Hacettepe University

Artificial Intelligence Engineering Department

AIN 421 Fuzzy Logic - 2024 Fall

Project 2 – Extending an AutoML Platform with a Mamdani-Type FIS

December 10, 2024

Student Name: Süleyman Yolcu

Student Number: 2210765016

1. Introduction: What is AutoML and Why is it Necessary?

Automated Machine Learning (AutoML) refers to the set of methodologies, tools, and frameworks that automate the end-to-end process of developing machine learning (ML) models. Traditional ML workflows often require significant human intervention, including the selection of appropriate algorithms, hyperparameter tuning, feature engineering, and model evaluation. As the complexity and volume of data have grown, so too has the need for efficient, reliable, and repeatable ML development processes.

The necessity of AutoML becomes clear when considering that not every professional applying machine learning is an expert in the field. Constructing, optimizing, and updating models by hand demands substantial effort, from carefully selecting model architectures and tuning hyperparameters to repeatedly validating performance. This is a time-consuming process that can be especially challenging for teams that need to regularly incorporate new data and maintain their models over time. AutoML solutions address these issues by automating key steps and using systematic search techniques—such as Bayesian optimization or evolutionary algorithms—to discover effective model configurations. As a result, even non-specialists can benefit from advanced machine learning capabilities, reducing human bias, speeding up experimentation, and frequently achieving performance levels rivaling or surpassing those of manually tuned models deep background in data science.

2. Three AutoML Solutions: Sources, Developers, Capabilities, and ML Workflow Execution

(a) auto-sklearn

- **Source and Developers:** auto-sklearn is an open-source Python package built on top of the scikit-learn ecosystem. It is primarily developed and maintained by researchers at the University of Freiburg.
- **Capabilities:** auto-sklearn automates model selection and hyperparameter tuning over a wide range of scikit-learn compatible estimators. It leverages Bayesian optimization and meta-learning to efficiently explore the model space. Users can define the time and memory budget, and the tool returns an optimized pipeline with preprocessing steps, chosen algorithms, and tuned parameters.
- **ML Process Execution:** Using auto-sklearn typically involves Python code within a Jupyter Notebook or a Python script. The general workflow includes: loading the dataset, specifying time constraints, fitting auto-sklearn with training data, and retrieving the best pipeline after the search concludes. The user interaction is programmatic and code-driven, with limited graphical user interface options.

(b) MATLAB’s Fuzzy Logic Toolbox

- **Source and Developers:** This toolbox is developed and maintained by MathWorks, the company behind MATLAB. It is a commercial product integrated into the MATLAB environment.

- **Capabilities:** The Fuzzy Logic Toolbox provides an environment to design, analyze, and simulate fuzzy inference systems (FIS). It supports membership function design, fuzzy rule editing, inference methods, and defuzzification techniques. However, it does not directly offer the standard machine learning pipeline components such as automatic feature selection, hyperparameter tuning of classical ML algorithms, or the same level of automation for supervised tasks like classification and regression. Its core value lies in enabling fuzzy reasoning rather than end-to-end ML.
- **ML Process Execution:** The user typically works within MATLAB's graphical interfaces or command-line environment. While it excels at designing and simulating fuzzy systems, the lack of automated traditional ML workflows (e.g., model selection, preprocessing, and hyperparameter tuning across multiple supervised learning models) makes it less suitable as a general-purpose AutoML tool.

(c) PyCaret

- **Source and Developers:** PyCaret is an open-source, low-code machine learning library in Python developed by the open-source community and backed by a strong contributor base. It is frequently updated and well-documented.
- **Capabilities:** PyCaret is designed as a comprehensive AutoML suite that simplifies the entire ML pipeline. It can handle data preprocessing (e.g., missing value imputation, encoding, normalization), feature engineering, model selection from a broad set of algorithms (including tree-based methods, linear models, and ensemble techniques), automatic hyperparameter tuning, and model evaluation. PyCaret supports multiple ML tasks such as classification, regression, clustering, and anomaly detection.
- **ML Process Execution:** PyCaret's workflow usually involves Python code execution within a Jupyter Notebook environment. A typical process starts with importing PyCaret, choosing a ML module (e.g., classification), loading data, and calling functions like `setup()` which automatically preprocesses data. Next, the `compare_models()` function compares multiple algorithms and returns a best-performing model. Finally, one can fine-tune the chosen model and deploy it. PyCaret abstracts away much of the complexity, requiring only a few lines of code to complete the end-to-end pipeline.

3. Selected Platform and Extension Design

Reason for Choosing PyCaret: Among the three examined solutions, PyCaret emerges as the most suitable tool for an integrated AutoML experience. Unlike MATLAB's Fuzzy Logic Toolbox, which specializes in fuzzy inference systems but does not offer a standard, automated ML pipeline for tasks like classification and regression, PyCaret provides a broad range of features including preprocessing, model selection, hyperparameter tuning, and model evaluation all in one place. Compared to auto-sklearn, PyCaret provides a more user-friendly, low-code interface, and is highly extensible with clear APIs. Its active community and compatibility with standard Python ecosystems make it a strong choice for both prototyping and proof-of-concept extensions.

Extension Design Overview: The proposed extension to PyCaret involves integrating a Mamdani-type Fuzzy Inference System (FIS) into its workflow at a proof-of-concept level. Currently, PyCaret focuses on standard ML algorithms. The goal is to illustrate how fuzzy reasoning can complement or augment model interpretation or decision-making. In practice, this could involve using a fuzzy layer to handle uncertain or imprecise input data, or to provide interpretable rules associated with model predictions.

Extension Integration Plan:

1. **Integration Layer:** Create a small module within PyCaret's internal structure (or as an external add-on) that can be called after a model is trained. This module will integrate fuzzy rules or membership functions into the decision-making process.
2. **Data Input and Triangular Membership Functions:**
   - Users (or the extension pipeline) will specify linguistic variables and their associated triangular membership functions. For instance, if dealing with a classification problem, input features can be represented as fuzzy sets (e.g., "Low," "Medium," "High" for a numerical feature).
   - Implement a simple syntax or configuration dictionary that maps numerical ranges to triangular membership functions.
3. **Fuzzy Rule Base and Mamdani Inference:**
   - A minimal set of rules will be hardcoded or defined by the user. For example, "If feature_x is High and feature_y is Low, then output is Class A."
   - The inference engine will apply these rules to the input data using Mamdani fuzzy inference, producing fuzzy output sets.
4. **Defuzzification (COA):**
   - Implement the Center of Area method to convert the aggregated fuzzy output set into a crisp value. For classification tasks, this could be aligned with selecting the class whose membership is highest after defuzzification, or for regression tasks, providing a numeric prediction directly.
5. **Output and Model Interpretation:**
   - The results of the fuzzy inference can be compared to those of the standard PyCaret model pipeline. Although this is a PoC, the final goal is to show that PyCaret can be extended to incorporate fuzzy logic steps to provide interpretable logic-based decisions alongside or in place of standard ML model predictions.

4. Conclusion: AutoML platforms are invaluable in accelerating ML development and increasing accessibility for non-experts. After evaluating three solutions—auto-sklearn, MATLAB's Fuzzy Logic Toolbox, and PyCaret—PyCaret was chosen due to its integrated ML pipeline capabilities, ease of use, and flexibility. The proposed enhancement involves adding a Mamdani-type FIS with triangular membership functions and COA defuzzification at a proof-of-concept level. This work aims to demonstrate how fuzzy logic can be integrated into an existing AutoML framework, potentially offering more interpretable and flexible decision-making mechanisms in future implementations.

References

1. **Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2015).** Auto-sklearn: Efficient and robust automated machine learning. *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*. University of Freiburg, Germany. Available at: https://github.com/automl/auto-sklearn.
2. **MathWorks. (2024).** MATLAB Fuzzy Logic Toolbox Documentation. MathWorks. Available at: https://www.mathworks.com/products/fuzzy-logic.html
3. **Ali, M., Jabeen, F., & Contributors. (2024).** PyCaret: An Open-source, Low-code Machine Learning Library. GitHub Repository. Available at: https://github.com/pycaret/pycaret.