information

Article

Applying DevOps Practices of Continuous Automation for Machine Learning

Ioannis Karamitsos [1,*], Saeed Albarhami [1] and Charalampos Apostolopoulos [2]

[1] Department of Computing, Rochester Institute of Technology, Dubai Campus, Dubai 341055, UAE; sat5006@rit.edu
[2] Department of Management Science, University of Strathclyde Business School, Glasgow G1 1XQ, UK; charalampos.apostolopoulos@strath.ac.uk
* Correspondence: ixkcad1@rit.edu

Received: 1 June 2020; Accepted: 5 July 2020; Published: 13 July 2020

Abstract: This paper proposes DevOps practices for machine learning application, integrating both the development and operation environment seamlessly. The machine learning processes of development and deployment during the experimentation phase may seem easy. However, if not carefully designed, deploying and using such models may lead to a complex, time-consuming approaches which may require significant and costly efforts for maintenance, improvement, and monitoring. This paper presents how to apply continuous integration (CI) and continuous delivery (CD) principles, practices, and tools so as to minimize waste, support rapid feedback loops, explore the hidden technical debt, improve value delivery and maintenance, and improve operational functions for real-world machine learning applications.

Keywords: CRISP-DM; CI; CD; DevOps; machine learning; pipeline; SEMMA; TDSP

1. Introduction

In the data science world, machine learning (ML) is becoming a core approach for solving complex real-world problems, transforming industries, and delivering value in respective domains. Therefore, there are many data science teams in the scientific and machine learning communities trying to improve the overall business value using descriptive and predictive models. As a consequence, ML data scientists and ML operations engineer teams study how to apply DevOps principles to their ML systems under study.

DevOps [1] is a set of practices and tools based on software and systems engineering. Software engineering can be defined as a discipline dedicated to developing tools and techniques which allow creation and use of sophisticated software systems. Data science is less about program development and more about analyzing and getting insights from the data. Agile [2], on the other hand, refers to an iterative approach which focuses on collaboration, customer feedback, and small and rapid releases. DevOps and Agile are two pillars to support in achieving business strategy and overlap the traditional operational and developmental teams to create an environment that is continually improving operations through a cross-functional team of developers and operators. The DevOps strategic objective is to investigate methods for improving service quality and features in a manner that satisfies their customer needs (Farroha [3]).

ML data scientists and ML operations engineering teams have introduced manual steps for the delivery of ML pipeline model. This method may likely produce unexpected results due to the dependency on data collection, preparation and preprocessing, model training, validation, and testing. Moreover, this method led to the conclusion that no apparent advantage exists in utilizing our manual approach for ML projects. Also, in terms of quality results, this ML manual pipeline method produces high operational costs and delays, which directly or indirectly affect the revenue or quality reputation of the business.

We can conclude that there are four research questions associated with the above ML manual pipeline approach.

- RQ1: Which DevOps tools have been selected to design and implement automate deployment pipelines?
- RQ2: What challenges have been reported or adopting DevOps continuous practices?
- RQ3: How does DevOps impact the ML manual pipeline method in terms of performance, scalability, and monitoring?
- RQ4: How can we find a new approach for the ML manual pipeline method to improve the ML tracking lifecycle?

To address these research questions, a novel approach is proposed in this paper to introducing the two DevOps principles in the MLOps approach. The main idea is to design a machine learning automate pipeline using two DevOps [1] principles: the continuous integration (CI) and the continuous delivery (CD). Practicing MLOps means that we advocate for automation and monitoring at all steps of ML system construction, including integration, testing, releasing, deployment, and infrastructure management. This specific operation of ML complex systems incorporates two important DevOps principles [1], continuous delivery (CD) and continuous integration (CI). The functionality of CI is not limited only for testing, and validating code and components, but also testing and validating data, data schemas, and models. CD is no longer about a single package or service, but an ML pipeline that should automatically deploy another ML service.

The research method used in this study is the systematic literature review (SLR) [4] as the most widely used in the software engineering area. SLR aims to provide a well-defined process for identifying, evaluating, and interpreting all available evidence relevant to a particular research questions or topic. Following the SLR guidelines reported in [4], our review protocol consisted of (i) research questions, (ii) study selection, (iii) assignment criteria, and (v) data extraction and synthesis.

The target audience of this paper is ML data scientists who want to apply the ML trained models using CI and container principles in the testing environment, as well as ML operations engineers working on tested ML models in the production environment. The paper makes the following contributions:

- a review of the two principal DevOps components—continuous integration (CI) and continuous delivery (CD)—in the ML context,
- a ML manual pipeline design with the components in details,
- a ML automate pipeline design with CI/CD components in details

The rest of the paper is organized as follows. Section 2 includes the background information of DevOps and the related works on different DevOps and ML scenarios. In Section 3, we present different ML lifecycle methodologies. In Section 4, we design the proposed ML pipeline manually and ML with CI/CD automation. Section 5 presents ML scalability for different pipelines. Finally, Section 6 concludes the paper.

2. Background

In this section, we provide an overview of DevOps principles as part of continuous software engineering paradigm.

2.1. Continuous Software Engineering

Continuous software engineering is an emerging area of research and practice. It refers to development, deployment, and getting quick feedback from software and customers in a very rapid cycle [5,6]. Continuous software engineering involves three phases: business strategy and planning, development, and operation with the following software development activities: continuous integration and continuous delivery.

Continuous integration (CI) is a widely established development practice in software development industry [5], in which members of a team integrate and merge development work (e.g., code) frequently, for example multiple times per day. CI enables software companies to have shorter and frequent release cycle, improve software quality, and increase their teams' productivity [6]. This practice includes automated software building and testing [7].

Continuous delivery (CD) is aimed at ensuring an application is always at production-ready state after successfully passing automated tests and quality checks [8,9]. CD employs a set of practices e.g., CI, and deployment automation to deliver software automatically to a production-like environment [10]. CD is a push-based approach [11]. According to [11,12], this practice offers several benefits such as reduced deployment risk, lower costs, and getting user feedback faster.

2.2. DevOps

DevOps [1] stands for development and operations. It's a practice that aims at merging development, quality assurance, and operations (deployment and integration) into a single, continuous set of processes. This methodology is a natural extension for Agile and continuous delivery approaches from the software engineering paradigm. However, DevOps is not merely a set of actions. It is more of a culture or even a philosophy that fosters cross-functional team communication. One of the main benefits of DevOps is that it does not require substantial technical changes being rather oriented to changing the way a team works. Teamwork is a crucial part of DevOps culture: the whole success of a process depends on it, and there are principles and practices that DevOps teams use.

2.2.1. DevOps Principles

In short, the main principles of DevOps are automation, continuous delivery, and fast reaction to feedback. You can find a more detailed explanation of DevOps pillars in the CAMS acronym [1]:

Culture represented by human communication, technical processes, and tools
Automation of processes
Measurement of KPIs
Sharing feedback, best practices, and knowledge

Adherence to these principles is achieved through a number of DevOps practices that include continuous delivery, frequent deployments, QA automation, validating ideas as early as possible, and in-team collaboration.

2.2.2. DevOps Model and Practices

DevOps [1] requires a delivery cycle that comprises planning, development, testing, deployment, release, and monitoring with active cooperation between different members of a team as depicted in Figure 1.

Continuous delivery is an approach that merges development, testing, and deployment operations into a streamlined process as it heavily relies on automation. During the development phase, engineers commit code in small chunks multiple times a day for it to be easily tested. A quality assurance team sets committed code testing using automation tools. If bugs and vulnerabilities are revealed, they are sent back to the engineering team. This stage also entails version control to detect integration problems in advance. A version control system (VCS) allows developers to record changes in the files and share them with other members of the team, regardless of its location. The code that passes automated tests is integrated in a single, shared repository on a server. Frequent code submissions prevent a so-called 'integration hell' when the differences between individual code branches and the mainline code become so drastic over time that integration takes more than actual coding. The most popular tools for continuous integration are Jenkins [13] and GitLab CI [14]. Then, the code is deployed to run in production on a public server. Code must be deployed in a way that it does not affect already functioning features and can be available for a large number of users. Frequent deployment allows for a 'fail fast' approach, meaning that the new features are tested and verified early. There are various automated tools that help engineers deploy a product increment. The most popular are Chef [15], Puppet [15], Azure Resource Manager [16], Google Cloud Deployment Manager [17], and Amazon S3 [18]. The final stage of the DevOps lifecycle is oriented to the assessment of the whole cycle. The goal of monitoring is detecting the problematic areas of a process and analyzing the feedback from the team and users to report existing inaccuracies and improve the product's functioning.

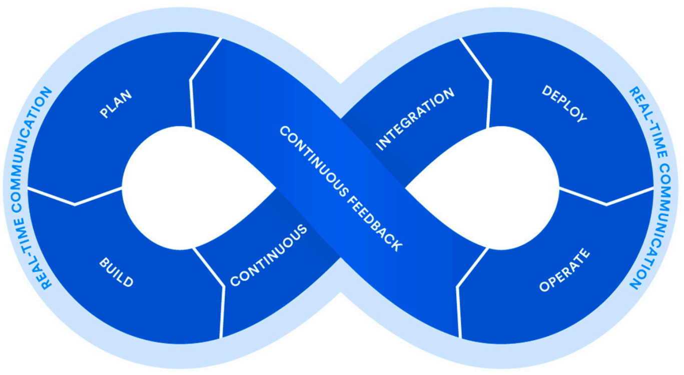

Figure 1. DevOps lifecycle (Source: Atlassian).

2.3. *Existing Literature Reviews*

Machine learning system developments are characterized by some challenges that need to be dealt with, in order to achieve continuous improvements; one of the most critical challenges is data collection, data extraction, and data cleansing. Few research papers have been noted towards warning companies from the dangers of quick wins when developing machine learning models and suggesting frameworks to highlight technical debt (Sculley [19]; Kontsevoi, Soroka, & Terekhov [20]).

Sculley [19] claimed that machine learning models have a remarkable capacity for causing technical debt, due to problems related to maintenance of software engineering code as well as an extra set of machine learning systems specific issues. Technical debt is the result of prioritizing speed over perfection by choosing a quick and easier way to a solution that needs to be refactored or reworked in the future, leading to higher operational costs.

Kontsevoi et al. [20] shared the details of an approach to improve and manage the quality of the software product using a set of practices to manage and avoid technical debts. The technical debt is not easy to detect because it occurs at the system level and not in the code level.

In addition, several researchers (Karamitsos [21], Virmani, [22]; Erich [23]; Lwakatare [24]) have agreed that Agile transformation is essential to improve the efficiency of the companies to optimize the lifecycle delivery, to break the gap and to create a continuous feedback loop between the business users and development teams. They highlight the importance of Agile principals and guidelines as well as DevOps practices and tools to adopt continuous integration and continuous delivery pipelines which will result in an increment in the pace of the development process and improve the quality. It is critical to remember the fact that data preprocessing influences machine learning systems behavior and the standard methods for minimizing the code level technical debt in traditional software development are not always perfect for reducing machine learning related technical debt at the system level.

The underlying problems that DevOps practice tries to address are the flexibility to change, speed up the value delivery to market and maintain high quality while keeping the cost low; these are universal business problems at any business domain of software engineering projects.

One of the essential goals of DevOps is to change the culture towards increasing collaboration and creating cross-functional teams. The culture is abstract; even though there might be different definitions and models of it. According to Forsgren [25], the main challenge is to discover a model of culture that is well-defined in the scientific literature, that could be effectively measured, as well as have the predictive capability in our domain. Not only did we achieve these objectives, but we also discovered that it is possible to influence and improve culture by implementing DevOps practices.

The overall goal of DevOps is to improve the business value of the work done in IT and to focus on delivering measurable business value through continuous and high-quality service delivery; emphasizing simplicity and agility in all areas—including human factors, technology, process; breaking down barriers between development and operations by enabling trust and shared ownership; and supporting innovation and encouraging collaboration (Farroha, [3]).

3. Machine Learning Lifecycle Methodologies

This section describes the methodology and methods for the development of machine learning lifecycle. These different methodologies may work better in different scenarios and data types. CRISP-DM [26], which stands for "CRoss-Industry Standard Process for Data Mining", is the most commonly used approach by data mining experts and it was introduced in 1996 by Daimler Chrysler (then Daimler-Benz). SEMMA [27], which stands for "Sample, Explore, Modify, Model and Assess", is a popular project methodology developed by the SAS Institute. TDSP [28], which stands for "Team Data Science Process", uses the team data science process with Azure Machine Learning developed by Microsoft. In this study, the TDSP methodology is selected.

3.1. CRISP-DM Methodology

The CRISP-DM methodology [26] typically has six iterative phases:

- Business Understanding Phase: In this phase, determine business objectives, assess the situation, establish data mining goals, and produce the project plan.
- Data Understanding Phase: In this phase, the initial data is available for the exploratory analysis, and evaluation of the data quality.
- Data Preparation Phase: In this phase, the preparation of data is a multistage process that comprises several individual steps. These steps are feature extraction, data cleaning, data reduction, data selection, and transformation.
- Modeling Phase: In this phase, the machine learning model is selected for the specific problem.
- Evaluation Phase: In this phase, the results can be processed by the selection of the ML model. Also, a review may be performed to check if the business understanding is achieved.
- Deployment Phase: In this phase, the steps are plan deployment, plan monitoring, and maintenance.

A diagram of CRISP-DM methodology with six steps is illustrated in Figure 2.

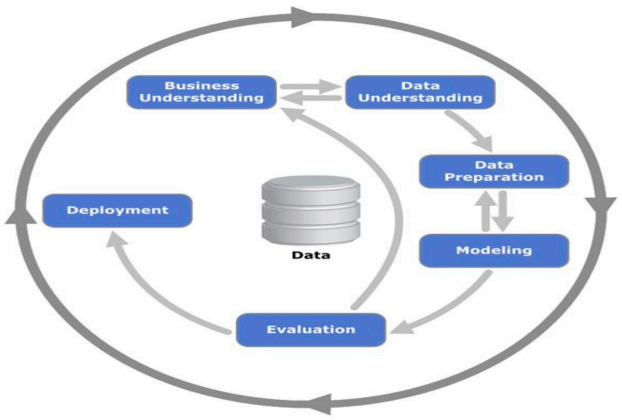

Figure 2. CRISP-DM methodology (Source: [26]).

CRISP-DM [26] has been consistently the most common used methodology for analytics, data mining and data science projects. Despite its popularity, CRISP-DM has not been revised since its creation.

3.2. SEMMA Methodology

SEMMA [27] methodology is defined by the SAS Institute as the process of sampling, exploring, modifying, modeling, and assessing a large amount of data to uncover previously unknown patterns which can be utilized as a business advantage. The main differences between the CRISP-DM methodology are as follows: the steps are not iterative and focus only on the procedures instead of results. To understand the SEMMA [27] methodology, the following steps are presented.

- Sample step: In this step, sample data is limited to collection and analysis of the data contained in form.
- Explore step: Understand the data exploring the outliers, patterns, and relationships.
- Modify step: Modify the data by selecting, transforming and deriving the required feature to enable reaching an outcome.
- Model step: Model the data using data analytics algorithms and tools to establish the results.
- Assess step: In this step, the resulting outcome is assessed in multiple stages by evaluating the usability and reliability of the findings from the data mining process.

A diagram of SEMMA methodology is illustrated in Figure 3.

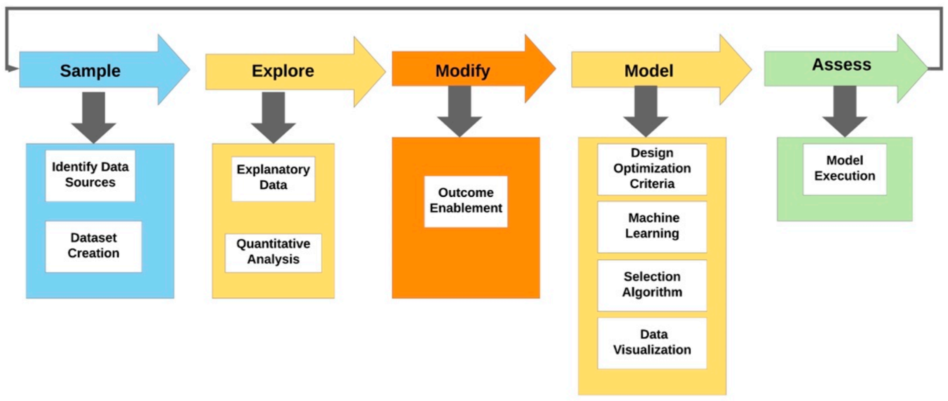

Figure 3. SEMMA methodology (Source: Adapted [27]).

3.3. Team Data Science Process (TDSP) Methodology

Team data science process (TDSP) [28] is an agile, iterative data analysis methodology designed to produce effective predictive analytics technologies and intelligent applications. TDSP includes best practices and structures from Microsoft to help implement the data science initiatives successfully. TDSP provides the lifecycle to structure the development process of data science projects. The TDSP lifecycle has been designed for data science projects and can be used as part of intelligent applications that deploy machine learning or artificial intelligence models. In addition, this method may also help exploratory data science projects or improvised analytics projects. The lifecycle outlines the major stages that projects typically execute, often iteratively in Figure 4.

Although the lifecycle graphic in Figure 4 looks quite different; TDSP's project lifecycle is similar to CRISP-DM [26] and includes five iterative stages:

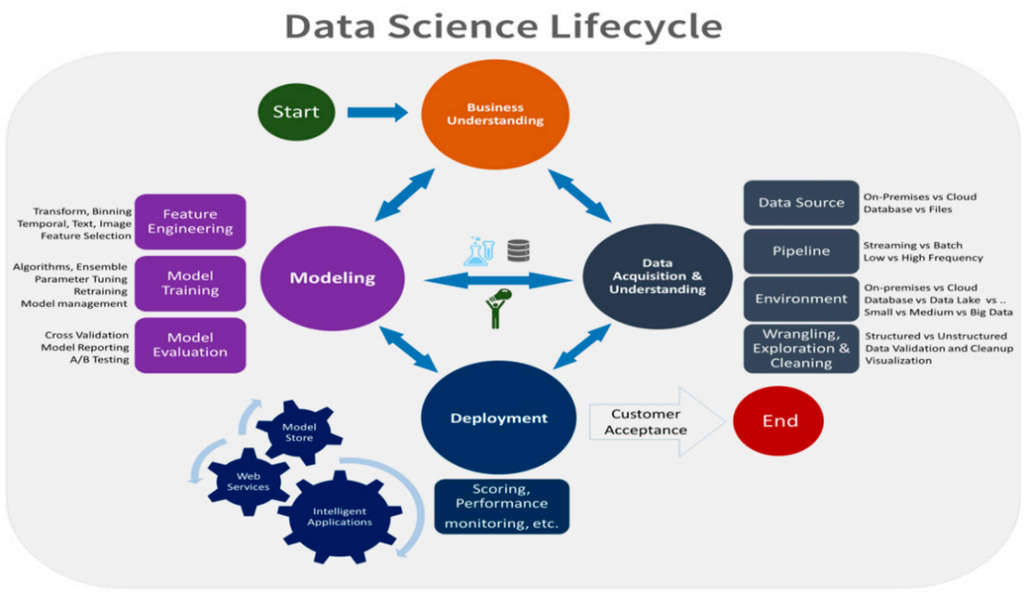

Figure 4. TDSP methodology (Source [28]).

3.3.1. Business Understanding

In this stage of the Team Data Science Process, there are two main tasks to focus on:

Defining objectives: Data scientists and customers, as well as other stakeholders, work together to identify and understand the business problems, and formulate questions that define the business goals. A core objective of this step is to name the key business variables that the analysis is required to predict. These variables are considered as the 'model targets', metrics of which are used to determine the success of the project. Define the business goals by asking and refining questions that are relevant, specific, and unambiguous. Define the team by specifying the roles and responsibilities and create a high-level milestone plan that iterates as more information is discovered. Define success metrics as SMART (specific, measurable, achievable, relevant, time-bound).

Identifying sources of the data: Locate the relevant data that help to solve the questions that define the objectives of the project. Identify data sources that contain examples of answers to questions and make sure that the data is relevant to the problem. Also, this data must have an accurate measure of the target model and the features that matter.

3.3.2. Data Acquisition and Understanding

This critical phase focuses on fact-finding about the data. It starts by ingesting the data, exploring and setting up a data pipeline. The goals of this phase are to produce a clean, high-quality dataset that can be related to the target variables. Find the data set in the appropriate analytics environment to start the modeling process. Create a solution architecture of the data pipeline that updates the data regularly.

Ingest the data: Configure the pipeline to move the data from the source locations to destination locations where we run analytics operations.

Explore the data: To train the model, we need to develop an understanding of the data. The real-world dataset is often noisy and has many missing values and outliers. Data visualization and summarization can be used to audit the quality of the data; furthermore, it provides the information that is required to prepare the data before it is ready for modeling.

Setup data pipeline: In addition to the initial ingestion and cleansing of the data, we typically require setting up a process to update new data or refresh the data regularly as part of a continuous learning process.

3.3.3. Modeling

In this stage of the team data science process, the focus is on determining the best data features for the machine learning model that predicts the target most accurately and suitable for production use.

Feature engineering: This step involves the inclusion, aggregation, and transformation of raw variables to create the features used in the analysis. This step requires domain expertise, working in a cross-functional team with data scientists and analysts to observe the insights obtained from the data exploration step and decide the features to introduce if needed. Feature engineering is a balancing act of collecting and including informative variables, but at the same time trying to avoid unrelated variables because informative variables improve our result and the separate variables introduce unnecessary noise into the model.

Model training: This step depends on the type of question that we are trying to answer. There are many modeling algorithms available, this step includes splitting the input data, build the model, evaluate the model, and determine the optimal solution for the defined goals.

3.3.4. Deployment

The goal of this step is to deploy the created machine learning model with a data pipeline to a staging or production environment to verify the model for user acceptance. Once the model's performance fits the solution, then we can operationalize it for other applications. Depending on the business goals and requirements, predictions are made either in real-time or on a batch basis.

3.3.5. Customer Acceptance

The goal of customer acceptance phase is to finalize the project deliverables and confirm that the model, pipeline, and the deployment to production environment satisfy the customer's goals. In this step, the customer will validate that the system meets the business needs; and therefore, proceed to deployment of the system to production for use by the client's application.

3.3.6. Team Definition

Team data science process addresses the weakness of CRISP-DM's lack of team representation by defining four well-defined roles (solution architect, project manager, data scientist, and project lead) and their responsibilities during each stage of the project lifecycle.

4. Machine Learning Pipelines

In any machine learning (ML) project, teams of data scientists and ML engineers are working to build state-of-the-art models using manually or automatic driven processes. The data-scientist driven process selection depends on how ML models are changed or trained over time and their complexities. Also, the development/test environment is different from the staging/production environment and ML manual process models often fail to adapt to changes in the dynamics of production environment or changes in the data which describe the production environment.

4.1. Machine Learning Manual Pipeline

Machine learning (ML) manual pipeline process is a convenient and frequently used tool in the study of use cases. The process is script-driven, and every step is manual, including data analysis, data preparation, model training, and data validation. Manual execution of each step and manual transformation from one step to the next is necessary. This process is driven by known analytical methodologies such as CRISP-DM [26], SEMMA [27], and TSDP [28].

The purpose of this process is to present a generic approach that is MLOps manual pipeline. The process separates teams, data scientists who create the model, and ops engineers who serve the model as a testing service. Continuous integration (CI) is missing in this process due to few implementations changes. Testing coding is typically a part of the execution of notebooks or scripts.

The scripts and notebooks that implement the steps of the experiment are controlled sources and produce artefacts such as trained models, evaluation metrics, and visualizations. Continuous delivery (CD) is not considered in this process due to not frequent model version deployment. Also, the MLOps manual process does not track or log the model results or actions, which are required in order to detect the model performance degradation. Figure 5 shows the workflow of this process:

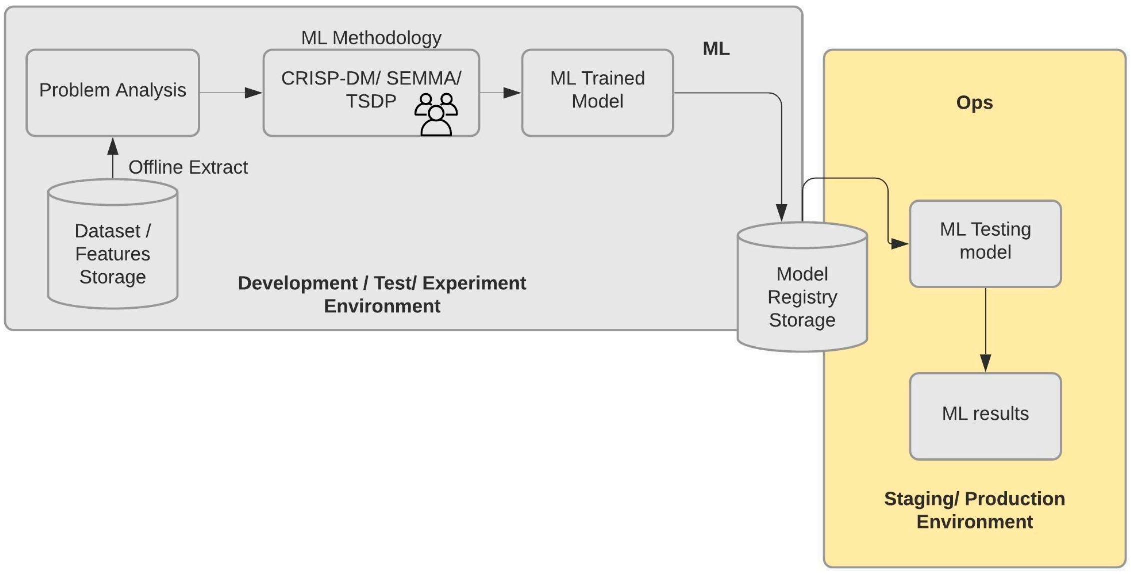

Figure 5. Machine learning manual pipeline process.

The MLOps pipeline process is composed by two teams: ML scientists responsible for the setup of the development environment and the ops engineers responsible for the production environment. The main components including the development and production environment of ML manual process may be summarized as follows. The MLOps manual setup includes the following components:

- Business problem framing
- Dataset features and storage
- ML analytical methodology
- ML trained model
- Model registry storage
- ML testing model
- ML results

Business problem framing: In any ML model, firstly, we define the business problem and establish the business understanding and the success criteria for the problem.

Dataset features and storage: In this step, a dataset with features/attributes is presented for the specific business problem.

ML analytical methodology: This step is essential for the selection of the ML methodology as presented in the previous section. During the analysis, the use of TSDP methodology has proven satisfactory in practice of MLOps pipeline.

ML trained model: This step is essential for making the proposed model feasible for MLOps manual pipeline. Data scientists create the ML trained model and deploy in the model registry storage using API infrastructure.

Model registry storage: This step is useful for the data storage. Data scientists will hand over a trained model as an artefact to the engineering team (ops) to deploy on their API infrastructure. This handoff incorporates two important techniques: putting the trained model in a storage location, checking the model object into a code repository or uploading it to a model registry. As a consequence, this reliable storage can be used for the generated models and the associated data, it can be an on-site file share system, Azure Storage [16], Google Cloud Storage [17], or Amazon S3 [18] ensuring that we can reproduce the same results at any time in the future.

ML testing model: This step is essential for making the testing model feasible for the MLOps pipeline. ML engineers (ops) retrieve the trained model and create the ML testing model with the required available features in the production environment.

ML results: This step is useful for the analysis of ML results in terms of performance and accuracy using a confusion matrix.

The dependency on data collection, preparation and preprocessing, model training, and testing produce unexcepted results on ML manual pipeline method. Also, in terms of quality results, this ML manual pipeline method is not adapted for dynamic changes of the model, therefore, it produces high operational costs and delays, which directly or indirectly affect the revenue or quality reputation of the business. With the proposed ML automate pipeline with CI/CD principles, the above issues of the ML manual approach are resolved in an efficient way. In the next subsection, we propose a ML automate pipeline using DevOps principles for the data.

4.2. Proposed Machine Learning Automate Pipeline with CI/CD

The main focus of our approach is on the continuous training and testing of the model by automating the ML pipeline. This framework of MLOps brings together various principles of continuous integration (CI) [5] and continuous delivery (CD) [9] in a coherent and practical way for the ML pipelines.

Process automation of retraining models in production using new data is introduced by two components from the DevOps framework [29] such as continuous integration (CI) and continuous delivery (CD). For a rapid and reliable update of the pipelines in production, we need a robust automated CI/CD system. The automated CI/CD system allows the data scientists to rapidly explore new ideas around feature engineering, model architecture, and hyperparameters. They can implement the process of building a model, checking in the code to a repository, building a deployment package such as scripts and packages, and then deploying the new pipeline components to a staging/production environment. Figure 6 shows MLOps pipeline deployment using CI/CD which has the features of automated ML pipelines setup plus the automated CI/CD routines.

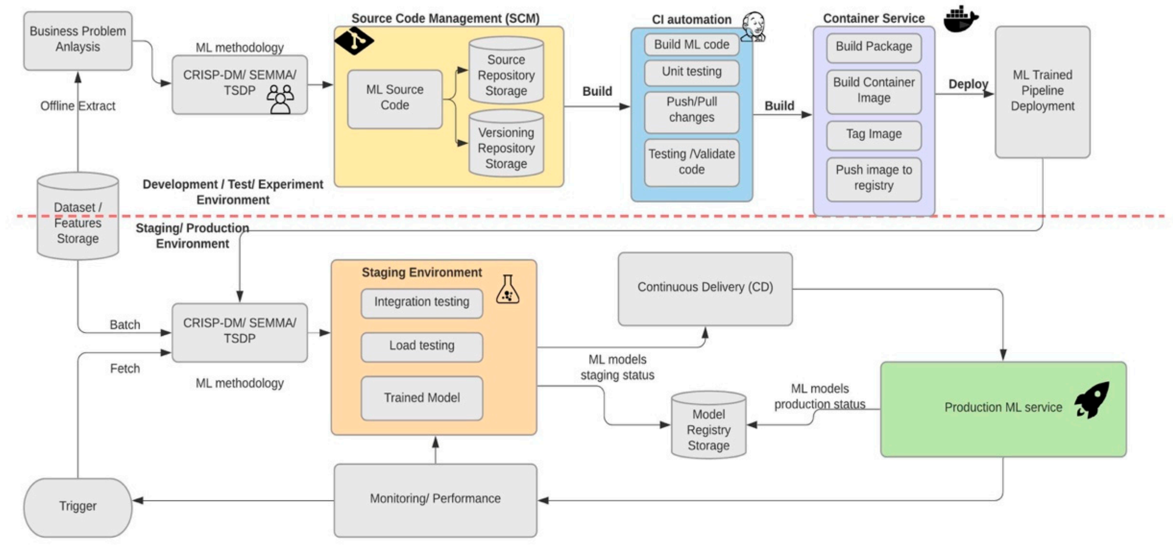

Figure 6. ML pipeline automation for CI/CD.

The main components required for the development and implementation of MLOps pipeline automation with CI/CD routines are as follows:

- Business problem analysis
- Dataset features and storage
- ML analytical methodology
- Pipeline CI components
- Pipeline CD components
- Automated ML triggering
- Model registry storage
- Monitoring and performance
- Production ML service

Business problem analysis: In any ML model, firstly, we define the business problem and establish the business understanding and the success criteria for the problem.

Dataset features and storage: In this step, a dataset with features/attributes is presented for the specific business problem.

ML analytical methodology: This step is essential for the selection of the ML methodology as presented in the previous section. During the analysis, the use of TSDP methodology has proven satisfactory in practice of MLOps pipeline.

Pipeline continuous integration (CI): In this stage, we build source code and run various ML trained models. The outputs of this stage are pipeline components (packages and artefacts) to be deployed in the staging/production environment of the continuous delivery (CD).

The machine learning code is just a small portion of real machine learning system; however, to the best of our knowledge, this complexity of the system requires more time and resources. An important component of the machine learning infrastructure is the configuration and data elaboration. The analysis of the data was performed with the help of cleaning, wrangling, and feature extraction. There is no doubt that the training process for the machine learning models may take many efforts; however, we need to invest also in the production process of the model in a way that allows us to retrain and tune the hyperparameters and to use the new data in order to improve the model.

The proposed continuous integration (CI) for our machine learning systems relies on having a substantial impact on the end-to-end pipeline to automate the delivery of our machine learning model with minimal efforts. The main steps for the continuous integration (CI) are summarized as follows:

- Source code management (SCM)
- Push/pull changes to the repository to trigger a continuous delivery build
- Check out the latest code and the associated data version from the data repository storage.
- Run the unit tests
- Build/run machine learning model code
- Testing and validation
- Package the model and build the container image
- Push the container image to the registry

A variety of software tools have been used for the deployment of machine learning models such as Jenkins [13], Git [14], Docker [30], Helm [31], and Kubernetes [32].

Source code management (SCM): In this stage, the source code and the data versioning are storage in the ML testing environment.

Push changes: The implementation of machine learning models is an incremental and iterative process. This process is computed in two steps. First, the model is built, second, the model is evaluated, and the process can be repeated several times to achieve their goals. The previously implemented process has been considered similar to the process of changing the code, compiling, and executing in software development, i.e., we should use software engineering tools to help data scientists in the process of developing machine learning models, these tools include version control.

Version control is the process of tracking each version of the code or data in a way that makes it easy to track the history of changes and return to previous versions at any time, and also to share code with the collaborators. This also applies to the dataset used during each incremental and iterative cycle, and we have to keep track of that data in each increment in order to reproduce the same results in the future if required. In this step, any changes to the source code of the model or changes to the version of the dataset used will be pushed (uploaded) to the repository (code or data) to trigger the continuous delivery build.

Pull changes: In this stage of continuous integration automation, once the build model is triggered, the latest code tests were performed and analyzed by the code repository and then the associated data version is pulled from the data repository storage.

Run unit tests: One of the primary features of this step is the ability to provide robust code that cannot break easily upon changes like package updates, bug fixes, and new features.

Build/run machine learning model code: The design of the machine learning code depends upon the language type used, whether the language is interpreted or compiled. Moreover, this step includes the training and retraining of the ML models. The goal behind the training and retraining is either introducing new data or adjusting the hyperparameters.

Testing and validating code: Machine learning models help improve the decision-making process; therefore, it is significantly essential to have sufficiently accurate models; otherwise, these might have a negative impact rather than positive. An essential part of the machine learning delivery pipeline is to understand how well a model works. In order to validate this ML model, a subset of training data as a test set is selected. This test data is not used to build the ML model, but once the ML model is built, it is used as a test data to evaluate how well the model performs. For validating the code, there are a number of different strategies that can be used to ML models.

- Recreate: Terminate the version A and then roll out the version B.
- Ramped: Version B is rolled out in an increment way to replace version A.
- Blue/Green: Version B is released beside version A, once B is validated and confirmed its functionality traffic is switched to the new version B.
- Canary: Make a subset of the traffic to go to version B while old version A is still serving then proceed to roll out to full users.
- A/B testing: Running both versions and comparing results

Package the model and build the container image: In this stage of the container service, all the dependencies of our machine learning model are containerized into a container image. This step was carried out using Helm [31] package manager with the latest container image. Kubernetes [32] an open-source system for automating deployment, scaling, and management of containerized applications. K8S groups [32] containers that make up an application into logical units to simplify the management, discovery, and monitoring. Shortly afterwards, the container image is pushed to the image repository.

Push the image to the container registry: After packaging the latest container image with the latest dependencies, the process pushes the image to the container registry.

The outputs of this continuous integration (CI) pipeline are different trained ML models to be deployed in the staging/production environment of the continuous delivery (CD).

Pipeline continuous delivery (CD): In this stage, we build the artefacts produced by a previous continuous integration (CI) to the staging/preproduction/production environment. The output of this stage is the delivery of ML testing models. Jenkins [13] is a popular tool for continuous automated deployment. It is an open-source tool written in Java that integrates well with version control systems such as Git [14]. Jenkins works with a pipeline metaphor that enables developers to deliver execution pipelines as code (Jenkins file) in multiple steps within a defined execution environment such as staging or production and executing test scripts and record test results. The components of the CD pipeline are summarized as follows:

Staging environment: In software engineering, it is very common and standard practice to deploy the ML trained model first to a staging environment. Software architecture is no longer limited to the deployed ML pipeline running at the customer infrastructure, but also includes the system being deployed for development and staging as well. The environment term is often used to describe the infrastructure required to run a service or an application (app servers, databases, caches). In this environment, the trained model is used as a modeling tool for the prediction of the testing ML models. The output of this stage is a testing ML model which it is pushed to the model registry storage.

Model registry storage: In this step, the ML models under staging status and ML models under production status are uploaded in a storage location. In addition, the ML engineers (ops) check the ML model objects into a model repository or upload them to a model registry.

Automated triggering: This step is automatically executed on schedule or in response to a trigger in the production environment. The output of this stage is a ML testing model which it is pushed to the staging environment.

Performance monitoring: Statistical analysis was performed on live data to ascertain which ML model deployed in efficient way. Once the model is deployed to production, data scientists still need to continue validating or testing it because patterns in the data can change over a while, and the model might become less accurate or drift because the data used in training the model is no longer representative of the new data that exist in production. The output of this stage is a trigger to execute the pipeline or to execute a new experiment cycle.

The performance monitoring with which ML model operates is of the utmost importance for data scientists. Performance monitoring consists of a wide range of tasks, involves understanding the level of resource utilization, measuring throughput and operating efficiently, and knowing what services are available to users.

Monitoring resources is essential for making the method feasible for critical resources as CPU, memory, and persistent storage. There are many GUI tools for monitoring and visualizing the status of resources. Alternatively, the command-line tools are also used to collect, and display information targeted to our particular interests.

5. Techniques for Scalable Machine Learning Models

There is a wide range of techniques for scaling machine learning models. Using a cluster of servers to run machine learning models has its advantages if we want to deploy a scalable, highly available machine learning model; however, deploying this approach on multiple servers manually—especially if we want to perform it based on the demand of the service—is a very challenging and time-consuming task. In order to have a scalable model, we need first to deploy the model with an API as a microservice, which will facilitate the process of sharing the service across different applications. Then, we run the API microservice that contains the model in a container which includes all dependencies required to run the model. Finally, we use an orchestration tool to manage to monitor the containers in the cluster. The REST interface is a widely used protocol for calling services over HTTPS, returning JSON result, the programming language we want to use to build the machine learning model does not matter as this approach is universal (Java, Python, R). For example, in Python, Flask is available to serve as a lightweight web development framework, and R developers can make use of Plumber package to make their machine learning models available as APIs. Once we have built a machine learning model and we want to deploy it in a scalable environment, we do it by exposing the model through an API, putting the model and API in a container, and managing the container with an orchestration platform. Containerization is the process of bundling the application with all of its related files, library configurations, and dependencies which are required to run efficiently across different environments running the same container's engine.

6. Conclusions

In this paper, we presented a ML automate pipeline with CI/CD principles. The method is based on DevOps practices, which are responsible for the integration and delivery of ML trained and tested models. We presented different ML methodologies and the TSDP methodology is more appropriate for this study. We discovered that the manual ML method produces high operational costs and delays in business organizations. The proposed automated ML method improves many areas such as time to market, integration across business units, and breakdown departmental silos; it also increases code and deployment quality, productivity, and visibility. However, it is not straightforward; many companies struggle and stuck at the start of the journey while others abort the implementation halfway through the process due to challenges like resistance to change, isolated teams (silos), lack of skillsets, etc. It is critical to understand that DevOps does not stand alone, but it relies on the adoption and integration of multiple frameworks and methodologies like ITSM, Agile, and Lean.

The machine learning models lifecycle is different and more complex than traditional software development; it requires extensive work with data extraction, preparation and verification, infrastructure configuration, provisioning, post-production monitoring, and enhancements. Therefore, Agile principles/values and DevOps practices and tools are highly recommended to provide continuous delivery and co-creation of value to customers, increase the model quality, minimize waste, highlight the importance of supporting a rapid feedback loop, accommodate early changes, as well as explore the hidden technical debt that leads to an enormous increase in operational costs of real-world machine learning systems. Our proposed automated ML pipeline method has included DevOps principles and practices; therefore, the above benefits are added value in the lifecycle of ML models.

In order to succeed in the DevOps journey, it is imperative to first focus on changing your business' culture, seek executive-level support (buy-in), choose the right toolchain that fits your business needs, automate (continuous integration, continuous testing, and continuous delivery) and promote success.

Leveraging our method, future work may be focused on specific aspects of the model for complex systems and develop a useful software to help testing ML models.

Author Contributions: Conceptualization, I.K. and S.A.; Methodology, C.A. and I.K.; Software, S.A.; Validation, I.K. and S.A.; Formal analysis, S.A.; Resources, S.A.; Data curation, C.A. and S.A.; Writing—original draft preparation, I.K. and S.A.; Writing—review and editing, I.K., C.A., and S.A.; Visualization, I.K.; Supervision, I.K. and C.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. DevOps Documentation. Available online: https://devops.com (accessed on 10 March 2020).
2. Moreira, M. *The Agile Enterprise: Building and Running Agile Organizations*, 1st ed.; Apress: Berkeley, CA, USA, 2017.
3. Farroha, B.S.; Farroha, D.L. A framework for managing mission needs, compliance, and trust in the DevOps environment. In Proceedings of the IEEE Military Communications Conference, Baltimore, MD, USA, 6–8 October 2014; pp. 288–293.
4. Kitchenham, B.; Chartes, S. Guidelines for performing systematic literature reviews in software engineering. *EBSE Technical Report (EBSE-2007-01); Keele University and Durham University Joint Report*. 2007. Available online: https://edisciplinas.usp.br/pluginfile.php/4108896/mod_resource/content/2/slrPCS5012_highlighted.pdf (accessed on 7 July 2020).
5. Fitzgerald, B.; Stol, K.J. Continuous Software Engineering: A Roadmap and Agenda. *J. Syst. Softw.* **2017**, *123*, 176–189. [CrossRef]
6. Bosch, J. Continuous Software Engineering: An Introduction. In *Continuous Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 3–13.
7. Leppanen, M.; Makinen, S.; Pagels, M.; Eloranta, V.P.; Itkonen, J.; Mantyla, M.V.; Mannisto, T. The Highways and Country Roads to Continuous Deployment. *IEEE Softw.* **2015**, *32*, 64–72. [CrossRef]

8. Weber, I.; Nepal, S.; Zhu, L. Developing Dependable and Secure Cloud Applications. *IEEE Internet Comput.* **2016**, *20*, 74–79. [CrossRef]
9. Humble, J. Continuous Delivery vs. Continuous Deployment. Available online: https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment (accessed on 15 June 2020).
10. 2015 State of DevOps Report. Available online: https://puppetlabs.com/2015-devops-report (accessed on 15 June 2020).
11. Chen, L. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Softw.* **2015**, *32*, 50–54. [CrossRef]
12. Humble, J. What is Continuous Delivery? Available online: https://continuousdelivery.com/2010/02/continuous-delivery/ (accessed on 15 June 2020).
13. Jenkins. Build Great Things at any Scale. Available online: https://jenkins.io (accessed on 10 April 2020).
14. Chacon, S.; Straub, B. *Pro Git*, 2nd ed.; Apress: Berkeley, CA, USA, 2014.
15. Christof, E.; Gallardo, G.; Hernantes, J.; Serrano, N. DevOps. *IEEE Softw.* **2016**, *33*, 94–100.
16. Azure Documentation. Available online: https://www.azure.microsoft.com (accessed on 10 March 2020).
17. Google Documentation. Available online: https://google.com (accessed on 10 March 2020).
18. AWS Documentation. Available online: https://aws.amazon.com (accessed on 10 March 2020).
19. Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.F.; Dennison, D. Hidden technical debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems*; Curran Associates: New York, NY, USA, 2015; Volume 28, pp. 2503–2511.
20. Kontsevoi, B.; Soroka, E.; Terekov, S. TETRA as a set of techniques and tools for calculating technical debt principal and interest. In Proceedings of the IEEE/ACM International Conference on Technical Debt, TechDebt, Montreal, QC, Canada, 26–27 May 2019; pp. 64–65.
21. Karamitsos, I.; Albarhami, S.; Apostolopoulos, C. Tweet Sentiment Analysis (TSA) for cloud providers using classification algorithms and latent semantic analysis. *J. Data Anal. Inf. Process.* **2019**, *7*, 276–294. [CrossRef]
22. Virmani, M. Understanding DevOPs and bridging the gap from continuous integration to continuous delivery. In Proceedings of the 5th International Conference on Innovative Computing Technology INTECH, Pontevedra, Spain, 22–25 May 2015; pp. 78–82.
23. Erich, F.M.; Amrit, C.; Daneva, M. A qualitative study of DevOps usage in practice. *J. Softw. Evol. Process* **2017**, *29*, 1–20. [CrossRef]
24. Lwakatare, L.E.; Kuvaja, P.; Oivo, M. Relationship of DevOps to Agile, Lean and Continuous Deployment. In *Product-Focused Software Process Improvement, Proceedings of the 17th International Conference, PROFES 2016, Trondheim, Norway, 22–24 November 2016*; Springer: Cham, Switzerland, 2016; Volume 10027, pp. 399–415.
25. Forsgren, N.; Humble, J.; Gene, K. *The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*; IT Revolution Press: Portland, OR, USA, 2018.
26. Shearer, D. The CRISP-DM Model: The new Blueprint for Data Mining. *J. Data Wareh.* **2000**, *5*, 13–18.
27. SAS Enterprise Miner- SEMMA SAS Institute Inc. Available online: http://www.sas.com/technologies/analytics/datamining/miner/semma.html (accessed on 20 May 2020).
28. *Microsoft Azure Team Data Science Process*. Available online: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/overview (accessed on 10 March 2020).
29. Humble, J.; Farley, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*; Pearson Education, Inc.: Boston, MA, USA, 2011.
30. Docker Documentation. Available online: https://docs.docker.com (accessed on 10 April 2020).
31. Helm Team: The Package Manager for Kubernetes. Available online: https://helm.sh (accessed on 15 April 2020).
32. Google: Kubernetes. Available online: https://kubernetes.io (accessed on 15 April 2020).