Refactoring Machine Learning Workflow with Snakemake

PROGRAMMING 6

Romana Mahjabin Eshita

1 Introduction

Snakemake is a Python language-based human-readable workflow management tool that is used for reproducible and scalable data analysis [2]. Without modifying the workflow definitions, it can be easily scalable to server, cluster, grid, and cloud environments. For this assignment, I have chosen an existing machine learning project to explore different machine learning models to evaluate its performance. This report summarizes the refactoring process, design choices, and results of implementing a machine learning workflow using Snakemake. The workflow automates a complete ML pipeline from data retrieval through preprocessing, model training, evaluation, and reporting.

2 ML Application

The project aims to predict the histology of lung cancer using clinical and gene expression data. The dataset [1] consists of 89 samples, each containing both clinical and gene expression data. The clinical metadata includes 16 variables, while the gene expression dataset contains 66,000 features, making it highly dimensional. The target variable is categorical; that's why we selected Random Forest and multi-class logistic regression to assess the performances of these models.

3 Refactoring Process

Initially, the solution was implemented in a Jupyter Notebook rather than a script. The first step in refactoring was to convert it into a script-based solution that sequentially handled data processing, model training, and performance evaluation. To improve structure and maintainability, the pipeline was further broken down into modular Snakemake rules, with each rule managing a specific stage of the workflow.

- **Data Retrieval:** Extracts and combines datasets from raw sources, then splits them into training and testing sets.
- **Data Preprocessing:** Transforms the input data using feature processing techniques to make it suitable for model training.
- **Model Training:** Implements parallelized training of multiple models.
- **Model Evaluation:** Computes performance metrics for each trained model.
- **Result Aggregation:** Combines evaluation metrics into a summary table.
- **Benchmark Analysis:** Analyzes the computational efficiency of model training and evaluation.
- **Visualization:** Generates plots for comparing model performance.
- **Report:** Produces a final HTML report containing detailed information about the models, their performance, and comparison plots. The HTML format was chosen because it allows easy integration of plot images for better visualization and analysis.

Directory Structure

To keep the project organized, we set up a clear directory structure:

- **raw_data:** Stores the original datasets before processing.
- **transformed_data:** Contains cleaned and processed data ready for modeling.
- **models:** Saves trained models and preprocessing steps.

- **results:** Stores model performance scores and evaluation results.
- **plots:** Holds graphs and visualizations comparing model performance.
- **logs:** Keeps track of workflow execution details.
- **benchmarks:** Records system performance and resource usage.
- **reports:** Stores the final HTML report with all results and insights.
- **scripts:** Contains Python scripts for each step of the workflow.

4 Design Choices

The workflow was built specifically for SnakeMake, so all scripts follow its input-output system. This makes it easy for SnakeMake to handle dependencies and run tasks in the right order. Research emphasized that workflows should be scalable, portable, and reproducible [2]. To match this, we used a modular script setup, a clear directory structure, and parallel execution where possible. The entire workflow was built based on the Snakemake documentation [3]. In particular, the use of wildcards for flexibility and benchmarking for performance tracking made the workflow much easier to manage. Wildcards enabled dynamic rule generation, making it simple to add new models without modifying the core workflow. Benchmarking helped track computational efficiency, ensuring the workflow remained optimized. Key design choices include:

- **Configuration-driven Workflow:** No hard-coded path or any kind of setting was in the code. A configuration file ('config.yaml') is used to manage paths, model settings, and data sources, so it becomes easy to update the parameters without changing the code.
- **Modular Script Structure:** Each rule (data retrieval, preprocessing, training, evaluation, etc.) has a dedicated script; hence, it becomes easier for debugging and testing. Dependencies were explicitly defined to ensure smooth execution. Model evaluation metrics were saved for future visualizations. The trained models, along with feature processing pipelines, were stored for reuse on new datasets. The encoded target labels were also saved to maintain consistency in future applications.
- **Logging:** A centralized log file was implemented with timestamps so that we can track log details of all steps in one file, along with their error reports.
- **Benchmarking**: Performance benchmarking was implemented during model training and evaluation to track the model's computational performance. This allows easy comparison of models based on resource usage.
- **Wildcards for Model Flexibility:** Snakemake wildcards were used to dynamically select the model based on the configuration and simultaneously execute the model training and evaluation, allowing easy addition of new models.

5 Result Summary:

The implemented workflow successfully automates the entire machine learning pipeline from raw data to final report. The following summarizes key outcomes and findings:

- Table 1 presents the evaluation metrics for the trained models. The results indicate that Random Forest performed better in terms of accuracy and execution time. The Snakemake automation significantly reduced manual intervention and improved workflow efficiency.

Model Name	Accuracy	F1 Score	Total Execution Run Time (s)
RandomForest	0.869	0.869	2.08
MulticlassLogisticsRegression	0.826	0.826	4.33

Table 1: Quantitative Performance Metrics

- Parallel execution of model training and evaluation reduced overall execution time.
- Benchmarking allowed for a detailed runtime analysis of each stage, helping identify computational bottlenecks.
- The workflow also generated an HTML report summarizing model performance with visualizations, making it easy to interpret and compare results.

6 Conclusion:

This refactored workflow demonstrated the advantages of using Snakemake for automating machine learning pipelines. The modular script structure improved maintainability, making it easier to debug and extend the workflow. Parallel execution optimized runtime, reducing the overall execution time for model training and evaluation. Benchmarking provided insights into computational performance, while the configuration-driven approach allowed for easy modifications without altering the code. Future improvements could include dynamic resource allocation and integration with cloud computing platforms for scalable execution.

References

[1] Hugo J. W. L. Aerts, Emmanuel Rios Velazquez, Ralph T. H. Leijenaar, Chintan Parmar, Patrick Grossmann, Sara Carvalho, Johan Bussink, René Monshouwer, Benjamin Haibe-Kains, Derek Rietveld, Frank Hoebers, Michelle M. Rietbergen, C. René Leemans, Andre Dekker, John Quackenbush, Robert J. Gillies, and Philippe Lambin. Data from nsclc-radiomics-genomics, 2015.

[2] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, August 2012.

[3] Johannes Köster. *Snakemake Workflow Management System*, 2024. Accessed: March 23, 2025.