.3.2 Abstract Evaluator Class and Specialized Subclasses

To enhance the modularity and extensibility of the EvoFlow system, an abstract class for evaluators was introduced. This class provides a foundational framework for implementing various evaluation strategies tailored to different machine learning tasks.

AbstractEvaluator Class The `AbstractEvaluator` class is an abstract base class designed to standardize the evaluation process across different machine learning tasks. This class includes essential methods and attributes that need to be implemented by its subclasses. The core components of the `AbstractEvaluator` class are as follows:

- `__init__`: Initializes the evaluator with parameters such as `timeout`, `measure`, `cv` (cross-validation folds), `outdir` (output directory), `seed`, and `ml_type`. It also sets placeholders for `fold` and `invalid_value`.
- `evaluate`: Executes the evaluation of a given individual `ind` using the provided data `X` and target `y`. It tracks execution time, writes results to a file, and returns the fitness score, predictions, and elapsed time.
- `evaluate_cv`: Handles cross-validation by splitting data into training and testing sets, training the model, and calculating fitness scores. It also manages timeouts and error handling during model fitting and prediction.
- `fit_predict`: Trains the model and makes predictions, taking into account memory limits and data preparation steps.

- `try_fit_predict`: Manages exceptions during model training and prediction, ensuring that errors are properly handled and reported.
- `valid_prediction`: A placeholder method for validating predictions, which can be customized in subclasses.
- `prepare_data`, `fit_transform_y`, `fit_transform`, `fit_resample`, `predict`, `fit_ml`, `split`, `init_args`, and `get_train_data`: Abstract methods that must be implemented by subclasses to handle specific data processing and model training tasks.

Specialized Subclasses Derived from the `AbstractEvaluator` class, specialized subclasses cater to distinct machine learning tasks, providing tailored evaluation methods and metrics.

- **ClassificationEvaluator:** This subclass is designed to evaluate classification models. It mainly uses the previous code of Evoflow that evaluates a pipeline.
- **ForecastingEvaluator:** A new addition for evaluating forecasting models. The `ForecastingEvaluator` class is tailored to handle time series data and forecasting-specific evaluation procedures.

The introduction of the `AbstractEvaluator` class and its specialized subclasses allows EvoFlow to support a variety of evaluation methods in a structured and extensible manner. This design ensures that new evaluation strategies can be seamlessly integrated into the system, enhancing its adaptability to different machine learning tasks.

4.4 Evaluation and Testing

The adapted version of EvoFlow is avalaible on Github.

4.4.1 Testing EvoFlow with Diverse Datasets

Diverse Dataset Sizes EvoFlow was tested across datasets of varying sizes to assess its scalability and performance. Smaller datasets were used to evaluate the framework's ability to handle limited data while ensuring that models do not overfit. Conversely, larger datasets allowed for the examination of EvoFlow's efficiency in processing substantial amounts of data and its capacity to maintain performance with increased complexity.

Inclusion of Exogenous Variables Several datasets included exogenous variables—external factors that can influence the primary time series being forecasted. For instance, datasets related to financial markets were supplemented with economic indicators, while weather forecasting datasets incorporated variables such as humidity and wind speed. By integrating exogenous variables, the framework's ability to leverage additional information and improve forecasting accuracy was tested.

Compatibility with Classification Tasks In addition to forecasting tasks, EvoFlow's functionality in classification tasks was verified. Datasets designed for classification problems were utilized to ensure that EvoFlow could seamlessly switch between forecasting and classification modes with the new modifications.

4.4.2 Examples of Datasets Used

Those datasets available on skforecast, were chosen to ensure robustness of the adapted version of Evoflow:

- **H2O Expenditure Data**
  Description: Monthly expenditure on corticosteroid drugs by the Australian health system (1991-2008) with simulated exogenous variables. **Source:** Hyndman R (2023). fpp3package (GitHub).
- **Fuel Consumption Data**
  Description: Monthly fuel consumption in Spain (1969-2022). **Source:** Cores.
- **Air Quality in Valencia**
  Description: Hourly air pollutant measurements in Valencia. **Source:** GVA.
- **Website Visits Data**
  Description: Daily website visits data from cienciadedatos.net. **Source:** Zenodo.
- **Bike Sharing Data**
  Description: Hourly bike-sharing usage in Washington, D.C., with weather and holiday info. **Source:** UCI Machine Learning Repository.
- **Australia Tourism Data**
  Description: Quarterly overnight trips in Australia (1998-2016). **Source:** Journal.
- **UK Daily Flights Data**
  Description: Daily flight numbers in the UK (2019-2022). **Source:** ONS.

5 Results and Analysis

The results obtained from testing EvoFlow demonstrated promising outcomes when applied to various forecasting tasks. EvoFlow's approach, leveraging grammar-guided genetic programming, was able to generate workflows that performed well against traditional forecasting methods. The ability to automatically construct and optimize workflows specifically tailored for time series data highlighted EvoFlow's potential as a robust tool in the AutoML space.

However, while EvoFlow showed strong performance in comparison to some conventional forecasting techniques, a comprehensive evaluation against other AutoML tools, such as AutoGluon-TS and BOAT, could not be completed. These tools are well-known for their advanced capabilities in automated model selection, hyperparameter optimization, and ensemble learning, which are critical aspects of effective forecasting. Unfortunately, due to time constraints, the comparison of EvoFlow's performance against these state-of-the-art AutoML frameworks could not be fully realized, leaving a gap in the assessment of EvoFlow's relative effectiveness.

6 Conclusion

In conclusion, the adaptation of EvoFlow for forecasting tasks yielded positive results, demonstrating the system's capability to handle time series data effectively. EvoFlow's performance suggests that it can be a valuable addition to the suite of tools available for automated forecasting. However, to fully ascertain its competitiveness, a detailed comparison with other AutoML frameworks like AutoGluon-TS and BOAT is necessary. Due to the limited time available for this study, such a comprehensive comparison could not be undertaken. Future work should focus on completing this comparative analysis to better understand EvoFlow's strengths and areas for improvement within the broader context of AutoML solutions for forecasting.

References

[1] David E. Goldberg and John H. Holland. "Genetic Algorithms and Machine Learning". In: *Machine Learning* 3.2 (Oct. 1988), pp. 95–99. ISSN: 1573-0565. DOI: 10.1023/A:1022602019183. URL: https://doi.org/10.1023/A:1022602019183.

[2] Ciencia de Datos. *Time Series Forecasting in Python with Scikit-learn*. https://cienciadedatos.net/documentos/py27-time-series-forecasting-python-scikitlearn.html. Accessed: 2024-08-25. 2023.

[3] Oleksandr Shchur et al. "AutoGluon–TimeSeries: AutoML for Probabilistic Time Series Forecasting". In: *Proceedings of the Second International Conference on Automated Machine Learning*. Ed. by Aleksandra Faust et al. Vol. 224. Proceedings of Machine Learning Research. PMLR, Nov. 2023, pp. 9/1–21. URL: https://proceedings.mlr.press/v224/shchur23a.html.

[4] John Joy Kurian et al. "BOAT: A Bayesian Optimization AutoML Time-series Framework for Industrial Applications". In: *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*. 2021, pp. 17–24. DOI: 10.1109/BigDataService52369.2021.00008.

[5] R. Barbudo, A. Ramírez, and J.R. Romero. "Grammar-based evolutionary approach for automated workflow composition with domain-specific operators and ensemble diversity". In: *Applied Soft Computing* 153 (2024), p. 111292. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2024.111292.

[6] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[7] Joaquin Amat Rodrigo and Javier Escobar Ortiz. *skforecast*. Version 0.13.0. Aug. 2024. DOI: 10.5281/zenodo.8382788. URL: https://skforecast.org/.

[8] Franz Király et al. *sktime/sktime: Release v0.31.2*. Version v0.31.2. Aug. 2024. DOI: 10.5281/zenodo.13310241. URL: https://doi.org/10.5281/zenodo.13310241.

[9] Ahmad Alsharef et al. "Review of ML and AutoML Solutions to Forecast Time-Series Data". In: *Archives of Computational Methods in Engineering* 29.7 (Nov. 2022), pp. 5297–5311. ISSN: 1886-1784. DOI: 10.1007/s11831-022-09765-0. URL: https://doi.org/10.1007/s11831-022-09765-0.

[10] Skipper Seabold and Josef Perktold. "statsmodels: Econometric and statistical modeling with python". In: *9th Python in Science Conference*. 2010.