Animal Shelter Adoption Prediction

1. Problem Definition

Every year, a large number of stray animals are admitted to animal shelters across the country. Despite ongoing efforts to increase adoption rates, many animals are not adopted due to the shelters' limited resources. These unadopted animals are often euthanized or transferred to other facilities. This project aims to predict the likelihood of animal adoption in advance, enabling shelters to operate more strategically and intervene earlier when necessary.

This is a binary classification problem, where a machine learning model is used to predict whether an animal will be adopted. The target variable is Adopted, which is labeled as 1 for adopted animals and 0 for those that are not. The input data includes various features such as age, animal type, breed, color, sex upon outcome, and other relevant characteristics.

- **Learning Type:** Supervised Learning
- **Prediction Task:** Binary Classification
- **Target Variable:** Adopted (1: Adopted, 0: Not Adopted)
- **Input Features:** Age, Animal Type, Breed, Color, Sex upon Outcome, Intake Type, Time in Shelter, etc.
- **Output:** Adoption probability (P(adopted = 1)), with final classification based on a threshold

Since the proportion of adopted animals is relatively low compared to the total, this dataset suffers from class imbalance. Therefore, instead of relying solely on accuracy, this project uses F1-score as the primary performance metric to better balance precision and recall in the evaluation of predictions.

2. AutoML Tool Description

This project employed the MLJAR-Supervised AutoML framework to automate the training, tuning, and evaluation of multiple machine learning algorithms. MLJAR is a robust and user-friendly tool that streamlines the entire machine learning workflow—including data preprocessing, model selection, hyperparameter optimization, ensembling, performance evaluation, and report generation—all with minimal code. By automating repetitive experimentation processes, MLJAR significantly enhances efficiency and allows researchers to focus on high-level analysis and interpretation.

MLJAR offers several notable features:

- **Automated algorithm exploration and comparison**, covering a wide range of models such as Decision Tree, Random Forest, XGBoost, and CatBoost.
- **Automatic hyperparameter tuning** to optimize model performance without manual intervention.
- **Feature engineering capabilities**, including the generation of Golden Features (derived variables), KMeans-based clustering features, and advanced ensembling and stacking techniques.
- **Automatic generation of visual outputs**, such as performance summaries, training time comparisons, and confusion matrices for each model.

For this study, the framework was configured with mode="Compete" and eval_metric="f1" to focus on selecting models that perform best based on the F1-score. To ensure clarity and efficiency in comparison, the experiment was restricted to four representative algorithms: Decision Tree, Random Forest, XGBoost and CatBoost.

3. Development Environment

This project was developed and executed in a Python 3.8.20 conda virtual environment on macOS 14.4.1. The primary tools and libraries used in the development process are listed below:

- Operating System: macOS 14.4.1 (Apple Silicon M4)
- Python Version: 3.8.20 (Conda environment)
- Jupyter Notebook: Used for experiment tracking, code execution, and data visualization

Key libraries and their versions:

- pandas (v2.0.3) – Data handling and preprocessing
- scikit-learn (v1.3.2) – Model training and evaluation
- mljar-supervised (v1.1.6) – AutoML framework for automated modeling
- xgboost, catboost, lightgbm – Gradient boosting models
- matplotlib, seaborn – Visualization and EDA tools

Model training and analysis were conducted within the Jupyter Notebook environment, leveraging MLJAR's built-in reporting features to compare model performance and interpret results. Additionally, MLJAR's automatic feature engineering and hyperparameter tuning functionalities were actively utilized throughout the experimentation process.

4. Results and Interpretation

4-1. AutoML-Based Model Performance Overview and Interpretation

In this study, the performance of five model types—Decision Tree, Random Forest, XGBoost, CatBoost, and Ensemble—was compared using the MLJAR AutoML framework. Each model was trained under identical experimental conditions on the same dataset, and performance evaluation was conducted primarily based on cross-validated F1-scores.

Model Type	Best F1-score	Training Time(s)
Ensemble (Best)	0.8841	6.3s
CatBoost	0.8806	13s~21s
XGBoost	0.8691	9.3s
Random Forest	0.8513	6.0s
Decision Tree	0.8101	4.5~5.0s

Table 1. Performance Summary of AutoML-Generated Models : *Comparison of F1-score and training time across selected models generated by MLJAR AutoML.*

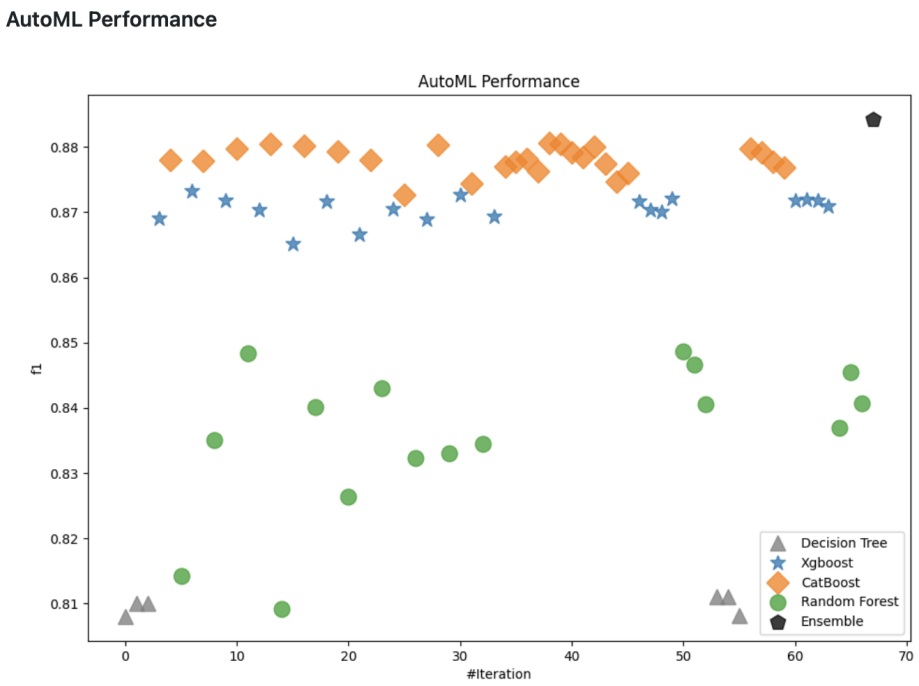

Figure 1. F1-score distribution of models generated by AutoML This plot visualizes the F1-scores of various models generated during the AutoML iterations. CatBoost models (orange diamonds) consistently achieved the highest scores, followed by XGBoost models (blue stars) with competitive performance. Random Forest (green circles) and Decision Tree models (gray triangles) showed relatively lower performance. The final Ensemble model (black pentagon), located at the top-right, achieved the highest overall F1-score, confirming its position as the best-performing model selected by AutoML

The **Ensemble model**, which achieved the **highest F1-score of 0.8841**, is a notable example of improving generalization performance by reducing overfitting through the integration of individual model predictions. In particular, MLJAR's automated stacking technique combined high-performing models such as CatBoost and XGBoost in a complementary manner, leveraging their differing predictive biases and feature-handling strategies. This ensemble structure proved highly effective in capturing various characteristics of the dataset and exhibited strong performance even under the challenge of class imbalance.

The **CatBoost model** demonstrated consistent F1-scores above 0.88, thanks to its inherent ability to handle categorical features without the need for explicit encoding. When combined with KMeans-based derived features generated by MLJAR, CatBoost showed robust performance across different feature interactions and class imbalance scenarios. These results highlight CatBoost as a reliable and practical choice for real-world deployment, especially in imbalanced classification tasks.

XGBoost showed strong baseline performance without tuning, but its limited handling of categorical features led to a slightly lower F1-score (0.8691) than CatBoost. Random Forest offered stable predictions but lacked the ability to model complex feature interactions, resulting in a lower F1-score (0.8513). Decision Tree, while interpretable and fast, had the lowest F1-score (0.8101) due to its inability to capture complex patterns.

Among these, CatBoost and the Ensemble model stood out for their robustness in dealing with class imbalance and high applicability in real-world scenarios. Most importantly, the fact that these optimal model structures were automatically identified through an AutoML framework—without any manual tuning—carries significant meaning. High-performing models were discovered without repetitive

experimentation or subjective intervention, thereby enhancing the reproducibility, efficiency, and objectivity of the modeling process.

4-2. Performance Comparison: Assignment 1 Model vs. MLJAR Best Model

This section compares the manually developed model from Assignment 1 with the best-performing model automatically selected by MLJAR AutoML (hereafter referred to as the "MLJAR Best Model"). Evaluation metrics include Accuracy, Precision, Recall, and F1-score, with both models evaluated on the same test dataset.

For fairness in evaluation, the performance of both models was assessed using the same train/test split initially defined in Assignment 1. As a result, the F1-score of the MLJAR Best Model presented here may differ slightly from previously reported values due to consistent dataset partitioning.

(1) Performance Comparison

	Best Model in MLJAR	Model in Assignment1
Accuracy	0.8850	0.8853
Precision	0.8401	0.8480
Recall	0.8983	0.8869
F1-score	0.8682	0.8670

Table 2. Comparison Best Model in MLJAR and Model in Assignment1

The MLJAR Best Model showed a slight advantage in F1-score and notably outperformed the Assignment 1 model in Recall. This indicates that the MLJAR model was better at correctly identifying adopted animals, which is a crucial metric in the context of adoption targeting. On the other hand, the Assignment 1 model achieved higher Precision, suggesting it was more effective at minimizing false positives—cases where non-adoptable animals were incorrectly predicted as adoptable.

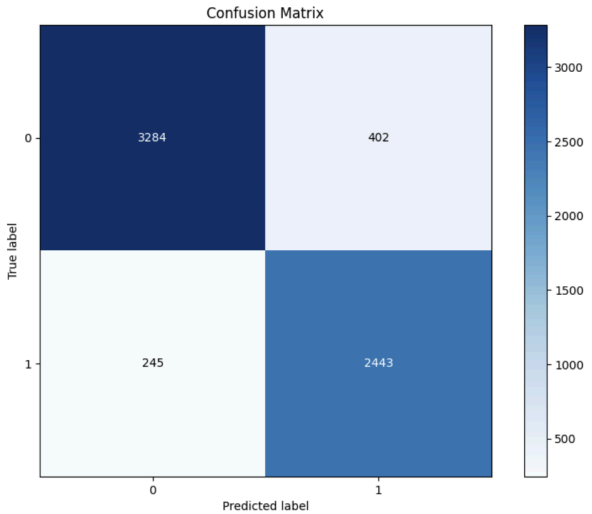

Figure 2. Confusion Matrix of MLJAR Best Model

Additionally, the confusion matrix of the MLJAR Best Model (an Ensemble model) highlights its strong performance in predicting Class 1 (Adopted), with a particularly high Recall of 0.8983. However, the presence of some false positives led to a slight reduction in Precision. This indicates the model is proactive in identifying adoptable animals but may require adjustments to reduce false positives in real-world deployment.

(2) Interpretation and Conclusion

While both models demonstrated similar overall accuracy, they each had distinct strengths. The MLJAR Best Model excelled in Recall and F1-score, making it advantageous for systems focused on promoting adoption. In contrast, the Assignment 1 model's higher Precision could be more beneficial in reducing resource waste at shelters.

Notably, MLJAR achieved this level of performance without any manual feature engineering or hyperparameter tuning, thereby demonstrating the practical efficiency of AutoML. The fact that its performance was comparable to or better than the manually crafted Assignment 1 model supports the viability and potential of AutoML in similar classification tasks.

5. Discussion of the Advantages and Limitations of AutoML

In this study, we utilized MLJAR AutoML to automatically train and evaluate a variety of models, ultimately selecting the most optimal one. Through this process, we identified several strengths and limitations of using AutoML.

The most notable advantage of AutoML is its ability to save time and resources. By automating complex tasks such as model selection, hyperparameter tuning, and feature engineering, high-performance models can be developed with minimal manual effort. Experiments showed that AutoML-generated models often achieved higher F1-scores compared to manually crafted models, indicating its effectiveness in performance optimization. Additionally, results were automatically organized into tables and visualizations, enhancing experiment management and reproducibility. AutoML is particularly valuable in that it allows even non-experts to engage in machine learning.

However, there are also limitations. The automatically generated models tend to have complex internal structures, making them difficult to interpret—an important drawback in practical applications where explainability is essential. Furthermore, users have limited control over algorithm selection and feature handling, and constraints related to real-world deployment (e.g., model size, inference speed) are not always accounted for. Even if performance metrics appear strong during training, there is still a risk of overfitting when evaluated on separate test data.

In conclusion, AutoML is a powerful tool for fast and efficient model development, but its interpretability and flexibility are still limited compared to manual modeling. While highly effective in the exploratory and prototyping stages, careful human judgment remains necessary for deployment in real-world applications.

6. Remote Source Repository

: https://github.com/llishyun/MLops_Assignment2.git