Machine Learning Operations

Predicting Hospital Readmission in Diabetic Patients

Eldar Medvedev (20181162)
Pedro Campino (20240547)

Contents

Contents

1 Python Packages Used

This project relies on several Python packages. Below is a summary of the key libraries and their versions, grouped by purpose:

- **Data manipulation and analysis**: `pandas==2.1.4`, `numpy==1.26.4`, `pyarrow==17.0.0`, `tables==3.9.2`, `numexpr==2.10.1`.
- **Machine Learning and Visualization:** `scikit-learn==1.4.2`, `xgboost==2.1.1`, `altair==4.2.2`, `matplotlib==3.9.4`, `shap==0.48.0`.
- **Databases and Storage:** `SQLAlchemy==1.4.48`, `aiomysql==0.2.0`, `PyMySQL==1.1.1`, `pyodbc==5.2.0`, `deltalake==1.0.2`, `db-dtypes==1.4.3`, `fsspec==2025.5.1`.
- **Cloud and API Integration:** `google-cloud-bigquery==3.34.0`, `google-auth==2.40.3`, `google-api-core==2.25.1`, `pandas-gbq==0.29.1`, `httpx==0.28.1`, `requests==2.32.4`, `hopsworks==4.2.6`.
- **Configuration and Validation:** `pydantic==1.10.13`, `dynaconf==3.2.11`, `python-dotenv==1.1.1`, `great-expectations==0.18.12`.
- **Development Tools and Environment:** `jupyterlab==4.4.3`, `ipython==8.18.1`, `ipykernel==6.29.5`, `notebook==7.4.3`, `cookiecutter==2.6.0`, `pre-commit-hooks==5.0.0`.
- **Experiment Tracking and Orchestration:** `kedro==0.19.14`, `kedro-telemetry==0.6.3`, `kedro-datasets==4.1.0`, `kedro-mlflow==0.14.4`, `mlflow==2.22.1`.
- **Web Framework / API serving:** `fastapi==0.111.0`, `starlette==0.37.2`, `uvicorn==0.34.3`, `Flask==3.1.1`.

A complete list of dependencies with pinned versions can be found in the project repository (see Appendix A).

2 Project Planning

To structure our MLOps project effectively, we adopted an agile-inspired workflow, organizing our work into three main sprints, each spanning approximately two weeks. The planning and coordination were managed collaboratively using a combination of GitHub Projects, allowing for transparent task tracking, priority assignment, and sprint retrospectives.
Sprint 1 – Data Understanding and Initial Setup: The first sprint focused on understanding the dataset, identifying missing values, exploring key variables, and assessing class distribution. Simultaneously, we set up the base project structure using the `cookiecutter-kedro` template, integrated version control with GitHub, and configured the environment with Docker to ensure consistent development setups across team members.
Sprint 2 – Pipeline Development and Experiment Tracking: In this sprint, we developed the modular pipeline using Kedro, incorporating data preprocessing, validation, splitting, and model training nodes. We also integrated MLflow for experiment tracking and Hopsworks for feature storage. The model candidates (Logistic Regression and XGBoost) were trained and evaluated using cross-validation and tracked via MLflow runs. Unit tests were written for data quality checks and pipeline nodes.

Sprint 3 – Model Evaluation, Deployment, and Monitoring: The final sprint focused on finalizing model selection and evaluation. We simulated data drift, logged artifacts, and performed SHAP-based explainability analysis. A FastAPI-based service was created to expose the model, and Docker images were built for containerized deployment. We also discussed CI/CD possibilities and monitored key metrics for potential model degradation.
Throughout the process, both met weekly to review progress, assign responsibilities, and adapt priorities based on the latest findings. GitHub issues and pull requests facilitated code reviews and collaborative development.

3 Problem Definition and Introduction

Context

Hospital readmissions within 30 days after discharge are a significant concern for healthcare systems, as they often indicate issues in quality or continuity of care. They also represent a considerable financial burden for hospitals and insurance providers. Patients with chronic diseases, such as diabetes, are particularly at risk due to the complexity of their treatment and follow-up care.

Objective

The aim of this MLOps project is to simulate the real-world process of deploying a machine learning model. Specifically, we develop a pipeline to predict whether a diabetic patient will be readmitted to the hospital within 30 days after discharge. This task is framed as a binary classification problem. Our approach includes modular pipeline design, version control, testing, model monitoring, and deployment considerations.

Success Metrics

Due to class imbalance and the clinical relevance of identifying patients at risk of early readmission (class 1), we evaluate models using precision, recall, F1-score, AUC-ROC, and AUC-PR.While all metrics offer useful insights, we select the best model based on the F1-score of class 1, as it balances precision and recall for this critical group.

Dataset Overview

The analysis is based on the *Diabetes 130-US hospitals for years 1999–2008* dataset, which contains over 100,000 records of hospitalizations for diabetic patients in 130 hospitals in the United States. Each record includes demographic information, diagnostic codes, laboratory test results, medication data, and administrative attributes.

Problem Formulation

The target variable in this study is `readmitted`, which contains three possible values:

- `<30` – the patient was readmitted within 30 days,
- `>30` – the patient was readmitted after 30 days,
- `NO` – the patient was not readmitted.

For the purposes of this project, the problem is simplified into a binary classification task, where:

- Class 1: `<30` (readmitted within 30 days),
- Class 0: `>=30` or `NO` (not readmitted within 30 days).

This simplification was made because readmissions in 30 days are especially important in the healthcare setting. They are often seen as a sign of potential problems in the care process, such as complications, inadequate discharge planning, or insufficient follow-up. By focusing on this 30-day window, the model can help identify high-risk patients early on, allowing for timely interventions that improve care and help avoid unnecessary hospital costs.

Motivation

Accurate prediction of early readmissions can help healthcare providers identify high-risk patients and take preventive actions, such as more intensive discharge planning, patient education, or follow-up appointments. This contributes to improving patient outcomes and reducing unnecessary hospital costs.

4 Data Exploration Insights

Missing Data

Some features, such as `weight`, `payer_code`, and `medical_specialty`, contain a high proportion of missing values. These likely reflect inconsistencies in clinical documentation or administrative omissions. While some can be imputed, variables with excessive missingness may need to be excluded from modeling.

Target Variable Distribution

Only about 11% of patients fall into this high-risk category, highlighting a significant class imbalance (see Figure 1 in Appendix B). This can impair model performance if unaddressed; resampling strategies or cost-sensitive learning will likely be necessary.

Categorical Variables and Association with Readmission

Key categorical features—such as `admission_type_id`, `discharge_disposition_id`, and `A1Cresult`—were explored through frequency analysis and chi-square tests. Results show that emergency admissions and discharges to care facilities are significantly associated with higher readmission rates. Elevated A1C values (>8) were also linked to increased risk, suggesting that glycemic control plays a critical role.

High-Cardinality Categorical Features

Some features, including `diag_1`, `diag_2`, `diag_3`, and `medical_specialty`, contain an overwhelming number of unique values, making them difficult to visualize and model directly. Grouping ICD-9 codes into broader diagnostic categories and consolidating rare specialties may help mitigate sparsity and improve interpretability.

Numerical Feature Distributions

The numerical variables reveal markedly skewed distributions, with most patients concentrated at the lower end and a smaller group showing much higher values (see Figure 2 in Appendix B). For example, `number_inpatient`, `number_emergency`, and `time_in_hospital` are typically low, but some patients experienced frequent admissions or extended hospital stays, possibly reflecting complications or poorly managed conditions.
There is also a wide variation across patients when it comes to `num_medications`. Higher counts may indicate more complex treatment regimens or comorbidities, a pattern commonly observed in polypharmacy among vulnerable diabetic populations.

Correlation Analysis

Correlation between numerical variables was generally modest, with the highest pairwise value around 0.47 (see Figure 3 in Appendix B). This suggests limited multicollinearity and reduces the need for aggressive feature pruning. However, moderate relationships (e.g., between `num_medications`, `time_in_hospital`, and `num_procedures`) may still reflect overlapping aspects of healthcare burden.

Feature Relevance

Statistical tests were used to identify the most informative features. ANOVA and mutual information highlighted `number_inpatient`, `number_diagnoses`, and `time_in_hospital` as top contributors to readmission prediction. While mutual information scores were low in absolute terms, they clearly distinguished a subset of more predictive features.

5 Modeling Approach

The modeling pipeline was implemented using the `Kedro` framework, which provides a modular and scalable structure for reproducible data science projects. The entire project workflow was executed through the following nine key steps:

1. **Data Ingestion:** Raw data was ingested and feature sets were stored in `Hopsworks`, a data platform for managing features and experiments efficiently.
2. **Data Quality Testing:** A data unity test was performed to identify and log bad data points. For instance, any invalid entries in the `gender` column (values other than "male" or "female") were detected and logged for further inspection.
3. **Data Splitting:** Before preprocessing, the dataset was split into training and testing sets (80% training, 20% testing) to prevent data leakage during feature engineering and transformation.
4. **Data Preprocessing:** All preprocessing steps were applied including handling missing values, outlier detection, and feature engineering with label encoding and one-hot encoding.
5. **Validation Split:** A further split within the training data was performed to create a validation set, allowing us to select the best model configuration without touching the test set.
6. **Model Training:** This step involved training the model while leveraging `MLflow`, an open-source platform that facilitates experiment tracking, model management, and reproducibility by logging parameters, metrics, and artifacts. Additionally, a baseline model was established using a simple `XGBClassifier`.

7. **Model Selection:** We compared multiple algorithms, specifically *Logistic Regression* and *XGBoost classifiers*, using grid search to tune hyperparameters and select the best performing model.

8. **Final Prediction:** The selected model was used to generate predictions on the test set, and performance metrics were computed to evaluate the final results.

9. **Drift Simulation:** An additional step involves simulating data drift to evaluate how our model performs under changing data conditions.

6 Modeling Results

Two model versions were initially trained and evaluated using their default hyperparameters, with results properly tracked using version control tools. In the final comparison between the best *Logistic Regression* and *XGBoost* models, *XGBoost* was selected as the best performer, achieving a class 1 F1-score of 0.27, higher than the 0.25 obtained by *Logistic Regression.*
Subsequently, the *XGBoost* model was further optimized through hyperparameter tuning to improve performance. Since XGBoost was the chosen model, its final results were stored, and the remaining performance metrics are shown below.

Model	**Accuracy**	**AUC ROC**	**Prec (1)**	**Rec (1)**	**F1 (1)**	**Prec (0)**	**Rec (0)**	**F1 (0)**
XGB Classifier	0.633	0.673	0.176	0.619	0.273	0.930	0.635	0.755

Table 1: Model performance metrics for the *XGBoost Classifier*

	Predicted 0	**Predicted 1**
Actual 0	11475	6602
Actual 1	865	1406

Table 2: Confusion matrix for the *XGBoost Classifier*

The *XGBoost Classifier* achieved an accuracy of 63.3% and an *AUC-ROC* of 0.673, indicating a reasonable ability to discriminate between readmitted and non-readmitted patients.
Focusing on the minority class (class 1 — readmitted patients), the precision is low (0.176), meaning many positive predictions are false positives. However, the recall is relatively high (0.619), showing that the model correctly identifies most patients who will be readmitted. The F1-score for class 1 is 0.273, reflecting a balance between precision and recall, although there is room for improvement.
For the majority class (class 0 — non-readmitted patients), the model attains high precision (0.930) but moderate recall (0.635), indicating the model tends to be conservative in classifying a patient as non-readmitted but still misses some cases in this class.
The confusion matrix confirms this behavior: there are 6602 false positives (non-readmitted patients classified as readmitted) and 865 false negatives (readmitted patients not detected). This distribution suggests the model is more inclined to flag patients as readmitted, which may be an appropriate strategy to prioritize monitoring these patients despite the cost of false positives.

As shown in Table 3, the differences in all metrics between the normal and drifted data are minimal, however, a slight degradation in performance can be observed.

Model	Accuracy	AUC ROC	Prec (1)	Rec (1)	F1 (1)	Prec (0)	Rec (0)	F1 (0)
XGB Classifier	0.617	0.671	0.171	0.633	0.269	0.930	0.615	0.740

Table 3: Model performance metrics for the *Drifted Data*

7 Feature Importance

Figure 4 in Appendix C shows the SHAP summary plot, which illustrates the impact of each feature on the *XGBoost Classifier*'s predictions. Each point represents a patient case, with the SHAP value on the x-axis indicating how much that feature influenced the prediction for that individual. The color reflects the feature value, from low (blue) to high (red).
The most influential feature is `discharge_disposition_id`, where high values tend to significantly reduce the model's output. This suggests that certain discharge outcomes (e.g., discharge to a facility or death) are strongly associated with the predicted target. Other key features include `number_inpatient`, `num_lab_procedures`, and `num_medications`. For example, a high number of previous inpatient visits tends to increase the predicted risk. In contrast, `num_lab_procedures` shows both positive and negative impacts depending on the context, which indicates the model considers interactions with other variables.
This visualization helps identify which variables most influence the model's decision-making, and in what direction.

8 Conclusion

This project demonstrates a complete Machine Learning Operations (MLOps) pipeline focused on predicting hospital readmissions within 30 days for diabetic patients. Through the use of a structured Kedro pipeline, MLflow for experiment tracking, SHAP for explainability, and Hopsworks for feature storage and governance, we successfully simulated the deployment process of a real-world healthcare application. Emphasis was placed on modularity, version control, validation, and monitoring to align with industry practices.
Despite the promising performance of the XGBoost model, challenges such as class imbalance and data quality limitations remain. Addressing these issues through advanced sampling techniques and improved data validation would further increase model robustness.
One of the key technological enablers in this project was Docker, which allowed us to encapsulate the entire environment—dependencies, configurations, and runtime—into reproducible containers. This ensures consistency between development and production environments and facilitates collaborative workflows. Docker also paves the way for CI/CD pipelines and deployment scalability, particularly when integrated with tools like Kubernetes or cloud-native solutions.
While pandas and NumPy were sufficient for our current data volume, this architecture may not scale efficiently with larger datasets. In a production scenario, we propose extending the pipeline to support distributed processing using Apache Spark. We estimate that adapting the pipeline to PySpark would require an additional 2–3 weeks of development, including testing and performance benchmarking.
In summary, this proof of concept not only achieves its predictive goal but also lays the foundation for a production-ready MLOps system. The selected technologies provide a balance between agility and scalability, while our modular design ensures maintainability and adaptability to future requirements.

A Project Repository

The complete source code for this project is available at the following GitHub repository: `https://github.com/ExodiaViolet/Exodia-MLOPS/`

B Data Exploration

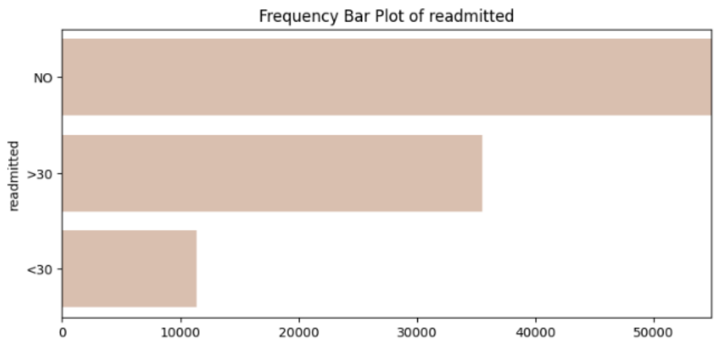

Figure 1: Distribution of the target variable

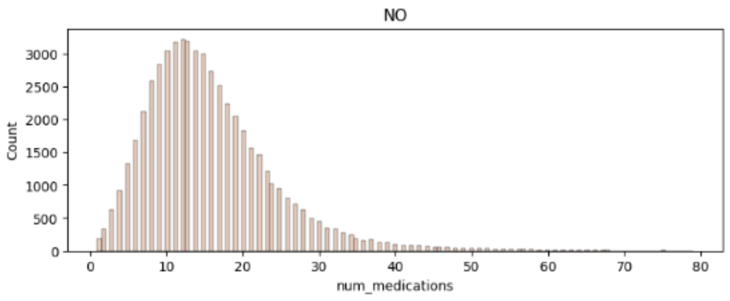

Figure 2: Example of a right skewed distribution

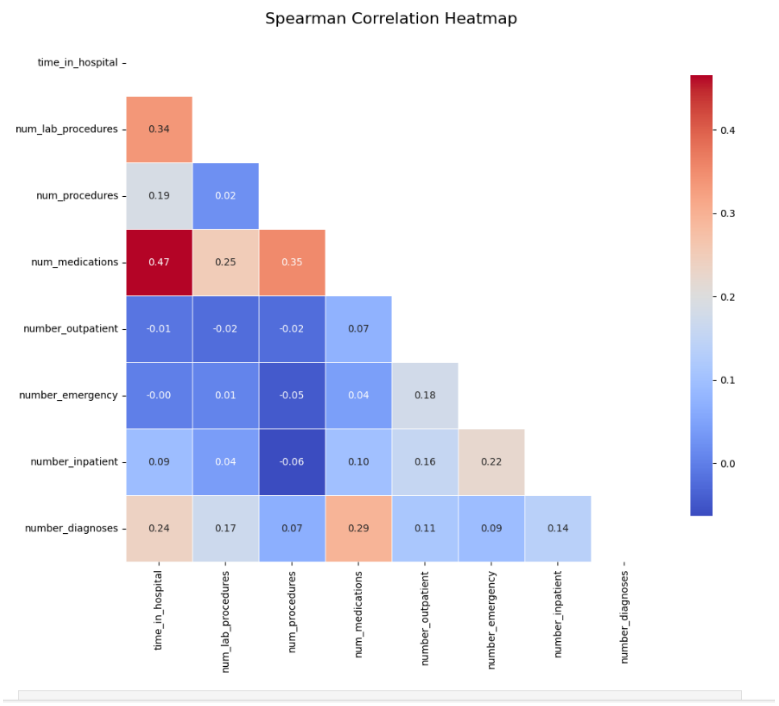

Figure 3: Spearman Correlation Heatmap

C SHAP Feature Importance Plot

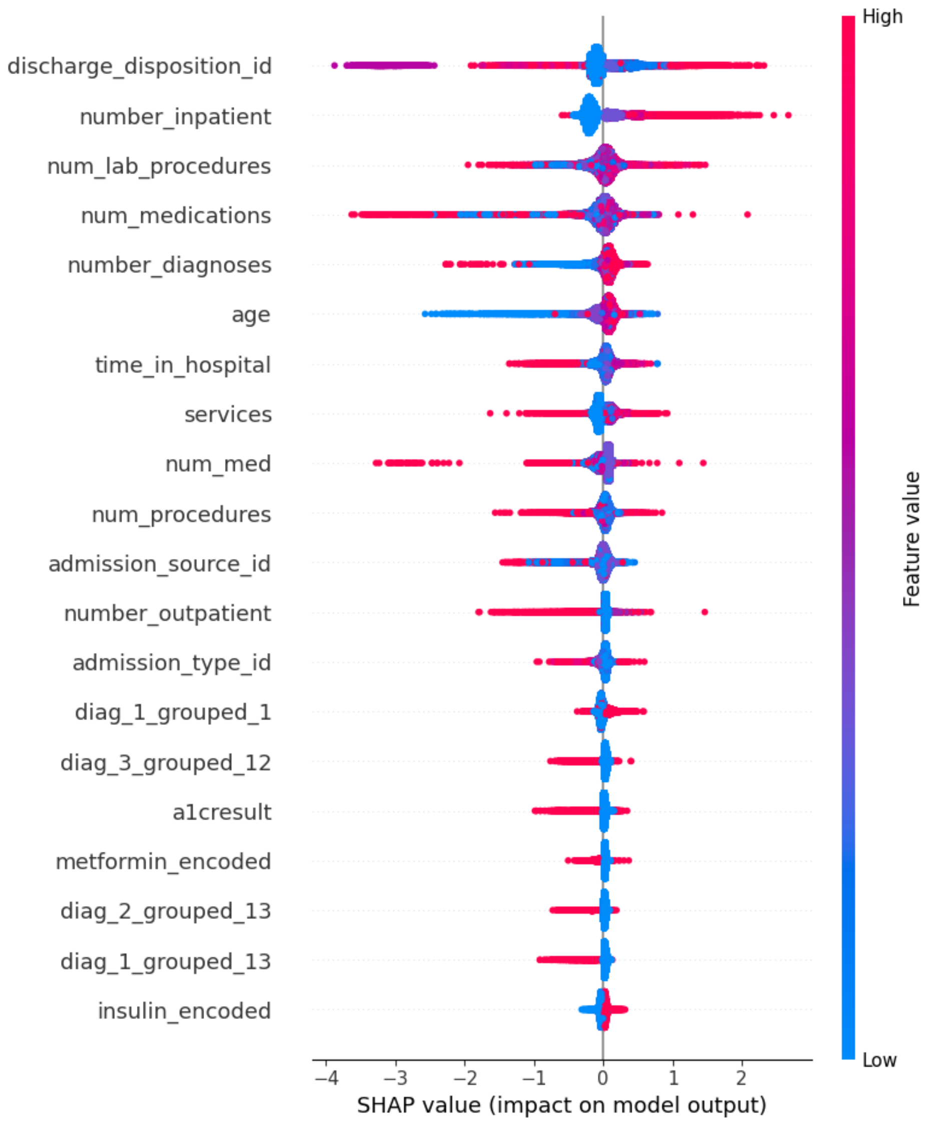

Figure 4: SHAP summary plot showing the impact of each feature on the model predictions.