Ministere de l'Enseignement Superieur et de la Recherche Scientifique

Exploring MLOps (MLFlow with a Movie Recommendation System

Software Engineering for Data Science Course Mini-Project

Instructor: Belkacem KHALDI

Team Members:
SAADI Ahmed
TOUMI Adem

Submission Date: Janury 23, 2025

Contents

Abstract

The objective of this mini-project is to explore MLOps principles and practices through a movie recommendation system using the Apriori algorithm. By integrating MLflow, an open-source MLOps platform, the system managed 60 experimental runs with varying hyperparameters, achieving a maximum lift score of 53.2875 and generating over 427,000 association rules. The project highlights the role of MLOps in enhancing reproducibility, scalability, and efficiency in machine learning workflows.

1 Introduction

1.1 Defining MLOps

Machine Learning Operations (MLOps) is an evolving discipline that bridges the gap between machine learning model development and production deployment. It integrates principles from machine learning, DevOps, and data engineering to address challenges such as model versioning, automated deployment, performance monitoring, and scalability.

1.2 Project Objectives

This project aims to:

- Implement a movie recommendation system using the Apriori algorithm.
- Integrate MLflow for experiment tracking, model management, and deployment.
- Demonstrate the benefits of MLOps in improving reproducibility and efficiency.

2 MLOps Architecture and Platform

2.1 Chosen Architecture: MLflow

MLflow was selected as the MLOps platform for this project due to its lightweight design, flexibility, and comprehensive support for experiment tracking, model management, and deployment. The architecture of MLflow is modular and framework-agnostic, making it suitable for integrating with the Apriori algorithm and the MovieLens dataset.

Core Components of MLflow

- **MLflow Tracking**:
  - Logged hyperparameters (e.g., `min_support`, `min_threshold`) and metrics (e.g., lift score) for all 60 experimental runs.
  - Enabled real-time monitoring of experiments through the MLflow UI.
- **MLflow Projects**:
  - Packaged the Apriori algorithm implementation into a reproducible format.
  - Managed dependencies using Conda environments to ensure consistency across runs.
- **MLflow Models**:
  - Stored trained models in a standardized format for easy deployment.
  - Supported versioning to track changes in model performance over time.

- **MLflow Model Registry**:
  - Centralized management of model versions and lifecycle stages (e.g., staging, production).
  - Facilitated collaboration between team members by providing a shared repository for models.

2.2 Implementation of MLflow in the Project

The integration of MLflow into the movie recommendation system involved the following steps:

- **Experiment Tracking**:
  - Logged hyperparameters, metrics, and artifacts for each run.
  - Used the MLflow UI to compare results and identify the best-performing configurations.
- **Model Packaging**:
  - Packaged the Apriori algorithm as an MLflow Project for reproducibility.
  - Managed dependencies using Conda to ensure consistent execution across environments.
- **Model Deployment**:
  - Deployed the best-performing model as a REST API using MLflow's serving capabilities.
  - Containerized the model using Docker for portability and scalability.

2.3 Architectural Diagram

2.4 Findings and Insights

- **Strengths of MLflow**:
  - Simplified experiment tracking and reproducibility.
  - Provided a centralized platform for managing models and artifacts.
  - Enabled seamless deployment of models as REST APIs.
- **Limitations of MLflow**:
  - Required manual configuration for custom logging and artifact storage.
  - Limited support for advanced deployment scenarios (e.g., Kubernetes orchestration).
- **Comparison with Other Platforms**:
  - Compared to Kubeflow, MLflow is easier to set up and use for small to medium-scale projects.
  - Unlike SageMaker, MLflow is open-source and does not require cloud vendor lock-in.

3 Methodology

3.1 Dataset

The MovieLens dataset was chosen for its richness in movie ratings and user interactions. It includes user IDs, movie IDs, ratings, and genres, which were processed to create transaction-based data suitable for the Apriori algorithm.

3.2 Preprocessing

Key preprocessing steps included:

- Filtering movies with insufficient ratings to ensure quality data.
- Transforming the dataset into transactions where each transaction represents a user's preferred movies.
- Normalizing and encoding data for compatibility with the Apriori algorithm.

3.3 Apriori Algorithm Implementation

The Apriori algorithm, implemented using the `mlxtend` Python library, generates frequent itemsets and association rules. Hyperparameters such as `min_support`, `min_threshold`, and `metric` were tuned to optimize performance.

4 MLflow Integration

4.1 Experiment Tracking

MLflow's Tracking feature was utilized to log and monitor experiments. For each of the 60 runs:

- **Hyperparameters** such as `min_support`, `metric`, and `min_threshold` were logged.
- **Metrics** including lift scores and the number of generated rules were recorded.
- **Artifacts** like models and outputs were saved for future reference.

4.2 MLflow Dashboard

The MLflow UI enabled visualization of results, facilitating comparison across runs. Figure 1 showcases the experiment dashboard, highlighting key metrics and configurations.

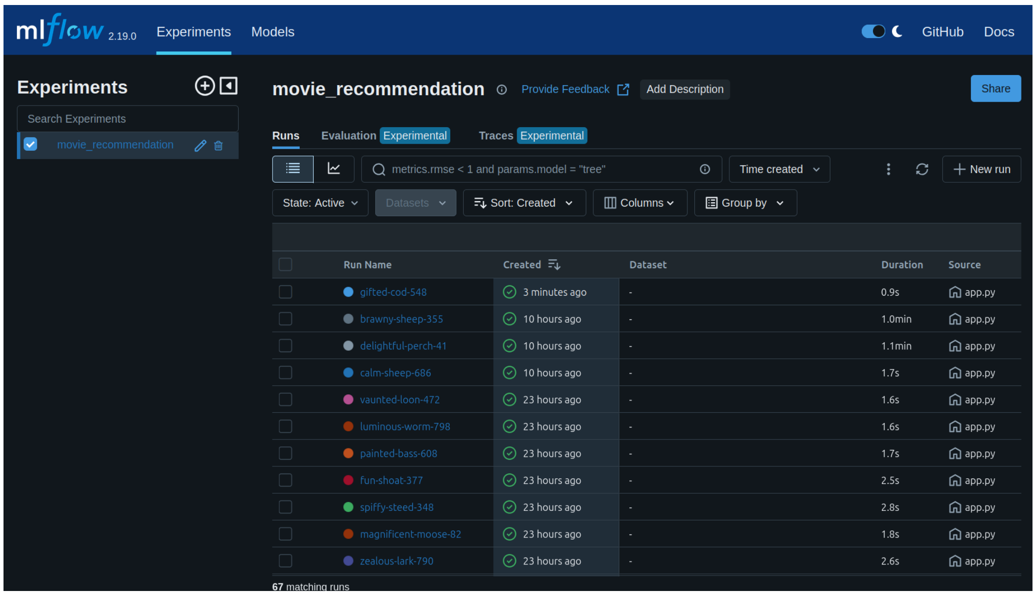
5 Experimentation and Results

5.1 Hyperparameter Tuning

Sixty experiments were conducted with varying `min_support`, `min_threshold`, and `metric`. Table 1 summarizes the best-performing runs.

5.2 Analysis of Results

Lower `min_support` values resulted in more rules with higher diversity, while higher values yielded fewer but stronger rules. The best configuration achieved a lift score of 53.2875, validating the importance of fine-tuned hyperparameters.



movie_recommendation

metrics.rmse < 1 and params.model = "tree"

Run Name	Created	Dataset	Duration	Source
gifted-cod-548	3 minutes ago	-	0.9s	app.py
brawny-sheep-355	10 hours ago	-	1.0min	app.py
delightful-perch-41	10 hours ago	-	1.1min	app.py
calm-sheep-686	10 hours ago	-	1.7s	app.py
vaunted-loon-472	23 hours ago	-	1.6s	app.py
luminous-worm-798	23 hours ago	-	1.6s	app.py
painted-bass-608	23 hours ago	-	1.7s	app.py
fun-shoat-377	23 hours ago	-	2.5s	app.py
spiffy-steed-348	23 hours ago	-	2.8s	app.py
magnificent-moose-82	23 hours ago	-	1.8s	app.py
zealous-lark-790	23 hours ago	-	2.6s	app.py

67 matching runs

Figure 1: MLflow Experiment Dashboard

Run ID	min_support	Metric	Lift	Number of Rules
b7c947c8b5b6439fab271e3f296e739f	0.01	Lift	53.2875	427,088

Table 1: Top Runs Summary

6 Discussion

6.1 Benefits of MLflow

- **Reproducibility**: Detailed logging ensured consistent experiment replication.
- **Efficiency**: Automated tracking reduced manual effort.
- **Visualization**: The MLflow UI simplified result comparison and analysis.
- **Collaboration**: Centralized tracking enhanced teamwork.

6.2 Challenges

- **Initial Setup**: Required careful environment configuration.
- **Custom Logging**: Needed tailored scripts for specific metrics and artifacts.

7 Conclusion

This project demonstrated the integration of MLflow as an MLOps platform in a movie recommendation system. By tracking experiments, automating workflows, and enabling detailed analysis, MLflow enhanced reproducibility and efficiency. The Apriori algorithm achieved optimal results with a lift score of 53.2875, showcasing the power of systematic experimentation.

8 Future Work

Potential directions include:

- Deploying the recommendation system as a production-grade application.
- Experimenting with larger datasets for improved generalization.
- Combining association rules with collaborative filtering for hybrid recommendations.
- Incorporating real-world user feedback for iterative refinement.

References

[1] MLflow Documentation, *MLflow*, `https://www.mlflow.org/docs/latest/index.html`.