# MLOps Report: IMDb Sentiment Analysis Pipeline

UPC MLOps Team

October 13, 2025

## 1 Introduction

This report summarises the development of an end-to-end machine learning (ML) component for IMDb film review sentiment classification. The objective is to deliver a reproducible pipeline that ingests raw reviews, trains a text classifier, and exposes evaluation artefacts suitable for integration into a broader ML-based product. The project embraces MLOps practices, including data versioning, configuration management, and automated evaluation.

## 2 Dataset Overview

The pipeline sources the public `imdb` dataset from Hugging Face via the `datasets` library (`mlops_imdb/data/download_imdb.py`). The dataset consists of 50,000 English-language movie reviews labelled as positive (1) or negative (0), already split into equally sized training and test partitions. After download, raw files are stored under `data/raw/` for reproducibility.

## 3 Exploratory Data Analysis

Preliminary exploration, performed in `notebooks/1.0-exploration-imdb.ipynb`, guides feature design and model assumptions. The balanced label split helps ensure unbiased optimisation (Figure 1), while the long-tailed review length distribution (Figure 2) motivated normalising whitespace and retaining bi-grams during TF–IDF feature extraction.

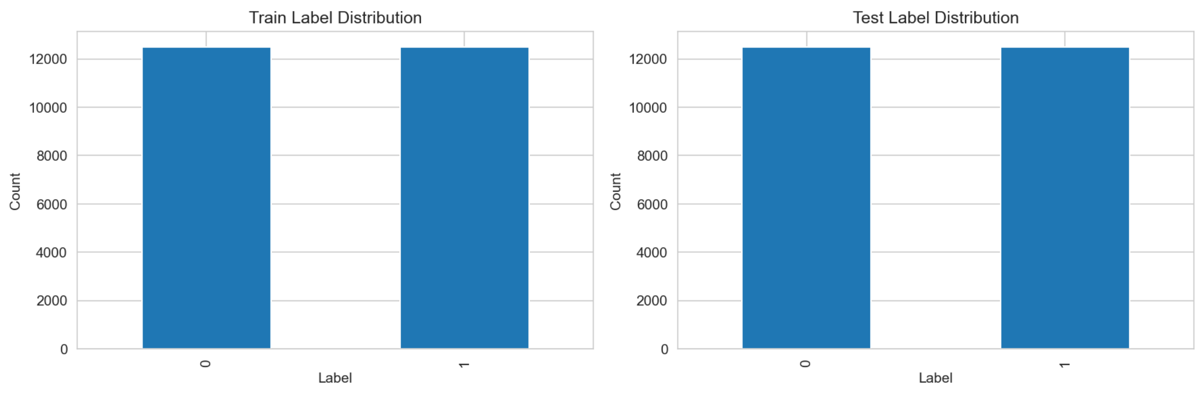

Figure 1: Sentiment label distribution in the IMDb dataset (train + test).

## 4 Pipeline Orchestration and Tooling

- **Pipeline management:** DVC orchestrates the stages defined in `dvc.yaml` (`prepare` → `features` → `train` → `eval`). Parameterisation is centralised in `params.yaml`, enabling

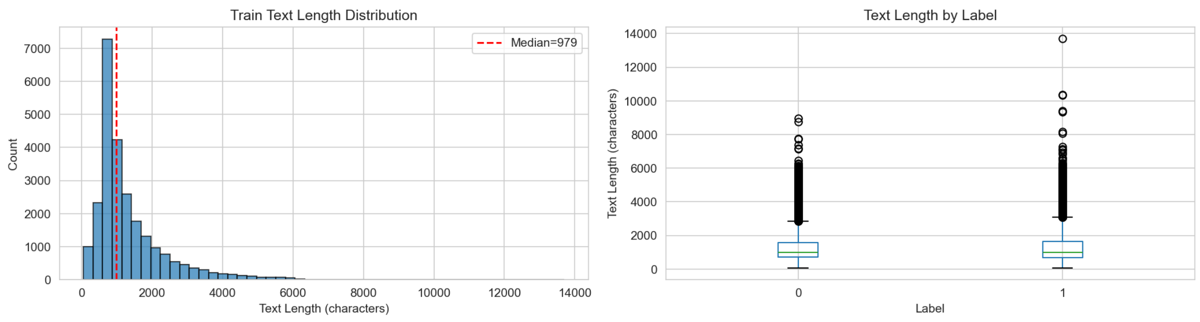

Figure 2: Distribution of tokenised review lengths highlighting long-form texts.

consistent configuration across stages.

- **Energy tracking:** Each stage optionally instantiates a CodeCarbon `EmissionsTracker`, configured via the `energy.codecarbon` section of `params.yaml`, to capture environmental impact.
- **Testing:** Pytest suites (e.g. `tests/data/test_download_imdb.py`) validate critical behaviours such as dataset retrieval, safeguarding the pipeline against regressions.

## 5 Data Preparation

The preparation step (`mlops_imdb/data/prepare.py`) performs deterministic cleansing:

- Lowercases text and removes HTML tags.
- Decodes HTML entities and normalises whitespace, ensuring compact token sequences.
- Validates required schema columns (`text`, `label`) in both train and test CSVs.

The cleaned outputs are saved to `data/processed/train_clean.csv` and `data/processed/test_clean.csv`. A fixed random seed (42) is maintained for any stochastic operations, although the current pipeline leverages the canonical IMDb split.

## 6 Feature Engineering

TF–IDF features are produced in `mlops_imdb/features/build_features.py`. Key configuration values include:

- Maximum vocabulary size of 10,000 terms.
- Bi-gram coverage via $n$-gram range $[1, 2]$.
- Frequency cut-offs with `min_df` $= 2$ and `max_df` $= 0.95$.

The resulting sparse matrices are persisted as `.npz` files for both training and test sets, alongside the serialised `TfidfVectorizer`. This design minimises memory footprint and keeps model training decoupled from textual pre-processing.

## 7 Model Training

A logistic regression classifier (`mlops_imdb/modeling/train.py`) is trained on the TF–IDF features. The configuration leverages the `liblinear` solver to handle sparse input efficiently, with `max_iter` $= 1000$ and `random_state` $= 42$. The fitted model is persisted to `models/model.pkl` using `joblib`, ensuring compatibility with downstream inference scripts such as `mlops_imdb/modeling/predict`

## 8 Model Evaluation

The evaluation stage (`mlops_imdb/modeling/eval.py`) loads the trained model and test features to compute standard classification metrics. Results are stored in `reports/metrics.json` and a confusion matrix figure (`reports/figures/baseline_confusion_matrix.png`). Table 1 summarises the latest evaluation run based on that artefact.

Table 1: Binary classification performance on the IMDb test split.

| Metric | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Value | 0.8943 | 0.8907 | 0.8990 | 0.8948 |

The confusion matrix recorded in `reports/metrics.json` indicates balanced performance across classes:

$$\begin{bmatrix} 11121 & 1379 \\ 1263 & 11237 \end{bmatrix}$$

which corresponds to 22,358 correct predictions out of 25,000 test samples. The model makes 1,379 false positives and 1,263 false negatives, highlighting slightly better recall than precision. Figure 3 illustrates the confusion matrix generated during evaluation.

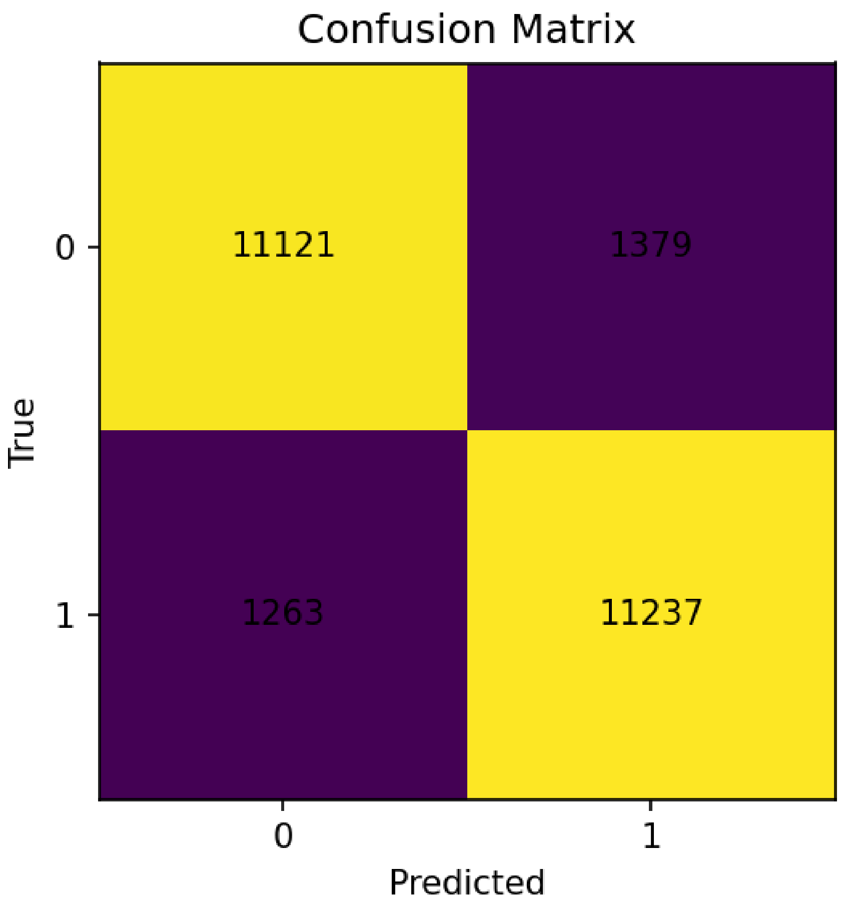

Figure 3: Confusion matrix for the logistic regression classifier.

QA thresholds specified in `params.yaml` (`min_accuracy` = `min_f1` = 0.85) are satisfied, supporting promotion of this model to downstream environments.

## 9 Energy and Carbon Monitoring

DVC runs executed with CodeCarbon tracking (`reports/codecarbon_emissions.csv`) show that the full pipeline emitted only 7.15 mg $CO_2$eq over 27.1 seconds on the Apple M4 development laptop. Table 2 details the per-stage footprint, highlighting feature extraction as the dominant contributor. Power traces stored in `reports/powermetrics_log.txt` confirm modest CPU draw (approximately 3.4 W) with no GPU utilisation, consistent with the lightweight logistic-regression workload.

Table 2: CodeCarbon emissions per pipeline stage (values in mg $CO_2$eq).

| 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** |

## 10 Coding Standards and Tooling

Static analysis relies on Ruff, configured in `pyproject.toml` to enforce a project-wide line length of 99 characters and to treat `mlops_imdb` as the first-party import namespace. The configuration enables both linting and auto-formatting, with import ordering (`I` checks) turned on so that `ruff check` harmonises third-party and local imports. These conventions are codified in the Make targets `make lint` (audit only) and `make format` (auto-fix then format). The `dev` dependency group also pins Ruff for local development and integrates with optional pre-commit hooks, ensuring contributors receive consistent feedback before code reaches the repository.

## 11 Operational Considerations

- **Reproducibility:** DVC pipelines combined with version-controlled parameters guarantee deterministic runs. Processed artefacts (cleaned data, features, model, metrics) are stored under tracked directories, simplifying handoffs.
- **Deployment readiness:** The project includes inference scripts (`mlops_imdb/modeling/predict_one.py`) and configurable prediction paths (`params.yaml`) to facilitate batch or online scoring workflows.
- **Energy monitoring:** CodeCarbon hooks produce emissions reports (e.g. `reports/codecarbon_emissions.` enabling sustainability tracking for future iterations.

## 12 MLflow Tracking (Placeholder)

MLflow integration is planned but not yet implemented. This section will outline experiment tracking configuration, run metadata, and metric logging once the team enables MLflow in the pipeline.

## 13 Conclusions and Next Steps

The current pipeline delivers a robust baseline sentiment model with near 90% accuracy on IMDb reviews. Immediate next steps include:

- Integrating MLflow to manage experiment metadata and model registry operations.
- Evaluating alternative models (e.g. linear SVM, transformer-based encoders) to benchmark gains over logistic regression.
- Expanding automated tests to cover feature generation and evaluation logic, further strengthening quality assurance.