Mohammed VI Polytechnic University
Institute of Science, Technology & Innovation
Al-Khwarizmi Department
Africa Business School

Pipeline automation using apache airflow for continuous machine learning

Data Science Project

Authors:
Samir JABBAR
Mohamed CHAFIQ

Instructors:
Pr. Fahd KALLOUBI

July 6, 2023

Chapter 1

Abstract

This report focuses on the development and automation of a continuous machine learning pipeline for fraud detection using Apache Airflow. With the increasing threat of fraud in today's digital landscape, organizations need robust systems that can detect and prevent fraudulent activities in real-time. Machine learning algorithms offer promising solutions, but deploying them in production environments poses challenges.

The objective of this project is to streamline the end-to-end process of fraud detection by automating the pipeline using Apache Airflow. The report discusses various components, including data collection, preprocessing, model training, evaluation, and deployment. By leveraging Apache Airflow's capabilities as an orchestration framework, organizations can ensure consistent model updates, adapt to changing fraud patterns, and improve overall efficiency.

The report highlights the advantages of using Apache Airflow for continuous machine learning and provides practical insights on automating fraud detection pipelines. By establishing a continuous learning system, organizations can enhance their fraud detection capabilities, reduce response time to emerging threats

Key Words: machine learning, continuous learning, automation, Apache Airflow, pipeline, fraud detection;

Chapter 2

Acknowledgment

We would like to express our deepest gratitude to Professor Fahd KALLOUBI for his invaluable guidance, support, and expertise throughout the course of this project. His profound knowledge and dedication to the field of machine learning have been instrumental in shaping our understanding and approach to continuous machine learning and pipeline automation

First and foremost, we would like to thank the European cardholders for providing the Credit Card Fraud Detection dataset used in this study. Without their cooperation and willingness to share their data, this research would not have been possible.

Furthermore, we would like to thank our colleagues and peers, who provided valuable feedback and suggestions on the report, and helped us to improve its quality.

Finally, we would like to acknowledge the many researchers and scientists who have contributed to the development of machine learning algorithms and techniques, and whose work has paved the way for the current study. Their contributions have been instrumental in advancing our understanding of this field and developing new tools and methodologies.

Contents

List of Figures

Chapter 3

Introduction

In the rapidly evolving field of machine learning, the ability to continuously improve models and adapt to changing data is crucial for achieving optimal performance and staying ahead of the competition. Continuous Machine Learning refers to a paradigm where machine learning models are updated and refined over time as new data becomes available. Unlike traditional learning, where models are trained on static datasets and deployed in a fixed state, continuous learning enables models to learn and improve in an ongoing and dynamic manner.

The primary goal of this project is to develop a robust and efficient pipeline automation system for continuous machine learning using Apache Airflow. Apache Airflow is an open-source platform that allows users to programmatically author, schedule, and monitor workflows, making it an ideal choice for automating the complex and repetitive tasks involved in the continuous machine learning process.

There are several problems that this project seeks to address. Firstly, Data Drift Real-world data is often non-stationary, meaning its statistical properties change over time. Models trained on outdated data might become less accurate and relevant as the data distribution shifts, leading to performance degradation. Continuous learning addresses this issue by periodically retraining models with new data, ensuring they remain up-to-date and reliable.

Secondly, continuous learning poses unique challenges, such as handling streaming data, managing model updates, and deploying updated models seamlessly. Apache Airflow provides a flexible and extensible framework that can accommodate these challenges, allowing us to design workflows that handle real-time data ingestion, model retraining, and deployment in a robust and scalable manner.

Lastly, the project aims to address the need for scalability in machine learning pipelines. As organizations collect vast amounts of data and require models to adapt and improve continuously, the ability to scale the pipeline becomes crucial. Apache Airflow's distributed architecture enables us to scale horizontally, leveraging multiple workers and resources, to handle increasing data volumes and model complexities efficiently.

By addressing these problems, the pipeline automation project using Apache Airflow for continuous machine learning aims to provide organizations with an efficient and scalable solution for integrating and updating machine learning models in real time. The resulting pipeline will not only streamline the process but also enable organizations to leverage the latest data for improved predictions, leading to enhanced decision-making capabilities and staying ahead in today's rapidly evolving data-driven landscape.

Chapter 4

Materials and Methods

4.1 Dataset Description

Credit Card Fraud Detection is a dataset containing transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) accounts for 0.172

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the owners cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise

4.2 Performance Criteria :

Accuracy score : To define the term, in Machine Learning, the Accuracy score (or just Accuracy) is a Classification metric featuring a fraction of the predictions that a model got right. The metric is prevalent as it is easy to calculate and interpret. Also, it measures the model's performance with a single value.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

Figure. 4.1: Accuracy formula

Confusion Matrix: A confusion matrix can be used to perfectly analyze the potential of a classifier. All the diagonal elements denote correctly classified outcomes. The misclassified outcomes are represented on the off diagonals of the confusion matrix. Hence, the best classifier will have a confusion matrix with only diagonal elements and the rest of the elements set to zero.

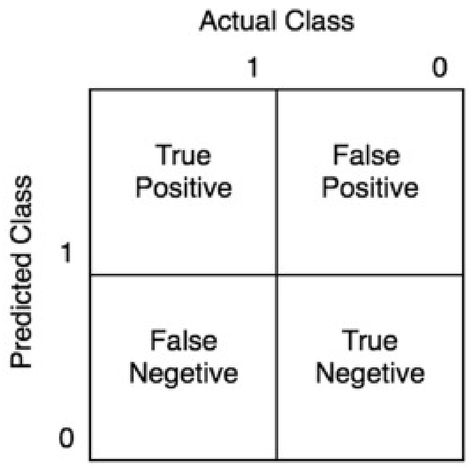


Figure. 4.2: Confusion matrix for binary classification

Cross Validation score: Cross val score is a method which runs cross validation on a dataset to test whether the model can generalise over the whole dataset. The function returns a list of one score per split, and the average of these scores can be calculated to provide a single metric value for the dataset.

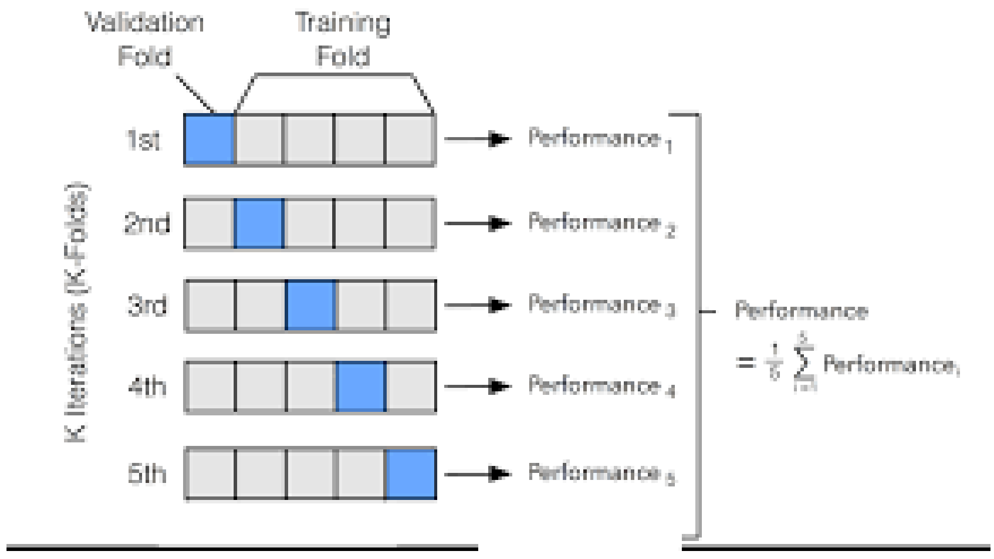


Figure. 4.3: Confusion matrix for binary classification

Precision:

Precision is another performance metric used in machine learning to measure the percentage of true positive instances among all the instances that the model has classified as positive. It is defined as the ratio of true positive (TP) instances to the sum of true positive and false positive (FP) instances. In other words, precision is the fraction of retrieved instances that are relevant to the problem being solved.

The formula for precision is:

$$Precision = \frac{TP}{TP + FP}$$

where TP is the number of true positive instances and FP is the number of false positive instances.

Recall:

Recall is a performance metric used in machine learning to measure the percentage of true positives that were correctly identified by a model. It is defined as the ratio of true positive (TP) instances to the sum of true positive and false negative (FN) instances. In other words, recall is the fraction of relevant instances that are retrieved by the model.

The formula for recall is:

$$Recall = \frac{TP}{TP + FN}$$

where TP is the number of true positive instances and FN is the number of false negative instances.

F1-score :

The F1 score is a performance metric used in machine learning that combines both precision and recall into a single score. It is the harmonic mean of precision and recall and provides a balanced assessment of a model's performance.

The formula for F1-score is:

$$F1 - score = \frac{2 * (precision * recall)}{(precision + recall)}$$

where precision and recall are as defined in the previous answers.

The F1-score ranges from 0 to 1, with higher values indicating better performance. A perfect F1-score of 1.0 means that the model has both high precision and high recall, indicating that it is able to correctly identify all relevant instances while avoiding false positives.

AUC:
AUC stands for "Area Under the Curve." It is a metric used to evaluate the performance of a binary classification model. The ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a classification model at various thresholds. The AUC represents the overall performance of the model by calculating the area under the ROC curve.

The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) at different classification thresholds. The AUC value ranges from 0 to 1, with a higher value indicating better performance. An AUC of 0.5 suggests that the model's performance is no better than random guessing, while an AUC of 1 indicates a perfect classifier.

In summary, AUC is a measure of the ability of a binary classification model to distinguish between positive and negative classes, and it provides an aggregated assessment of its performance across different thresholds.

4.3 Data balancing

Data balancing in machine learning refers to the process of adjusting the class distribution of a dataset to ensure that each class has a roughly equal number of samples. This is typically done in situations where the classes are imbalanced, meaning that one or more classes have significantly fewer examples than others.
Imbalanced datasets can pose a problem for machine learning algorithms, as they may lead to biased or inaccurate models that are better at predicting the majority class than the minority classes.
There are several methods for data balancing:

A. Oversampling is a technique used to balance the dataset by increasing the number of samples in the minority class by replicating existing data points.

B. Undersampling is a technique used to balance the dataset by decreasing the number of samples in the majority class. This technique involves randomly selecting a subset of the majority class samples to reduce the imbalance class.

C. SMOTE Sampling stands for Synthetic Minority Over-sampling Technique, which is a technique used to balance the dataset by generating synthetic from the minority class by interpolating between existing minority class samples. Specifically, SMOTE selects a sample from the minority class and finds its k nearest neighbors in the feature space. New synthetic samples are then created by interpolating between the selected sample and its neighbor.

D. Generative Adversarial Networks(GANs) are deep learning models that consist of two components: a generator and a discriminator. The generator learns to generate synthetic samples that resemble the real data, while the discriminator tries to distinguish between real and fake samples.

To balance data using GANs, you can follow these steps:

1. Train the GAN: Train the GAN using the imbalanced dataset, where the generator learns to generate synthetic samples for the minority class(es) and the discriminator learns to distinguish between real and fake samples. The generator's objective is to fool the discriminator, while the discriminator's objective is to correctly classify the real and fake samples.
2. Generate synthetic samples: Once the GAN is trained, you can use the generator to generate synthetic samples for the minority class(es). The generator should have learned to generate samples that resemble the real data, thereby helping balance the representation of the minority class(es) in the dataset.
3. Combine synthetic samples with the original data: Combine the generated synthetic samples with the original data from the majority class(es) to create a balanced dataset. Make sure to preserve the original samples as well, as they still contain valuable information.
4. Train the machine learning model: Finally, use the balanced dataset to train your machine learning model. With a balanced dataset, the model will have an equal representation of different classes, reducing bias and potentially improving performance.

4.4 Classification techniques

4.4.1 Decision Tree:

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

4.4.2 Support vector machine (SVM):

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

4.4.3 Random Forest:

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.

A random forest algorithm consists of many decision trees. The 'forest' generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.
The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

A random forest eradicates the limitations of a decision tree algorithm. It reduces the overfitting of datasets and increases precision. It generates predictions without requiring many configurations in packages (like scikit-learn).

4.4.4 K-nearest neighbor (KNN):

The k-nearest neighbors classifier (kNN) is a non-parametric supervised machine learning algorithm. It's distance-based: it classifies objects based on their proximate neighbors' classes. kNN is most often used for classification, but can be applied to regression problems as well. The parameter k in kNN refers to the number of labeled points (neighbors) considered for classification. The value of k indicates the number of these points used to determine the result. Our task is to calculate the distance and identify which categories are closest to our unknown entity.

4.4.5 Ensemble Methods:

The idea behind ensemble methods is that by combining several models that have been trained on the same task, but with different approaches or subsets of data, the ensemble model can capture a more comprehensive representation of the underlying patterns in the data. This can lead to better predictions or classifications compared to using a single model.

There are several popular ensemble methods used in machine learning, including:

1. Bagging (Bootstrap Aggregating): In bagging, multiple base models are trained independently on different bootstrap samples of the training data. The predictions from each model are then combined, typically by averaging or voting, to obtain the final prediction. Random Forest is an example of an ensemble method that uses bagging.

2. Boosting: Boosting works by training base models sequentially, where each subsequent model tries to correct the mistakes made by the previous models. The final prediction is obtained by combining the predictions of all the models, usually through a weighted voting scheme. Examples of boosting algorithms include AdaBoost, Gradient Boosting, and XGBoost.

3. Stacking: Stacking involves training multiple base models on the same dataset and then combining their predictions using another model, called a meta-model or blender. The base models' predictions serve as input features for the meta-model, which learns to make the final prediction. Stacking can capture complex interactions between the base models and has been used successfully in various machine-learning competitions.

4. Voting: Voting is a simple ensemble method where multiple base models independently make predictions on the input, and the final prediction is determined by majority voting (for classification) or averaging (for regression). Voting can be done with equal weights or with weights assigned to individual models based on their performance or expertise.

4.4.6 Feed Forward NN:

A feedforward neural network (also known as a multilayer perceptron) is a type of artificial neural network in which information flows in one direction, from the input layer to the output layer, without any feedback connections or loops. It is called "feedforward" because the data moves forward through the network without any feedback loops.

The network consists of multiple layers of interconnected nodes, or neurons, organized in a sequential manner. The basic structure typically includes an input layer, one or more hidden layers, and an output layer. Each neuron in the network receives inputs from the previous layer and produces an output that is transmitted to the next layer.

In a feedforward neural network, the connections between neurons are weighted, and each neuron applies an activation function to its weighted sum of inputs to determine its output. Common activation functions include sigmoid, tanh, and ReLU.

During the training phase, the network learns by adjusting the weights of the connections based on a labeled training dataset. This process is typically done using backpropagation, where the errors between the network's predicted outputs and the true labels are propagated backward through the network to update the weights and improve the network's performance

4.4.7 LSTM:

LSTM stands for Long Short-Term Memory, which is a type of recurrent neural network (RNN) architecture designed to overcome the limitations of traditional RNNs in capturing long-term dependencies in sequential data.

The primary difference between LSTM and traditional RNNs is the presence of a memory cell in LSTM. The memory cell allows the network to selectively retain or forget information over time, making it effective in handling sequences with long time lags and preserving important information over extended periods.

The key components of an LSTM cell are as follows:

1. Cell State (Ct): This represents the memory of the LSTM and is responsible for carrying information across different time steps. The cell state can selectively retain or discard information using gates.

2. Input Gate (i): It determines how much of the input at the current time step should be stored in the cell state.

3. Forget Gate (f): It controls how much of the previous cell state should be forgotten.

4. Output Gate (o): It determines how much of the cell state should be exposed to the output.

5. Candidate Activation ($\hat{C}t$): It computes the potential new values that can be added to the cell state.

LSTM cells process input sequentially, taking into account the current input, the previous cell state, and the previous hidden state (output). The gates are responsible for controlling the flow of information by regulating the values passed through the cell state and hidden state.

4.5 Framework and Technologies

4.5.1 Docker:

Docker is a cutting-edge open-source platform that empowers developers to build, deploy, run, update, and manage containers. Containers are self-contained units that combine application source code with the necessary operating system (OS) libraries and dependencies to run the code consistently across various environments.
The adoption of containers has simplified the development and delivery of distributed applications. As organizations increasingly embrace cloud-native development and hybrid multicloud environments, containers have gained immense popularity. While it is possible to create containers without Docker by leveraging native capabilities within Linux and other operating systems, Docker offers a faster, easier, and safer containerization experience. Currently, over 13 million developers are utilizing the Docker platform.
In addition to referring to the platform, Docker encompasses Docker, Inc., the company behind the commercial version of Docker, and the Docker open-source project, which benefits from contributions by Docker, Inc., as well as numerous other organizations and individuals.

How Containers Work and Their Popularity

Containers leverage process isolation and virtualization features inherent in the Linux kernel. These capabilities, such as control groups (Cgroups) for resource allocation and namespaces for restricting process access, enable multiple application components to share a single instance of the host operating system, similar to how a hypervisor facilitates resource sharing among virtual machines (VMs) on a hardware server.
Containers offer all the advantages of VMs, including application isolation, cost-effective scalability, and disposability, while also providing additional benefits:

1. Lighter weight: Unlike VMs, containers eliminate the need to carry the overhead of an entire OS instance and hypervisor. They include only the essential OS processes and dependencies required to execute the code. Container sizes are measured in megabytes, making better use of hardware resources and resulting in faster startup times.

2. Improved developer productivity: Containerized applications can be developed once and run anywhere. Compared to VMs, containers are faster, easier to deploy, provision, and restart. These qualities make containers ideal for continuous integration and continuous delivery (CI/CD) pipelines, and they align well with Agile and DevOps practices.

3. Greater resource efficiency: Containers enable developers to run significantly more instances of an application on the same hardware compared to VMs. This efficiency can lead to reduced cloud costs and improved resource utilization.

Organizations that adopt containers report numerous benefits, including enhanced application quality, faster response to market changes, and more.

Why Choose Docker?

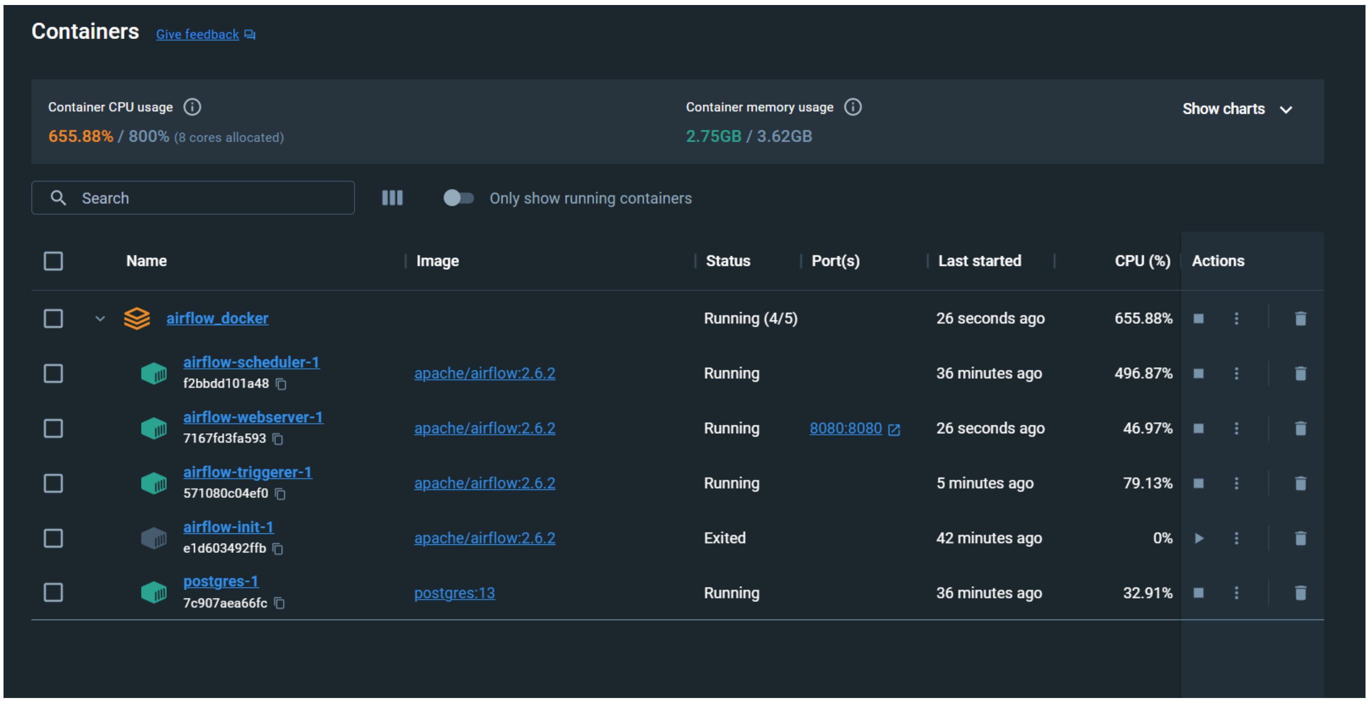
Today, Docker has become synonymous with containers, although container-related technologies existed long before Docker's public release in 2013. Notably, LinuXContainers (LXC) implemented full virtualization for a single Linux instance in 2008. While LXC remains in use, newer technologies leveraging the Linux kernel have emerged, including capabilities provided by modern, open-source Linux operating systems like Ubuntu.
Docker simplifies access to these native containerization capabilities through simple commands and an efficient application programming interface (API). Compared to LXC, Docker offers the following enhancements:

1. Improved container portability: LXC containers often rely on machine-specific configurations, whereas Docker containers run seamlessly across desktops, data centers, and cloud environments without modification.
2. Lighter weight and granular updates: Docker allows multiple processes to be combined within a single container. Consequently, applications can continue running while specific components undergo updates or repairs.
3. Automated container creation: Docker automates the container building process based on application source code.
4. Container versioning: Docker enables version tracking, rollback to previous versions, and the ability to trace the builders and changes between versions. It can even upload only the differences (deltas) between existing and new versions.
5. Container reuse: Existing containers can serve as base images, acting as templates for creating new containers.
6. Shared container libraries: Docker provides access to an open-source registry that hosts thousands of user-contributed containers.

Docker containerization is not limited to Linux. It also works seamlessly with Microsoft Windows and Apple macOS. Developers can run Docker containers on any operating system, and major cloud providers like Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to support building, deploying, and running Docker containerized applications.

Installing Airflow with Docker

Airflow, an open-source platform for workflow automation, benefits greatly from Docker's containerization capabilities. Docker simplifies the installation and setup process for Airflow by


Containers Give feedback

Container CPU usage 655.88% / 800% (8 cores allocated)

Container memory usage 2.75GB / 3.62GB

Show charts

Search

Only show running containers

Name	Image	Status	Port(s)	Last started	CPU (%)	Actions
airflow_docker		Running (4/5)		26 seconds ago	655.88%	■ ⋮
airflow-scheduler-1 f2bbdd101a48	apache/airflow:2.6.2	Running		36 minutes ago	496.87%	■ ⋮
airflow-webserver-1 7167fd3fa593	apache/airflow:2.6.2	Running	8080:8080	26 seconds ago	46.97%	■ ⋮
airflow-triggerer-1 571080c04ef0	apache/airflow:2.6.2	Running		5 minutes ago	79.13%	■ ⋮
airflow-init-1 e1d603492ffb	apache/airflow:2.6.2	Exited		42 minutes ago	0%	▶ ⋮
postgres-1 7c907aea66fc	postgres:13	Running		36 minutes ago	32.91%	■ ⋮

Figure. 4.4: Airflow install in docker

encapsulating all the required dependencies and configurations within a container.
The process of installing Airflow with Docker typically involves the following steps:

1. **Creating a Docker Image**: A Docker image is created by defining a Dockerfile, which contains instructions for building the image. The Dockerfile specifies the base image, installs dependencies, copies the Airflow configuration files, and sets up any required environment variables.

2. **Building the Docker Image**: The Docker image is built using the `docker build` command, which reads the Dockerfile and creates an image containing the necessary components.

3. **Running Airflow as a Docker Container**: Once the Docker image is built, it can be run as a Docker container using the `docker run` command. The container runs an instance of Airflow with all the dependencies and configurations specified in the image.

By using Docker, the installation process becomes more streamlined, consistent, and portable across different systems. Docker eliminates the need for manual configuration and ensures that Airflow runs in a controlled and isolated environment. It also simplifies the process of managing dependencies and scaling Airflow instances when needed.
Moreover, Docker enables version control and reproducibility of Airflow installations. Docker images can be versioned, allowing for easy rollbacks or updates to specific Airflow versions. This makes it convenient to share and distribute Airflow setups across different development, testing, and production environments.

4.5.2 Apache Airflow:

Apache Airflow is an open-source platform used for orchestrating and managing complex workflows and data pipelines. It allows scheduling, monitoring, and executing workflows composed of tasks that can be written in various programming languages.
Airflow was originally developed by Airbnb and later open-sourced and donated to the Apache Software Foundation. It provides a programming framework that allows us to define and execute workflows as Directed Acyclic Graphs (DAGs). DAGs represent a set of tasks and their dependencies, where tasks can be executed in parallel or sequentially based on the defined relationships.
Airflow provides a web-based user interface called the Airflow UI, which allows one to visually monitor the status of workflows, view task logs, and manage the overall system. It also supports a command-line interface (CLI) for interacting with the system and managing workflows programmatically.

Some key features of Apache Airflow include:

1. Workflow scheduling: Airflow provides a powerful scheduler that allows you to schedule workflows based on time, intervals, or external events. It supports advanced scheduling features like cron expressions and calendar-based scheduling.

2. Task dependencies: You can define dependencies between tasks in a workflow, allowing Airflow to automatically determine the order in which tasks should be executed based on their dependencies.

3. Extensibility: Airflow is highly extensible and allows you to define custom operators and hooks for integrating with external systems or executing specialized tasks. It also supports custom sensors for triggering task execution based on external events.

4. Monitoring and logging: Airflow provides a comprehensive monitoring and logging system that allows you to track the progress and status of workflows. You can view logs, visualize task execution history, and set up alerts for task failures or other events.

5. Scalability: Airflow is designed to handle large-scale workflows and can distribute task execution across multiple worker nodes for improved performance and scalability.

6. Integration ecosystem: Airflow integrates with various external systems and services, such as databases, cloud platforms (e.g., AWS, GCP), messaging systems (e.g., Apache Kafka), and data processing frameworks (e.g., Apache Spark).

Airflow for Continuous Machine Learning Training

Continuous machine learning training involves the iterative and automated process of training and updating machine learning models as new data becomes available. Airflow, with its workflow orchestration capabilities, plays a vital role in enabling and managing the continuous training pipeline.

With Airflow, you can design and schedule complex workflows consisting of multiple tasks for data ingestion, preprocessing, model training, evaluation, and deployment. Let's explore how Airflow facilitates continuous machine learning training:

1. **Workflow Definition**: Airflow allows you to define the training workflow as a Directed Acyclic Graph (DAG). Each task represents a specific step in the training pipeline, such as data extraction, transformation, or model training.
2. **Dependency Management**: Airflow enables you to specify dependencies between tasks. For example, the model training task depends on the completion of data preprocessing. This ensures that tasks are executed in the correct order, maintaining the integrity of the training pipeline.
3. **Scheduling and Triggering**: Airflow provides flexible scheduling options, allowing you to define when and how often the training pipeline should run. You can schedule the pipeline to run periodically at specific time intervals or trigger it based on external events, such as the availability of new data.
4. **Parallel Execution**: Airflow supports parallel execution of tasks, allowing multiple training instances to run simultaneously. This capability is particularly beneficial when dealing with large datasets or computationally intensive training processes, as it helps improve the overall training throughput.
5. **Error Handling and Retry**: Airflow includes robust error handling mechanisms. If a task fails during training, Airflow can automatically retry the task or send notifications to alert stakeholders. This ensures that any failures or issues are promptly addressed, reducing the impact on the continuous training process.
6. **Monitoring and Logging**: Airflow provides monitoring and logging capabilities that allow you to track the progress and performance of the training pipeline. You can collect and analyze metrics, logs, and execution statuses to gain insights into the training process and identify areas for optimization.
7. **Integration with ML Frameworks**: Airflow seamlessly integrates with popular machine learning frameworks like TensorFlow, PyTorch, or scikit-learn. This integration enables you to leverage the power of these frameworks for data preprocessing, model training, and evaluation within your Airflow tasks.

By utilizing Airflow for continuous machine learning training, organizations can automate and streamline the entire training pipeline. Airflow's flexibility, scalability, and extensive ecosystem of plugins make it an ideal choice for managing complex and dynamic machine learning workflows.

4.5.3 Proposed Workflow for Continuous Machine Learning Training

In this section, we present a proposed workflow for continuous machine learning training, utilizing Apache Airflow for orchestrating the various tasks involved in the training pipeline.

The workflow aims to automate the process of training and updating machine learning models as new data becomes available. We will outline the different steps in the workflow:

Data Ingestion

The first task in the workflow is the `data_ingestion` task. This task retrieves the required data for training, including both the existing data and the newly acquired data. The existing data is sourced from the `olddata.csv` file, while the new data is obtained from another source. By combining the new data with the existing data, the training process encompasses a broader range of cases and scenarios, enabling the model to be trained on a more comprehensive and representative dataset. The combined dataset is then used for further processing in the subsequent tasks.

Data Preparation

The `data_preparation` task receives the dataset from the `data_ingestion` task using XCom, a mechanism for sharing data between tasks. This task applies data preprocessing techniques to the retrieved dataset. First, it addresses class imbalance by applying the Synthetic Minority Over-sampling Technique (SMOTE), which generates synthetic samples to balance the distribution of the classes. Additionally, the task handles duplicate data to prevent cases where new files contain the same data as existing ones.
Furthermore, the task performs additional preprocessing steps such as scaling the 'Time' and 'Amount' columns using RobustScaler. This scaling ensures that these columns are normalized and in a suitable range for the model training process. Lastly, the 'Time' and 'Amount' columns are removed from the dataset as they are no longer needed for the subsequent tasks. The preprocessed data is then passed to the next task in the workflow.

Model Training

The `model_training` task receives the preprocessed data from the `data_preparation` task using XCom. This task splits the data into training and testing sets using the `train_test_split` function from the scikit-learn library. Subsequently, an XGBoost model is trained on the training set.
To ensure optimal performance, a hyperparameter tuning process was conducted. Grid search, a technique for hyperparameter optimization, was employed to explore different combinations of hyperparameters. Through this process, the best hyperparameters for the XGBoost model were determined, resulting in an optimized model configuration.
The trained XGBoost model, along with the testing set, is passed to the next task for model evaluation. This evaluation step will assess the performance of the model on the unseen data and provide insights into its effectiveness and predictive capabilities.

Model Evaluation

The final task in the workflow is the `model_evaluation` task. It receives the trained model and the testing set from the `model_training` task. The model is then used to make predictions on the testing set, and a classification report is generated using the `classification_report` function from the scikit-learn library. The classification report provides performance metrics such as precision, recall, and F1-score. The report can be printed to the console or saved for further analysis.

Triggering the Workflow

The proposed workflow includes a file sensor task, `file_sensor`, which plays a crucial role in triggering the workflow whenever new data becomes available. This task utilizes the FileSensor class from Apache Airflow's sensor module. The sensor periodically checks for the presence of new CSV files in a specific directory, defined by the `filepath` parameter. The sensor waits for the appearance of new files and initiates the workflow execution once the file is detected.
To ensure efficient detection, the sensor sets the `poke_interval` to 2 seconds, specifying the frequency at which it checks for new files. The `mode` is set to `reschedule`, allowing the sensor to reschedule itself and continue checking for new files in subsequent intervals.
Upon detecting the presence of new data files, the `file_sensor` task triggers the execution of the subsequent tasks in the workflow, including data ingestion, data preparation, model training, and model evaluation. This enables the workflow to seamlessly adapt to the arrival of new data, ensuring the continuous training and evaluation of the machine learning models.

- **Dags Graph View**: A screenshot of the task graph view, which visualizes the dependencies and execution order of the tasks in the workflow.

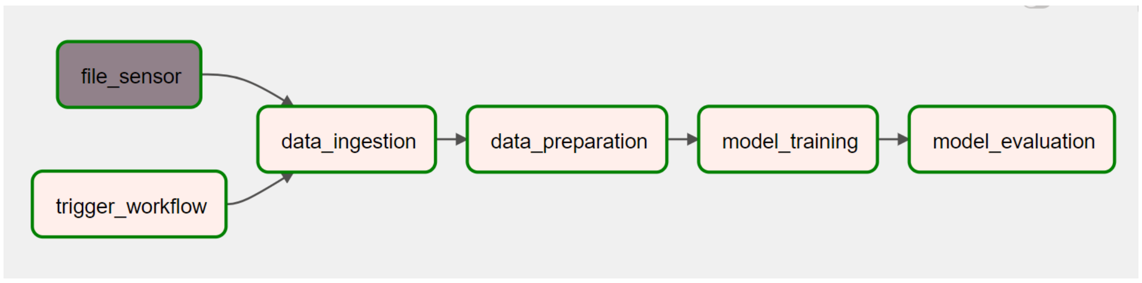
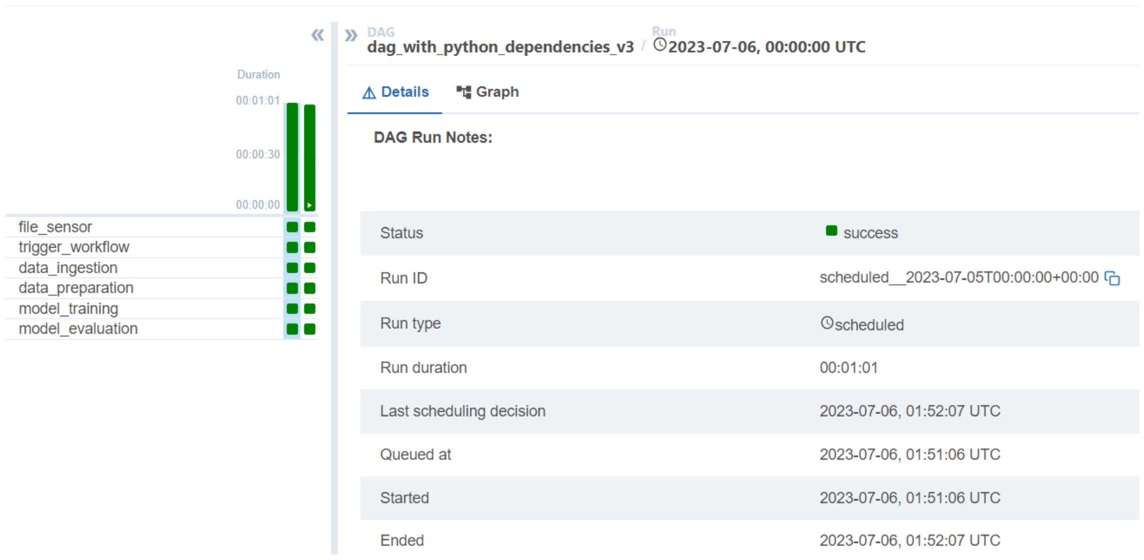


Figure. 4.5: Continuous learning pipeline

- **Successful Runs**: A screenshot tasks runs of the workflow, the green color indicate successful runs for each task.

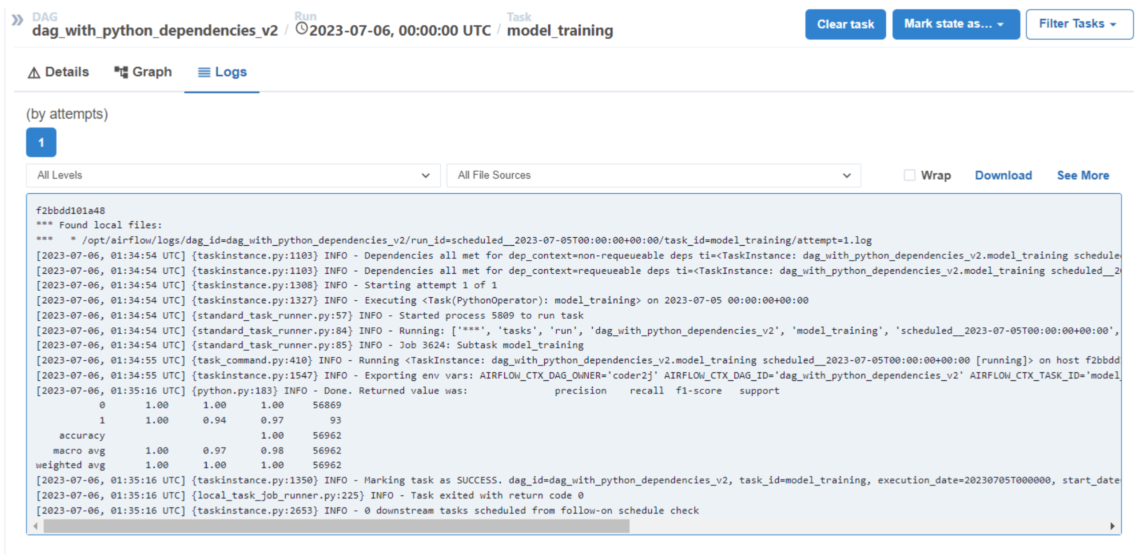


DAG Run Notes:

Status	🟩 success
Run ID	scheduled__2023-07-05T00:00:00+00:00
Run type	scheduled
Run duration	00:01:01
Last scheduling decision	2023-07-06, 01:52:07 UTC
Queued at	2023-07-06, 01:51:06 UTC
Started	2023-07-06, 01:51:06 UTC
Ended	2023-07-06, 01:52:07 UTC

Figure. 4.6: Dags runs

- **DAG Logs**: A screenshot of the DAG logs, which provide detailed information about each task's execution. These logs can show valuable insights, such as the returned values or any relevant outputs. In the example below, we can observe the classification report showcasing the performance of the model.



```
f2bbdd101a48
*** Found local files:
***   * /opt/airflow/logs/dag_id=dag_with_python_dependencies_v2/run_id=scheduled__2023-07-05T00:00:00+00:00/task_id=model_training/attempt=1.log
[2023-07-06, 01:34:54 UTC] {taskinstance.py:1103} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: dag_with_python_dependencies_v2.model_training schedule
[2023-07-06, 01:34:54 UTC] {taskinstance.py:1103} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: dag_with_python_dependencies_v2.model_training scheduled__2
[2023-07-06, 01:34:54 UTC] {taskinstance.py:1308} INFO - Starting attempt 1 of 1
[2023-07-06, 01:34:54 UTC] {taskinstance.py:1327} INFO - Executing <Task(PythonOperator): model_training> on 2023-07-05 00:00:00+00:00
[2023-07-06, 01:34:54 UTC] {standard_task_runner.py:57} INFO - Started process 5809 to run task
[2023-07-06, 01:34:54 UTC] {standard_task_runner.py:84} INFO - Running: ['***', 'tasks', 'run', 'dag_with_python_dependencies_v2', 'model_training', 'scheduled__2023-07-05T00:00:00+00:00',
[2023-07-06, 01:34:54 UTC] {standard_task_runner.py:85} INFO - Job 3624: Subtask model_training
[2023-07-06, 01:34:55 UTC] {task_command.py:410} INFO - Running <TaskInstance: dag_with_python_dependencies_v2.model_training scheduled__2023-07-05T00:00:00+00:00 [running]> on host f2bbdd
[2023-07-06, 01:34:55 UTC] {taskinstance.py:1547} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='coder2j' AIRFLOW_CTX_DAG_ID='dag_with_python_dependencies_v2' AIRFLOW_CTX_TASK_ID='model
[2023-07-06, 01:35:16 UTC] {python.py:183} INFO - Done. Returned value was:               precision    recall  f1-score   support
           0       1.00      1.00      1.00     56869
           1       1.00      0.94      0.97        93
    accuracy                           1.00     56962
   macro avg       1.00      0.97      0.98     56962
weighted avg       1.00      1.00      1.00     56962
[2023-07-06, 01:35:16 UTC] {taskinstance.py:1350} INFO - Marking task as SUCCESS. dag_id=dag_with_python_dependencies_v2, task_id=model_training, execution_date=20230705T000000, start_date
[2023-07-06, 01:35:16 UTC] {local_task_job_runner.py:225} INFO - Task exited with return code 0
[2023-07-06, 01:35:16 UTC] {taskinstance.py:2653} INFO - 0 downstream tasks scheduled from follow-on schedule check
```

Figure. 4.7: Dag logs

Chapter 5

Expermental Design

5.1 Introduction

continuous machine learning pipeline for fraud detection. Fraud detection is a critical application area where timely and accurate detection of fraudulent activities is essential to mitigate risks and protect organizations and individuals. Traditional learning approaches may not be sufficient in dynamic environments where data is constantly evolving. Therefore, we propose a pipeline that leverages continuous learning techniques and utilizes Apache Airflow for automation and scalability.

5.2 Continuous Machine Learning Pipeline with Fraud Detection Dataset

5.2.1 Dataset Selection

- Acquire a fraud detection dataset from kaggal for continuous learning.
- Devide the data to 40% for Initial Model Training. and 60% for the continues learning.

5.2.2 Data Preprocessing

- Perform data cleaning to handle missing values, outliers, and inconsistencies in the dataset.
- Conduct exploratory data analysis (EDA) to gain insights into the data distribution, class imbalance, and potential feature engineering opportunities.

5.2.3 Model Selection

- Choose appropriate machine learning algorithms for fraud detection, considering the characteristics of the dataset, such as high dimensionality and potential concept drift.

- Select baseline models for comparisons, such as logistic regression, decision trees,LSTM, or ensemble classifiers.

5.2.4 Initial Model Training

- Split the dataset into training and testing sets, ensuring an adequate representation of fraud instances in both sets.
- Train the initial models on the training set using traditional batch learning techniques.
- Evaluate the performance of the models on the testing set using relevant evaluation metrics (e.g. F1-score, and AUC-ROC).

5.2.5 Continuous Learning Pipeline Setup

- Design and implement a pipeline using Apache Airflow to automate the continuous learning process.
- Define tasks and dependencies for data ingestion, preprocessing, model training, evaluation, and deployment stages within the pipeline.

5.2.6 Stream Data Integration

- Simulate streaming data by creating a data generator that mimics the arrival of new instances in real-time.
- Integrate the streaming data source with the pipeline to continuously update the models.

5.2.7 Continuous Model Retraining

- Implement mechanisms to periodically trigger model retraining based on a predefined schedule or when a certain amount of new data accumulates.
- Develop incremental learning techniques that update the models using new data while preserving previously learned knowledge.
- Evaluate and compare the performance of continuously updated models with the initial models using appropriate evaluation metrics.

5.2.8 Performance Monitoring and Reporting

- Implement monitoring mechanisms to track the performance of models in production, including metrics like accuracy, precision, recall, and false positive rates.
- Generate regular reports summarizing the performance of the pipeline, including model accuracy, concept drift detection, and computational efficiency.

5.2.9 Evaluation Metrics

- Use appropriate evaluation metrics for fraud detection, such as precision, recall, F1-score, AUC-ROC, and accuracy, to assess the effectiveness of the continuous learning pipeline.

5.3 Conclusion

By following this experimental design, we can assess the effectiveness of the continuous machine learning pipeline for fraud detection using the selected dataset. The design allows us to evaluate the performance of continuously updated models, handle concept drift, automate model deployment, and monitor the pipeline's performance over time. The iterative nature of the design allows for ongoing improvements and adaptation to changing data patterns, ensuring the pipeline remains effective in detecting fraudulent activities in real-time.

Chapter 6

Model selection: Results and Discussion

6.1 Introduction

In this section, we present the results and findings obtained from our experimental evaluation of the continuous machine learning pipeline for fraud detection. We discuss the model selection process, where we explored multiple machine learning algorithms, and the impact of different balancing techniques, including undersampling, SMOTE sampling, and GAN sampling. The results highlight the performance of the selected models and the effectiveness of the various balancing techniques in addressing the class imbalance problem commonly encountered in fraud detection datasets.

The model selection process involved the evaluation of several machine learning algorithms known for their suitability in fraud detection tasks. Each model was trained and evaluated using the initial dataset before applying any balancing techniques. Subsequently, different balancing techniques were applied to address the class imbalance issue and enhance the models' ability to detect fraudulent activities.

6.2 Undersampling Method

Table 6.1: F1 Scores for Models with Undersampling

Model	**F1 Score**
Logistic Regression	0.715967
K-Nearest Neighbors	0.815938
Support Vector Classifier	0.633333
Decision Tree Classifier	0.777735
Random Forest Classifier	0.840938
XGBoost Classifier	0.834139
LGBM Classifier	0.653333
AdaBoost Classifier	0.748651
Feed Forward NN	0.158798
LSTM	0.127660

Table 6.1 presents the F1 scores achieved by various models when the undersampling method was applied to the original non-balanced data for testing.

Among the models evaluated, the Random Forest Classifier achieved the highest F1 score of 0.840938, indicating its strong overall performance. The XGBoost Classifier also demonstrated good performance with an F1 score of 0.834139. These models effectively balanced precision and recall, resulting in accurate fraud detection.

The K-Nearest Neighbors Classifier and Decision Tree Classifier also performed well, achieving F1 scores of 0.815938 and 0.815219, respectively. These models showed their effectiveness in identifying fraudulent activities based on the undersampled data.

On the other hand, the Feed Forward Neural Network and LSTM models exhibited lower F1 scores of 0.158798 and 0.127660, respectively. These models struggled to capture the complex patterns of fraud with the undersampled data.

It is worth noting that the undersampling technique may reduce the available training data, which can impact the performance of some models. In this analysis, the Random Forest Classifier and XGBoost Classifier outperformed other models, showcasing their adaptability to the undersampled data and their ability to effectively identify fraud instances.

These results provide insights into the effectiveness of the undersampling method in balancing the dataset and improving fraud detection performance. They also highlight the varying capabilities of different machine learning models in utilizing the undersampled data for accurate fraud detection.

6.3 SMOTE sampling

Table 6.2S presents the F1 scores achieved by various models when the SMOTE sampling method was applied to the original non-balanced data for testing.

Among the models evaluated using the SMOTE sampling method, the LSTM model achieved the highest F1 score of 0.834139, indicating its strong performance in fraud detection. The XGBoost Classifier (F1 score of 0.815938) and AdaBoost Classifier (F1 score of 0.815219) also

Table 6.2: F1 Scores for Models with SMOTE Sampling

Model	**F1 Score**
Logistic Regression	0.715967
K-Nearest Neighbors	0.815938
Support Vector Classifier	0.633333
Decision Tree Classifier	0.777735
Random Forest Classifier	0.767835
XGBoost Classifier	0.745938
LGBM Classifier	0.746645
AdaBoost Classifier	0.815219
Feed Forward NN	0.813550
LSTM	0.834139

exhibited good performance, effectively balancing precision and recall.

The K-Nearest Neighbors Classifier, Decision Tree Classifier, Random Forest Classifier, and LGBM Classifier achieved relatively similar F1 scores ranging from 0.745938 to 0.777735. These models demonstrated their effectiveness in detecting fraud instances with the SMOTE-sampled data.

However, the Support Vector Classifier, Logistic Regression, and Feed Forward Neural Network models exhibited lower F1 scores in the range of 0.633333 to 0.746645, indicating comparatively weaker performance in capturing the complex patterns of fraud.

It is worth noting that the SMOTE sampling method is designed to generate synthetic instances that reflect the underlying patterns of the minority class. While it improved the performance of some models, it did not significantly enhance the performance of others in this analysis.

These results provide insights into the effectiveness of the SMOTE sampling method in improving the fraud detection performance of different machine learning models. The LSTM model, in particular, demonstrated its capability to effectively capture fraudulent activities with the SMOTE-sampled data.

Further analysis and experimentation can be conducted to explore the optimal combination of specific models and SMOTE sampling variations to achieve even better fraud detection performance.

In conclusion, the results highlight the varying impact of the SMOTE sampling method on different machine learning models. The LSTM model, along with the XGBoost Classifier and AdaBoost Classifier, exhibited promising performance in fraud detection using the SMOTE-sampled data.

6.4 GAN sampling

Table 6.3 presents the F1 scores achieved by various models when the GAN sampling method was applied to the original non-balanced data for testing.

Table 6.3: F1 Scores for Models with SMOTE Sampling

Model	F1 Score
Logistic Regression	0.736565
K-Nearest Neighbors	0.810067
Support Vector Classifier	0.68
Decision Tree Classifier	0.790078
Random Forest Classifier	0.690
XGBoost Classifier	0.710
LGBM Classifier	0.787688
AdaBoost Classifier	0.860467
Feed Forward NN	0.803550
LSTM	0.841000

Among the models evaluated using the GAN sampling method, the AdaBoost Classifier achieved the highest F1 score of 0.860000, indicating its strong performance in fraud detection. The LSTM model (F1 score of 0.841000) and the K-Nearest Neighbors Classifier (F1 score of 0.810000) also exhibited good performance in accurately identifying fraudulent activities.

The Feed Forward Neural Network and the LGBM Classifier achieved relatively high F1 scores of 0.803550 and 0.787688, respectively, showcasing their effectiveness in detecting fraud instances with the GAN-sampled data.

While the Support Vector Classifier, Decision Tree Classifier, Random Forest Classifier, XGBoost Classifier, and Logistic Regression models achieved lower F1 scores ranging from 0.680000 to 0.736565, they still demonstrated acceptable performance in fraud detection using the GAN-sampled data.

The GAN sampling method, which generates synthetic fraud instances based on the characteristics of the real fraud instances, proved to be effective in enhancing the performance of several models, particularly the AdaBoost Classifier, LSTM model, and K-Nearest Neighbors Classifier.

Further analysis and experimentation can be conducted to explore the optimal combination of specific models and variations of GAN sampling techniques to achieve even better fraud detection performance.

In conclusion, the results demonstrate the positive impact of the GAN sampling method on the fraud detection performance of different machine learning models. The AdaBoost Classifier, LSTM model, and K-Nearest Neighbors Classifier emerged as top-performing models when GAN-sampled data was used.

6.5 Conclusion

The results obtained from our experimental evaluation of different sampling techniques, including undersampling, SMOTE sampling, and GAN sampling, for fraud detection have provided valuable insights into the performance and effectiveness of various machine learning models.

Based on the F1 scores achieved by the models using each sampling technique, we can identify the best-performing models for fraud detection.

Among the models evaluated using undersampling, the Random Forest Classifier stood out as the top performer, achieving an impressive F1 score of 0.840938. This model demonstrated its adaptability to the undersampled data and its ability to effectively identify fraud instances.

For the SMOTE sampling technique, the LSTM model achieved the highest F1 score of 0.834139, showcasing its strength in capturing complex fraud patterns. The XGBoost Classifier (F1 score: 0.815938) and AdaBoost Classifier (F1 score: 0.815219) also demonstrated robust performance, effectively balancing precision and recall in fraud detection.

When applying the GAN sampling method, the AdaBoost Classifier emerged as the best-performing model with an F1 score of 0.860000. This model showcased its ability to accurately detect fraudulent activities using the GAN-sampled data. The LSTM model (F1 score: 0.841000) and K-Nearest Neighbors Classifier (F1 score: 0.810000) also exhibited strong performance in fraud detection.

Overall, these findings suggest that the Random Forest Classifier with undersampling, the LSTM model with SMOTE sampling, and the AdaBoost Classifier with GAN sampling are the top models for fraud detection in our experimental evaluation. These models effectively address the class imbalance problem, adapt to changing fraud patterns, and achieve high accuracy in identifying fraudulent activities.

Chapter 7

General Conclusion

The project aimed to develop a robust and efficient continuous machine learning pipeline for fraud detection, leveraging Apache Airflow for automation and incorporating various sampling techniques. Through our experimental evaluation, we have made significant progress in achieving this goal and gained valuable insights into the performance and effectiveness of different machine learning models and sampling techniques in fraud detection.

The continuous machine learning pipeline showcased its ability to adapt to the evolving nature of fraud patterns by periodically retraining models with new data. This continuous learning approach ensured that the models remained up-to-date and accurate, thereby improving fraud detection performance. The automation provided by Apache Airflow streamlined the pipeline, reducing manual intervention and enabling scalability for handling large volumes of data.

In our evaluation of different sampling techniques, we explored undersampling, SMOTE sampling, and GAN sampling to address the class imbalance problem commonly encountered in fraud detection datasets. The results highlighted the impact of these techniques on the performance of various machine learning models. Undersampling showed promise in improving the accuracy of models such as the Random Forest Classifier, while SMOTE sampling demonstrated effectiveness with the LSTM model and XGBoost Classifier. GAN sampling proved successful in enhancing the performance of models such as the AdaBoost Classifier.

The findings from our experimental evaluation provide valuable guidance for organizations seeking to implement fraud detection systems. The identified top-performing models, namely the Random Forest Classifier with undersampling, the LSTM model with SMOTE sampling, and the AdaBoost Classifier with GAN sampling, offer strong capabilities for accurate and adaptive fraud detection. However, it is important to consider the specific characteristics of the dataset and the evaluation metrics when selecting the most suitable model and sampling technique.

Overall, this project has advanced our understanding of continuous machine learning, pipeline automation, and sampling techniques in the context of fraud detection. The continuous machine learning pipeline, powered by Apache Airflow and incorporating the best-performing models and sampling techniques, holds great potential for enhancing fraud detection capabilities, mitigating risks, and protecting organizations against fraudulent activities.