Mini AutoML System

(Automated Machine Learning - Project 2)

Ada Wojterska, Katarzyna Skoczylas, Miłosz Zieliński

26.01.2026

1 Introduction

The goal of this project was to create a simple AutoML system capable of performing binary classification tasks on any given tabular dataset. A key part of our work was the development and use of a model portfolio.

2 Stage 1 & 2: Algorithms and Hyperparameters Selection

This stage describes how we found 50 algorithms and sets of hyperparameters to our model portfolio. The maximum size of the portfolio was set to 50, and we utilized it fully. The entire process can be found in a file `finding_hyperparameters`.
We decided to build our final set of models from five classification algorithm families: **Logistic Regression**, **Decision Tree**, **Random Forest**, **XGBClassifier**, and **LGBMClassifier**.

The selection of classification algorithms was based on the need to use diverse learning strategies and ensure satisfactory performance across varying data characteristics. During our experiment, we observed that boosting models performed better on datasets with a larger number of features. We were also inspired by the **LightAutoML** library, which we analyzed earlier for our presentation. As a result, we decided to use Logistic Regression, LightGBM, and XGBoost in our system. Moreover, we included Decision Trees and Random Forests to provide a balance between simple models and more complex boosting algorithms.

2.1 Datasets and Training Procedure

For each of the five families of algorithms, we randomly selected 50 sets of hyperparameters (250 in total). To build an effective knowledge base for the AutoML system, we evaluated each configuration on a diverse set of datasets from the OpenML repository. These datasets were chosen to represent different levels of complexity and feature characteristics, including both balanced and imbalanced classes, as well as continuous and categorical features. Table 1 presents the datasets used in the experiment along with their dimensions.

We trained the models using all hyperparameter sets on all datasets. The evaluation metric employed during this process was *Balanced Accuracy*, as it will serve as the final evaluation metric in our AutoML system. For this experimental phase, we utilized a simple train-test split with a 70:30 ratio.

Table 1: Overview of datasets used in the experiment (Source: OpenML)

Dataset Name	Instances	Features	Source
breast_cancer	286	9	`https://www.openml.org/d/13`
adult	48,842	14	`https://www.openml.org/d/1590`
madelon	2,600	500	`https://www.openml.org/d/1485`
phoneme	5,404	5	`https://www.openml.org/d/1489`
amazon_employee_access	32,769	9	`https://www.openml.org/d/4135`
ozone_level_8hr	2,534	72	`https://www.openml.org/d/1487`

2.2 Methodology for Optimal Configuration Selection

To create a universal model portfolio for the AutoML system, a naive selection of the top 50 models based solely on global scores was rejected. Such an approach would have resulted in the dominance of a single algorithm type (Logistic Regression). Instead, we applied a diversity-oriented strategy described below.

The *Balanced Accuracy* score was calculated for each model across all six datasets. A new aggregated metric, *Total Balanced Accuracy*, was defined as the sum of *Balanced Accuracy* scores obtained by a hyperparameter set across all datasets. To ensure diversity, we first selected the top 3 sets from each of the 5 algorithm families based on *Total Balanced Accuracy*. The remaining slots (to reach a total of 50) were filled by selecting the best-performing remaining configurations, regardless of the model type, also based on *Total Balanced Accuracy*.

From the selected configurations, a JSON file was generated to use in the final system.
The final model portfolio includes the following algorithms:

- Logistic Regression (25 models)
- XGBoost (12 models)
- LGBM (7 models)
- Decision Tree (3 models)
- Random Forest (3 models)

3 Stage 3: Mini-AutoML System Implementation

The core of the project is the `MiniAutoML` class, designed to automate the end-to-end pipeline for binary classification tasks. The system consists of three main components: a preprocessor, a multi-stage model screening process, and an ensembling phase.

3.1 Data Preprocessing

The system assumes input data is in a standard tidy data format, where categorical variables are explicitly provided in string or object formats. The `Preprocessor` class automates the preprocessing pipeline, assuming that numerical columns represent continuous or discrete values. The pipeline automatically handles the following operations:

- **Missing Values:** Numerical gaps are filled with the median, while categorical missing values are replaced with the mode.
- **Categorical Encoding:** Non-numerical features are transformed using `OneHotEncoder` with a safety mechanism (`handle_unknown='ignore'`) to prevent errors on new data.

- **Feature Scaling:** All numerical features are processed using `StandardScaler`.

3.2 Model Selection Strategy (fit method)

The selection process is executed in the `fit` method and follows a two-tier approach to balance efficiency and predictive power. In the first stage, all 50 configurations from the portfolio are evaluated using 3-fold cross-validation on the training data. To maintain performance on large datasets, a subsampling heuristic was applied: if the dataset exceeds 10,000 rows, a random subset of 10,000 instances is used during the screening phase.

In the second stage, the selection procedure prioritizes model diversity rather than relying solely on performance ranking. Instead of selecting only the top five configurations overall, the system first identifies the best-performing model from three distinct algorithm families (e.g., the best XGBoost, the best Random Forest and the best Logistic Regression models). The remaining slots of the final Top-5 set are then filled with the next best-performing models across all configurations.

3.3 Ensembling and Final Evaluation

After identifying the Top-5 candidates during the screening phase, the system constructs and evaluates several ensembling strategies to further improve Balanced Accuracy. This stage focuses on combining the complementary strengths of different models. At first, the system builds **soft-voting** ensembles using both the Top-3 and Top-5 selected models, named accordingly as *Voting_ Top3* and *Voting_ Top5*.

Later three **stacking** architectures are created, in which a final-estimator is trained to combine predictions from the Top-3 candidates. To ensure fair and fully reproducible comparisons, all stacking configurations are assessed using 3-fold cross-validation with *shuffle=true* and constant *random_ state*. Those final-estimators with their names are:

- **LogisticRegression** (Stacking_ Top3)
- **LogisticRegression with *passthrough=True*** (Stacking_ Top3_ Passthrough)
- **RandomForestClassifier** (Stacking_ RF)

Those three were chosen to compare a linear combination of base model predictions (LogisticRegression), the inclusion of original input features (passthrough variant), and the ability to capture non-linear interactions between base models (RandomForestClassifier).

To determine the final winner, the system performs an evaluation of all candidate models, including the best single model, voting ensembles, and stacking variants, using 5-fold cross-validation on the training data. The model configuration achieving the highest Balanced Accuracy is selected. Once the optimal architecture is identified, it is retrained on the full training dataset to ensure optimal performance.

3.4 Time Management

The `MiniAutoML` system dynamically manages execution time to deliver a model within the time limit specified by the user. If the time limit is reached during the model portfolio testing phase, the process is interrupted and the system proceeds to the next stage using the best results obtained so far. More complex models, such as voting and stacking ensembles, are constructed only if at least 30% of the total time budget remains. Additionally, as mentioned earlier, for datasets exceeding 10,000 rows, the system applies random subsampling to enable rapid candidate selection while maintaining evaluation stability. Moreover, for the datasets

used in our experiments, the system performed efficiently enough to allow the evaluation of all models in the portfolio and the construction of multiple ensemble variants without exceeding the imposed time constraints.

3.5 MiniAutoML Evaluation

To evaluate our system, we tested it on several binary classification datasets, focusing on four primary ones: Pima Indians Diabetes, Bank Marketing, Airline Passenger Satisfaction and a dataset provided by course coordinators.

For benchmarking, we compared our results with two popular AutoML libraries: **LightAutoML** and **MLJAR**. In MLJAR, the models were trained using the ROC-AUC metric because Balanced Accuracy was not available. LightAutoML was trained using Balanced Accuracy. For a final comparison, we calculated the Balanced Accuracy on the test set for all three systems. Each time, the data are split into an 80/20 train-test set. The results are presented in the Table 2 below:

Table 2: Comparison of Balanced Accuracy scores.

Dataset	**LightAutoML**	**MLJAR**	**MiniAutoML**
Pima Indians	0.7272	**0.7644**	0.7361
Bank Marketing	0.8347	0.8568	**0.8662**
Provided Dataset	0.5390	0.5266	**0.5537**

Initially, we were surprised by the low scores (around 0.55) on the provided dataset, but the other two systems confirmed that the task was challenging. Our system performed well, achieving the highest scores on both the Bank Marketing and the provided datasets (that was a positive surprise). This demonstrates that our model portfolio and ensembling strategy are effective compared to professional AutoML tools. However, it is important to note that in MLJAR using ROC-AUC during training may have influenced its performance. To draw more reliable conclusions, testing on more datasets should be conducted.

Table 3 presents the total execution time, the number of samples and final model in each dataset. The system remains fast even when processing larger datasets, such as Airline, completing the entire AutoML pipeline in under five minutes. As observed, the system selected a variety of final solutions, ranging from standalone tuned models to ensembles. This diversity in selection confirms that implementing five different ensembling architectures was justified.

Table 3: System total execution time, the number of samples and final model.

Dataset	**Instances**	**Execution Time**	**Final Model**
Pima Indians	768	0:34 min	LogisticRegression_16
Bank Marketing	11,162	2:40 min	Stacking_RF
Provided Dataset	3,481	1:20 min	Stacking_Top3_Passthrough
Airline Passenger Satisfaction	129,880	4:30 min	Voting_Top5

4 Conclusion

In summary, we successfully created a simple AutoML system capable of performing binary classification on any given tabular dataset using a model portfolio. MiniAutoML proved to be a fast and effective solution. Our experimental results show that diverse model selection techniques (including standalone tuned models, voting, and various stacking architectures) combined with our portfolio of algorithms have a significant impact on final performance.