MLOps Report

IMDb Sentiment Analysis

Vasco Gouveia, Matthias Nadal, Vasily Apasov, Iker Jansa, Martin Peretti

https://github.com/mlops-2526q1-mds-upc/MLOps_IMDb
October 13, 2025

Machine Learning Operations – Academic Year 2024/2025

Contents

1 Introduction

1.1 Goal of the Project

The project delivers an end-to-end analytics platform that ingests comment sections from major movie-review websites, filters noise such as spam or off-topic content, and extracts sentiment signals that can support downstream dashboards. Starting from the `cookiecutter-data-science` template, we built a reproducible pipeline that downloads the IMDb benchmark dataset, cleans and vectorises text, trains a logistic regression model, and evaluates it against held-out examples. The success criteria agreed during inception were:

- Produce sentiment metrics (accuracy, F1) above 0.85 on benchmark movie-review datasets.
- Reproduce the full data and model workflow via DVC with a single `dvc repro`.
- Track experiments (hyperparameters, metrics, artefacts, emissions) in MLflow to support iterative analysis of comment streams.

The latest evaluation run (see Section 2.2) satisfies these targets, with the logistic regression classifier reaching 0.894 accuracy and 0.895 F1.

2 Methodology

2.1 Milestone 1: Inception

2.1.1 Problem Selection and Requirements

We selected binary sentiment classification for IMDb reviews because it combines natural language processing with clear business value (gauging audience sentiment) and publicly available benchmark data. Functional requirements include batch scoring of text reviews and traceable metrics. Non-functional requirements emphasise reproducibility, energy-awareness, and compatibility with container deployment.

2.1.2 Dataset Card Summary

The dataset card (`data/README.md`) follows the Hugging Face template. It documents:

- **Provenance and Licence:** IMDb reviews under the IMDb Terms of Use, crowdsourced annotations, English-only content.
- **Composition:** 50,000 labelled reviews (25k train / 25k test), binary sentiment.
- **Motivation and Uses:** Benchmarking sentiment analysis models, educational demonstrations, not suitable for nuanced or multilingual sentiment.
- **Collection Process:** Neutral reviews excluded, balance enforced between polarities.
- **Limitations & Biases:** Domain specificity, possible demographic bias, binary simplification of sentiment.
- **Citation:** Maas et al. (2011) reference for reproducible acknowledgement.

2.1.3 Model Card Summary

The model card (`models/README.md`) captures:

- **Model Description:** TF–IDF features with a scikit-learn logistic regression classifier.
- **Intended Uses:** Baseline benchmarking and teaching reproducible MLOps practices; not for high-stakes production.
- **Training Procedure:** 25k train / 25k test split, TF–IDF with 1–2 grams, logistic regression (liblinear solver).
- **Evaluation:** Accuracy and F1 typically above 0.82, with current run at 0.894/0.895.
- **Limitations and Ethics:** English-only, binary polarity, sensitivity to sarcasm, potential demographic bias.

2.1.4 Project Coordination and Communication

We operate with a lightweight Scrum cadence: weekly planning and review ceremonies guide backlog priorities, while daily asynchronous stand-ups surface blockers. Slack hosts coordination, and its GitHub and Jira integrations post activity notifications and ticket updates into a shared channel, keeping sprint status visible despite distributed schedules.

2.1.5 Selection of Cloud Provider

For Milestone 4 we plan to deploy the inference service on Google Cloud Run. Its fully managed container runtime, integration with Artifact Registry, and generous free tier make it a pragmatic choice for lightweight academic workloads while providing a path to autoscaling if needed.

2.2 Milestone 2: Model Building – Reproducibility

2.2.1 Project Structure

The repository inherits the `cookiecutter-data-science` scaffold, separating code under `mlops_imdb/`, configuration in `params.yaml`, data artefacts under `data/`, and generated reports beneath `reports/`. This consistent layout eases onboarding and tooling integration.

2.2.2 Code Versioning

We apply GitHub Flow on top of the enforced branch naming convention (`feature/`, `fix/`, `chore/`, `docs/`, `experiment/`). Conventional Commit messages feed changelogs and are validated via PR automation (Section 2.4).

2.2.3 Exploratory Data Analysis and Baseline Definition

Prior to pipeline industrialisation, we performed an exploratory analysis in `notebooks/1.0-exploration-imdb` to validate the integrity of the IMDb dataset, define success thresholds, and freeze preprocessing decisions for later implementation.

Data Integrity. The dataset comprises 25,000 training and 25,000 test samples, each with two columns: `text` (string) and `label` (integer, $\{0, 1\}$). No missing values were detected, and only a small fraction of duplicate rows were found (96 in train, 199 in test). Duplicates were retained for baseline reproducibility. The split is balanced (50% positive / 50% negative), as illustrated in Figure 3, with an average text length of 1,325 characters and 7.36% long-text outliers (range: [2,983, 13,704]).

Baseline Modelling. Two baselines were evaluated:

- **Majority classifier:** always predicts the most frequent class, achieving 0.50 accuracy and 0.00 F1-score.
- **TF–IDF + Logistic Regression:** 10,000 features, 1–2-gram range, min_df=2, max_df=0.95, trained with `max_iter=1000`. The model reached 0.893 accuracy and 0.894 F1-score on the IMDb test set (Table 1).

Table 1: Baseline performance on the IMDb test set.

Model	Accuracy	Precision	Recall	F1-score
Majority	0.500	0.000	0.000	0.000
TF–IDF + Logistic Regression	0.893	0.888	0.899	0.894

The confusion matrix (Figure 1) highlights balanced misclassification rates across classes. The model serves as the reproducible baseline to be reimplemented in the DVC pipeline.

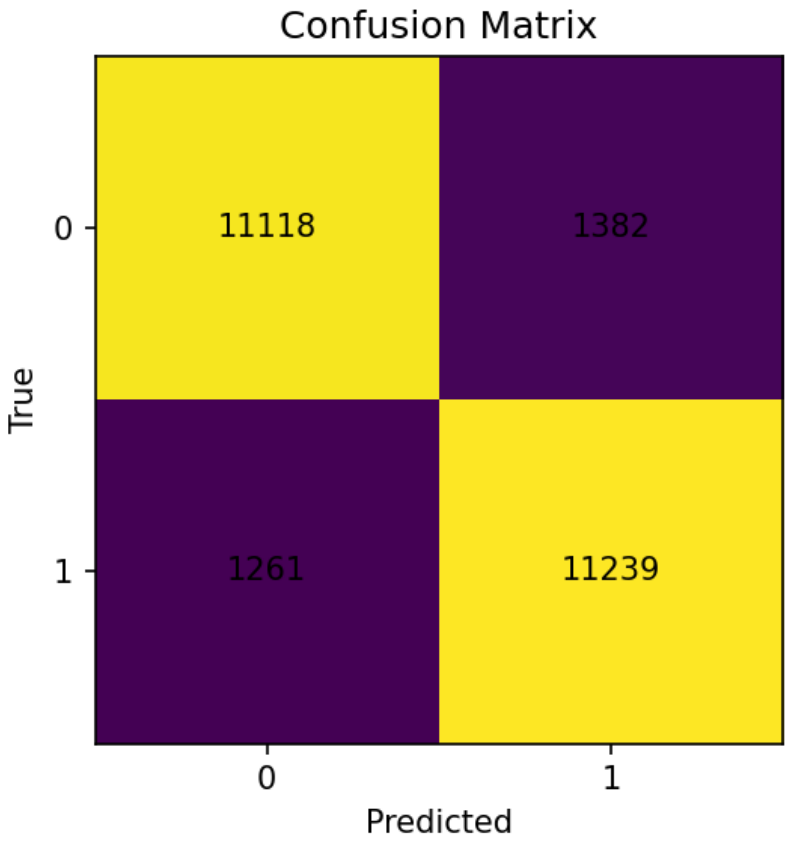

Figure 1: Confusion matrix for the baseline TF–IDF + Logistic Regression model.

2.2.4 Data Versioning

DVC stages (`prepare` → `features` → `train` → `eval`) encode dependencies and outputs. Regenerating the pipeline uses `dvc repro`, guaranteeing deterministic artefact creation (`data/processed`, `models/`, `reports/metrics.json`, figures).

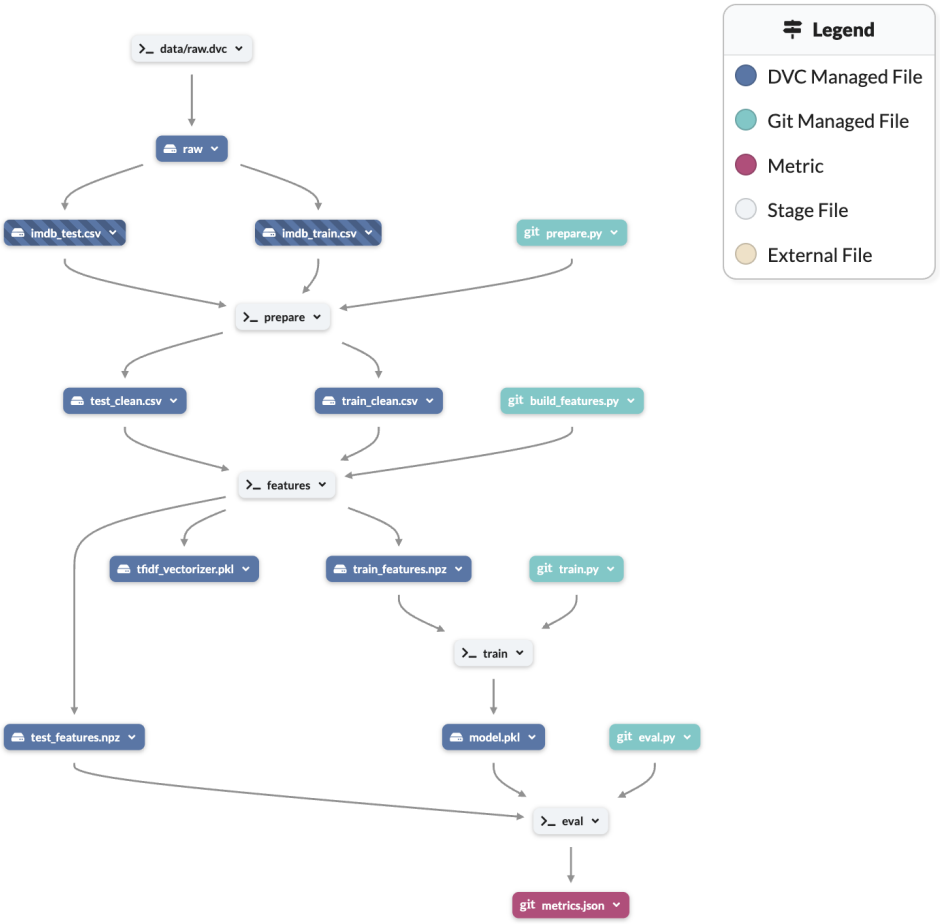

Figure 2: DVC pipeline stages from data preparation through evaluation.

2.2.5 DVC Pipeline and Reproducibility

Each processing step is versioned through DVC, ensuring full data and model reproducibility. The pipeline is defined in `dvc.yaml` with explicit dependencies, parameters, and outputs:

- **prepare:** cleans raw data and saves processed CSVs.
- **features:** builds TF–IDF matrices and serialises the vectoriser.
- **train:** fits the logistic regression model and exports the pickle artefact.
- **eval:** computes metrics, confusion matrix, and saves `reports/metrics.json`.

Each stage specifies inputs via `-d`, outputs via `-o`, and parameters via `-p` keys, allowing DVC to rebuild only the necessary stages when upstream data or configuration changes.

Running `dvc repro` triggers the entire pipeline from raw data to evaluation, producing identical artefacts across environments. The lock file (`dvc.lock`) captures the hashes of all dependencies, guaranteeing immutability and traceability of results.

2.2.6 Experiment Tracking

MLflow is configured through `mlops_imdb/config.py`. Both training and evaluation stages wrap their execution in `mlflow.start_run`, logging hyperparameters, dataset dimensions, metrics, confusion-matrix artefacts, and CodeCarbon emissions under the shared `emissions_kg` metric. Local runs default to `./mlruns`, while remote tracking URIs can be supplied via environment variables.

Frozen Decisions. From this notebook, the following parameters were locked for pipeline integration:

- Random seed = 42 for deterministic runs.
- Minimal preprocessing: lowercase, HTML tag removal, whitespace normalization.
- TF–IDF configuration as above; no lemmatization or stopword filtering.
- Logistic Regression hyperparameters: `max_iter=1000`, `random_state=42`.
- QA thresholds: accuracy ≥ 0.85, F1 ≥ 0.85.

2.3 Milestone 3: Model Building – Quality Assurance

2.3.1 Energy Efficiency Awareness

CodeCarbon contexts wrap every DVC stage. The run captured in `reports/codecarbon_emissions.csv` emitted 7.15 mg CO_2eq over 27.1 seconds (Table 2), with feature generation dominating the footprint. Emission figures are logged back to MLflow, enabling longitudinal monitoring.

Table 2: CodeCarbon emissions per stage (mg CO_2eq).

Stage	Duration (s)	Emissions (mg)
Prepare data	5.51	1.21
Build features	14.46	5.52
Train model	3.47	0.21
Evaluate model	3.63	0.22
Total	27.07	**7.15**

2.3.2 Static Code Analysis

Ruff enforces formatting and linting (line length 99, import sorting, code-style checks). The CI pipeline (Section 2.4) runs `uv run ruff format --check` and `uv run ruff check` on every push and pull request.

2.3.3 Model Testing

Model-level Pytests mirror the production workflow end to end. Synthetic sparse matrices exercise training, evaluation, and artefact persistence, yielding behavior-focused checks that cover roughly 88% of the critical paths while remaining hermetic and fast to run.

2.3.4 Data Testing

Data preparation tests (`tests/data/test_prepare.py`) exercise schema validation and text-cleaning behaviours, while `tests/data/test_download_imdb.py` mocks the dataset download to assure reproducible CSV outputs. Notebook `notebooks/1.0-exploration-imdb.ipynb` complements these checks by exploring label balance (Figure 3), text length distributions (Figure 4), and spam-like patterns; findings from the notebook inform assertions and thresholds in the automated tests. Although we do not yet employ Great Expectations, this combination of scripted checks and exploratory analysis catches regressions in preprocessing logic.

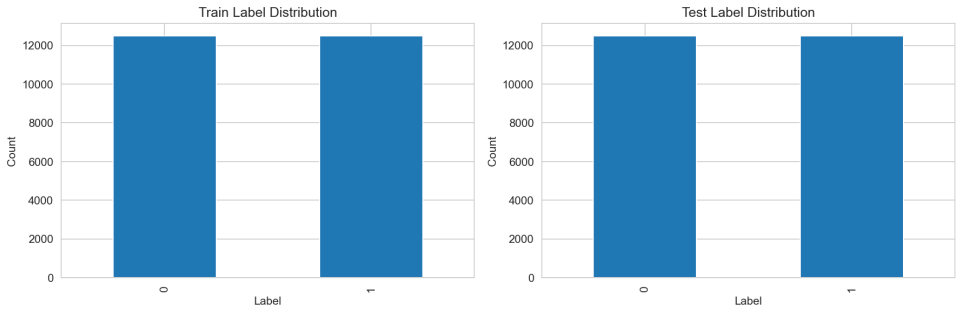

Figure 3: Sentiment label distribution across train and test splits (balanced 25k/25k).

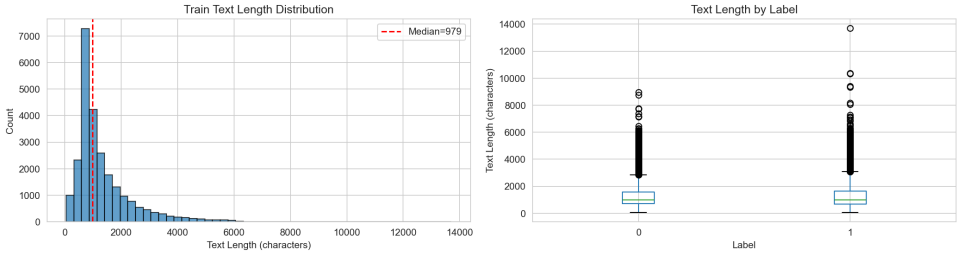

Figure 4: Distribution of cleaned review lengths, highlighting long-form tails and noisy spikes.

2.4 CI/CD Automation and Collaboration

Two GitHub Actions workflows underpin automation:

- `policy.yml` enforces collaboration rules by checking branch names, validating semantic PR titles (via `amannn/action-semantic-pull-request`), and preparing a uv toolchain for future commit-message linting.
- `ci.yml` installs Python 3.12, bootstraps dependencies with uv (using a cache keyed on `uv.lock`), and executes Ruff formatting, Ruff lint checks, and the Pytest suite on every push and pull request targeting `main`.
- Local git hooks powered by `.pre-commit-config.yaml` run Ruff before commits and validate Conventional Commit messages:
  - Subject line starts with one of `feat`, `fix`, `docs`, `chore`, `test`, `refactor`, `perf`, `build`, `ci`, `style`, or `revert`.
  - Subject length stays below 72 characters and is separated from the body by a blank line.
  - Body lines wrap below 80 characters with no trailing whitespace or hard tabs.

Integrated Workflow. The DVC pipeline acts as the orchestration backbone: each stage invokes MLflow logging and CodeCarbon tracking internally. This coupling ensures that every pipeline execution yields:

- reproducible artefacts and metrics (DVC),
- tracked experiments with metadata (MLflow),
- energy consumption records (CodeCarbon).

Together, these tools form a traceable, auditable MLOps stack ensuring both scientific and environmental reproducibility.

3 Future Work

3.1 Future Work

Milestone 4 will expose the sentiment model through a FastAPI service on Google Cloud Run, complete with authenticated endpoints and request/response validation. Milestone 5 focuses on packaging the service into Docker images, publishing them to Artifact Registry, and extending GitHub Actions to drive staging deployments and smoke tests. Milestone 6 emphasises observability and governed retraining by wiring Prometheus/Grafana dashboards, adding drift alerts, and scripting automated refreshes with DVC + MLflow triggers.

Beyond pipeline plumbing, we plan to introduce a lightweight spam-detection model that filters noisy comments before sentiment scoring, as well as an anonymisation layer that redacts personal identifiers prior to logging or storage.