

University of Modena and Reggio Emilia

Department of Engineering "Enzo Ferrari"

Bachelor's degree in computer engineering

MLOps

Practices to deploy Machine Learning models into production

Pratiche per implementare modelli di Machine Learning in produzione

Candidate

Giorgia Bertacchini
141810

Supervisor

Prof. Nicola Bicocchi

Academic Year 2021 - 2022

Summary (Italian language)

Introduzione a MLOps

Machine Learning Operations (MLOps) è una disciplina ingegneristica che raggruppa pratiche per progettare, sviluppare e mantenere le applicazioni di apprendimento automatico. MLOps è stato progettato per facilitare l'installazione del software ML in un ambiente di produzione. E' in grado di combinare le complessità dei modelli di Machine Learning con le complessità delle moderne aziende.

Gli algoritmi di Machine Learning generano i modelli di ML a partire dai dati di addestramento. Questi algoritmi hanno l'obiettivo di trovare una rappresentazione sintetica dei dati che vengono loro forniti. I modelli di ML possono generare previsioni corrette solo quando i dati richiesti assomigliano ai dati di addestramento.

MLOps deriva dall'applicazione della metodologia dettata da DevOps in ambito dell'apprendimento automatico. Il movimento DevOps ha iniziato a svilupparsi tra il 2007 e il 2008. La parola DevOps è la combinazione dei termini "sviluppo" e "operazioni", rispettivamente quindi "development" e "operations". DevOps descrive l'adozione dello sviluppo iterativo del software, dell'automazione e della distribuzione e manutenzione dell'applicativo prodotto.

Per rimanere al passo coi tempi e per trarre vantaggi provenienti dall'applicazione del Machine Learning, le aziende moderne vogliono ampliare l'uso dell'apprendimento automatico. Purtroppo molte imprese non dispongono di processi di ciclo di vita standardizzati causando spesso il fallimento di molti progetti di ML. Viene quindi in loro aiuto MLOps, che aumenta la capacità di installare con successo nuovi modelli di Machine Learning nelle organizzazioni.

I principi fondamentali di MLOps

Per la realizzazione di modelli di Machine Learning durevoli e tracciabili è importante attenersi ai principi di MLOps, i più importanti dei quali sono descritti in seguito.

Il progetto deve essere l'implementazione iterativa di tre fasi, quali la progettazione di applicazioni basate sul ML, la sperimentazione e sviluppo del ML e

le operazioni di ML. Il software deve essere automatizzato, che corrisponde al concetto di maturità del progetto per la quale esistono tre livelli per i processi di Machine Learning. Maggiore è la maturità maggiore sarà la velocità di formazione di nuovi modelli. Il principio di "Continuous Deployment" è un sistema che distribuisce automaticamente un altro servizio, in questo caso il servizio di predizione del modello. Un servizio di deployment fornisce in genere orchestrazione, registrazione, monitoraggio e notifica per garantire la stabilità dei modelli di ML, del codice e dei dati. Per permettere il confronto e l'analisi approfondita dei modelli è necessario implementare il "versioning" di dati e di modelli. Questo permette di tenere traccia di tutti gli esperimenti effettuati dai data scientist riguardanti i modelli. Simile a questo principio è presente anche la riproducibilità, quindi la capacità di ripetere facilmente lo stesso esperimento. Questa capacità richiede che il modello sia accompagnato da una documentazione dettagliata, da dati di addestramento e di test, dall'implementazione del modello e dalle specifiche complete dell'ambiente in cui è stato eseguito. Durante il monitoraggio in produzione del modello, vengono calcolate quattro metriche importanti, quali la frequenza di distribuzione, "lead time for changes" (tempo per passare dal codice impegnato alla produzione), "mean time to restore" (tempo per ripristinare il servizio quando si verifica un incidente di servizio o un difetto) e "change fail percentage" (percentuale di modifiche apportate alla produzione o al prodotto rilasciato).

Ultimo concetto molto importante in MLOps è la collaborazione tra il personale dell'azienda. All'interno dell'intero processo che riguarda MLOps sono presenti persone esperte in campi tra loro eterogenei. MLOps coinvolge in particolare data scientist, sviluppatori DevOps, ingegneri del software e architetti del Machine Learning. Tutte queste figure devono collaborare e, spesso, ciascuno deve essere coinvolto in tutti gli step di progettazione e installazione del modello di ML.

I tre pilastri di MLOps

L'applicazione di MLOps si sviluppa su tre importanti concetti, quindi il "data engineering", il "model engineering" e il "model deployment".

Il primo è l'insieme delle seguenti fasi: "data ingestion", "exploration and validation", "data wrangling" e "data splitting". Per spiegare meglio, il "data

engineering" vuole collezionare tabelle di dati da diverse sorgenti, per estrarne il materiale utile per il modello da sviluppare successivamente e adattarlo nella forma più conveniente. L'ultimo intervento consiste nello spartire il risultato ottenuto in tre set di dati detti "training", "validation" e "test", ciascuno utilizzato in maniera diversa nel seguito. Concetto che rappresenta bene lo scopo di tale prima fase è il "feature engineering", cioè il processo che parte dai dati grezzi e li trasforma in "feature" che rappresentano meglio il problema da risolvere.

Il "model engineering" si implementa attorno le seguenti fasi: "model training", "model evaluation", "model testing" e "model packaging". Si occupa in particolare di eseguire algoritmi di Machine Learning, di scrivere le pipeline e l'architettura del modello, di addestrare i modelli, di tracciare gli esperimenti eseguiti e di valutare sulla base di metriche. In più tratta anche la questione dell'esportazione del modello, quindi di impacchettarli in specifici formati che descrivono il modello alle aziende interessate a installarlo nel loro ambiente di produzione.

Il concetto di "model deployment" racchiude in sé le fasi: "model serving", "model performance monitoring" e "model performance logging". Dopo aver generato e valutato il modello di Machine Learning questo deve essere distribuito per poi essere messo in produzione. Il modello viene servito sotto forma di pacchetto contenente una collezione di artefatti, che sono il codice, gli iperparametri, il set di dati, il modello addestrato e la documentazione. A seguire è importante il monitoraggio del modello durante il suo utilizzo perché, in caso di suo degrado, venga richiesto un nuovo addestramento del modello.

Caso d'uso

Per sostenere e accompagnare MLOps nel suo obiettivo sono presenti numerosi strumenti di lavoro che vogliono porre l'attenzione verso i suoi principi e i suoi fondamenti, descritti precedentemente.

Ho quindi accompagnato lo studio di questa materia con un'implementazione pratica, che vuole mettere in atto la teoria concepita. Ho così notato che l'utilizzo di librerie e strumenti adeguati è sostanziale e questi permettono un'implementazione maggiormente strutturata e veloce. E' stata infatti importante la scelta accurata dei "tools" usati, spesso classificati in base alla tipologia di gestione eseguita.

Perché venga preso in considerazione l'intero ciclo di vita del modello di Machine Learning è possibile distinguere due fasi, una di sviluppo e una di produzione. In seguito spiego come ho implementato tali due fasi, soffermandomi sugli strumenti di lavoro usati, ritenuti più convenienti.

Implementazione della fase di sviluppo

La fase di sviluppo comprende il tema della ingegneria dei dati, dell'ingegneria del modello di Machine Learning e della sua relativa esportazione.

Per costruire la struttura base di tale progetto ho installato e inizializzato un progetto, generato dall'orchestratore Kedro. Questo strumento di lavoro permette di automatizzare il flusso di lavoro e di definire la pipeline dei compiti. Mette anche a disposizione un'interfaccia utente che mostra in maniera grafica tale pipeline, costituita da funzioni e dai relativi input e output.

Per iniziare ho salvato il dataset che verrà manipolato per poi essere usato per addestrare e valutare il modello. Perché questo dataset originale venga archiviato per permettere la riproducibilità del modello di machine learning, ho installato e inizializzato DVC, data version control. Il suo utilizzo permette una facilitata esportazione e archiviazione dei dati utili nel personale drive di Google.

Per implementare al meglio il data engineering, nominato precedentemente, ho creato una pipeline di Kedro con funzioni riguardanti l'esportazione, la validazione e la pulizia dei dati. Per semplificare tale lavoro ho usato lo strumento Pandas per agevolare l'analisi e la manipolazione dei dati, adottando convenientemente il concetto di DataFrame. A fine di questa pipeline ricavo il dataset lavorato contenente solo le informazioni utili per la costruzione del modello.

Per quanto riguarda lo sviluppo e l'addestramento del modello di Machine Learning ho scritto un'ulteriore pipeline e ho impiegato la libreria Scikit-learn o sklearn. Questa si basa sugli iperparametri che riceve e si occupa anche di valutare il modello e di calcolare le relative metriche. Così eseguendo anche questa pipeline otteniamo un modello di machine learning addestrato e pronto all'uso.

Per consentire il tracciamento delle metriche e dei modelli di machine learning prodotti, ho installato e adoperato lo strumento di lavoro MLflow. Per specificare quali informazioni e modello voglio che tenga traccia, nella pipeline

riguardante il model engineering ho specificato i log che deve salvare, come quali file, quale modello, quali parametri, quali valori di metriche e altri tag utili. Il suo utilizzo è favorito dalla cooperazione che si instaura a pennello con l'orchestratore Kedro, rendendo la loro unione complementare. Anche MLflow riserva una propria interfaccia dalla quale vedere l'intera collezione di esperimenti e le loro specifiche.

Altro strumento da me usato è BentoML, che semplifica la preparazione del modello e il suo servizio di previsione. Quindi fornisce una propria interfaccia utente per il servizio che esso crea quando gli viene importato il modello di ML. Infatti richiedendo il comando di "build" a BentoML viene creato un bento, cioè un file con modello, servizio, codice sorgente, configurazioni di dipendenze e altro. Per distribuire il bento, è possibile porlo in un container attraverso un semplice metodo di BentoML e si ricava un'immagine di docker del bento.

Per favorire la distribuzione delle pipeline di Kedro ho adottato il plugin kedro-docker, che ne crea direttamente l'immagine Docker. Come ultimo si fa presente che è implementata anche una applicazione Flask capace di rispondere a certe API utili per la fase di produzione, approfondita successivamente.

Implementazione della fase di produzione

La fase di produzione racchiude in sé il compito di offrire il servizio di predizione, monitorare il modello e, se serve, allertare l'utente. Questo progetto parte usando il servizio di predizione nel bento costruito nella fase precedente. Quindi attivando il servizio da Docker possiamo interagire con questo e mandargli richieste di predizione. Per le azioni di ricerca di "data drift" e di "output drift" che vengono implementate nel seguito, è necessario che si conosca il dataset sul quale il modello è stato precedentemente addestrato. Quindi per riprendere il training dataset ho fatto uso, di nuovo, di DVC che lo scarica dal personale Drive di Google dove era stato archiviato.

Per quanto riguarda il monitoraggio sono partita installando lo strumento Evidently, in particolare per il calcolo di drift, o derive dei dati. Sfruttando la collezione di test e di report appartenenti, ad Evidently, ho ricavato i loro risultati esportandoli in tre pagine HTML. Questi ultimi possono poi in futuro essere archiviati o distribuiti ad altri settori a seconda delle esigenze dell'azienda.

Ma per avere il monitoraggio in real-time del modello e dei servizi è necessaria la piattaforma Grafana. Esistendo una integrazione di Evidently con Grafana e Prometheus sono partita dal progetto di integrazione reperibile dalla documentazione ufficiale di Evidently, al quale ho apportato le necessarie modifiche.

Perché Prometheus raccolga le metriche da varie sorgenti, ho impostato nelle sue configurazioni il servizio di predizione bento, il servizio di evidently e il servizio di alertmanager. Affinché il software avverta l'utente di anomalie e di drift, ho implementato varie regole di Prometheus, ad esempio di "dataset drift", di "feature drift" e di un servizio non attivo. Offre quindi un'interfaccia grafica dalla quale l'utente può effettuare query per avere indietro i valori di tali metriche da esso raccolti. Contiene anche una pagina di avvisi che informa quando un allarme si è attivato.

Legato a esso ho usato Alertmanager, che gestisce tali segnalazioni e manda notifiche di alertamenti a certe destinazioni, che possono essere ad esempio una email, una chat di Slack o Telegram. Anche questo ultimo strumento dispone di una piattaforma per l'utente.

Per avere finalmente l'interfaccia con i dati real-time raccolti da Prometheus ho fatto uso dello strumento Grafana. Grazie a questo ultimo gli utenti, anche senza capacità informatiche, possono creare e personalizzare le proprie dashboard adattandole al meglio alle proprie esigenze.

Pensando che tale progetto fosse destinato a un utilizzo da parte di tecnici dell'azienda che monitorano il modello in produzione, ho scritto anche una applicazione usando la libreria Streamlit. Gli utenti attraverso tale applicazione possono richiedere operazioni alla fase di sviluppo, dalle azioni più piccole fino a chiedere la costruzione di un nuovo modello di Machine Learning. Questa può essere meglio adattata alle esigenze dell'azienda o al tipo di utente che ne andrà a fare uso.

Table of contents

1. Introduction	10
1.1 Machine Learning	11
1.2 DevOps	11
1.3 Machine Learning project life cycle	12
1.4 Machine Learning Engineering (MLE)	12
1.5 MLOps in enterprises	12
1.6 Gartner report	13
2. The principles of MLOps	14
2.1 Iterative-Incremental Process	14
2.2 Automation	15
2.3 Continuous Deployment	15
2.4 Data and model versioning	15
2.5 Reproducibility	16
2.6 ML-based Software Delivery Metrics	16
2.7 Collaboration	16
3. The three pillars of MLOps	18
3.1 Data engineering	18
3.1.1 Data ingestion	18
3.1.2 Exploration and validation	19
3.1.3 Data wrangling (cleaning)	19
3.1.4 Data splitting (or data partitioning)	19
3.2 Model engineering	20
3.2.1 Model training	20
3.2.2 Model evaluation	20
3.2.3 Model testing	21
3.2.4 Model packaging	21
3.3 Model deployment	22
3.3.1 Model serving	22
3.3.2 Model performance monitoring	22
3.3.3 Model performance logging	23

4. MLOps processes and tools	24
4.1 Workflow orchestration	24
4.2 Data versioning	26
4.3 Data analysis and manipulation	27
4.4 Model training	28
4.5 Experimentation management	29
4.6 Model packaging and serving	31
4.7 Drift and model monitoring	32
4.8 Systems monitoring and alerting	34
4.9 Alert management	36
4.10 Managed observability platform	37
4.11 Building application	38
5. Use case	39
5.1 MLOps stack canvas	39
5.2 MLOps stack template	40
5.3 Develop phase	40
5.4 Production phase	45
6. Conclusion	52
Bibliography	54

Chapter 1

Introduction

Machine Learning Operations (MLOps) defines practices to design, develop, and maintain Machine Learning applications. It can be easily used in many environments since it has the principle of being language, framework, platform and infrastructure agnostic. MLOps has the aim to solve the complexities of Machine Learning models and the complexities of the modern organization.

MLOps has the goal to assist the installation of Machine Learning software in a production environment. "Machine Learning" (or ML) combines two concepts, artificial intelligence and data science fields. This is to say that the aim of a Machine Learning project is to build a statistical model by using collected data and applying Machine Learning algorithms to them.

It derives from the application of the methodology dictated by DevOps in the field of Machine Learning, thus defining effective practices and processes for the design, construction and deployment of Machine Learning models in production.

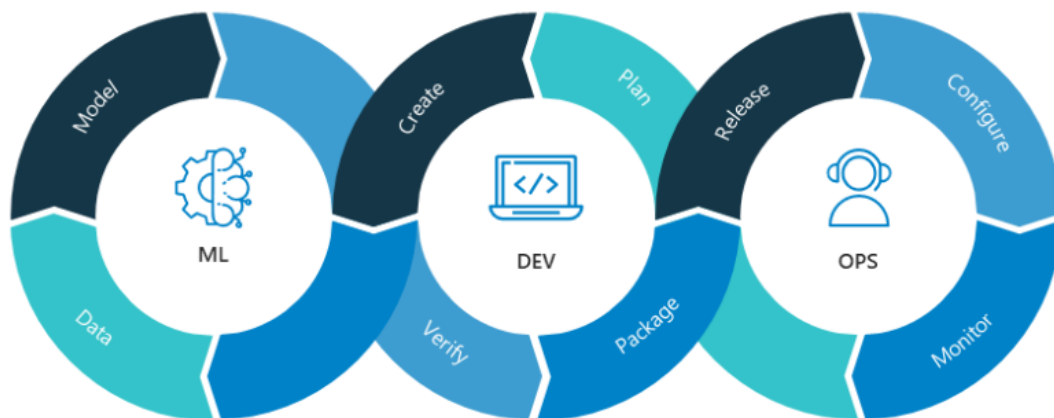


Figure 1.1 - MLOps combine Machine Learning, Applications development and Operations.

Both DevOps and MLOps pipelines include a code-validate-deploy loop. But the MLOps pipeline also includes additional steps regarding data and models that are necessary to build/train a Machine Learning model.

1.1 Machine Learning

Machine Learning software, as opposed to traditional software that is explicitly programmed, are computer algorithms that automatically learn and improve from experience. ML-type algorithms produce a software model capable of making predictions, useful in certain cases. An important step in ML is the training of the model through sample data, known as "training data", because its accuracy depends on it. If you have a problem with numerous elements with different representations, the application of Machine Learning (deep learning) is advisable.

Machine Learning algorithms build Machine Learning models from the training data. Such models are based on statistical theory. These algorithms have the goal to find a synthetic representation of the data given to them, and these data represent the world as it was at the time of collection. Only when their synthetic representation is still valid (thus when the future resembles the past), can the Machine Learning models generate correct predictions.

1.2 DevOps

The term MLOps is defined as:

"the extension of the DevOps methodology to include Machine Learning and Data Science assets as first-class citizens within the DevOps ecology".

Quote 1.1 - <https://github.com/tdcox/mlops-roadmap/blob/master/MLOpsRoadmap2020.md>

MLOps is the design of the DevOps extension that establishes effective practices and processes around designing, building, and deploying ML models into production. The DevOps movement started to coalesce somewhere between 2007 and 2008. DevOps is a fusion of the terms "development" and "operations". DevOps is the methodology based on iterative software development, automation, and programmable infrastructure deployment and maintenance. The implementation of software changes and updates are simplified by DevOps. This movement founded on the principles of:

- continuous integration and continuous delivery or deployment (CI/CD);
- robust automation and trust between teams;
- the idea of collaboration and increased communication between teams;

- the end-to-end service life cycle (build, test, release).

1.3 Machine Learning project life cycle

As a first activity of model development it is good practice to pay attention to business questions. From these the business needs are defined, avoiding the risk of starting the Machine Learning life cycle process trying to solve problems that don't serve the business.

The life cycle of a Machine Learning project involves the following steps:

1. goal definition,
2. data collection and preparation,
3. feature engineering,
4. model training,
5. model evaluation,
6. model deployment,
7. model serving,
8. model monitoring and maintenance.

1.4 Machine Learning Engineering (MLE)

In addition to the term MLOps, we can use the expression "Machine Learning Engineering" (MLE). MLE encompasses all stages from data collection to model building to make the model available for use by the product or consumers. MLE is the use of scientific principles, tools, and techniques from Machine Learning and traditional software engineering to design and build complex computer systems.

The Machine Learning engineer must be able to contribute to the entire life cycle of an ML project and understand how to work with the data, the model, and all aspects of it. In addition, the MLE often works with different stakeholders, so they often want to combine the work of the often disparate teams involved.

1.5 MLOps in enterprises

Is now widespread and confirmed that MLOps will deliver DevOps-like speed and agility to the ML lifecycle. Today's companies understand the potential

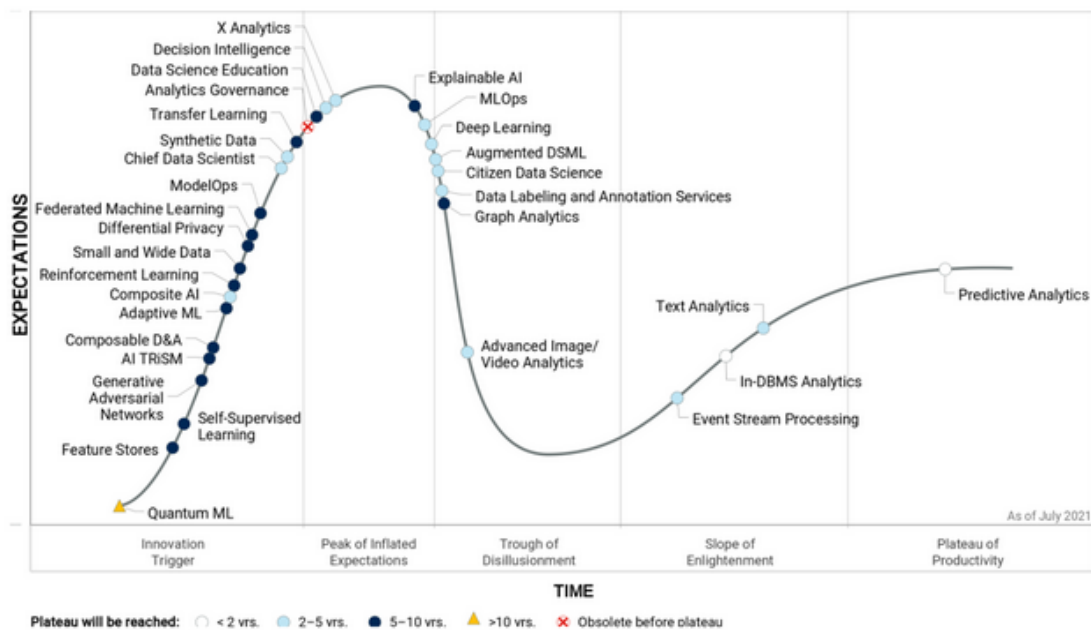
advantages of ML and want to expand its use. However, because many enterprises lack standardized life cycle processes, many data science projects fail.

MLOps increase the chance to successfully deploy new data science models. Incorporating these standardized processes and tools eliminate the need to deploy infrastructure, allowing data scientists and data engineers to focus on model development.

1.6 Gartner report

Below is a Hype Cycle model with respect to Data science and Machine Learning, where the maturity, adoption and application of specific technologies, such as MLOps, are graphically displayed. A Gartner Magic Quadrant is the culmination of research on a specific market, offering a wide-angle view of the relative positions of market competitors.

We can see that MLOps is still in the "Peak of Inflated Expectations" area but close to moving to the next stage, the so-called "Trough of Disillusionment". This means that MLOps is coming out of the prototype stage and its popularity is taking off. What happens then is that many companies are thinking of jumping into this new business, but most are afraid of the risks involved.



Gartner

Figure 1.2 - Hype Cycle for Data Science and Machine Learning, 2021

Chapter 2

The principles of MLOps

All MLOps principles are directed toward the realization of a development process for durable and traceable Machine Learning models. To approach the definition of MLOps, let us explore the principles already present in DevOps and apply them in a Machine Learning project.

2.1 Iterative-Incremental Process

The MLOps process is designed on the basis of three phases which are well linked and interconnected. These phases are "ML-powered application design", "ML experimentation and development", and "ML operations".

The first phase, "ML-powered application design", is dedicated to the study of the problem and the design of its Machine Learning solution and the subsequent evaluation of the project development. The attention is on understanding the business and the data, and designing the Machine Learning software. The design phase also has the aim to analyze the available data that will be used to train the model and specify the requirements needed to design the ML model architecture.

Experimentation is a useful procedure which can be carried out throughout the entire model development process. Experiments are intended to study various decisions or hypotheses in depth to draw conclusions about:

- the usefulness of the constructed model;
- the choice of the best parameters for the model (algorithms, hyperparameters, feature preprocessing, etc.);
- the calculation of the balance between model improvement and computational cost.

The last one, "ML operations", is based on DevOps practices which dictates principles and methods for testing, versioning, continuous delivery, and monitoring. The latter are necessary to deploy the Machine Learning model previously developed in production.

2.2 Automation

One of the MLOps principles is to automate the deployment of ML models in the main software system or as a service component. To make it automated, triggers are used for automated model training and deployment can include calendar events, messaging, monitoring events.

A parallel concept also appeared, ML process maturity, which represents the level of automation of the Data, ML Model, and Code pipelines. With increased maturity, the speed of the training of new models is also increased. There are three levels of automation of MLOps:

1. MLOps level 0 (Manual process)
2. MLOps level 1 (ML pipeline automation)
3. MLOps level 2 (CI/CD pipeline automation)

2.3 Continuous Deployment

The principle of Continuous Deployment (CD) is helpful when there are frequent model release deployments. In fact it concerns a system that should automatically deploy another service (model prediction service).

To define Model deployment, it is important to first specify the "ML assets", that is ML model, parameters and hyperparameters, training scripts, training and testing data. Managing deployment requires attention to the components, versioning, and dependencies of these Machine Learning artifacts. Often these Machine Learning artifacts are shipped in a service or in certain components of the infrastructure. A deployment service typically provides orchestration, logging, monitoring, and notification to guarantee the stability of ML models, code, and data.

2.4 Data and model versioning

The focus of versioning is to track ML models and datasets with version control systems. Since data scientists do a lot of experiments, they build, test and iterate on different versions of models in order for them to compare the models with any other versions they need to keep track of. Data versioning becomes a

requirement to deeply analyze predictions for each model version, so they are both important and connected.

To facilitate the versioning of ML model specifications, the use of VCS, or version control system, is recommended to make the training of ML models verifiable and reproducible.

2.5 Reproducibility

Reproducibility in MLOps refers to the ability to easily repeat the same experiment. It means that data scientists are able to go back to different stages of experiments to restore an earlier state of a project.

This ability requires that the model be accompanied by detailed documentation, training and test data, and an artifact representing the model implementation and full specifications of the environment in which it was run. Model metadata management, useful for ML reproducibility, includes algorithm type, features and transformations, data snapshots, hyperparameters, performance metrics, verifiable code from source code management, and training environment.

2.6 ML-based Software Delivery Metrics

In the monitoring phase, four main metrics are analyzed to measure and improve ML-based software delivery:

- Deployment Frequency, how often does your organization deploy code for production or to release it to end-users?
- Lead Time for Changes, how long does it take to go from committed code to successfully running code in production?
- Mean Time To Restore, how long does it generally take to restore service when a service incident or a defect that impacts users occurs?
- Change Fail Percentage, what percentage of changes to production or released product result in degraded service and requires remediation?

2.7 Collaboration

The picture below represents the current life cycle of a Machine Learning model inside an average organization. We can observe that it involves many different

people with completely different skill sets and who are often using entirely different tools. All these figures must collaborate and each must be involved in all steps of designing and installing the ML model. For example, collaboration between functional and technical teams allows for the diffusion of information across departments, enabling data scientists to introduce domain-specific knowledge into the model.

Related to the scope of ML the concept of HITL is used. Human-in-the-loop ML is a term used during the development process therefore one person is required to inspect, validate or modify some parts of the process in which to train and deploy a model into production. It includes all people who work with the data, including people who collect, label, and perform quality control (QC) on the data.

An example of work tools that facilitate communication are Miro or Mural. These are suitable for working collaboratively and are used when analyzing data product design.

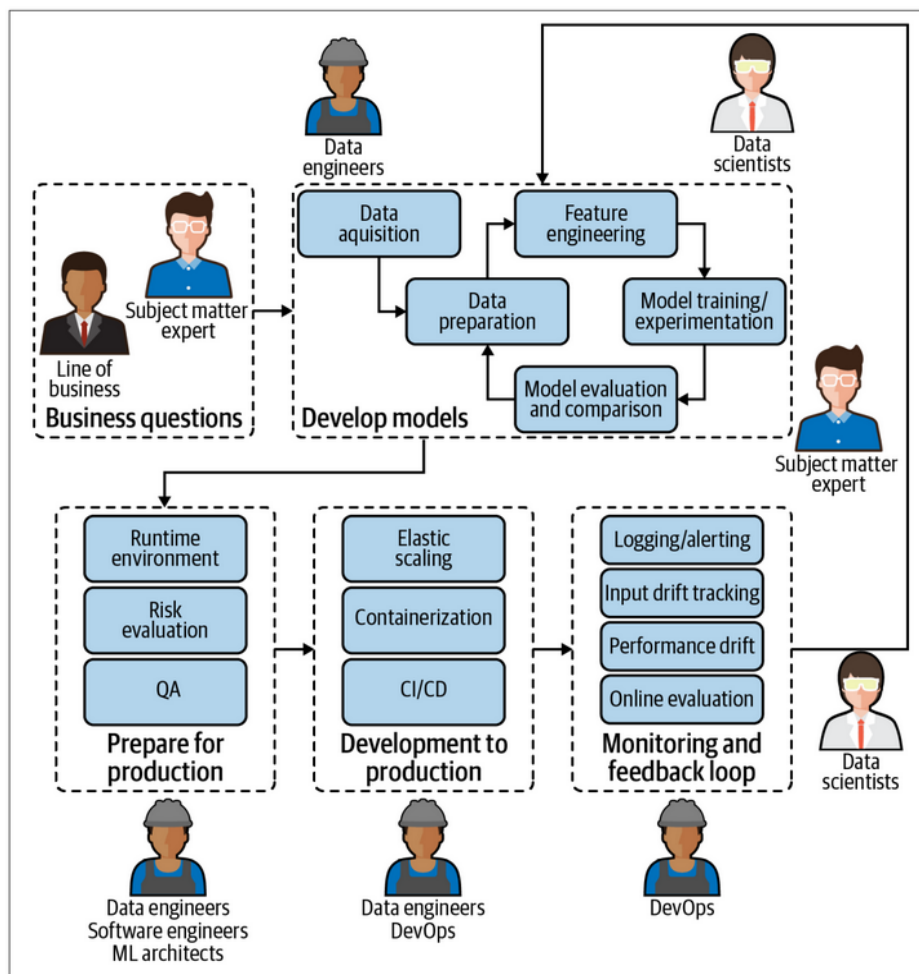


Figure 2.1 - All people involved in the life cycle of the machine learning model

Chapter 3

The three pillars of MLOps

MLOps is a process that helps organizations and business leaders to generate long-term value and to reduce risk associated with data science, Machine Learning, and AI initiatives. MLOps is the result of ModelOps, DataOps and DevOps, which fall into the three phases described below.

3.1 Data engineering

Data engineering is the phase of acquisition and preparation of the data for analysis. To begin with, data is collected from various resources and has different formats. Choosing good data sets has a huge impact on the quality and performance of the ML model. Thus, the performance of the ML model and, consequently, the production system is affected by the training data. Feature engineering is the process which starts with raw data and then transforms them into "features" that better represent the underlying problem to be solved. Features are the format and appearance of the data that are presented to a model so that they enable it to derive additional useful information.

Stages of the data engineering pipeline: data ingestion, exploration and validation, data wrangling (cleaning) and data splitting.

3.1.1 Data ingestion

Collecting data by using various frameworks and formats, such as internal/external databases, data marts, OLAP cubes, data warehouses, OLTP systems, Spark, HDFS, CSV, etc. To better explain, a data warehouse is a single and central data store that aggregates data from multiple sources. Data marts are like data warehouses but concerning a particular line of business or subject area. The OLAP (Online Analytical Processing) cube is an array multidimensional database that processes data much faster and more efficiently compared to a traditional relational

database. OLTP (online transaction processing) is essential in execution of a large number of database transactions by a large number of people, usually on the Internet.

This step might also include synthetic data generation or data enrichment.

3.1.2 Exploration and validation

Operations of data validation are user-defined functions of error detection which scan the dataset in order to spot some errors. The validation is a process of assessing the quality of the data by running dataset validation routines (error detection methods). When data scientists or analysts consider data sources to train a model, exploratory data analysis (EDA) steps in, which aims to first understand what the data look like.

The data exploration process can include: documentation regarding how data was collected and distributed and various actions such as cleaning, filling, reshaping, filtering, sampling, comparing etc.

3.1.3 Data wrangling (cleaning)

Data cleaning involves a process of reformatting or restructuring particular attributes and correcting errors in the data. For example, data with empty or null-valued elements, or duplicate data or data with values that are not possible for the problem, are considered and then transformed or deleted from the dataset.

The implementation of unit tests on data is recommended to avoid propagation of errors in the data to the next stage. Otherwise there are other ways to facilitate such data preprocessing. For example they can be implemented as simple SQL views or as lambda functions that operate on topics.

3.1.4 Data splitting (or data partitioning)

Splitting the data into training, validation, and test datasets to be used during the core Machine Learning stages to produce the ML model. The validation and test data are only used to calculate statistics reflecting the performance of the model.

The training set is used to train the model. It is the data the Machine Learning algorithm "sees". Instead, the data analyst uses the validation set to estimate the

performance of different Machine Learning algorithms or models when applied to new data. The remaining test set, which is also unseen by the learning algorithm, is used at the end of the project to evaluate and report the performance of the model.

3.2 Model engineering

Model engineering is about the Machine Learning workflow and its core is the writing and executing Machine Learning algorithms to obtain a ML model. It focuses on model architecture, Machine Learning pipelines, Machine Learning experiments, defining model metrics, model training and packaging. Overtime, changes occur in the actual data that then deviate from the data seen during model training. As a consequence, it happens that the performance of ML models in production degenerates. This problem is called "model decay".

Stages of the model engineering pipeline: model training, model evaluation, model testing and model packaging.

3.2.1 Model training

This is the process in which the ML model is trained. This function is done by educating the Machine Learning algorithm on the training data. It is essential to have training data as rich and varied as possible, so as to have better predictions generated by the model. To better define this training activity, it also includes feature engineering and hyperparameter tuning.

ML model training can be implemented in two forms: offline or online. In offline learning (or static learning) the model is trained on a set of already collected data. Once installed in production, the ML model slowly becomes obsolete. This phenomenon needs to be monitored and is called "model decay". Instead in online learning (or dynamic learning) the model is regularly retrained with updated data. therefore avoid becoming obsolete and making incorrect predictions.

3.2.2 Model evaluation

Model validation takes place before serving the ML model to the end user. This process is focused on ensuring that it meets the original objectives predicted.

It is often helpful to use a model evaluation store, which is a system that centralizes the data related to the life cycle of the model. A model version is obtained by training a model template on a given dataset, that aims to solve a business problem. The model evaluation archive has two main purposes: versioning of logical models over time. such an action needs all the essential information relating to its stage of formation; comparing the performance between different versions of the logical models, to decide which version to use.

3.2.3 Model testing

Performing the final "Model Acceptance Test" by using the hold backtest dataset to estimate the generalization error. Model testing is the application of the model to selected data to validate measurements against requirements. The metrics refer to two aspects:

1. statistical, such as accuracy, precision, F1 score, recall, false positives;
2. computational, such as average latency, 95th percentile of latency.

Going on to explain more about the major model statistics. F1 score considers both accuracy and recall. The F1 score can be considered as a weighted average of the precision and recall values. Accuracy is a measure calculated by dividing the number of correctly labeled annotations by the total number of annotations added by the ML model. Recall is a measure calculated by dividing the number of correctly labeled annotations by the number of annotations that should have been created.

Often this is tracked in order to be able to compare different versions of the model.

3.2.4 Model packaging

The process of exporting the final ML model into a specific format (e.g. PMML, PFA, or ONNX), which describes the model in order to be consumed by the business application.

PMML (Predictive Model Markup Language) is a format for model packaging based on XML. It uses a file with .pmml extension, that describes a model and pipeline in XML. PMML is not widely used for two reasons: it does not support all ML algorithms and it requires licensing which limits its use in open source tools.

PFA (Portable Format for Analytics) replaces PMML. A PFA document is text in JSON format that describes an executable. This last one has an input, an output and related functions.

ONNX (Open Neural Network eXchange) allows any ML tool to share a single model format, it is independent of the file format. This format is supported by many major technology companies such as Microsoft, Facebook and Amazon.

3.3 Model deployment

Once we train a ML model, attend the model deployment phase and we need to deploy it as part of a business application. The to-be-deployed models depend on the technology chosen and comprise a set of code and data artifacts. It is important that, during the production environment, the model finds the versions on which he depends because the use of different versions can cause the model's predictions to differ. The final stage of the Machine Learning workflow is the integration of the previously engineered ML model into the production phase.

This stage includes the following operations: model serving, model performance monitoring and model performance logging.

3.3.1 Model serving

It is the process of addressing the ML model artifact in a production environment.

A testable and distributable bundle of the project containing the code and data must be constructed. These bundles are usually called artifacts. There are various elements that need to be grouped into artifacts, which go through a test pipeline and are made available for production deployment, such as code, hyperparameters, dataset, trained model and documentation.

3.3.2 Model performance monitoring

After the ML model is distributed, the monitoring process takes over. This aims to make sure that the model performs as predicted, on the basis of real data and unpublished data.

One of the consequences to be most careful about is the deviation of the predictions from the previous model's performance. Receipt of such alerts may be followed by a trigger the retraining of the model with more representative data. ML models need to be monitored at two levels: at the resource level, and at the performance level.

3.3.3 Model performance logging

Log records are generated with each request and provide information about their states. These logs can be collected from different servers in the production environment. From this collection it is then possible to make further considerations about system performance.

An event log of a Machine Learning system is a record with a the following information:

1. model metadata, identification and version of the model;
2. model inputs, allowing for detection of data drift;
3. model outputs, give a concrete idea about the model performance in production;
4. model explanation, used in some highly regulated domains such as finance or healthcare.

Chapter 4

MLOps processes and tools

4.1 Workflow orchestration



Figure 4.1 - Kedro logo

This tool applies CI/CD methodology. The desire in MLOps is to automate the CI/CD pipeline as far as possible.

As Workflow orchestration is used Kedro, an open-source Python framework which helps while creating reproducible, maintainable and modular data science code. Kedro is a popular template for new data engineering and data science projects. This tool is used to organize all MLOps steps in a well-defined pipeline. Kedro is spread over three factors: data catalog, node and pipeline.

Data catalog makes the datasets declarative, rather than imperative. So all the information related to a dataset is highly organized. In the project Data Catalog is implemented in `conf/base/catalog.yml`. Here is defined each type, destination file path and if it is versioned about the data sets in output from the nodes. Here you can define all datasets using simple YAML syntax.

```
model_input_table:
  type: pandas.CSVDataSet
  filepath: data/04_feature/model_input_table.csv
  versioned: true
```

```
regressor:
  type: pickle.PickleDataSet
  filepath: data/06_models/regressor.pickle
  versioned: true
```

```
metrics:
  type: tracking.MetricsDataSet
  filepath: data/09_tracking/metrics.json
  versioned: true
```


Node is a Python function that accepts input and optionally provides outputs;

```
def preprocess_activities(activities: pd.DataFrame) -> Tuple[pd.DataFrame, Dict]:
    """Preprocesses the data for activities.

    Args:
        activities: Raw data.
    Returns:
        Preprocessed data and JSON file.
    """
    activities = _validation(activities)
    activities = _wrangling(activities)

    return activities, {"columns": activities.columns.tolist(), "data_type": "activities"}
```

Pipeline is a collection of nodes. It creates the Kedro DAG (Directed acyclic graph).

```
def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=preprocess_activities,
                inputs="activities",
                outputs=["preprocessed_activities", "activities_columns"],
                name="preprocess_activities_node",
            ),
            node(
                func=exploration_activities,
                inputs="activities",
                outputs="exploration_activities",
                name="exploration_activities_node",
            ),
            node(
                func=create_model_input_table,
                inputs=["preprocessed_activities", "params:table_columns"],
                outputs="model_input_table",
                name="create_model_input_table_node",
            ),
        ]
    )
```

Kedro also has a GUI, called Kedro-Viz. Kedro-Viz offers an interactive view of the entire pipeline, and interactive graphs in an informative way focusing on the connections between datasets and nodes. It is a tool that can be very useful for explaining graphically the code workflow to users.

Kedro-Viz also provides a page for live tracking and experiment tracking.

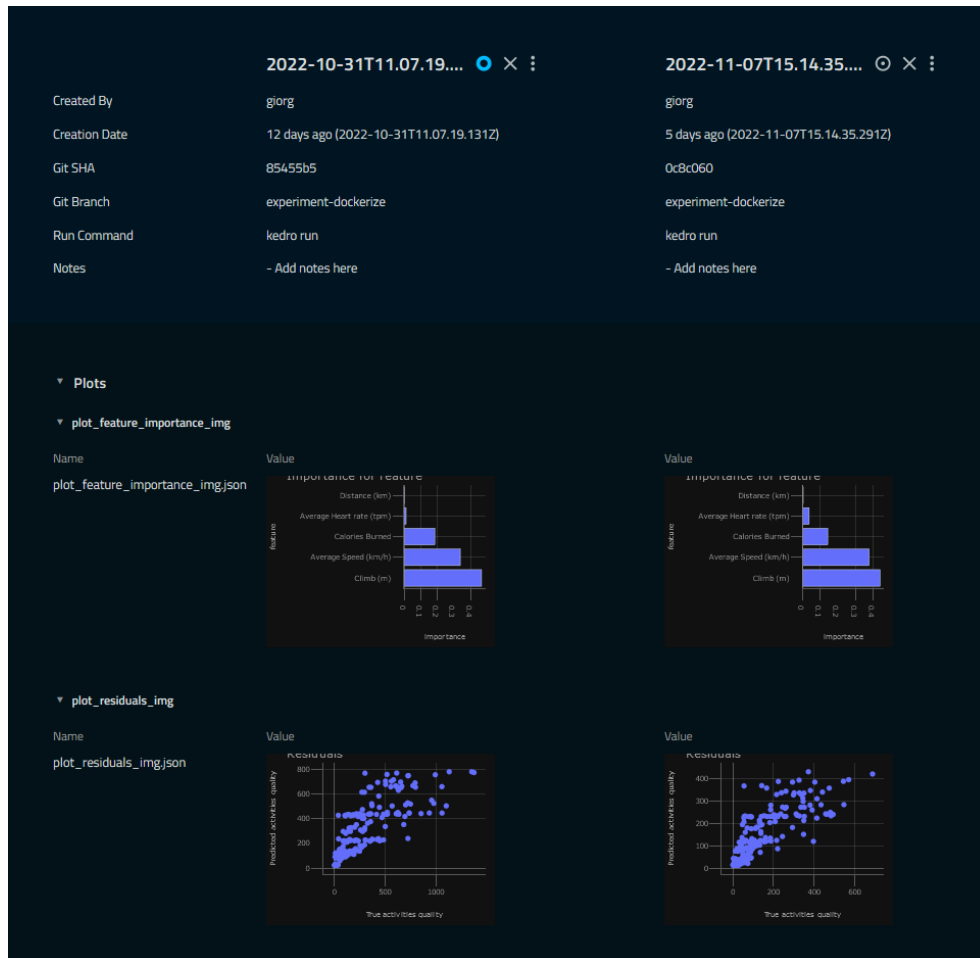


Figure 4.2 - Kedro-Viz shows a comparison between two my experiments.

4.2 Data versioning



Figure 4.3 - DVC logo

Versioning is essential to reproduce the experiments. Reproducibility in MLOps also involves the ability to easily rerun the exact same experiment.

DVC is a data versioning management. This provides a way to handle large files, data sets, machine learning models, and metrics.

Remote storage is necessary to set the remote storage of the data to save them. DVC stores information about the added file in a special .dvc file named "data/data.xml.dvc", this metadata file is a placeholder for the original data.

An example of Google Drive as remote storage:

```
[core]
  remote = storage
  autostage = true
['remote "storage"']
  url = gdrive://1LMUFVzJn4CNaqVbMsVGZazii4Mdxsanj
```

4.3 Data analysis and manipulation



Figure 4.4 - Pandas logo

Perform exploratory data analysis (EDA) is when data scientists or analysts consider available data sources to train a ML model.

Pandas is a Python package providing fast, flexible, and expressive data structures.

"It has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language.

It is already well on its way toward this goal."

Quote 5.1 - https://pandas.pydata.org/docs/getting_started/overview.html

Pandas will help the developer to explore, clean, and process data. In pandas, a data table is called a DataFrame. If the data table is 1-D, it is called Series. The functions of this tool about DataFrame are for example:

```
apps: pd.DataFrame

# Drop duplicates
apps.drop_duplicates(inplace = True)

# Calculate the MEAN, and replace any empty values with it
x = apps["Average Heart rate (tpm)"].mean()
apps["Average Heart rate (tpm)"].fillna(x, inplace = True)

# Clean rows that contain empty cells
apps.dropna(inplace = True)
```

```
activities: pd.DataFrame

totalNumber = activities.size
maxDistance = activities["Distance (km)"].max()
meanAverageSpeed = activities["Average Speed (km/h)"].mean()
minAverageHeartRate = activities["Average Heart rate (tpm)"].min()
```

4.4 Model training



Figure 4.5 - sklearn logo

The data scientist implements different algorithms with the prepared data to train various ML models. In addition, you subject the implemented algorithms to hyperparameter tuning to get the best performing ML model.

Scikit-learn, or sklearn, is an open source Machine Learning library. It is a tool heavily used in Machine Learning at the moment, and it also provides various tools for model fitting, data preprocessing, model selection, and model evaluation.

The main functions of this tool are listed below.

Split the dataset into validation, testing and training datasets:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, \
    test_size=parameters["test_size"], random_state=parameters["random_state"])
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, \
    test_size=parameters["val_size"], random_state=parameters["random_state"])
```

Model creation and fitting:

```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(max_depth=parameters["max_depth"], \
    random_state=parameters["random_state"])
regressor.fit(X_train, y_train)
```

Evaluate the model, with reference to its parameters:

```
from sklearn.model_selection import GridSearchCV

# define search space
space = dict()
space['max_depth'] = [1,2,3]
space['random_state'] = [41,42,43,44]

# define search
search = GridSearchCV(regressor, space, scoring='neg_mean_absolute_error')
# execute search
result = search.fit(X_train, y_train)
```

Calculate Machine Learning model metrics:

```
from sklearn import metrics

# MAE to measure errors between the predicted value and the true value.
mae = metrics.mean_absolute_error(y_val, y_pred)
# MSE to average squared difference between the predicted value and the true value.
mse = metrics.mean_squared_error(y_val, y_pred)
# ME to capture the worst-case error between the predicted value and the true value.
me = metrics.max_error(y_val, y_pred)
```

4.5 Experimentation management



Figure 4.6 - MLflow logo

ML experiment steps are orchestrated and done automatically. Experiment environment is used in the pre-production and production environment, which is a key aspect of MLOps practice for unifying DevOps.

MLflow is an open source platform for managing experiments on end-to-end Machine Learning projects. MLFlow can be very useful in terms of tracking ML

model metrics over time. MLFlow centralizes all these metrics and the models generated in its platform, allowing them to be visualized as well.

MLflow and Kedro are complementary tools which do not conflict: Kedro is the foundation of the project, building a pipeline of nodes; MLflow creates a centralized repository of metrics and progress over time.

Feature	Kedro	MLflow
Artifact Versioning	Yes	Yes
Metric Tracking	No	Yes
Parameter Versioning	Yes	Yes
Experiment Comparison	No	Yes
Code Organisation	Yes	No
Pipeline Construction	Yes	No
Pipeline Visualisation	Yes	No
Data Abstraction	Yes	No
Deployment	Yes	Yes

Figure 4.7 - How MLflow and Kedro balance each other out

Afterwards are the most used functions for logging useful information regarding the Machine Learning model.

Log file:

```
mlflow.log_artifact(local_path=os.path.join("data", "01_raw", "DATA.csv"))
```

Log model:

```
mlflow.sklearn.log_model(sk_model=regressor, artifact_path="model")
```

Log key-value param:

```
mlflow.log_param('test_size', parameters["test_size"])
mlflow.log_param('val_size', parameters["val_size"])
mlflow.log_param('max_depth', parameters["max_depth"])
mlflow.log_param('random_state', parameters["random_state"])
```

Log key-value metric:

```
mlflow.log_metric("accuracy", score)
mlflow.log_metric("mean_absolute_error", mae)
mlflow.log_metric("mean_squared_error", mse)
mlflow.log_metric("max_error", me)
```

Set key-value tag:

```
mlflow.set_tag("Model Type", "Random Forest")
```

Also MLflow has its own MLflow UI where we can see the logs made about the experiments. From the home page of the user interface of MLflow it is possible to see all the experiments carried out with their relative metadata and other information, set with the functions seen in the Key elements paragraph. The image below shows the page referring to a single experiment with all its details.

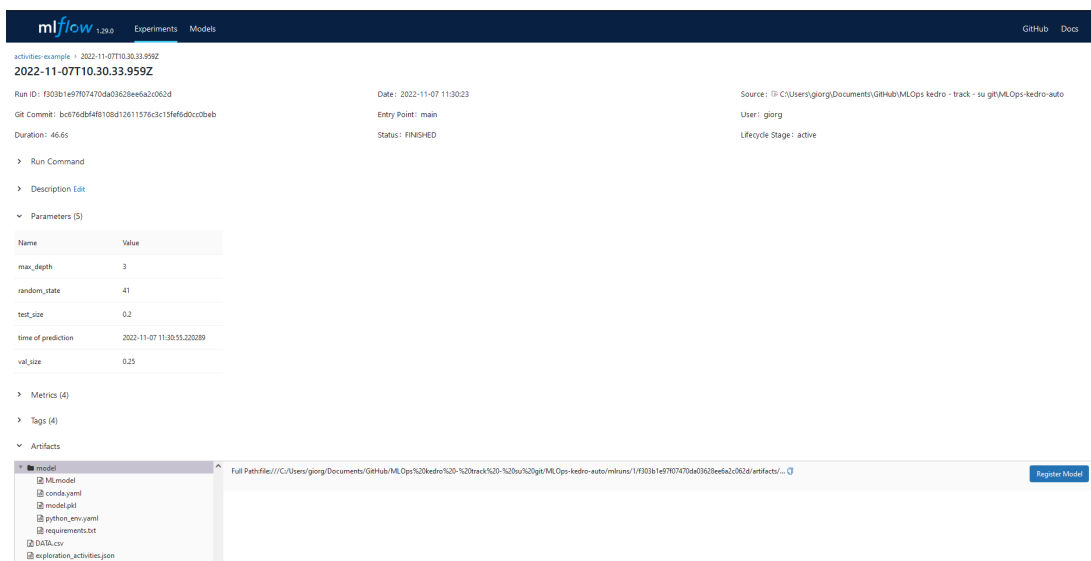


Figure 4.8 - MLflow interface shows the details of my each experiment.

4.6 Model packaging and serving



Figure 4.9 - BentoML logo

The validated model is deployed to a target environment to serve predictions. In Continuous Deployment (CD) the deployed system should automatically deploy the model prediction service.

The BentoML platform focuses on ML in the production environment. BentoML is intended only to serve and deploy trained models, independent of the model development environment.

BentoML works very well together with MLflow. MLflow focuses on loading a model, while BentoML provides an abstraction to build a prediction service. BentoML supports many essential model service features that are missing in MLflow, including multi-model inference, API server dockerization, built-in Prometheus metrics endpoint, and many more.

The BentoML basic steps are two: saving the machine learning model and creating a prediction service.

```
import bentoml

bentoml.mlflow.import_model("my_model", model_uri= os.path.join(os.getcwd(), 'my_model', dirname))


def predict(input_data: pd.DataFrame):
    with open(os.path.join("conf", "base", "parameters", "data_science.yml"), "r") as f:
        configuration = yaml.safe_load(f)
    with open('temp.json', 'w') as json_file:
        json.dump(configuration, json_file)
    output = json.load(open('temp.json'))

    parameters = {"header":output["model_options"]["features"]}
    input_data = create_model_input_table(input_data, parameters)
    input_data, dict_col = preprocess_activities(input_data)

    print("Start the prediction...")
    return model_runner.predict.run(input_data)
```

Bento is a user-oriented service packaged in a standardized format. It collects files and information necessary for running a BentoML, such as source code, models, data files, and dependency configurations. The three most common deployment options with BentoML are:

- generation of container images from Bento for custom docker deployment
- Yatai: Model Deployment at scale on Kubernetes
- bentocli: Fast model deployment on any cloud platform

From the interface about BentoML service the user can request a forecast.

4.7 Drift and model monitoring



Figure 4.10 - Evidently logo

The rationale is that if the data distribution (e.g., mean, standard deviation, correlations between features) diverges between the training and testing phases on one side and the development phase on the other, it is a strong signal that the model's performance won't be the same.

Evidently is a monitoring tool that helps evaluate, test, and monitor the performance of ML models applied in production. The tool generates interactive reports and tests from `pandas.DataFrame`.

Evidently have two integrations: with MLflow and with Grafana and Prometheus. Evidently with MLflow: Evidently is used to calculate the metrics and MLflow to log the results. From the MLflow interface you can access the metrics. Evidently with Grafana and Prometheus: Evidently is used to calculate the metrics, Prometheus is used to store the collected metrics, and Grafana is used to display the dashboards with metrics.

Evidently has three components: reports, tests, and monitors (in development).

The tests compare two sets of data: reference and current. They allow you to carry out ML model quality checks. As input they serve one or two datasets as `pandas.DataFrames` or `csv`. The output can take many forms, as an HTML inside a Jupyter notebook or Colab, as an exportable HTML file, as a JSON, or as a Python dictionary.

The reports calculate various metrics and provide rich interactive visualizations. As input, they serve one or two datasets as `pandas.DataFrames` or `csv`. The output can take many forms, as an HTML inside a Jupyter notebook or Colab, as an exportable HTML file, as JSON, or as a Python dictionary.

Evidently also has monitors that collect data and model metrics from a deployed ML service. The monitors are used in association with Prometheus and Grafana. Evidently calculates the metrics and outputs them in Prometheus format. Grafana dashboards are used to display them. As input, they serve POST live data from the ML service. The output is represented by data and quality metrics in the Prometheus format. If you decide not to use the help of the integration with Prometheus and Grafana, Evidently reports and tests are applied.

4.8 Systems monitoring and alerting



Figure 4.11 - Prometheus logo

Monitoring models need to collect metrics and to query on them.

Prometheus is a platform for monitoring application metrics and generating alerts based on these metrics. From this tool the user can make a query to receive the metrics grouped by it.

Prometheus is compatible with Docker and Kubernetes and is available on Docker Hub. The Prometheus server does not require much work to implement additional infrastructure or software because it does not depend on network storage or external services.

Typically, this tool monitors the following units with no integration with Evidently: CPU status, count request, duration request, and memory usage. Luckily, Prometheus can collect metrics from the Evidently source as well to provide drift detection data and other useful metrics.

BentoML offers integration with Prometheus. So Prometheus monitors and takes metrics about the BentoML service from the `"/metrics"` endpoint, from which essential metrics regarding ML models can be obtained. You can also create and customize new metrics if needed.

Each item you want to track is called "metric". The Prometheus server needs to read targets at a user-defined interval and collect metrics. The latter are stored in a time series database, which can be queried using the PromQL query language through the Prometheus browser interface. At the output of a query, you can see the results in the form of a table or graph.

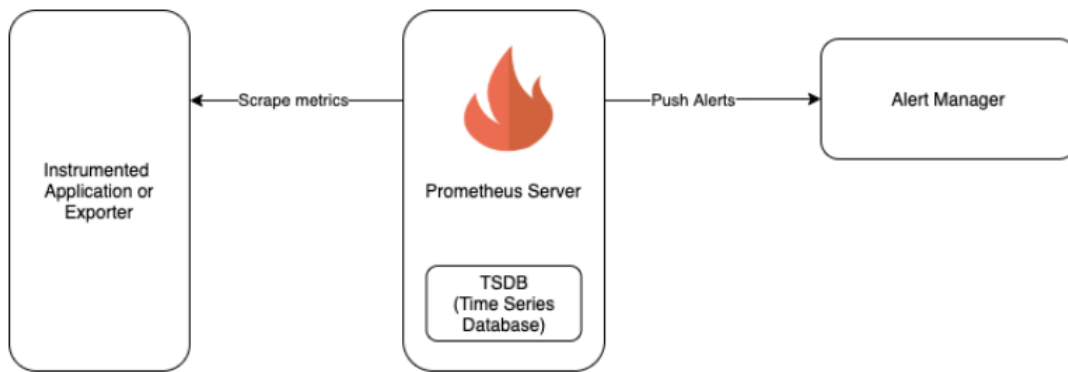


Figure 4.12 - Prometheus server, Alert Manager and Export.

To explain the picture above Prometheus has a server that reads and stores the metrics. Target is an application that exposes its metrics, or an exporter that exposes metrics of another application. Below the project code with specified targets.

```

# Load and evaluate rules in this file every 'evaluation_interval' seconds.
rule_files:
  - "prometheus_rules.yml"

alerting:
  alertmanagers:
    - scheme: http
      static_configs:
        - targets: [ 'alertmanager:9093' ]

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'
    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'service'
    scrape_interval: 10s
    static_configs:
      - targets: ['evidently_service.:8085']

  - job_name: 'bentoml'
    scrape_interval: 10s
    static_configs:
      - targets: ['bentoml.:3005']

  - job_name: 'alertmanager'
    scrape_interval: 10s
    static_configs:
      - targets: ['alertmanager.:9093']
  
```

Exporters are programs that ingest data from a variety of sources, and convert it to metrics that Prometheus can collect. Alertmanager generates alerts following pre-set rules. Below the project code with an example of a rule described.

```

- name: alert_rules
  rules:
    - alert: FeaturesDrift
      expr: evidently:data_drift:n_drifted_features > 0
      for: 1m
      labels:
        severity: critical
      annotations:
        summary: "Features drifting from (instance {{ $labels.instance }})"
        description: "The number of feature drifted is {{ $value }}"

```

From the Prometheus interface it is possible to make queries from the home page about the collected data on services and drifts. You can also see all the rules and which ones have been activated from its Alerts page

4.9 Alert management



Figure 4.13 - Prometheus logo.

To alert the data scientist of the change; that person can then diagnose the issue and evaluate the next course of action.

Alertmanager manages the alerts and sends out notifications via methods such as email, Slack, Webhook, Telegram. It takes care of deduplicating, grouping, silencing and inhibiting alarms. It is part of the Prometheus system and Alertmanager reacts to Prometheus rule activations.

It is possible to enrich the notification that is sent, by defining the alert annotations, to provide guidance to the humans handling the alert. These consist of a summary, i.e. a brief description of the problem, a description, i.e. a more extensive description of the symptom and its possible causes, and a dashboard, i.e. a link to the dashboard for the service.

Shown below is its home page where alerts triggered by Prometheus appear, and an example of an alert message that was sent to Slack's chat.

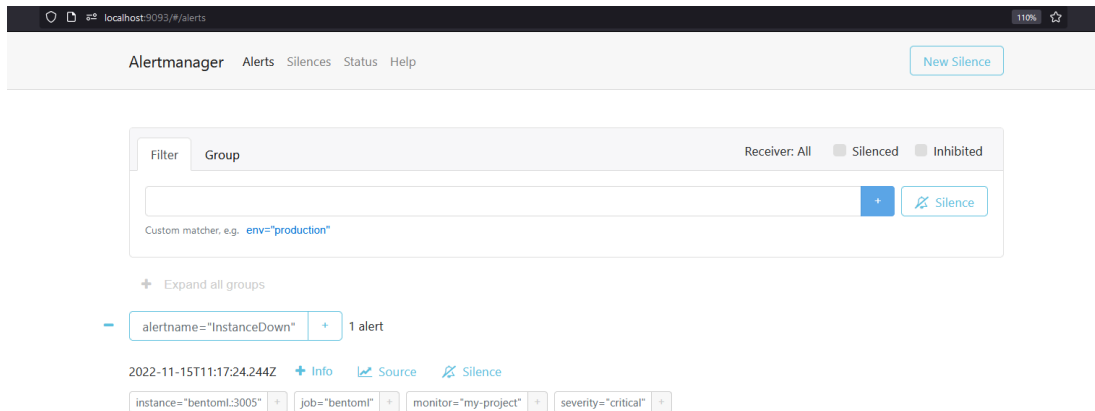


Figure 4.14- Home page of Alertmanager platform.

4.10 Managed observability platform



Figure 4.15 - Grafana logo.

Model monitoring: The model predictive performance is monitored to potentially invoke a new iteration in the ML process.

Grafana provides a platform for visualizing live. Grafana allows you to visualize monitoring metrics. It can visualize the results of monitoring work in the form of line graphs, heat maps, and histograms. You use Grafana GUI boards to request metrics from the Prometheus server and render them in the Grafana dashboard.

The code below is a part of the Grafana configuration where the data source is specified.

```
# list of datasources to insert/update depending
# what's available in the database
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus.:9090
```

From this tool the users can create their own dashboard with personalized graphs. Grafana dashboards are very flexible and easy to set up. You can also save the dashboards in JSON form to export them.

4.11 Building application



Figure 4.16 - Streamlit logo.

A drift alert can be followed by a trigger to the Develop phase to retrain and generate a new model.

Streamlit is an open-source Python library that facilitates building and deploying shareable web apps in minutes. It turns data scripts into customized, powerful, and shareable data apps.

Being a production-ready app framework, Streamlit offers the fastest way to build web apps for Machine Learning models.

Adding a widget is the same as declaring a variable. There is no need to write a backend, define routes, handle HTTP requests, connect a frontend, write HTML, CSS, JavaScript...

It is extremely valuable to visualize that data quickly, interactively, and from multiple different angles when you're working with data. You can display data via charts, and you can display it in raw form.

- Data display elements: DataFrame, Static tables, Metrics, Dicts, and JSON.
- Chart elements: Streamlit supports several different charting libraries, e.g. Matplotlib, and a few chart types that are "native" to Streamlit, e.g. `st.line_chart` and `st.area_chart`.

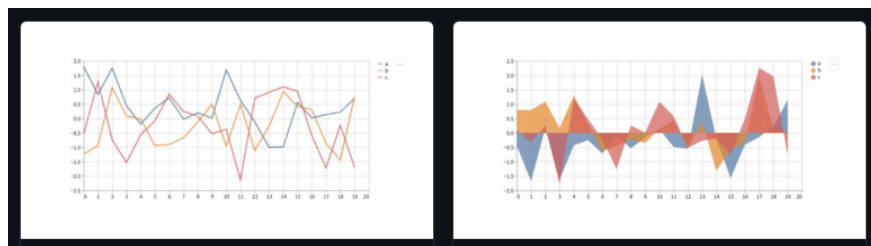


Figure 4.17 - Examples of Streamlit charts.

Chapter 5

Use case

The project is a practical implementation of MLOps practices. There are two projects that represent the phases throughout the entire life cycle of ML: develop and production.

To make it easier to consider what tools your organization could use to adopt MLOps and how these interact with each other, there are several templates and canvases for MLOps Stack.

5.1 MLOps stack canvas

As far as the first activity to be carried out in the MLOps cycle is regarded, it is recommended to use frameworks that guide the design of the MLOps infrastructure and its building blocks. MLOps Stack Canvas has the aim to help structure the workflows, architecture and infrastructure components for the MLOps stack in the ML project.

Project name: MLOps tirocinio			Team members: Giorgia Bertacchini		
Data analysis & experiment management For data analysis uses Pandas, a Python library. For experiment management uses MLFlow, an open source platform.	Data Sources & Data Versioning Data sources are static file in table format, as csv. Use specialized data versioning systems as DVC to save in a Google Drive folder the raw dataset. Another system use is Kedro that save each versions of intermediate and output data and of ml model.	Value Proposition This project would predict the quality of each case receive from csv file.	ML pipeline orchestration The DAG (directed acyclic graph) is create by Kedro tool, an open-source Python framework.	Model registry & Model versioning All model are collect in MLFlow ui and in BentoML platform.	
		MLOps Dilemmas Tooling: are all open-source. Platforms: using a hybrid solution, with more platforms.	Model & Data & Application Monitoring To observing the model there are Evidently and Prometheus tools, which can alerts user and Grafana provide a dashboard for users.	Model deployment ML model and its environment are deployment through BentoML, that create deployable artifacts (Bentos).	
		Foundations For source version control system uses GitHub.		Prediction serving BentoML provide a service, to require predictions to ml model in Bentos.	

Figure 5.1 - MLOps stack canvas regarding the project of this thesis.

5.2 MLOps stack template

This template is meant to streamline your workflow to a more manageable abstraction level. There are currently a lot of tools and technologies to help MLOps, and the MLOps Stack template is intended to help you understand which tools you need and how to connect them together. There are nine components in the stack which have varying requirements depending on the specific case of use.

Note that some technologies cover multiple components, whereas others are more singularly focused.

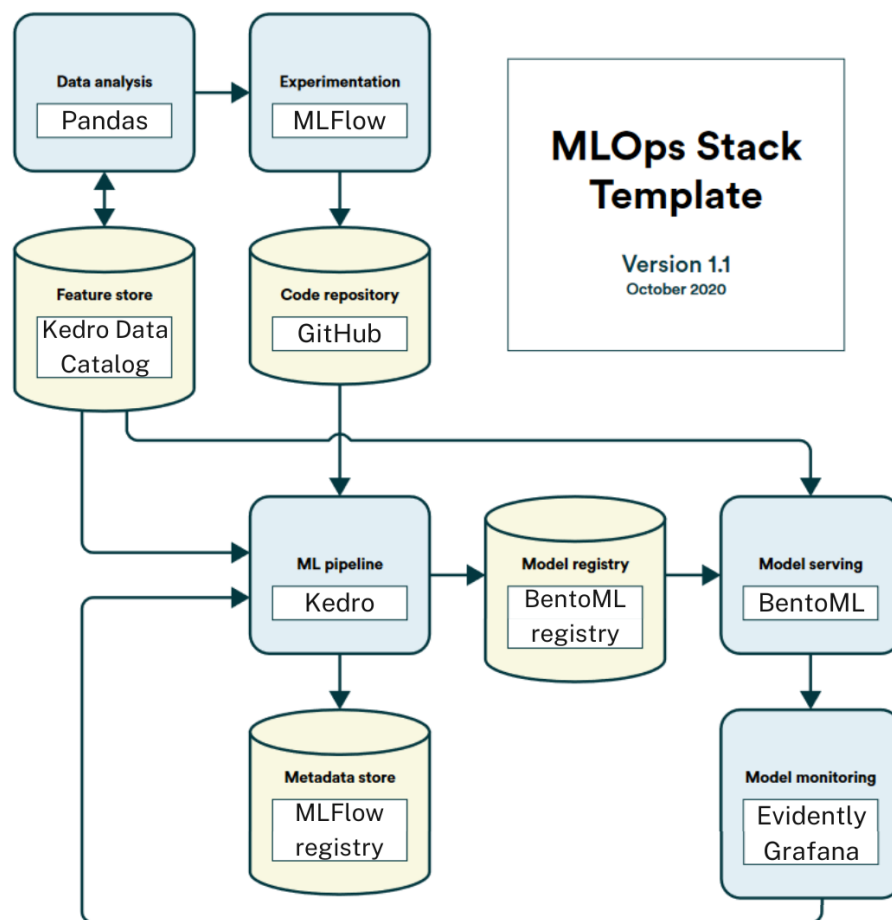


Figure 5.2 - MLOps stack template regarding the project of this thesis.

5.3 Develop phase

Themes implemented: data ingestion, data preparation, model building & training, experimentation, model deploy and model serving. The following image illustrates how the Develop phase works.

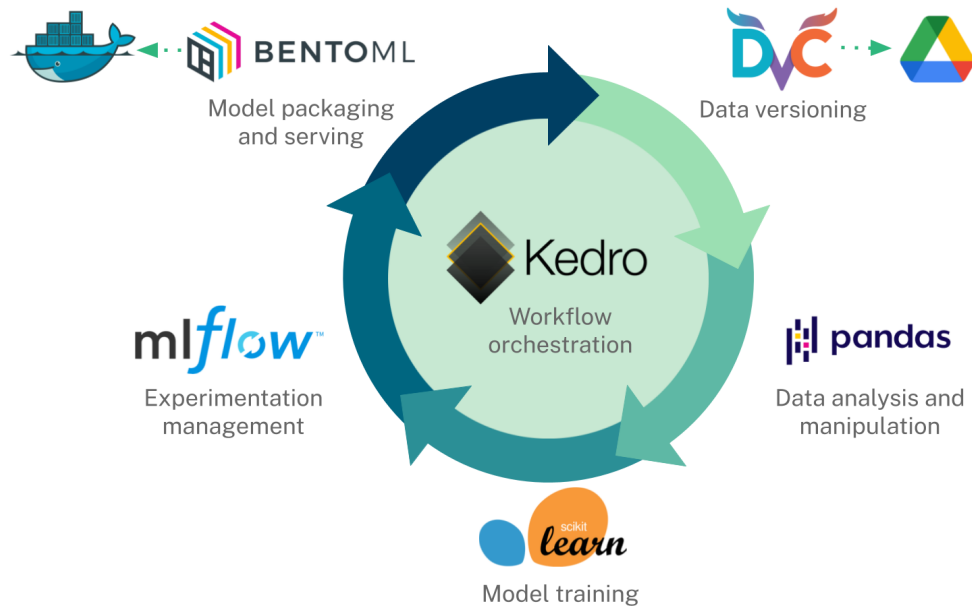


Figure 5.3 - Structure of the schema of the project development phase of this thesis.

Step 1.

To build the basic structure of this project, I installed Kedro and initialized a project generated by this tool. So I started with a software that was already organized in folders and capable of creating pipelines of functions and executing them in sequence.

Step 2.

To begin, I created a dataset that will be manipulated and then used to train and evaluate the model. This dataset should be saved in Kedro folder referring to raw data and marked in Kedro data catalog. I also stored it on Google Drive, to allow reproducibility of the ML model, with the help of DVC. To let the software know the destination, I set a Google personal Drive folder as remote storage in the DVC configuration file. Each time I add and push the file with DVC commands, Drive gets updated.

Step 3.

Moving on, the first macro step of MLOps to implement is data engineering. I then created a Kedro pipeline having the functions of exploration, validation, and data wrangling in this order. To simplify the implementation of these three nodes, as Kedro calls them, I installed Pandas library, which contains many useful methods for manipulating data tables using its DataFrames, which are widely used in Machine

Learning. At the end of this I obtained the processed dataset containing the useful information for building the model, also noted in Kedro data catalog.

Step 4.

The next macro step to be implemented is model engineering, which consists of construction, training and evaluation of the ML model. First I created an additional Kedro pipeline with new nodes and, to implement these functions, I used the scikit-learn tool for model fitting, evaluation and calculation of model metrics. I started from the previously manipulated dataset, which is always divided by it into the three validation, testing and training datasets. Related to this, I filled a Kedro file of parameters with hyperparameters used by the sk-learn methods.

At this point I was able to run the two Kedro pipelines, built in sequence, and I obtained a trained and ready-to-use Machine Learning model.

From Kedro user interface I can see the pipelines generated graphically.

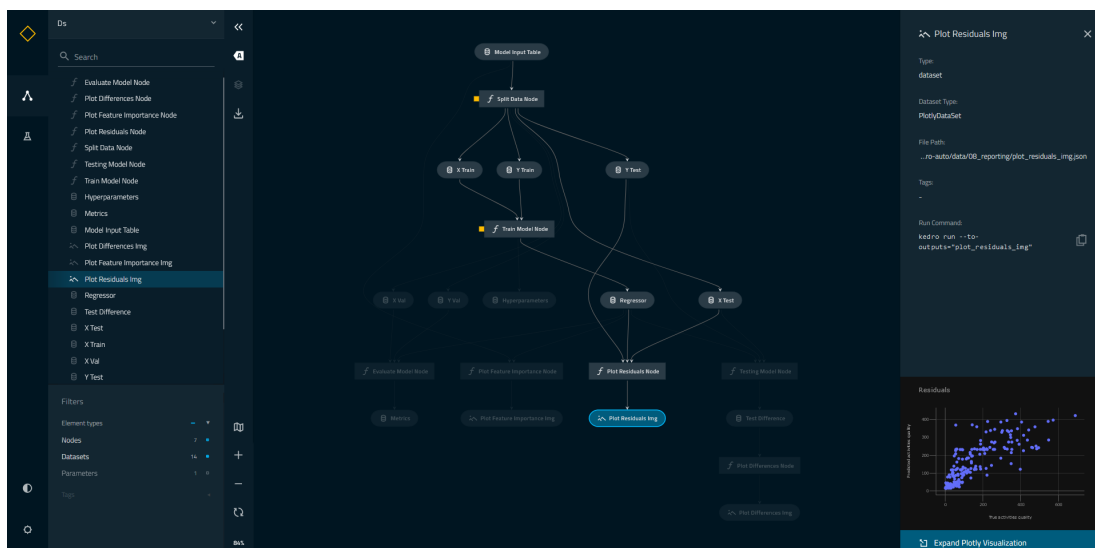
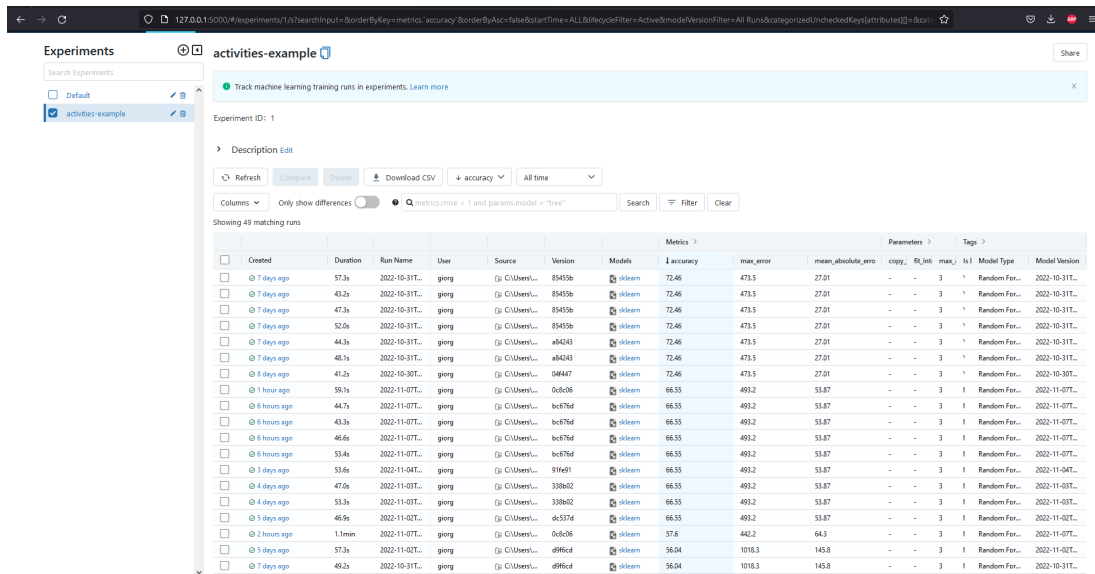


Figure 5.4 - Kedro-Viz shows a pipeline of my project with a viewable plot.

Step 5.

To keep track of all the machine learning models built, I installed MLflow. This creates an additional folder in the project, to save all the model versions and their information. In the pipeline of model engineering I specified MLflow logs referring to the model, parameters, metrics values, and other useful tags that I want to archive. To achieve this, in the MLflow configuration file I set up the entry point to run Kedro, so now it is possible directly from MLflow run command to launch the Kedro pipelines.

From the MLflow interface we can now see all the experiments done.



The screenshot shows the MLflow web interface. On the left, a sidebar lists experiments, with 'activities-example' selected. The main panel displays the details for this experiment, including a description, search filters, and a table of runs. The table has columns for 'Created', 'Duration', 'Run Name', 'User', 'Source', 'Version', 'Models', 'Metrics', 'Parameters', and 'Tags'. The 'Metrics' column is expanded, showing 'accuracy', 'max_error', and 'mean_absolute_error'. The table lists 49 matching runs, each with a checkbox, a clock icon, and various data points.

Created	Duration	Run Name	User	Source	Version	Models	Metrics	Parameters	Tags
7 days ago	57.3s	2022-10-31T...	giong	(1) C:\User\...	85455b	sklearn	accuracy: 72.46, max_error: 473.5, mean_absolute_error: 27.01	-	3
7 days ago	43.2s	2022-10-31T...	giong	(1) C:\User\...	85455b	sklearn	accuracy: 72.46, max_error: 473.5, mean_absolute_error: 27.01	-	3
7 days ago	47.3s	2022-10-31T...	giong	(1) C:\User\...	85455b	sklearn	accuracy: 72.46, max_error: 473.5, mean_absolute_error: 27.01	-	3
7 days ago	52.0s	2022-10-31T...	giong	(1) C:\User\...	85455b	sklearn	accuracy: 72.46, max_error: 473.5, mean_absolute_error: 27.01	-	3
7 days ago	44.3s	2022-10-31T...	giong	(1) C:\User\...	484243	sklearn	accuracy: 72.46, max_error: 473.5, mean_absolute_error: 27.01	-	3
7 days ago	48.1s	2022-10-31T...	giong	(1) C:\User\...	484243	sklearn	accuracy: 72.46, max_error: 473.5, mean_absolute_error: 27.01	-	3
8 days ago	41.2s	2022-10-30T...	giong	(1) C:\User\...	064647	sklearn	accuracy: 72.46, max_error: 473.5, mean_absolute_error: 27.01	-	3
1 hour ago	59.1s	2022-11-07T...	giong	(1) C:\User\...	0c8c06	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
6 hours ago	44.7s	2022-11-07T...	giong	(1) C:\User\...	bc676d	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
6 hours ago	43.3s	2022-11-07T...	giong	(1) C:\User\...	bc676d	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
6 hours ago	46.6s	2022-11-07T...	giong	(1) C:\User\...	bc676d	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
6 hours ago	53.4s	2022-11-07T...	giong	(1) C:\User\...	bc676d	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
3 days ago	53.6s	2022-11-04T...	giong	(1) C:\User\...	916f91	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
4 days ago	47.0s	2022-11-03T...	giong	(1) C:\User\...	338602	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
4 days ago	53.3s	2022-11-03T...	giong	(1) C:\User\...	338602	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
5 days ago	46.9s	2022-11-02T...	giong	(1) C:\User\...	4c537d	sklearn	accuracy: 66.55, max_error: 493.2, mean_absolute_error: 53.87	-	3
2 hours ago	1.1min	2022-11-07T...	giong	(1) C:\User\...	0c8c06	sklearn	accuracy: 57.6, max_error: 442.2, mean_absolute_error: 64.3	-	3
5 days ago	57.3s	2022-11-02T...	giong	(1) C:\User\...	d996cd	sklearn	accuracy: 56.04, max_error: 1018.3, mean_absolute_error: 145.8	-	3
7 days ago	49.2s	2022-10-31T...	giong	(1) C:\User\...	d996cd	sklearn	accuracy: 56.04, max_error: 1018.3, mean_absolute_error: 145.8	-	3

Figure 5.5 - MLflow interface shows all my experiments.

Step 6.

I now want to package the model and create its prediction service, so I installed BentoML. I then imported the model into BentoML through one of its explicit methods, and created a service that uses that model. So when I run the BentoML build command, it creates a bento, that is a file with model, service, source code, dependency configurations and other useful information to start the service. To deploy the bento, I packed this through a simple BentoML method, and I got a docker image of the bento. To prevent conflicts with ports already in use, I changed it in the bentofile.yaml configuration file.

Sending the command `BentoML serve`, turns on the service and the screenshot below appears.

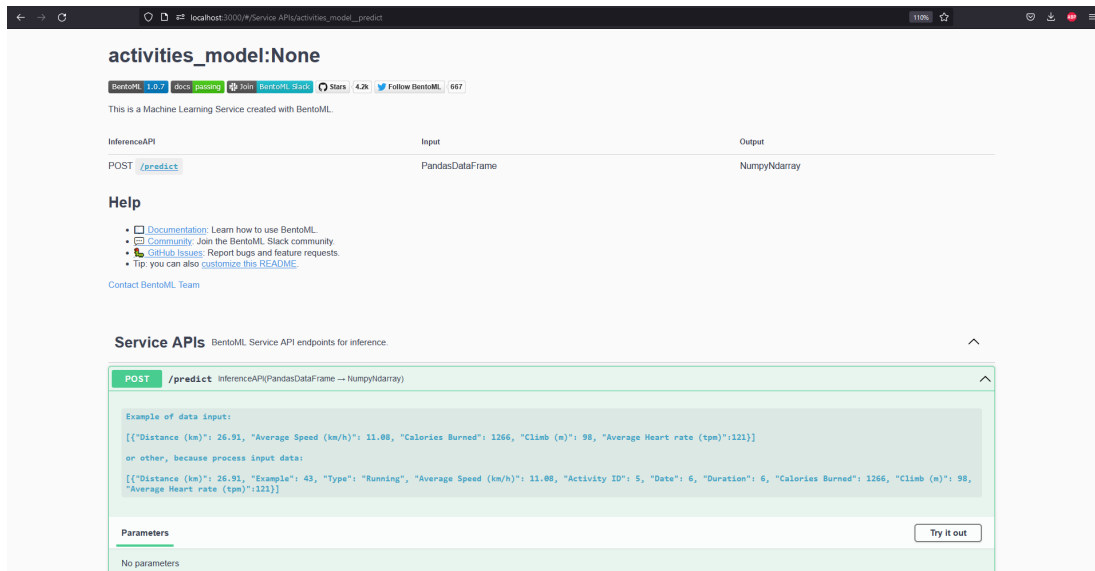


Figure 5.6 - My BentoML interface relates to a ML model with the available prediction service.

Step 7.

Finally I installed the kedro-docker plugin, to make Kedro pipelines distributable and to make it available to users quickly and accurately. I adjusted the settings in the Dockerfile, such as the port and the command to execute. Now it is possible to create a Docker image and run the Kedro project in a Docker environment.

Step 8.

To simplify the execution I wrote a Python script that answers to command lines in order to interact with pipeline and all steps. It makes it easier to fulfill tasks, such as opening tool GUI, creating a new model and its bento, and updating dataset. The available command lines are:

<code>python run.py pipeline</code>	open kedro ui to localhost:4141	
<code>python run.py exp</code>	open mlflow ui to localhost:5000	
<code>python run.py pipeline-docker</code>	create docker image about pipeline to create ml model	
<code>python run.py run</code>	run training model and building bento service	
<code>python run.py new-dataset <global_url></code>	upload dataset from file csv with <global_url>	

Figure 5.7 - Available commands to my Python script

5.4 Production phase

Themes implemented: prediction service, monitoring, alerting and application & trigger. The following image illustrates how the Production phase works.

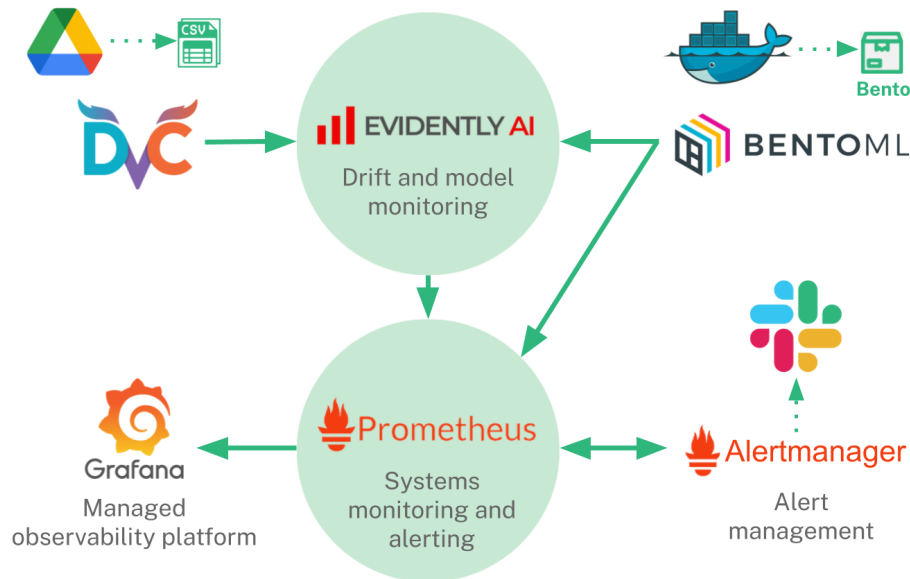


Figure 5.8 - Structure of the schema of the project production phase of this thesis.

Step 1.

This project starts using the prediction service in the bento built in the previous phase. Then by running the service from Docker we can interact with it and send it prediction requests.

My most relevant metrics are data drift and output drift. In fact, these allow the technician to know when it is necessary to create a new model because the current one has become too dated and distant from the current world. These metrics need to know and compare with the dataset on which the model was previously trained. I then extracted the requested dataset using DVC from my personal Google Drive where it was stored.

Step 2.

To write the code, I used the Evidently tool and its methods and classes to implement functions that calculate drift values and other information about the input dataset. Applying Evidently collection of tests and reports, I exported the results in HTML format, which can then in the future be archived or distributed to other areas

depending on the needs of the company. As mentioned above, instances of these output are HTML pages as shown below.

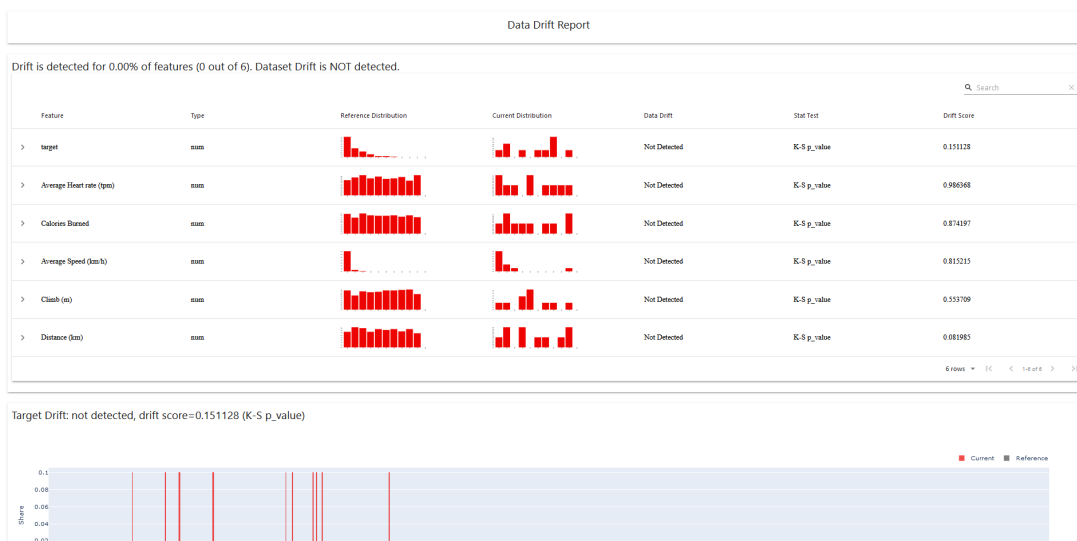


Figure 5.9 - Output page generated by my Evidently report with information about data drift.

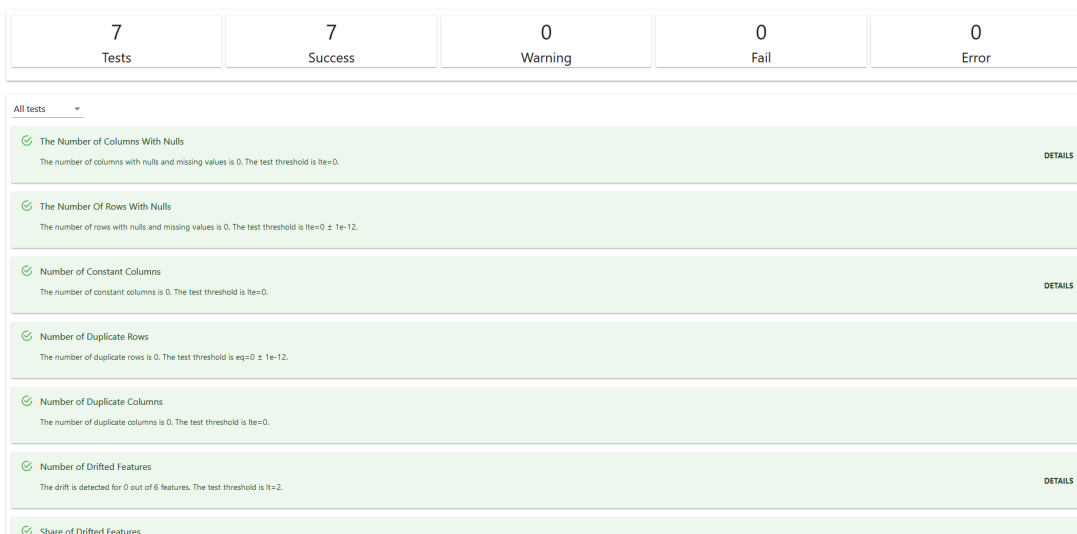


Figure 5.10 - Output page generated by my Evidently tests with information regarding data drift.

Step 3.

To have real-time monitoring of the model and services I need the Grafana tool. I noticed that there is an integration of Evidently with Grafana and Prometheus, but still in development, so I started from this integration project available at the official Evidently documentation, by adapting it to my experience.

There are several scripts that allow a chain of operations, one of which (example_run_request.py) allows interaction with the bento service, where it sends

row by row from the table with production data and then waits for the result. The submitted table is first manipulated and cleaned, by another script, to be accepted by the prediction service.

A key element for monitoring is the Flask application that encapsulates the interaction and data exchange between the Evidently and Prometheus tools. As mentioned above, Evidently calculates drifts, based on the request and prediction result, and collects them in "monitors", which are then collected by Prometheus.

Step 4.

Knowing that Prometheus collects data from various sources, I updated its configurations with the services that contain the metrics. The analyzed services are the bento prediction service, the Evidently service and the Alertmanager service. It is important that the software warns the user of anomalies and drift, so I have provided the prometheus rules, in the appropriate prometheus file. For example, I have written multiple rules, such as in the case of dataset drift, feature drift, and a service gone down.

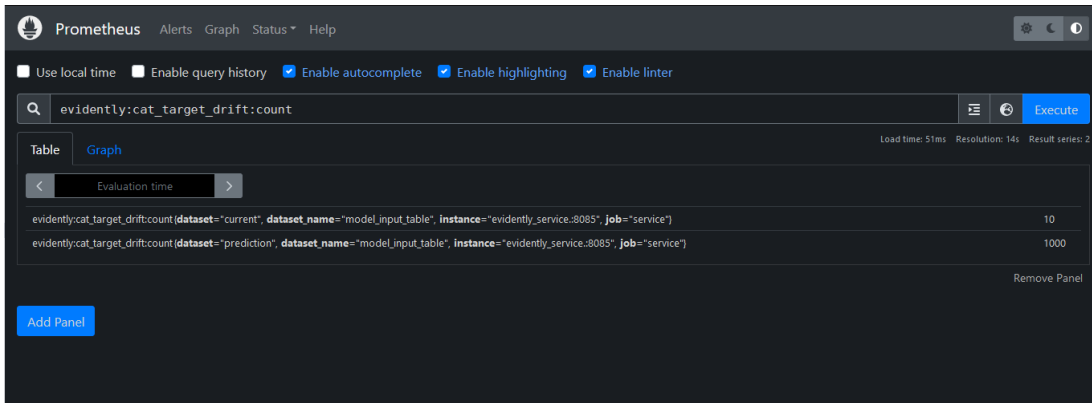


Figure 5.11 - Home page of my Prometheus platform.

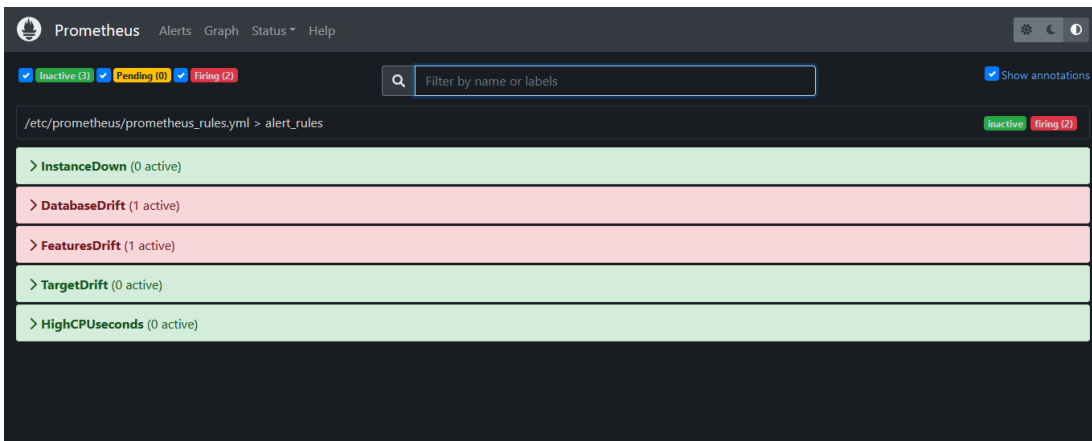


Figure 5.12 - Alerts page of my Prometheus platform.

Step 5.

I also want the occurrence of an anomaly or drift to be notified and then sent to the user to other destinations, in addition to the Prometheus interface. Then I set up the config file `alertmanager.yml` in which I specified the url api of my Slack chat where I want to receive such alerts.



Figure 5.13 - My Slack chat with alert notifications.

Step 6.

To finally have the interface with realtime data, there are two more Grafana configuration files, one where Prometheus is specified as the datasource and one with pre-created dashboards. So the user from the Grafana interface has the dashboards already in the project but can also create his own dashboards customizing them to his needs.

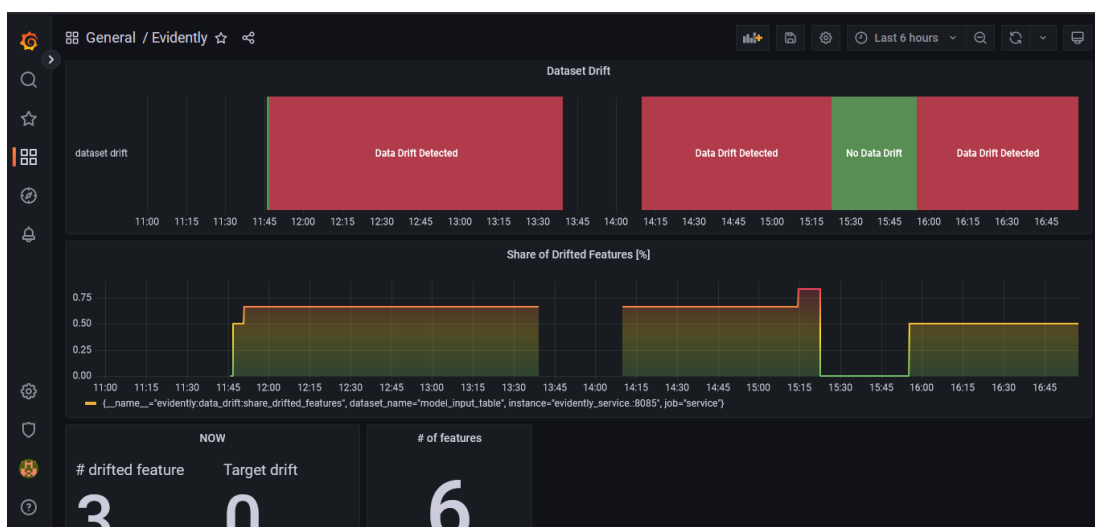


Figure 5.14 - A dashboard of my project on Grafana platform.

Step 7.

Thinking that this project is a support for technicians in companies to monitor production models, I also wrote an application using the Streamlit library. With this application I use the bento service, I can ask to pull down the training dataset from Drive, and I can make multiple requests to the development phase project. On the opposite side there is a Flask application which reacts to requests from Streamlit application.

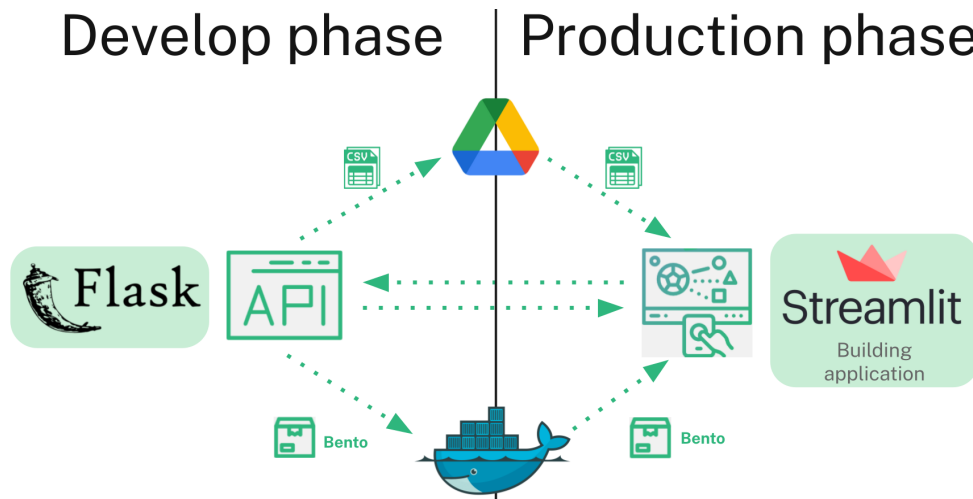


Figure 5.15 - Schema of the communication and exchange of features between the two phases.

I created a few example pages to show how the Streamlit application can best be suited to the needs or type of user who is going to use it. From the first page you can request a forecast from the ML model. From the second, the user can conveniently connect to the various interfaces to help monitor. From the third, it is possible to request a new model for the development phase from production. From the fourth page, the technician can request the development phase of the less complex actions, and the available API of Flask application are:

<code>@app.get("/dvc_file")</code>	return the code for take reference dataset from Google Drive.
<code>@app.route("/load_new_data", methods=["POST"])</code>	receive a dataframe to load as new reference dataset.
<code>@app.get("/retrain")</code>	load in Google drive new dataset, run mlflow to training model, build and dockerize bento.
<code>@app.get("/bento")</code>	build and dockerize bento.
<code>@app.get("/header")</code>	return name columns of dataset header.

Figure 5.16 - Available API to my Flask application

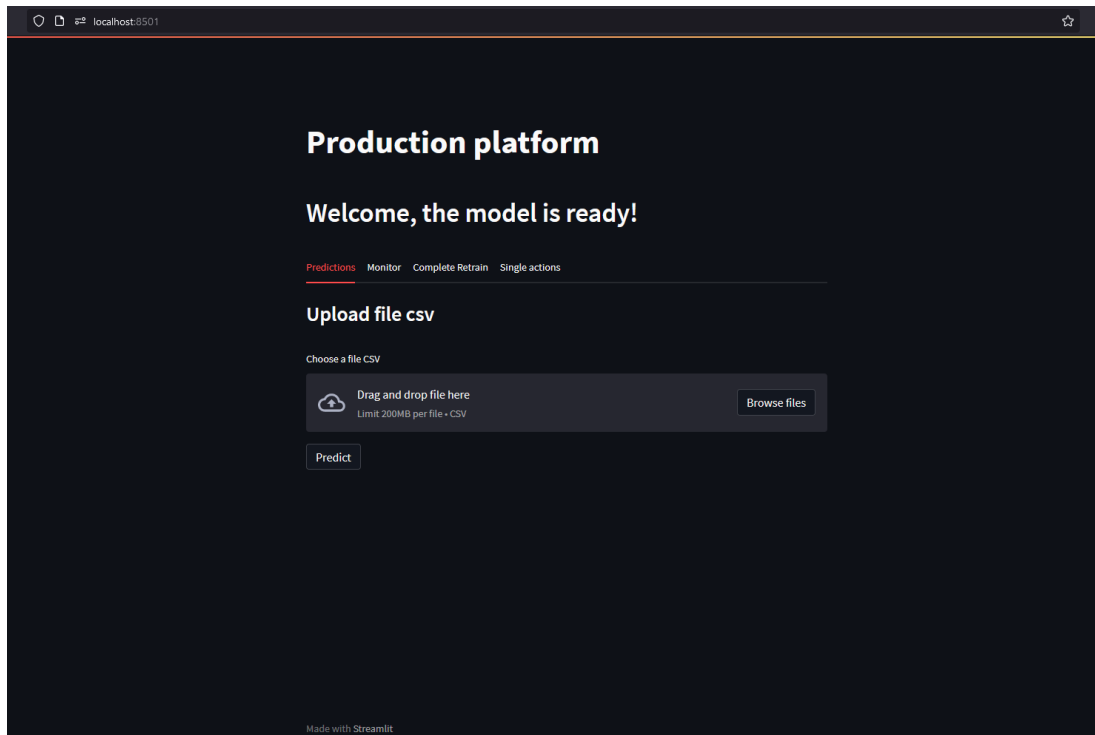


Figure 5.17 - My Streamlit application home page to request a prediction.

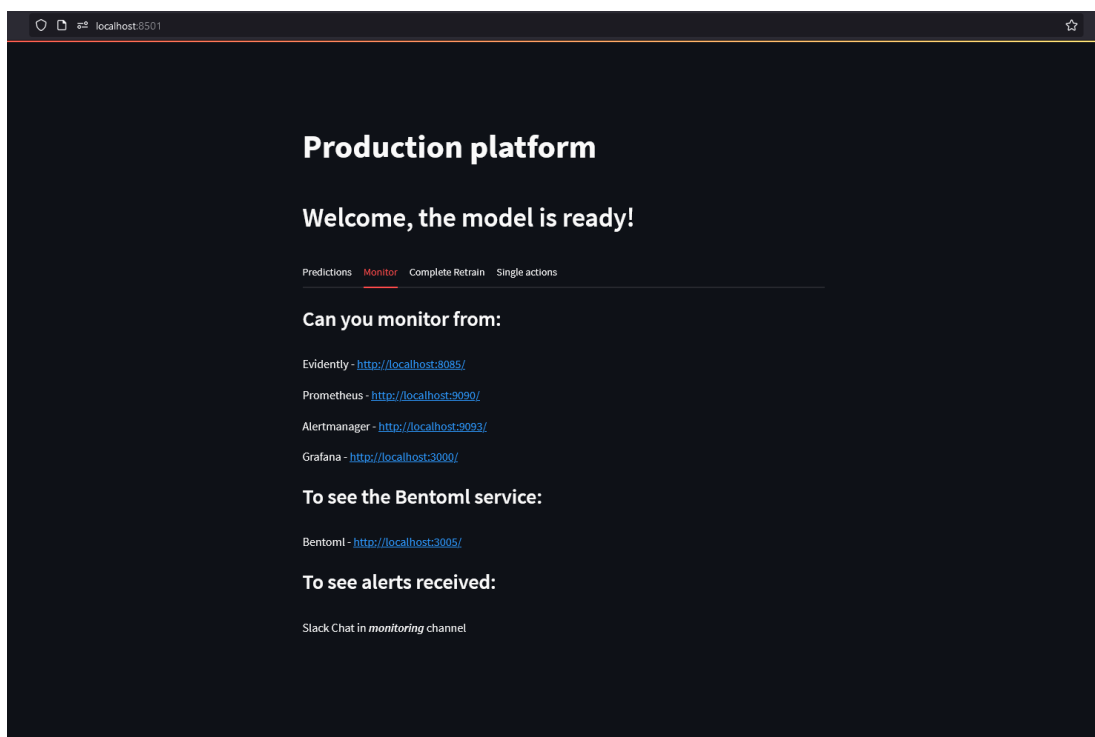


Figure 5.18 - My Streamlit application page containing links to tracking.

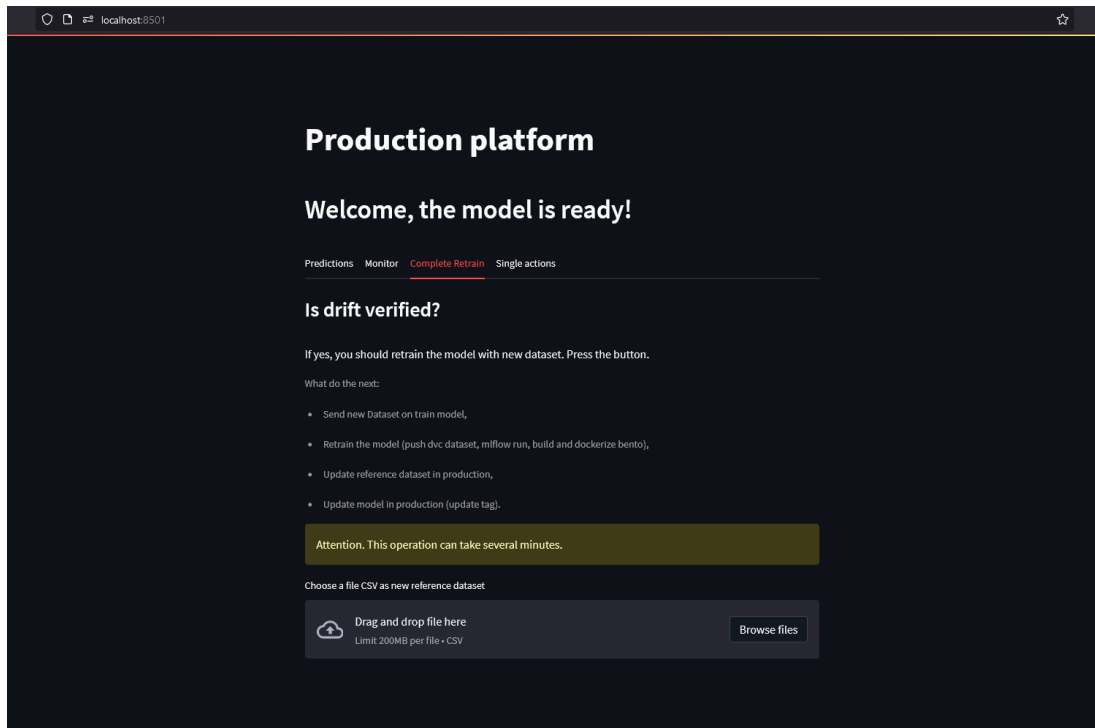


Figure 5.19 - My Streamlit application page to request ml model retrain.

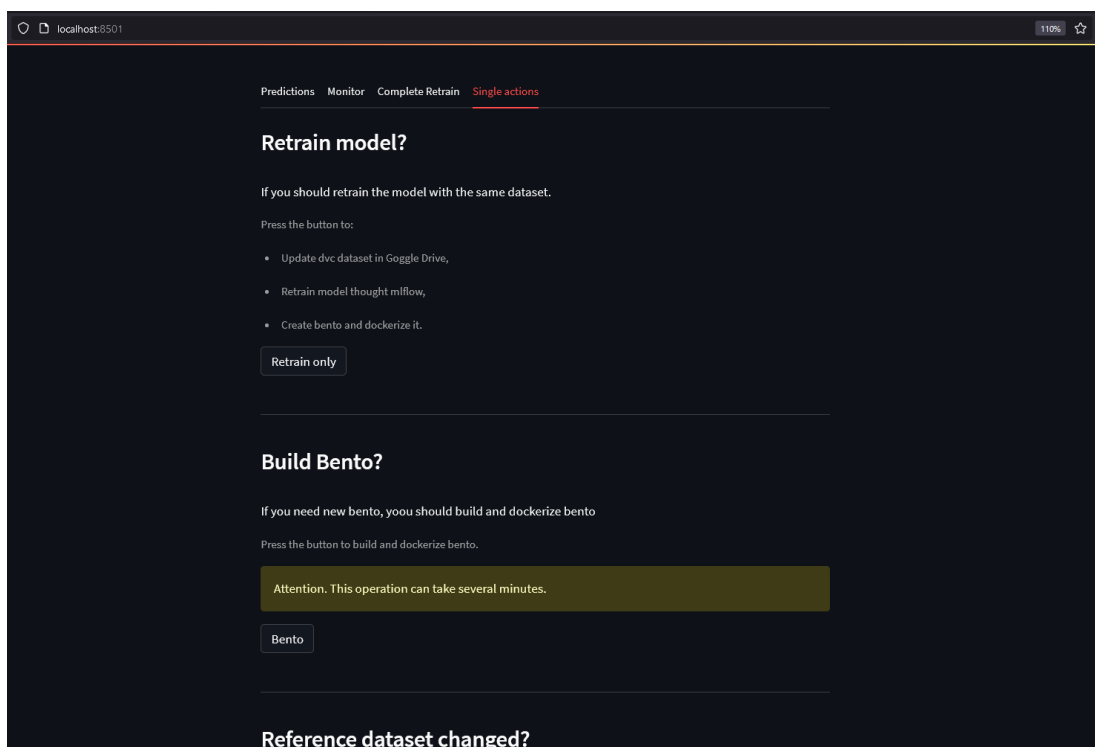


Figure 5.20 - My Streamlit application page to require less complicated actions.

Chapter 6

Conclusion

MLOps provides an engineering perspective on the entire process. To assist the success of Machine Learning projects, MLOps is a tool that comes to the rescue. In fact, it simplifies model development, leaving more time and human resources available to give attention toward ML model research to build better Machine Learning systems. We can conclude that designing an MLOps workflow provides benefits such as automation, cost and productivity optimization. But it is not just that, the uses of MLOps produce best practices for transparent, explainable and reliable AI.

Currently, MLOps is highly applied to build algorithms for the insurance, financial and pharmaceutical industries because they ensure compliance with governance policies in AI models. Governance is a collection of controls placed on a company to ensure that it meets its responsibilities to all stakeholders, like shareholders, employees, the public and national governments. These responsibilities cover financial, legal, and ethical obligations. Governance in MLOps can be categorized in two ways:

- data governance, to ensure the appropriate management and manipulation of datasets,
- process governance, to formalize the MLOps process via reviews, sign-offs, and documentation.

Lately, governments are beginning to take an interest in regulating ML, with the aim of mitigating the negative impact of its use. This is necessary because ML is embodied in many decision-making systems that impact our lives. The European Union is already working on planned legislation to define the acceptable uses of various forms of AI.

To provide a actual and future view of MLOps I quote below part of an article from the blog of Bitstrapped, a Canadian IT company that supports customers with MLOps solutions:

The market for MLOps solutions is expected to reach \$4 billion by 2025. We are in early days, but the strategy to adopt MLOps will take shape over the next few years. A well-structured MLOps practice makes it easier to align ML models with business needs, people, as well as regulatory requirements. While deploying ML solutions you will uncover sources of revenue, save time, and reduce operational costs. The efficiency of the workflow within your MLOps function will allow you to leverage data analytics for decision-making and build better customer experiences.

Quote 6.1 - <https://www.bitstrapped.com/blog/rise-of-ml-ops>

Bibliography

1. Dr. Larysa Visengeriyeva, Anja Kammer, Isabel Bär, Alexander Kniesz, and Michael Plöd. *Machine Learning Operations*.
URL. <https://ml-ops.org/>
2. Manasi Vartak. *What is MLOps? DataOps? And Why do They Matter?*
URL. <https://devops.com/what-is-mlops-dataops-and-why-do-they-matter/>
3. Abid Ali Awan. *The Machine Learning Life Cycle Explained*.
URL. <https://www.datacamp.com/blog/machine-learning-lifecycle-explained>
4. CloudCheckr. *The Cloud Infrastructure Report*.
URL. <https://click.cloudcheckr.com/rs/222-ENM-584/images/CloudCheckr-White-Paper-The-Cloud-Infrastructure-Report-2020.pdf>
5. Prince Canuma. *MLOps: What It Is, Why It Matters, and How to Implement It*.
URL. <https://neptune.ai/blog/mlops>
6. Andriy Burkov. *Machine learning engineering*.
URL. <http://www.mlebook.com/wiki/doku.php>
7. Google Architecture Center. *MLOps: Continuous delivery and automation pipelines in machine learning*.
URL. <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>
8. Google Cloud. *Google Cloud's AI Adoption Framework*.
URL. https://services.google.com/fh/files/misc/ai_adoption_framework_whitepaper.pdf
9. Henrik Skogström. *The MLOps Stack*.
URL. <https://valohai.com/blog/the-mlops-stack/>
10. Ajay Agrawal, Joshua Gans and Avi Goldfarb. *A Simple Tool to Start Making Decisions with the Help of AI*.
URL. <https://hbr.org/2018/04/a-simple-tool-to-start-making-decisions-with-the-help-of-ai>
11. IBM. *Analyzing machine learning model performance*.

- URL. <https://cloud.ibm.com/docs/watson-knowledge-studio?topic=watson-knowledge-studio-evaluate-ml>
12. IBM. *What is business intelligence?*
URL. <https://www.ibm.com/uk-en/topics/business-intelligence>
 13. Modzy. *Gartner® Report – Hype Cycle™ for Data Science and Machine Learning, 2021.*
URL. <https://www.modzy.com/reports/gartner-hype-cycle-for-data-science-machine-learning/>
 14. Rick Merritt. *What Is MLOps?*
URL. <https://blogs.nvidia.com/blog/2020/09/03/what-is-mlops/>
 15. Tom Goldenberg. *Kedro + MLflow - Reproducible and Versioned data pipelines at scale | PyData LA 2019.*
URL. <https://www.youtube.com/watch?v=ZPxuohy5SoU&t=823s>
 16. Streamlit. *API reference.*
URL. <https://docs.streamlit.io/library/api-reference>
 17. Saif Abid. *The Rise of MLOps.*
URL. <https://www.bitstrapped.com/blog/rise-of-ml-ops>
 18. Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, Lynn Heidmann.
Book: *Introducing MLOps: How to Scale Machine Learning in the Enterprise*