Master Thesis

DevOps for Machine Learning (MLOps)

A thesis submitted in fulfillment of the requirements for the degree of ***Master of Science (M.Sc.) in High Integrity Systems (HIS)***

to the

Frankfurt University of Applied Sciences
Department of Computer Science and Engineering

by

Shaikh Safir Mohammad Mustak
Matriculation No.: 1322554
safir.shaikh@stud.fra-uas.de

Academic Supervisor
Prof. Dr. Peter Nauth
pnauth@fb2.fra-uas.de

Industrial Supervisor
Hendrik Mende M.Sc.
hendrik.mende@ipt.fraunhofer.de

October 28, 2021
Frankfurt am Main

Declaration of Authorship

I, Shaikh Safir Mohammad Mustak, declare that this thesis titled **"DevOps for Machine Learning"** and the research work presented here are of my own. I confirm that:

- This work has been completed as a partial requirement for my Master's degree.
- I have properly mentioned and cited other's published works used for this thesis..
- I have acknowledged all of the prime sources of help.
- I have always provided the source wherever I have quoted the work of others. Except for such quotations, this thesis is entirely my work.

Place, Date

Signature

Abstract

In an ever-emerging era of artificial intelligence, various machine learning (ML) models are deployed and utilized in unique ways in the production environment. However, these models fail miserably as soon as they encounter new data or new features for prediction. The literature illustrates that, unlike traditional software development, ML systems demand special treatment and continuous attention to run without predicting wrong results. In this thesis, I describe several specialist techniques for a standardized model deployment process in order to analyze, monitor, understand, and improve the system. I have also implemented an automated application to demonstrate the same that focuses on faster deployment and higher quality operation of models. I expect this new approach towards dealing with the ML models and their life cycle would benefit many businesses and enterprise solutions using any kind of machine learning in their production.

Acknowledgments

The completion of this thesis would not have been possible without the help of a few people, and I would like to express my gratitude for them:

- First of all, I would like to thank the almighty Allah for blessing me with success in both research and implementation of this thesis.
- Next, I am thankful to Prof. Dr. Peter Nauth from Frankfurt University of Applied Sciences for providing me the opportunity to work on this thesis and for encouraging me during the work that resulted in this thesis.
- Then, I would like to thank my industrial supervisor, Hendrik Mende from Fraunhofer Institute for Production Technology for his guidance during my research.
- Moreover, I am grateful to my family, especially my parents, who supported me emotionally and financially, and my younger brother for continuously motivating me during my studies.
- Finally, I thank my friends who are also a family to me!

Thank you,
Safir Mohammad

Contents

1 Introduction **1**

1.1 Motivation 2

1.2 Aim of Thesis 3

1.3 Thesis Organisation 3

2 Background Theory **5**

2.1 DevOps vs. MLOps 6

2.2 Data Science Steps in ML Systems 8

3 State of the Art **11**

3.1 Levels in MLOps 11

3.1.1 Level 0: Manual Process 12

3.1.2 Level 1: Automating ML Pipeline 14

3.1.3 Level 2: Automating CI/CD Pipeline 18

3.2 Bonus for Data Scientists 21

3.2.1 AutoML [1] 21

3.2.2 MLflow [2] 25

3.2.3 Transparency 29

3.2.4 Monitoring 38

4 Implementation **41**

4.1 Dataset 42

4.2 CNC Mill Tool Wear App 44

4.2.1 ML Pipeline 44

4.2.2 Application Endpoints and Snapshots 48

4.3 How to Run? 62

5 Results and Discussion **65**

6 Conclusion and Future Scope **69**

Bibliography **74**

A. Definitions **76**

B. Formulae **78**

C. Public Access **79**

List of Figures

1.1 AI and ML Classification Tree 1

1.2 Microsoft's Tay chatbot initiated as a cool girl, but quickly converted into a hate-speech-spewing disaster [3]. 4

2.1 Vast and Complex elements of ML Systems [4] 5

2.2 Data Preparation Classification Tree 9

2.3 Model deployed as a Web Service 10

3.1 MLOps Level 0 [5] 12

3.2 MLOps Level 1 [5] 14

3.3 MLOps Level 2 [5] 18

3.4 Terminal Output 22

3.5 An MLflow Run 26

3.6 Artifacts recorded by the Run 27

3.7 Transparency Methods 30

3.8 LIME Example [6] 31

3.9 Anchor Example [7] 32

3.10 SHAP Illustration [8] 33

3.11 ICE Plots [9] 34

3.12 PDP Plots [9] 35

3.13 Direct Monitoring Scheme [10] [11] 38

4.1 Features with Constant Behavior 45

4.2 Decision Tree Hyperparameters 46

4.3 Random Forest Hyperparameters 46

4.4 Home 48

4.5 Data Distribution 49

4.6 Model Behavior (1) 50

4.7 Model Behavior (2) 51

4.8 Model Behavior (3) 52
4.9 Model Behavior (4) 52
4.10 Model Behavior (5) 53
4.11 Retrain 54
4.12 ML-Flow (1) 55
4.13 ML-Flow (2) 55
4.14 Metrics Endpoint 56
4.15 Prometheus (1) 57
4.16 Prometheus (2) 58
4.17 Grafana (1) 59
4.18 Grafana (2) 60
4.19 App Dashboard (1) 61
4.20 App Dashboard (2) 61

5.1 Selected Features 65
5.2 Optimized Training Time 66
5.3 Auto-Sklearn Logs 67

List of Tables

3.1 Properties of Transparency Methods 37

5.1 Performance Metrics 66
5.2 Decision Tree 67
5.3 Random Forest 67
5.4 Auto-Sklearn 67

Listings

3.1 Auto-Sklearn Installation . . . 21
3.2 Auto-Sklearn Implementation . . . 22
3.3 MLflow Installation . . . 25
3.4 MLflow Implementation . . . 26
3.5 Run MLflow . . . 26
3.6 MLflow Model Storage Format . . . 27
3.7 LIME Installation . . . 31
3.8 Anchor Installation . . . 32
3.9 SHAP Installation . . . 33
3.10 PDP Usage . . . 35
3.11 Counter Usage . . . 38
3.12 Gauge Usage . . . 39
3.13 Summary Usage . . . 39
3.14 Histogram Usage . . . 39
3.15 Exposing Metrics . . . 40
3.16 Installation of prometheus-client . . . 40
4.1 Application Installation . . . 62
4.2 Running Application . . . 62
4.3 Running in Debug Mode (1) . . . 63
4.4 Running in Debug Mode (2) . . . 63
4.5 Clear Containers and Images . . . 63

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
DRM	Disaster Risk Management
AutoML	Automated Machine Learning
CI	Continuous Integration
CD	Continuous Delivery
CT	Continuous Training
XP	Extreme Programming
EDA	Exploratory Data Analysis
AIC	Akaike Information Criterion
SBC/BIC/SBIC	Bayesian Information Criterion
ROC	Receiver Operating Characteristic
REST	Representational State Transfer
API	Application Programming Interface
NaN	Not a Number
QPS	Queries per Second
AutoDL	Automated Deep Learning
UI	User Interface
LIME	Local Interpretable Model-Agnostic Explanation
SHAP	SHapley Additive exPlanations
ICE	Individual Conditional Expectation
PD	Partial Dependence
PDP	Partial Dependence Plot
HTTP	Hypertext Transfer Protocol
WSGI	Web Server Gateway Interface
ASGI	Asynchronous Server Gateway Interface
CNC	Computerized Numerical Control

Chapter 1

Introduction

Artificial intelligence (AI), as opposed to natural intelligence, is the science to empower computers and machines to mimic human behavior. The field of AI contains multiple sub-domains such as robotics, machine learning, computer vision, natural language processing, expert systems, neural networks, evolutionary computation, and speech processing. Machine learning (ML) is the most widely used discipline of AI to enable machines to learn and improve at a given task with experience and data. Today, it is being manipulated in various fields like banks, insurance companies, health systems, telecommunication, and credit card companies for image recognition, speech recognition, recommendation systems, language translation, fraud detection, stock prediction, and many more. It can handle large volumes of multi-dimensional and multi-variate data to discover patterns out of it, which wouldn't be possible by humans. The key reason behind its success is, ML systems work automatically without being explicitly programmed. The classification tree for AI and ML is illustrated in figure 1.1 below:–

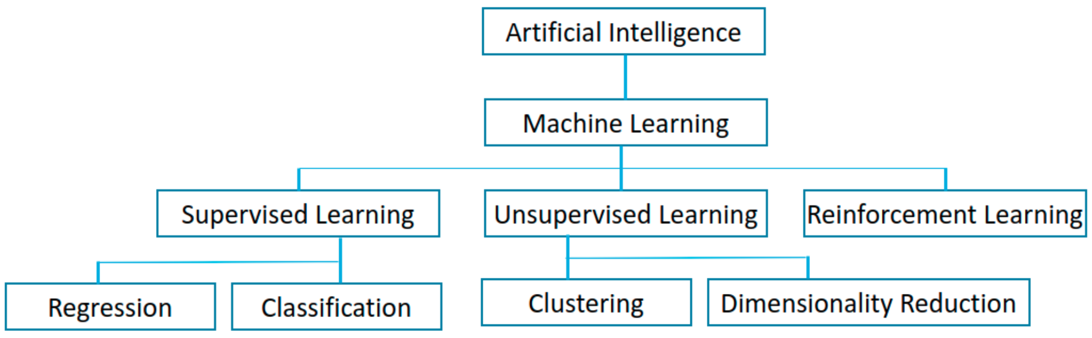


Figure 1.1 – AI and ML Classification Tree

When the software development and operation teams raised issues and concerns regarding the traditional software development process, the concept of DevOps came to the rescue as a solution around 2007. The term DevOps is a combination of development and operations, to depict the collaborative approach of working on any task. Throughout every phase of its lifecycle, development and operation teams work together and communicate to maintain alignment and quality of the product. Usually, DevOps principles apply fluently to a traditional software (non-ML project).

However, in an ML project, these principles change and must be followed accordingly to maintain the project. Therefore, to handle production-related issues in machine learning, a new concept is introduced, called as MLOps (DevOps for Machine Learning). It is a set of efficient and reliable practices for deploying and maintaining ML models in production. It delivers the capabilities to cover the domain gap and knowledge gap between data scientists and the operations team. It helps businesses to automate the system as much as possible and achieve richer insights.

1.1 Motivation

An ML model is built for solving a problem. And to do so, it must be deployed in production and usable by a specific group of end-users. According to many data scientists, model deployment is the most strenuous stage in the machine learning lifecycle [12]. A report by Algorithmia confers that many companies haven't yet figured out how to achieve their ML goals. As opposed to model operations (deploy and maintain), there is a lot of hype related to model building and training. No matter how good a model performs, when it is deployed in a production environment, unknown issues and unexpected challenges arise:–

- **Model Drifting**

  In model drift, the relationship between dependent and independent variable(s) changes over time, resulting in degradation of model performance. Due to this drift, the model starts capitulating and ends up in wrong predictions. Depending on the severity of the field of application, harmful (even deadly) impacts confront because of such results. Consider the multiple linear regression formula given below:–

  $$\hat{Y_i} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

  where,

$\hat{Y_i}$ = Target variable	X_i = Input variables	ϵ = Random error
β_0 = Intercept (Bias)		β_i = Correlation coefficient of X_i

  As the model experiences unforeseen data, it modifies the correlation coefficients and the intercept as well. And therefore, the decay of the relationship causes a threat to the business. This concept can be further classified into:–

  - **Data Drifting**

    In data drift, the data distributions differ from training data over time. The root cause behind this concept is **change in statistical properties of the predictors.** As soon as the model comes across new data (apart from training), such as personal preferences, seasonal changes, up to the minute data patterns, it becomes the victim of data drift.

  - **Concept Drifting**

    On the contrary to data drift, concept drift occurs when there is a **change in statistical properties of the output variable.** In this case, the users are completely lost and trying to predict things with completely wrong assumptions.

  - There are other problems like **feature drifting [13], label drifting,** and **prediction drifting** that lead to similar issues and cause heavy loss to the system.

- **Evolving Environment**

  "Productionizing" is a series of implementation tasks to bring the ML pipeline from the **research environment** into the **production environment**. When it comes to productionizing and testing, the environment becomes one of the key aspects for model degradation. Since the research environment and production environment are different, the model starts to malfunction as it gets evolved into the new environment. If an environment belongs to a safety-critical or security-critical system, and even if the model accuracy is 95%, it can be hazardous for the system to cause deaths, serious injuries, mega losses, etc.

- **Knowledge Gap between Data Scientists and Operations Team**

  Since machine learning is a new concept, the operation teams are not aware of the best practices to deal with it. And hence, bridging the gap between model building and model deployment is still a challenging task. There is a lot of difference between building a model in a Jupyter notebook and deploying it on a production server. Due to the disconnect between **data science and IT**, the models cannot run in the production environment for too long. The IT team tends to deploy stable applications with uninterrupted uptime at any cost. On the other hand, data scientists depend on iterations and experimentation, which becomes too tedious in the production environment.

Let us study some **case studies** to get more insights:–

- **Machine Learning for Disaster Risk Management (DRM):–**

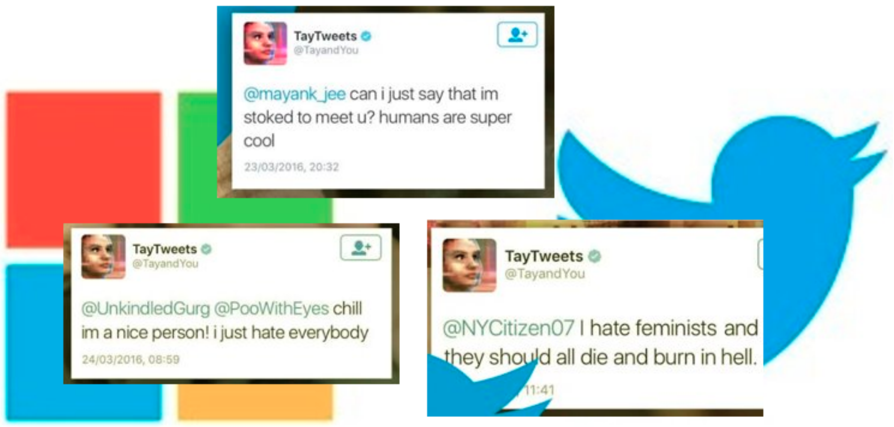
  Since ML models use past data to make decisions, they can aggravate preexisting issues available in data. In the field of **criminal justice**, a similar effect was traced, as discussed in a book by O'Neil [14]. The models for calculating criminal risk scores for individuals try to work by removing human bias. Nonetheless, they end up using superficially unbiased demographic details, and therefore, people are deprived based on race, color, and income.

  Similarly, Soden et al. [15] have mentioned a case to depict the exacerbation of **social inequalities** due to the use of biased datasets while applying ML in the field of DRM. Moreover, machine learning causes issues related to security and privacy in Violence, Fragility, and Conflict settings. Some of these cross-cutting security concerns are studied by its branch, called Fairness in ML [16].

- **Security in ML:–**

  Machine learning imposes new threat vectors during its deployment workflow as a security vulnerability. Kumar et al. [17] have covered specialized adversarial attacks on the algorithms and data. It may lead to confidentiality, integrity, or availability, which results in **intentional or unintentional failures**. Moreover, recent literature by Kumar et al. illustrates that industry professionals are not provisioned with tactical and strategic tools to detect, react and handle the attacks on their ML systems [18].

  Also, there have been cases of data poisoning, model stealing, and model inversion. One of the reasons behind this can be a collaborative approach between companies. In 2016, Microsoft released a chatbot "Tay" to the public on Twitter, which caused data poisoning by exploiting feedback loops [3]. Initially, Tay engaged people with lame jokes and gained popularity. Within a few hours, the bot learned not only the language from people on Twitter but also their values and started tweeting highly offensive things as depicted in figure 1.2. Therefore, the bot was taken down immediately.


TayTweets @TayandYou
@mayank_jee can i just say that im stoked to meet u? humans are super cool
23/03/2016, 20:32

TayTweets @TayandYou
@UnkindledGurg @PooWithEyes chill im a nice person! i just hate everybody
24/03/2016, 08:59

TayTweets @TayandYou
@NYCitizen07 I hate feminists and they should all die and burn in hell.
11:41

Figure 1.2 – Microsoft's Tay chatbot initiated as a cool girl, but quickly converted into a hate-speech-spewing disaster [3].

- **Machine Learning in Medical Applications:–**

  In a study by Jagielski et al. [19], almost half of the patients were injected an **incorrect dosage** by the linear model. Moreover, this model had changed the dosage of 75% of the patients by an average of 93.49%. And, one-tenth of these patients had their dosages changed by 358.89%. The models used in the medical field are exposed to both **white-box attacks and black-box attacks** and can even cause death or at least critical health conditions.

To infer, organizations invest billions of dollars in ML projects; however, because of their non-deterministic behavior, they cannot strive for longer in production. And, the way DevOps overcame the drawbacks of the traditional software development lifecycle, there is a need for a standardized approach to deal with the machine learning lifecycle.

1.2 Aim of Thesis

As the name indicates, "DevOps for Machine Learning," the thesis explains techniques for approaching ML models in a production environment to prolong their lifespan. It provides data scientists with visualizations, monitoring tools, and model transparency methods to understand, analyze and improve the system. It focuses on overcoming existing drawbacks, such as rotten model, model drifting, wrong predictions, time consumption, and performance degradation. Therefore, this thesis covers standardized model deployment with optimized time consumption and as much automation as possible.

1.3 Thesis Organisation

- In Chapter 1, the topic is introduced. First, an overview of machine learning, DevOps, and MLOps is covered. Then, the Motivation section comprehends the need and reason behind choosing this demanding topic.
- In Chapter 2, as a base step, the background theory is incorporated for a reference to explain the difference between DevOps and MLOps and the steps included in an ML system.
- Chapter 3 is one of the most crucial topics of this thesis that encompasses state-of-the-art research outcomes like MLOps practices, tools like AutoML and ML-Flow, transparency methods, and a monitoring technique.
- Chapter 4 comprises the implementation part, containing an end-to-end data science project as an automated application to exemplify the research outcomes.
- Chapter 5 discusses the obtained results with set benchmarks for the models and their performance.
- Chapter 6 concludes the thesis with a small scope for extension in the future.
- The second last section cites and lists the references used during the research and implementation in this thesis.
- The last sections of this thesis encompass the appendices with definitions and formulae of new concepts used, along with the details of public access to this thesis.

Chapter 2

Background Theory

Nowadays, with several years of experience, knowledge, research, and a pertinent dataset for the use case, data scientists can construct an offline model with predictive performance. But, the real challenge arises when it's time to finally build an integrated AI/ML system and operate it continuously in a production environment. According to Google [20], based on their hosted services, there are many pitfalls in operating an ML-based system in production. While setting up an environment for MLOps, factors such as CI, CD, and CT should also be considered. As depicted in the figure 2.1, many of the real-world ML systems comprise **vast and complex elements** apart from ML code such as:–

- automation,
- configuration,
- serving infrastructure,
- monitoring,
- resource management,
- data collection and verification,
- feature engineering and selection,
- process and metadata management, and other tools.

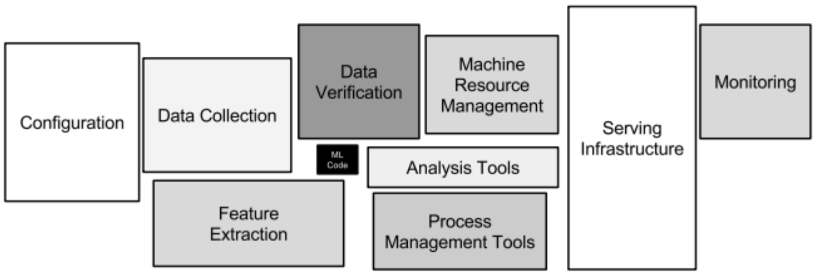


Figure 2.1 – Vast and Complex elements of ML Systems [4]

2.1 DevOps vs. MLOps

DevOps is a standardized practice to develop and operate large-scale applications and software. To accomplish the benefits of DevOps, the following 2 concepts are crucial in the system development:–

- **Continuous Integration (CI):–**

  CI is the primary and best practice of DevOps in software engineering for automating the merging of all developers' code changes into a shared mainline several times a day. This concept was adopted by extreme programming (XP) and advocated frequent integration as many as tens of times per day.

- **Continuous Delivery (CD):–**

  CD is a software engineering approach in which code changes automatically trigger a reliable release to the production. It is a practice to build, test, and release the software in short cycles with increased frequency and great speed.

An ML system is different from a software system in the following manner:–

- **Team Members and Skills:–**

  In an ML project, the team includes data scientists for model-related activities like exploratory data analysis (EDA), feature engineering, model development, and validation. They focus only on these tasks and might not be experienced in building production-level services.

- **Development:–**

  Machine learning is all about experimenting with things at different phases of their development. Depending on the constraints like speed, cost, and performance, data scientists try to find the best model for a problem after trying several features, parameter configurations, algorithms, and libraries. The challenge here is to keep track of what went well to maximize code reusability.

- **Testing:–**

  Testing in machine learning requires more involvement as compared to other software systems. Data validation, trained model quality evaluation, and model validation are also conducted, apart from unit and integration testing.

- **Deployment:–**

  Deployment in ML systems is not as straightforward as other systems. It requires the whole ML pipeline to be deployed for automatic retraining and re-deployment, as the trained model cannot be just directly deployed as a prediction service. Adding the entire pipeline is a complex and cumbersome task, and it requires the automation of manual model development steps.

- **Production:–**

  Once the model is in production, its performance automatically starts degrading due to constantly evolving data profiles and sub-optimal coding. Therefore, there are more chances and ways of decaying for ML models than traditional software systems. To tackle such issues, the summary and statistics of the model and data should be tracked, logged, and monitored. So that, in case of problems, alerts are generated to retrain or roll back the model.

- **CI:–**

  Continuous Integration in ML systems includes testing and validation of not only code and components, but also data, schemata, and models.

- **CD:–**

  Continuous Delivery in ML systems is no longer about a service or a software package, but the whole training pipeline that can automatically deploy another model prediction service.

- **Continuous Training (CT):–**

  CT is unique to ML systems, which focuses on retraining and serving the models automatically. This property is frequently triggered to retrain the model on new data and features so that the model can update its weights to adapt new behaviors.

2.2 Data Science Steps in ML Systems

In this section, typical steps are discussed for training and evaluating an ML model to serve as a prediction service. These steps can be realized manually or using an automated pipeline. Once a business use case is defined, and the success criteria is established, the following phases are carried out in delivering an ML model to production:–

- **Data Fetching/Extraction:–**

  In this step, relevant data is collected and integrated from authentic sources.

- **Data Analysis:–**

  In this phase, the available data is understood for building the ML model. This can be done manually or by using some approach, like EDA. It helps to figure out data preparation techniques needed for building the model.

- **Data Pre-processing:–**

  In this step, the data is prepared for the model. It is the most salient phase in model development. The outcome of this step is the processed data split in training, validation (optional), and testing sets. This step is necessary to avoid the outcomes of "garbage in, garbage out", and it includes:–

  – **Data Cleaning:–**

    In data cleaning, tasks like resolving inconsistencies, filling in missing values, identifying outliers, and smoothing noisy data are performed.

  – **Data Transformation:–**

    Data transformation is performed when one of the variables dominates other variables in the dataset. It includes techniques like normalization, aggregation, and removal of skewness.

  – **Variable Construction:–**

    In some cases, the existing variable(s) are used to derive new variables (variable construction), and the old variable(s) are removed.

  – **Data Reduction:–**

    Data reduction comprises of reduction of several records, variables, and variable levels to avoid issues like "Curse of Dimensionality".

  The figure 2.2 depicts the classification tree for the techniques used in data pre-processing and the colors represent respective levels:–

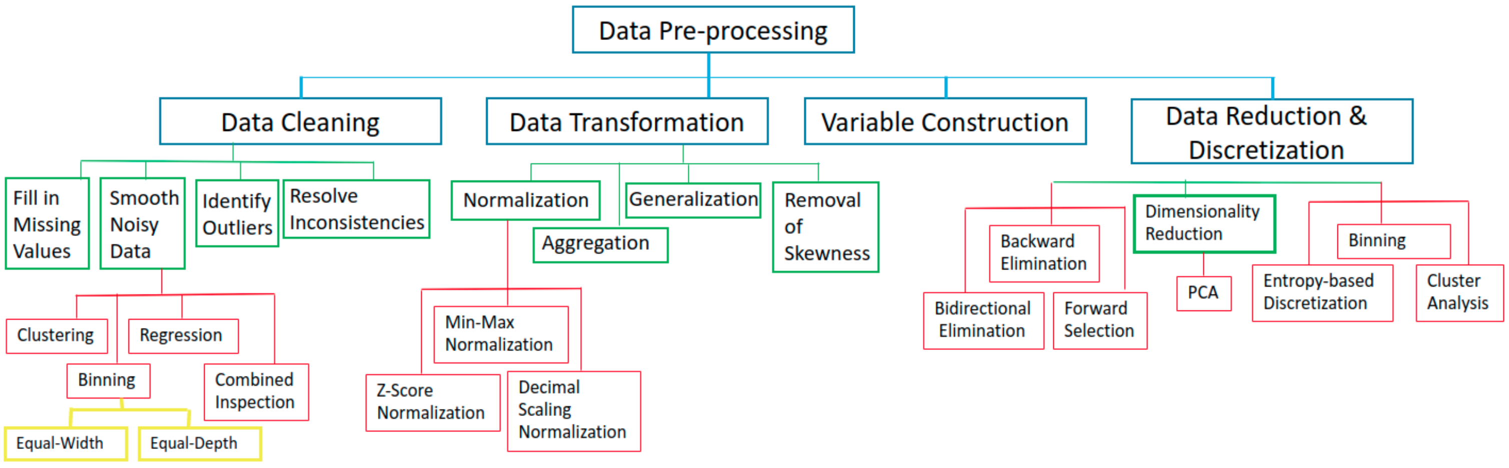


Figure 2.2 – Data Preparation Classification Tree

- **Model Building and Training:–**

  In this step, the actual model is built and trained. First, several algorithms are implemented with the processed data to figure out which yields the best results. Then, a set of hyper-parameters are tuned to obtain the optimized version of the model. The outcome of this step is a trained model.

- **Model Evaluation:–**

  The trained model is now evaluated on the test dataset to assess its quality. The quality/performance of a model is evaluated using a couple of metrics such as accuracy score, precision score, recall score, f1-score, confusion matrix, AIC, SBC, ROC curve, and accumulated lift curve.

- **Model Validation:–**

  In this step, it is decided whether the model is adequate for deployment, i.e., its predictive performance is better than a certain baseline such as a statistical significance of 95%, p-value <= 0.04, or accuracy >= 90%.

- **Model Post-processing:–**

  In some applications, the data needs to be transformed back to its original format, and this is achieved in this optional step.

- **Model Serving:–**

  The validated model is now deployed in a production environment to serve predictions for users. The model serving can be accomplished in one of the following forms:–

  – Web service / micro-service having a REST API

  – Embedded with an edge device

  – Batch prediction

The figure 2.3 illustrates a model deployed as a web service and being used by a web application:–

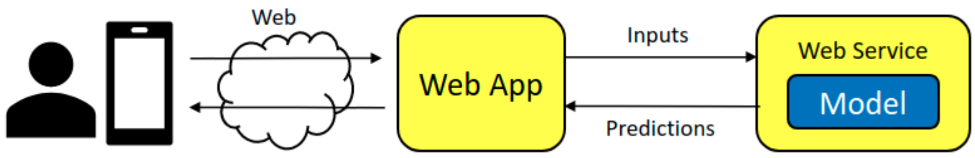


Figure 2.3 – Model deployed as a Web Service

- **Model Monitoring:–**

  Once the model is deployed, it is continuously monitored to invoke a new iteration depending on its predictive performance.

Chapter 3

State of the Art

In this chapter, the concept of MLOps is illustrated in detail so that data scientists and ML engineers can apply DevOps principles to ML systems. MLOps is an engineering practice for the fusion of ML development (Dev) and ML operations (Ops) that leads to automation of all the steps involved in its lifecycle.

3.1 Levels in MLOps

The maturity of an ML process is based on the level of automation of the aforementioned steps, which reflects the rate of expeditiousness of retraining models in a production environment, provided new data, or new implementation. In this section, three levels of MLOps are described that commence from the most common practices (with zero automation) up to the best practices (automating both ML and CI/CD pipelines).

3.1.1 Level 0: Manual Process

At level 0, or basic level of maturity, the entire building, and deployment of the model are accomplished manually by the data scientists or ML engineers. The figure 3.1 represents the workflow for this level:–

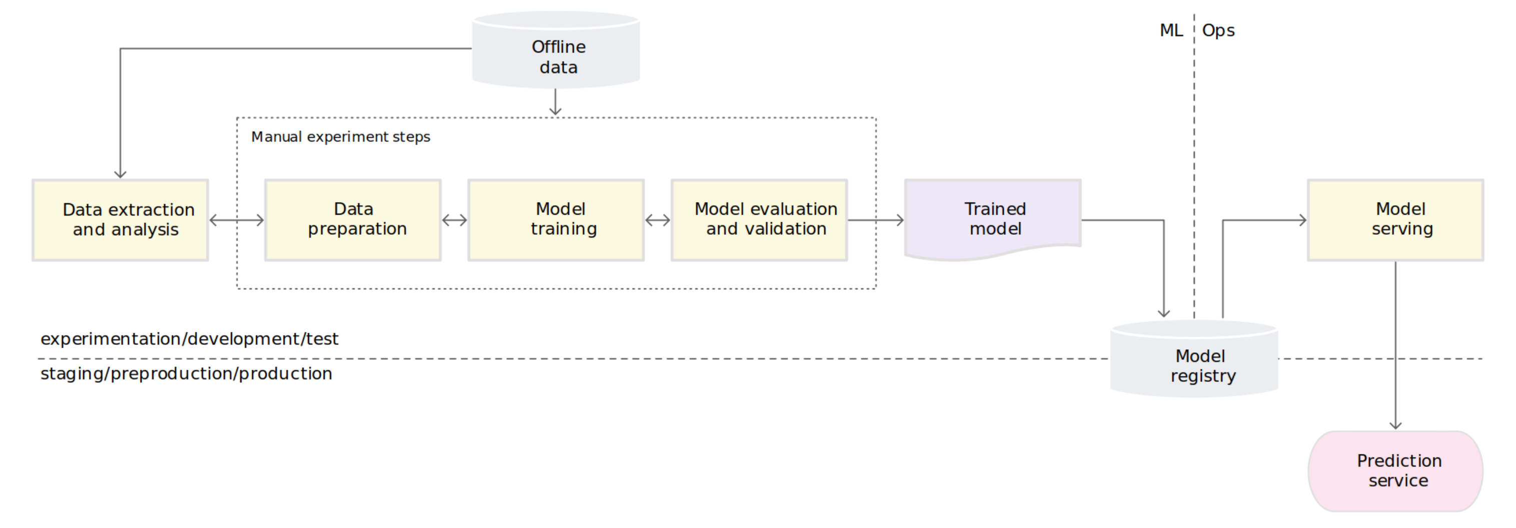


Figure 3.1 – MLOps Level 0 [5]

3.1.1.1 Characteristics/Challenges

MLOps level 0 is familiar in many businesses which are new to this field. Following are the highlights of characteristics possessed by this level:–

- **Script-based Process:–**

  At level 0, every step is implemented manually, including data analysis, data preparation, model training, and validation. Not only execution, but it also requires a manual transition from the current step to the next step. This process is driven by experimenting with several scripts in notebook interactively, until an optimized or at least workable model is obtained.

- **Separation between Development and Operation team:–**

  This process disconnects the data scientists who build the model and engineers who serve the model in a production environment as a prediction service. At this level, the data scientists are only concerned to come up with a working model and hand it over as an artifact to the engineering team for deployment. This handover can be realized in several ways, for example, uploading the trained model in a storage location or a model registry, and checking its executable (object code) to a code repository. Then, the engineers deploy the model and try to make the service available for low-latency serving. It usually leads to training-serving skew.

- **Few Releases:–**

  At this level, a new model version is rarely deployed in a year based on the assumption that models don't change frequently. Therefore, this level has infrequent release iterations.

- **Missing CI:–**

  Since only a few implementation changes are assumed, CI is often ignored at this level. The code is tested only once at the time of script execution. These scripts are source controlled and produce model artifacts such as trained models, evaluation metrics, and visualizations.

- **Missing CD:–**

  MLOps level 0 has no frequent model deployments and therefore, there is no continuous delivery involved in this process.

- **Incomplete Deployment:–**

  Instead of deploying the whole ML pipeline, only the trained model is deployed as a prediction service at this level.

- **Lack of Monitoring:–**

  The process doesn't track the live performance of model predictions, and the engineers are unaware of the underlying processes. This level lacks a monitoring service, which is essential for identifying the model's performance degradation and model drifts.

In practice, models often break as soon as they are deployed in a production environment because they fail to adapt to the dynamic behavior or changes in the data. To overcome these challenges, the next levels are preferred, which consist of active monitoring, CI/CD, and CT.

3.1.2 Level 1: Automating ML Pipeline

To take things one level up, MLOps level 1 is used to perform continuous training of the models by automating the whole pipeline. To automate the process of retraining models on new data, automated data and model validation steps must be introduced to the pipeline, apart from pipeline triggers and metadata management. The figure 3.2 is a schematic illustration of an automated ML pipeline for CT:–

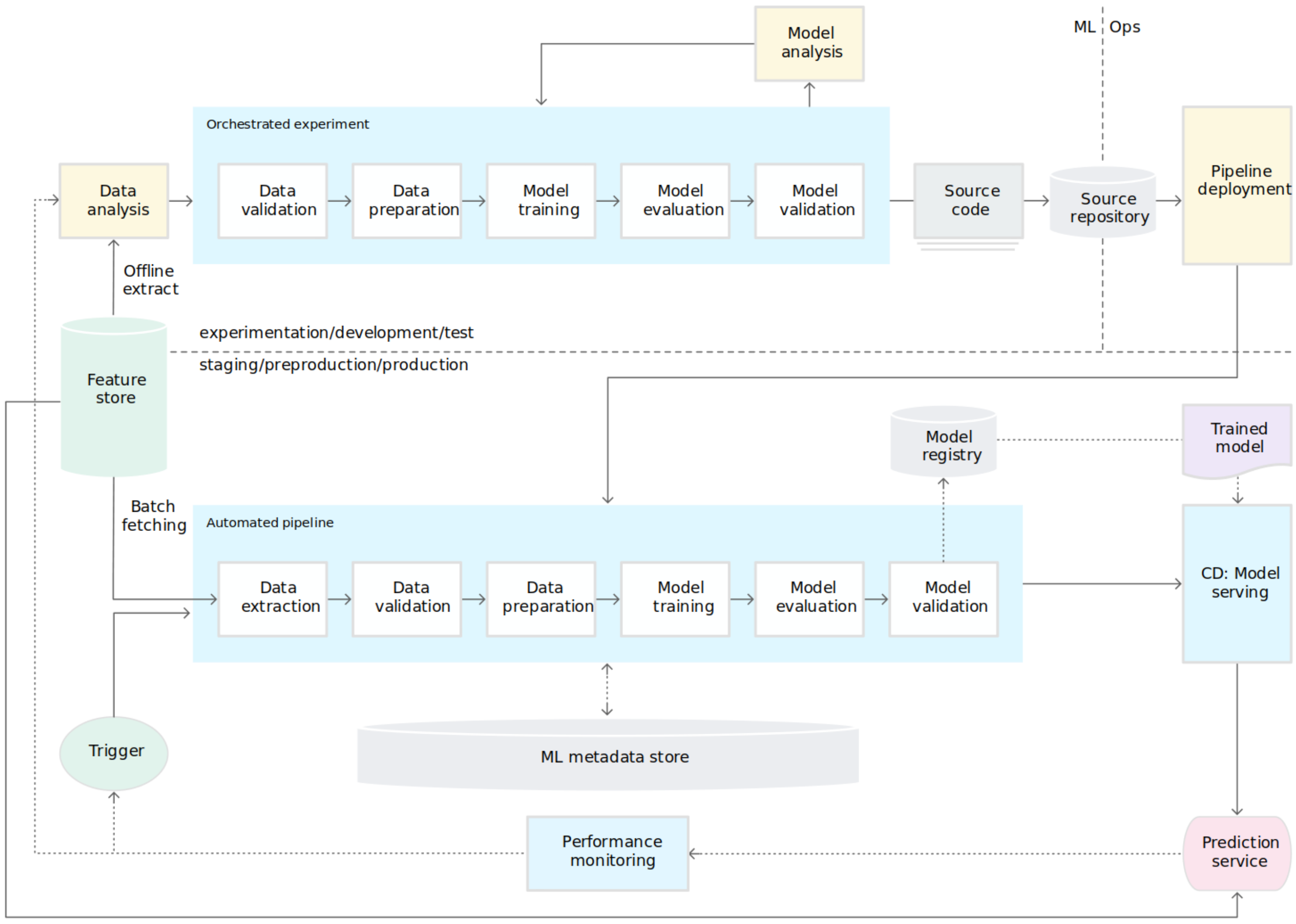


Figure 3.2 – MLOps Level 1 [5]

3.1.2.1 Characteristics

- **Quick Experimentation**

  The transition from one step to another is automated, which leads to the rapid execution of experiments. The steps of ML iteration are orchestrated to achieve better readiness to deploy the whole pipeline.

- **Modularized Code:–**

  The components must be reusable and shareable among other pipelines to build ML pipelines. Therefore, the source code for the elements needs to be modularized. Moreover, the components should be containerized to achieve the following benefits:–

  - To make code reproducible from development to the production environment.
  - To isolate each component in the pipeline since they can have their own versioning, languages, and libraries.
  - To decouple the execution environment from the custom environment.

- **Unified Pipeline:–**

  The pipeline used in a development environment is reused in staging and production environment, which is the main aim of MLOps practice, i.e., the fusion of DevOps.

- **Continuous Delivery:–**

  The production pipeline continuously offers prediction services to new models. At this level, the model deployment step to serve the trained and validated models for online predictions is automated.

- **Continuous Training:–**

  The model is automatically retrained in production on new data through pipeline triggers.

- **Deployment:–**

  At level 1 of MLOps, the whole pipeline is deployed, which runs automatically and recurrently to serve the trained model as a prediction service.

3.1.2.2 Added Advantage

With added advantages, come added implementations. Following components must be added to the implementation to pursue all the benefits of this level:–

- **Data Validation:–**

  Once the pipeline is deployed, the ML **pipeline triggers** automatically execute the pipeline on new data to produce a new model. Before model training, it is required to decide whether to retrain the model or stop the pipeline execution. And this decision is automatically taken in case of following skews:–

  - **Data Values Skews:–**

    As discussed in section 1.1, these skews are caused by changes in the statistical properties of data. In such cases, retraining must be triggered to capture and learn changes in data patterns.

  - **Data Schema Skews:–**

    Anomalies in the input data cause schema-related skews, such as the new data received doesn't comply with the expected schema. In such a case, the pipeline execution should be stopped to investigate the causes and then come up with the fix. These skews are also caused by receiving insufficient features, receiving unexpected features, receiving features with unexpected values, etc.

- **Model Validation:–**

  It is performed after the model is trained successfully on new data. Before promoting to production, it is evaluated and validated as follows:–

  - The predictive quality of the model is assessed by producing evaluation metric values on a test dataset.
  - The metric values of the new model are compared with the current model and made sure that the new model yields better results as compared to the current model.
  - It is made sure that the model performance is consistent across all data segments.
  - The model is tested for deployment and infrastructure compatibility.

- **Feature Store:–**

  A feature store is a centralized repository with standardized definition, storage, and access for the training and serving features. To support these workloads, it provides high-throughput batch serving and low-latency real-time serving. Following are the uses of a feature store:–

  - Instead of recreating feature sets for the entities, it allows to discover and reuse the same.
  - It avoids having similar features with several definitions by maintaining feature metadata.
  - It serves the latest feature values from the feature store.
  - More importantly, it avoids the training-serving skews, by using the store as a data source for experimentation, CT, and online serving. It also makes sure that the features used during training are identical to the ones used during serving. It is realized as follows:–
    - **Experimentation:–** The data scientists run their experiments by getting an offline extract from the feature store.
    - **CT:–** The automated ML pipeline fetches a batch of the latest feature values of the dataset used for training.
    - **Online Serving:–** The prediction service fetches a batch of the feature values related to the requested entity.

- **Metadata Management:–**

  In metadata management, data about each execution of the pipeline is tracked to help with data and artifacts lineage, debug errors and anomalies, etc. Whenever a pipeline is run, the metadata store records following metadata:–

  - Versions of the pipeline and components
  - Time, start date, end date, execution time by each step
  - Executor
  - The parameters/arguments passed
  - The pointers to the artifacts produced by each component (in case of failures, tracking these intermediate outputs are helpful to resume the pipeline execution from the most recent step)
  - A pointer to the previously trained model (in case of rollback or to produce evaluation metrics for comparison during model validation step)
  - Model evaluation metrics

- **Pipeline Triggers:–**

  The retraining of ML pipelines is automated using ML pipeline triggers, depending on the following use cases:–

  - **On Demand:–** The pipeline execution is triggered manually, ad hoc.
  - **Schedule Based:–** When new and labeled data is available in a timely manner, the retraining frequency can also be set accordingly based on how frequently the data patterns change, how frequently new data is available, how expensive it is to retrain the models, etc.
  - **Availability of New Data:–** When new data isn't systematically available, the pipeline can be triggered on an ad-hoc basis when new data is available in the source databases.
  - **Model Performance Degradation:–** When there is noticeable performance degradation, the pipeline is triggered, and the model is retrained.
  - **Changes in Data Distribution:–** If significant changes on the data distributions of the features are noticed, the model has gone stale and must be retrained on new data.

3.1.2.3 Challenges

At this level, new pipeline implementations are manually deployed and tested. The tested source code for the pipeline is submitted to the IT team for deployment. To automate these steps, try new ML ideas, and rapidly deploy new implementations of the ML components, there is a need for an automated CI/CD setup, which is achieved in the next level of MLOps.

3.1.3 Level 2: Automating CI/CD Pipeline

In the highest level of MLOps, a robust and automated pipeline is used for a rapid and reliable update of the pipeline in the production environment. It alleviates data scientists to scout new ideas around feature engineering, model structure, and hyper-parameters. After implementation, these ideas get built, tested, and deployed automatically as new pipeline components in the target environment. The figure 3.3 shown below depicts the implementation of an ML pipeline using CI/CD that offers automated pipeline setup and automated CI/CD routines:–

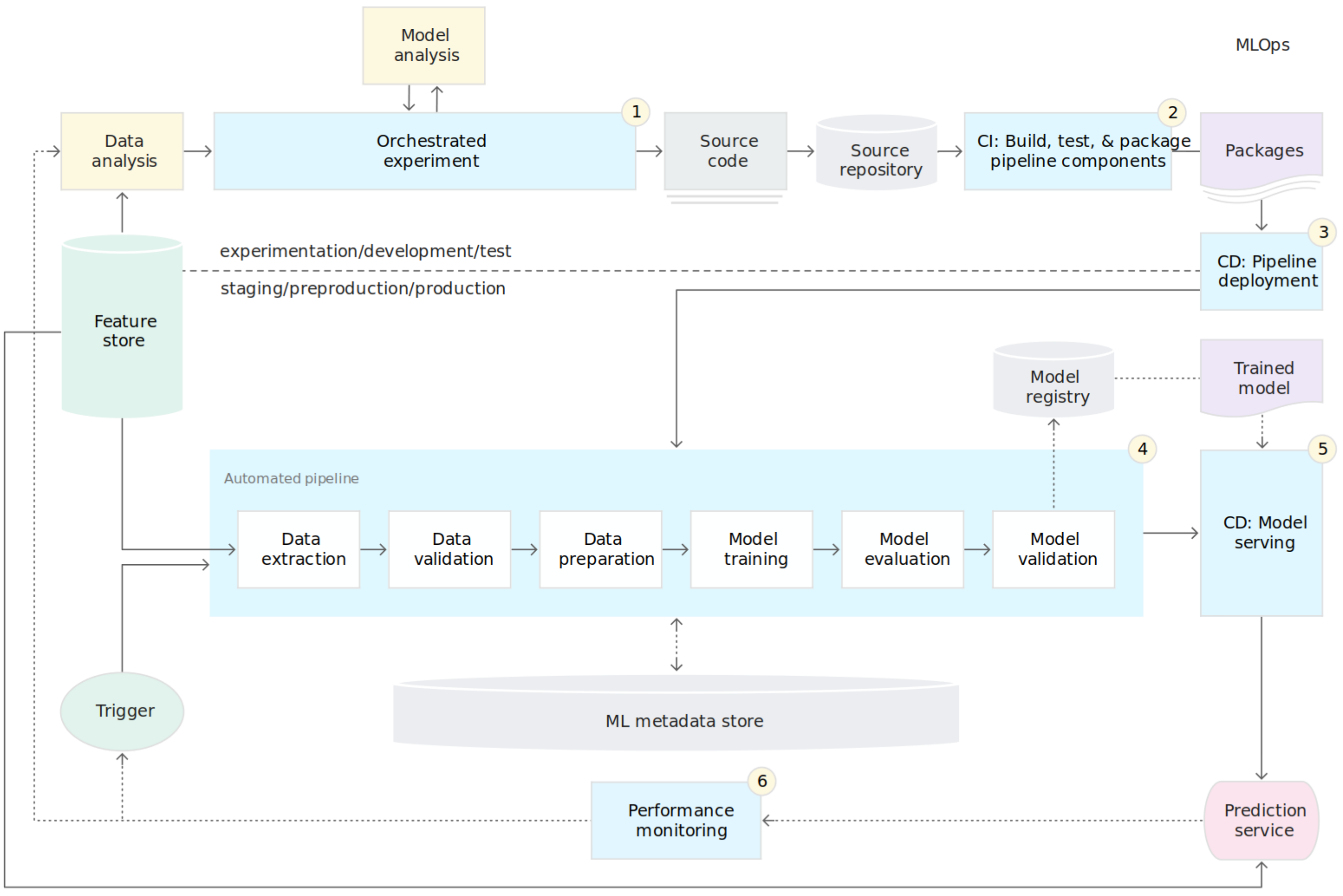


Figure 3.3 – MLOps Level 2 [5]

This setup consists of components such as source control, model registry, feature store, metadata store, pipeline orchestration, and build, test, and deployment services.

3.1.3.1 Characteristics

The ML CI/CD automated pipeline consists of the following stages:–

- **Development and Experimentation:–**

  At this stage, new algorithms and models are frequently tried out, and these experiments are orchestrated. This stage dispatches the source code of a pipeline that is pushed to a source repository.

- **Pipeline CI:–**

  After building the source code, numerous tests are run on it. The output of this stage is the pipeline components to be deployed. Apart from building packages and container images, the CI process performs the following tasks:–

  – Test feature engineering.

  – Test methods implemented in the model.

  – Test whether the model converges.

  – Test that the model doesn't produce NaN values.

  – Test whether each component in the pipeline generates expected artifacts.

  – Test integration between pipeline components.

- **Pipeline CD:–**

  Now, the artifacts produced by the CI stage are deployed in a production environment. This stage results in a new pipeline deployed with a new implementation of the model. Following best practices should be considered in this stage:–

  – Before deploying, the compatibility of the model with the target infrastructure is verified.

  – The prediction service is tested by calling the service API with valid inputs and, it is made sure that expected results are obtained, to avoid versioning conflicts.

  – The prediction service performance is tested in the form of queries per second (QPS), model latency, etc.

  – The data for retraining and batch prediction is validated.

  – It is verified whether the models meet predictive performance targets before deployment.

  – It is tested whether automated deployment works using a test environment.

- **Automated Triggering for Retraining:–**

  In response to a trigger, the pipeline is automatically executed in production. Therefore, a new version of the model is generated and pushed to the model registry.

- **Model CD:–**

  The trained model is served as a prediction service and this stage provides a deployed model prediction service.

- **Monitoring:–**

  Based on live data, model statistics are collected for monitoring. The output of this stage is a trigger to execute:–

  – a pipeline, or,

  – a new experiment cycle.

3.1.3.2 Summary

Implementing machine learning in a production environment does not mean deploying the models as an API for prediction. Rather, it means deploying a machine learning pipeline to automate the retraining and deployment of new models. Setting up a CI/CD system enables automatic testing and deployment of new pipeline implementations to cope with rapid changes in the business environment. And, there is no need to suddenly shift all the processes to the highest level. These practices must be implemented gradually to improve the automation of an ML system.

3.2 Bonus for Data Scientists

In this section, the research outcomes of this thesis are discussed, to help data scientists in various phases of MLOps. These components fit in the MLOps process and enhance the practice by bringing more understanding and transparency.

3.2.1 AutoML [1]

Automated Machine Learning (AutoML) is an enthusiastic helper that allows aspiring data scientists to develop optimized models without any prior knowledge/with only basic knowledge in this field. It is the major topic in the AI community that covers domains like machine learning, evolutionary algorithms, and reinforcement learning. AutoML generates the best-performing model out of provided inputs like data and training time; without worrying about data preparation, algorithm selection, and hyperparameter optimization. During recent years, several off-the-shelf packages have been developed that can be used easily, without expert knowledge:–

- **AutoWEKA [21]:–**

  Combined with the WEKA [22] package, AutoWEKA automatically yields adequate models for a wide variety of data sets. It is an approach for the simultaneous selection of an algorithm and its hyperparameters. It achieves this using an automated approach by leveraging the latest innovations in Bayesian optimization.

- **Auto-Sklearn [23]:–**

  Being an extension of AutoWEKA, Auto-Sklearn leverages recent advantages in Bayesian optimization, meta-learning, and ensemble construction to free an ML user from algorithm selection and hyperparameter tuning. As of now, it cannot be used on Windows OS. To install it on Linux systems, execute the following commands in the terminal:–

```
    # Install Dependencies
    sudo apt-get install build-essential swig
    # Install the Library
    pip install auto-sklearn
```

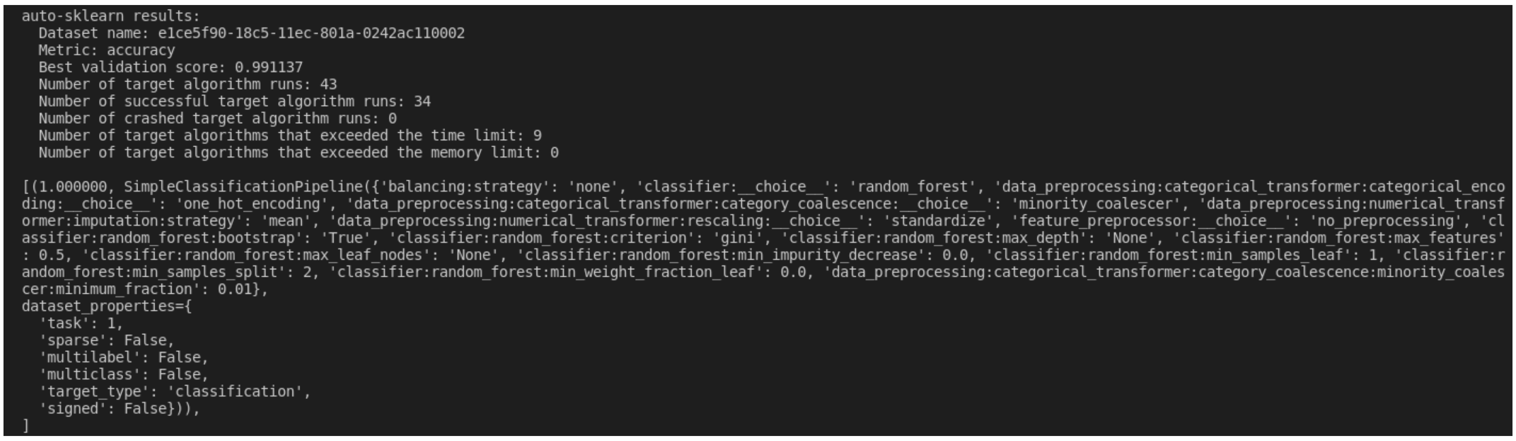
Listing 3.1 – Auto-Sklearn Installation

To use this automated machine learning toolkit in a code snippet, implement the following lines:–

```python
# Import the Library
from autosklearn.classification import AutoSklearnClassifier

# Create the Model
classifier = AutoSklearnClassifier(
                time_left_for_this_task=3 * 60,
                per_run_time_limit=30,
                ensemble_size=1,
                initial_configurations_via_metalearning=0,
                n_jobs=-1,
            )

# Train the Model
classifier.fit(X_train, y_train)

# Make new Predictions
y_pred = classifier.predict(X_test)
```

Listing 3.2 – Auto-Sklearn Implementation

The above code snippet creates a classification model, which takes 3 minutes to train. Depending upon the constraints, the model parameters can be set, for example, time to run, time per iteration, etc. During training, it tries multiple random possibilities with different algorithms and hyperparameters on given data and returns the optimized model. The results generated by the above code snippet can be illustrated as shown in figure 3.4:–



```
auto-sklearn results:
  Dataset name: e1ce5f90-18c5-11ec-801a-0242ac110002
  Metric: accuracy
  Best validation score: 0.991137
  Number of target algorithm runs: 43
  Number of successful target algorithm runs: 34
  Number of crashed target algorithm runs: 0
  Number of target algorithms that exceeded the time limit: 9
  Number of target algorithms that exceeded the memory limit: 0

[(1.000000, SimpleClassificationPipeline({'balancing:strategy': 'none', 'classifier:__choice__': 'random_forest', 'data_preprocessing:categorical_transformer:categorical_enco
ding:__choice__': 'one_hot_encoding', 'data_preprocessing:categorical_transformer:category_coalescence:__choice__': 'minority_coalescer', 'data_preprocessing:numerical_transf
ormer:imputation:strategy': 'mean', 'data_preprocessing:numerical_transformer:rescaling:__choice__': 'standardize', 'feature_preprocessor:__choice__': 'no_preprocessing', 'cl
assifier:random_forest:bootstrap': 'True', 'classifier:random_forest:criterion': 'gini', 'classifier:random_forest:max_depth': 'None', 'classifier:random_forest:max_features'
: 0.5, 'classifier:random_forest:max_leaf_nodes': 'None', 'classifier:random_forest:min_impurity_decrease': 0.0, 'classifier:random_forest:min_samples_leaf': 1, 'classifier:r
andom_forest:min_samples_split': 2, 'classifier:random_forest:min_weight_fraction_leaf': 0.0, 'data_preprocessing:categorical_transformer:category_coalescence:minority_coales
cer:minimum_fraction': 0.01},
dataset_properties={
  'task': 1,
  'sparse': False,
  'multilabel': False,
  'multiclass': False,
  'target_type': 'classification',
  'signed': False})),
]
```

Figure 3.4 – Terminal Output

- **Auto-PyTorch [24]:–**

  Being a deep learning framework, Auto-PyTorch optimizes the neural architecture along with hyperparameters. It is developed for the fusion of some of the early AutoML frameworks focused on traditional ML algorithms and some other frameworks focused on neural architecture search. The main reason behind its existence is to enable fully automated deep learning (AutoDL). Its latest version can work with tabular data and image data for regression and classification [25].

****Note:– Above listed packages are developed by AutoML.org, which is led by the academic research groups at the University of Freiburg and the Leibniz University of Hannover. Since AutoML is a new concept and it is still under development, it is important to apply these solutions with caution.***

Following are some other AutoML packages developed by other companies:–

- **AutoGluon [26]:–**

  AutoGluon is an easy-to-use and easy-to-extend AutoML library for tabular prediction, image prediction, object detection, text prediction, and multimodal prediction. It is intended for both, beginners and experts in this field. It enables data scientists to:–

  - Improve bespoke models and data pipelines easily,
  - Prototype deep learning and traditional ML solutions quickly,
  - Utilize state-of-the-art techniques automatically without expert knowledge, and
  - Leverage automatic model ensembling, hyperparameter tuning, and architecture search.

- **H2O AutoML [27]:–**

  The H2O AutoML interface is designed to automate the machine learning workflow. It is an open-source, distributed, in-memory, and scalable platform for predictive analytics and machine learning that allows building ML models on big data and provides easy productionalization of the models in an enterprise environment. It offers a number of model explainability methods that can be applied to individual models and groups of models.

- **MLBoX [28]:–**

  MLBox is a powerful AutoML library with three components, preprocessing, optimization and prediction. It provides the following features:–

  - Distributed data processing
  - Fast reading
  - Robust feature selection
  - Leak detection
  - High-dimensional hyper-parameter optimization
  - State-of-the-art models for classification and regression
  - Model interpretation

- **TransmogrifAI [29]:–**

  TransmogrifAI is an AutoML library for building reusable, modular, and strongly typed ML workflows based on Apache Spark with minimal hand-tuning. It is written in Scala.

- **TPOT [30]:–**

  TPOT is a data science assistant to optimize machine learning pipelines using genetic programming. It is built on top of scikit-learn and is still under active development.

3.2.2 MLflow [2]

MLflow is an open-source, ML lifecycle platform for experimentation, reproducibility, deployment, and a central model registry. It works with any ML library in any language and runs all the same in any cloud platform. It is developed to scale from a single user to large organizations. Currently, MLflow provides the following four components:–

- **MLflow Tracking:–**

  The MLflow Tracking is an API to record and query experiments that consist of code, results, data, and configurations. It provides a UI for logging parameters, code versions, metrics, and output files. Python, REST, R API, and Java APIs can be tracked with MLflow tracking. It deals with the concept of "runs" and each run is an execution of a piece of data science code. Following things are recorded by a run:–

  - **Code Version**, the git commit hash used for the run.
  - **Start & End Time**, start and end time of the run.
  - **Source**, name of the file to launch the run.
  - **Parameters**, input parameters for the model, for example, n_estimators, max_features, max_depth, criterion etc.
  - **Metrics**, model metrics such as accuracy score, confusion matrix, f1-score throughout the run.
  - **Artifacts**, output files to record images, models, and data files.

  To install MLflow Tracker, execute the below command:–

```
    pip install mlflow
```

Listing 3.3 – MLflow Installation

To use MLflow, integrate similar code in your program:–

```python
# Import the Library
import mlflow

# Set tracking URI
mlflow.set_tracking_uri('./mlruns')

# Create an experiment
raf_exp_id = mlflow.set_experiment("Random-Forest Experiment")

# Start Tracking
with mlflow.start_run(experiment_id=raf_exp_id):
        # Create the model
        # Train the model

        # Log parameters, metrics, and model
        mlflow.log_params({'n_estimators': n_estimators, ...})
        mlflow.log_metrics({'Accuracy Score': accuracy, ...})
        mlflow.sklearn.log_model(model, "RF Classifier")
```

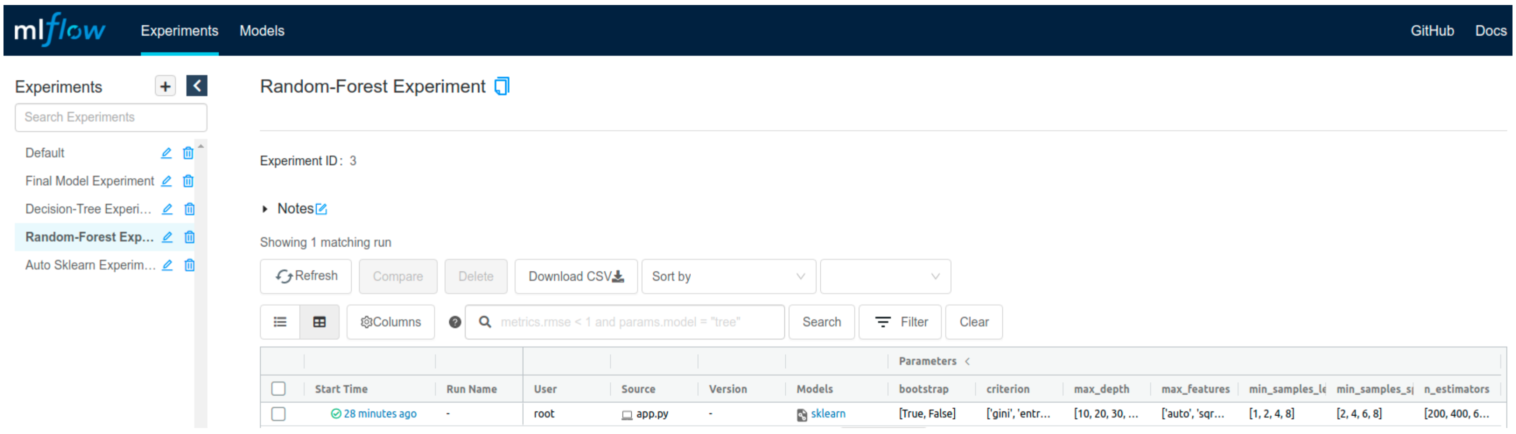
Listing 3.4 – MLflow Implementation

To run MLFlow Tracking in the web browser, execute the following command:–

```
    mlflow ui --host 0.0.0.0 --port 5000
```

Listing 3.5 – Run MLflow

The Figure 3.5 represents a run captured by MLflow and figure 3.6 represents the artifacts recorded by the run:–



Random-Forest Experiment

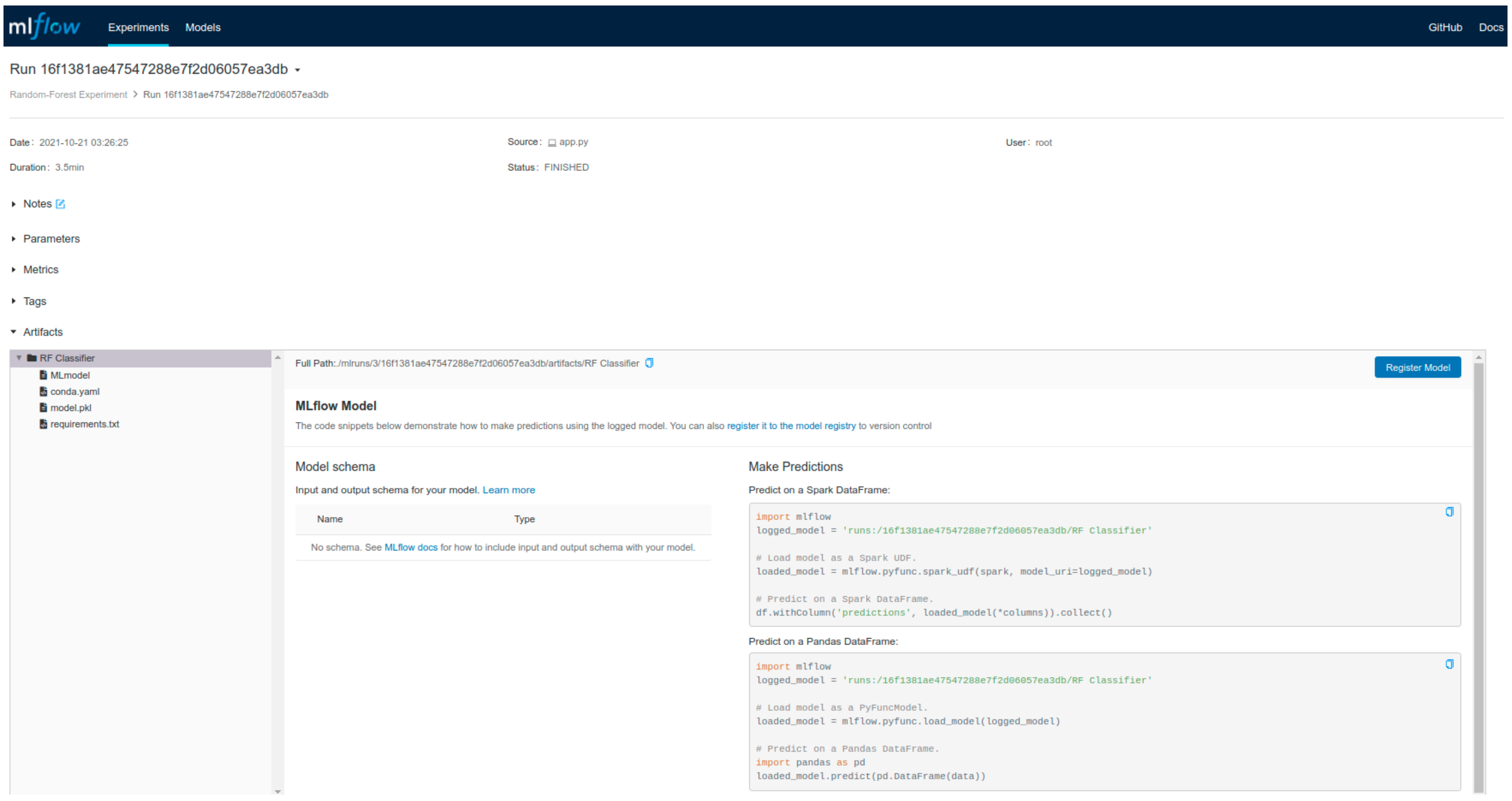
Experiment ID: 3

Notes

Showing 1 matching run

						Parameters						
Start Time	Run Name	User	Source	Version	Models	bootstrap	criterion	max_depth	max_features	min_samples_le	min_samples_s	n_estimators
28 minutes ago	-	root	app.py	-	sklearn	[True, False]	['gini', 'entr...	[10, 20, 30, ...	['auto', 'sqr...	[1, 2, 4, 8]	[2, 4, 6, 8]	[200, 400, 6...

Figure 3.5 – An MLflow Run



Run 16f1381ae47547288e7f2d06057ea3db

Random-Forest Experiment > Run 16f1381ae47547288e7f2d06057ea3db

Date: 2021-10-21 03:26:25
Source: app.py
User: root
Duration: 3.5min
Status: FINISHED

- Notes
- Parameters
- Metrics
- Tags
- Artifacts

RF Classifier
- MLmodel
- conda.yaml
- model.pkl
- requirements.txt

Full Path:./mlruns/3/16f1381ae47547288e7f2d06057ea3db/artifacts/RF Classifier

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control

Model schema

Input and output schema for your model. Learn more

Name	Type
No schema. See MLflow docs for how to include input and output schema with your model.	

Make Predictions

Predict on a Spark DataFrame:

```
import mlflow
logged_model = 'runs:/16f1381ae47547288e7f2d06057ea3db/RF Classifier'

# Load model as a Spark UDF.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model)

# Predict on a Spark DataFrame.
df.withColumn('predictions', loaded_model(*columns)).collect()
```

Predict on a Pandas DataFrame:

```
import mlflow
logged_model = 'runs:/16f1381ae47547288e7f2d06057ea3db/RF Classifier'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)

# Predict on a Pandas DataFrame.
import pandas as pd
loaded_model.predict(pd.DataFrame(data))
```

Figure 3.6 – Artifacts recorded by the Run

- **MLflow Projects:–**

  An MLflow Project packages ML code in a reusable and reproducible way by offering command-line tools to chain together projects into workflows. Each project is a directory of files, or a git repository, containing the code, and specifying the following properties:–

  – **Name**, to name the project.
  – **Entry Points**, to include commands run within the project with their parameters.
  – **Environment**, to mention the software environment used by the project, such as conda, and docker container environment.

- **MLflow Models:–**

  An MLflow Model is a standard format to package ML models and save them in different flavors that can be understood by and used in a variety of downstream tools. Each MLflow Model is a directory containing arbitrary files, together with an "MLmodel" file in the root of the directory, as shown below:–

```
my_model/
|-- MLmodel
|-- model.pkl
|-- conda.yaml
|-- requirements.txt
```
Listing 3.6 – MLflow Model Storage Format

- **MLflow Registry:–**

  An MLflow Model Registry component is a centralized model store to manage the complete life cycle of an MLflow Model. It provides model versioning, stage transitions, model lineage, and annotations. To achieve its functionality, it makes use of the following concepts:–

  - **Model**, an MLflow Model.
  - **Registered Model**, an MLflow Model registered with the Model Registry.
  - **Model Version**, version of the registered model.
  - **Model Stage**, assigned stage for the model at any given time.
  - **Annotations and Descriptions**, annotations for the model using Markdown.

Therefore, this simple tool can be very helpful in MLOps to monitor, log and analyze the models in production based on different factors such as experiments, runs in each experiment, metrics, and parameters in each run, etc.

3.2.3 Transparency

Transparency in AI systems is crucial in today's era to understand underlying implementation, as this field is flooded with third-party libraries. AI is intrinsically non-deterministic, especially ML systems that continually evolve and update. Apart from being the need of an hour, transparency offers many benefits for data scientists. The information gained through transparency can help data scientists to create more robust models and prevent them from creating models that can harm society. Therefore, the data, system, and business models should seek transparency. Moreover, ML systems and their decisions should be conveyed in a language understandable by the concerned stakeholders. People need to be aware that they are interacting with an AI system and must be informed about the system's capabilities and vulnerabilities.

Felzmann H. et al. [31] have described 9 principles, distributed in 3 phases for transparency as listed below:–

- **Phase 1: Design of AI Systems**
  - **Proactivity,** the systems should be proactive, not reactive, such that the realization of transparency should be included from the beginning of the development of an AI system.
  - **Integration,** to consider the complexity of transparency as an integrative process.
  - **Audience Focus,** to communicate in an **audience-sensitive** manner.
- **Phase 2: Information on Data Processing and Analysis**
  - **Data Processing,** to explain the data being used and how it is being processed.
  - **Decision-Making Standards,** to explain decision-making criteria.
  - **Risk Disclosure,** to explain the risks and risk mitigation measures.
- **Phase 3: Organizational and Stakeholder-Oriented Transparency Management**
  - **Inspectability,** to ensure inspectability and auditability.
  - **Responsiveness,** to be responsive to stakeholder queries and concerns.
  - **Reporting,** to report diligently about the system.

Traceability can facilitate the auditability as well as the explainability of the model. Explainability deals with how the system treats the input data, how the model transforms it into final prediction, and how to transform the technical part in a more friendly approach to the stakeholder. Moreover, trade-offs might happen between enhancing the explainability and reduced accuracy or vice-versa.

Including transparency in ML systems helps:–

- developers to debug, tune, and optimize models more easily.
- end-users to better understand why particular results are generated.

The methods used to improve the transparency in the models are broadly classified into:–

- **Local Methods:–**
  Local methods deal with explaining the predictions one by one and are used to check:–
  - what features were used to make a specific prediction,
  - how those features were used,
  - how those features contributed,
  - which features are important,
  - what could happen if some changes were made, and much more.

  Hence, data scientists can see the actual working and influence of all the features on the prediction. And, they can improve the decision-making by manually changing the order of some of the features (if requirement demands). The change in order can be based on relevance/importance, stakeholder requirement, etc. Moreover, local explanations are more accurate than global explanations.

- **Global Methods [32]:–**
  The global methods are applied by taking the group of instances and treating them as a complete dataset. They help to understand the distribution of the outcome based on the features. They describe the average behavior of the model. However, it is too difficult to achieve this in practice. While global interpretability is usually out of reach, at least some models are understandable on a modular level.

The figure 3.7 shown below illustrates the classification of transparency methods covered in this thesis:–

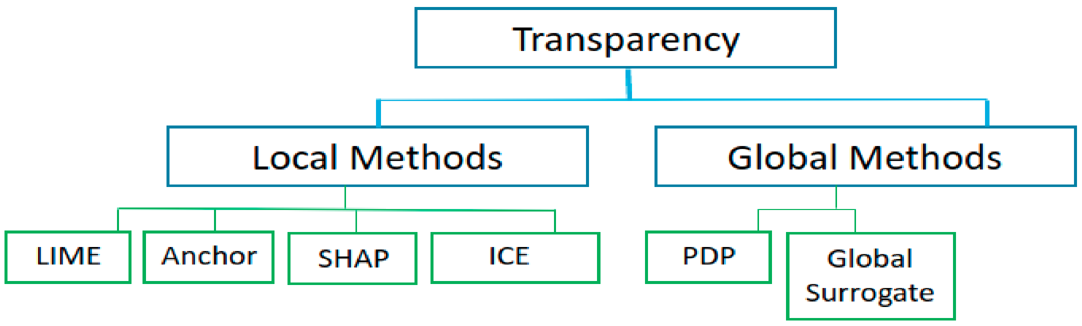


Figure 3.7 – Transparency Methods

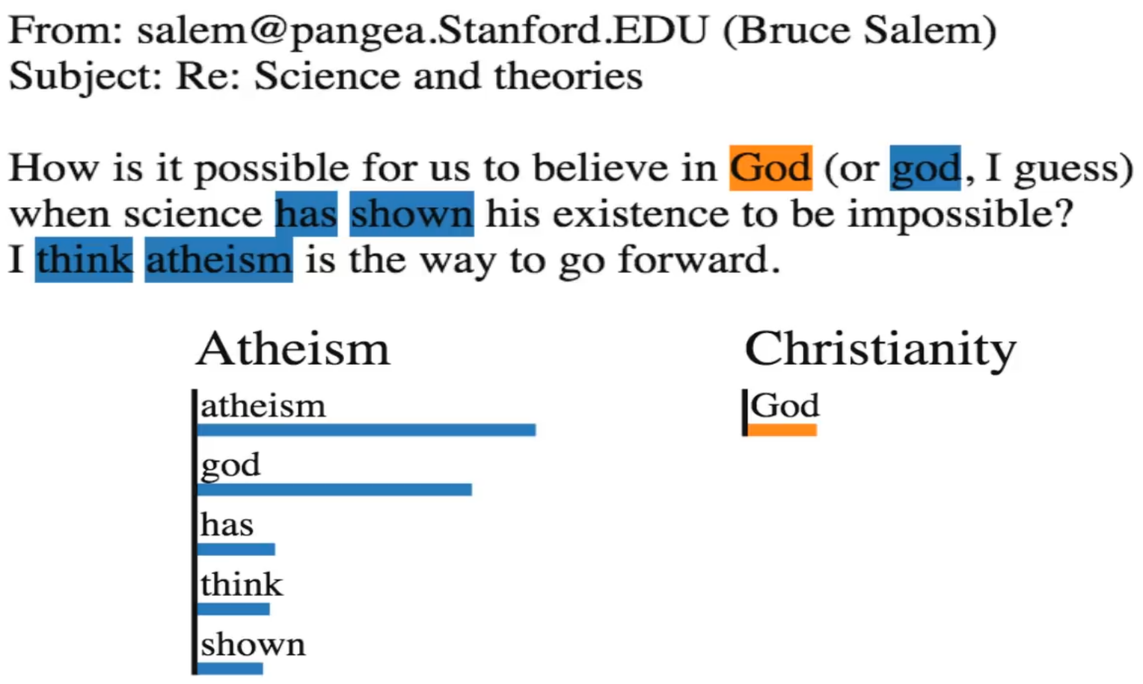
3.2.3.1 LIME [6]

LIME is an acronym for Local Interpretable Model-Agnostic Explanation. It is a novel explanation technique to unfold the predictions of any model in faithful and interpretable manner, by learning the model locally around the prediction. It works with any model since it treats the model as a black box. The explanation produced by LIME is simple enough for a human to understand. It approximates the black-box model locally in the neighborhood of the prediction being explained. It works on images, text, and tabular data. Therefore, LIME highlights easily understandable **factors**, based on which a particular decision is made by the model. These explanations help real users in the following manner:–

- to choose between competing models,
- to detect and improve untrustworthy models, and
- to get insights into the model.

Example:–

As shown in figure 3.8, LIME can easily explain why a particular model predicts this email is about Atheism. It picks up words such as atheism, or god (with lowercase G) and it also thinks that God (with uppercase G) is an evidence for Christianity.


From: salem@pangea.Stanford.EDU (Bruce Salem)
Subject: Re: Science and theories

How is it possible for us to believe in God (or god, I guess) when science has shown his existence to be impossible? I think atheism is the way to go forward.

Figure 3.8 – LIME Example [6]

Installation:–

To install LIME, execute following command in the command prompt:–

```
    pip install lime
```

Listing 3.7 – LIME Installation

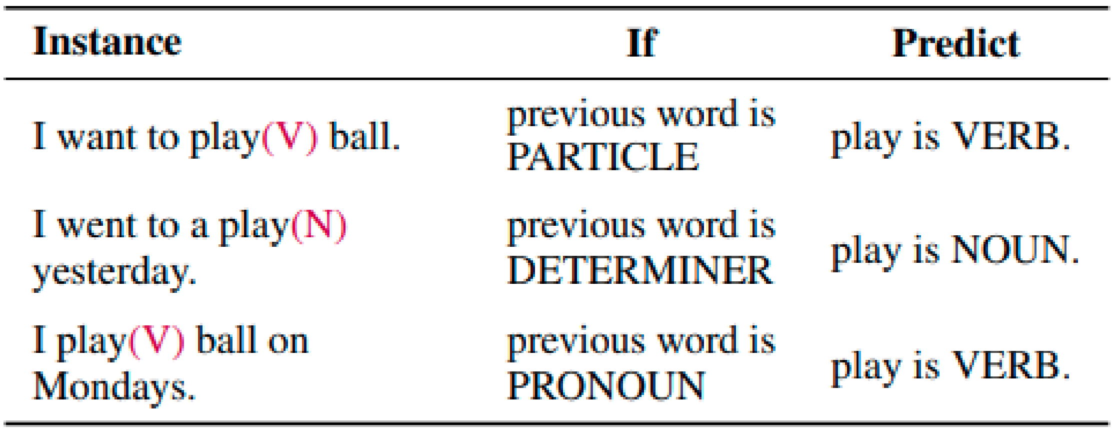
3.2.3.2 Anchor Explainator [7]

Anchor is also introduced by the authors of LIME. This method can explain any black-box classifier, with two or more classes. An anchor explanation is a novel model-agnostic method to explain the behavior of complex models with high-precision rules (anchors) that represent **sufficient** conditions for the prediction. Anchors help users to predict a model behavior on unseen instances with less effort, as compared to existing linear explanations.

Unlike LIME, this method doesn't use linearly weighted combinations to understand the prediction. Instead, it computes the anchors (if-then rules) to create an explanation. Anchor highlights a part of the input that is sufficient for the model to make the prediction, making them intuitive and easy to understand for novice users. It anchors the prediction of an instance in such a way that changes to the rest of the feature values do not affect the output. The authors have provided code templates for text and tabular data. However, (ideally) it supports image data as well.

Example:–

The figure 3.9 depicts an example of anchor explanation for the Part-of-Speech tag for the word "play". It has defined intuitive rules that are created/used by the model for predictions.



Instance	If	Predict
I want to play(V) ball.	previous word is PARTICLE	play is VERB.
I went to a play(N) yesterday.	previous word is DETERMINER	play is NOUN.
I play(V) ball on Mondays.	previous word is PRONOUN	play is VERB.

Figure 3.9 – Anchor Example [7]

Installation:–

To install Anchor, execute the following command(s) in the command prompt:–

```
  # Install the Library
  pip install anchor-exp

  # To use AnchorTextExplainer (Optional)
  python -m spacy download en_core_web_lg

  # To install transformers for using BERT (Recommended)
  pip install torch transformers spacy && python -m spacy
 download en_core_web_sm
```

Listing 3.8 – Anchor Installation

3.2.3.3 SHAP [8]

SHAP is an acronym for SHapley Additive exPlanations. It is an approach to explain the result of any machine learning model by connecting local explanations with optimal credit allocation. It uses the typical **shapley values** from game theory and their related extensions. SHAP values interpret the impact of having a certain feature value as compared to the prediction with baseline value of that feature. Following are several implementation use cases of SHAP:–

- **TreeExplainer,** a robust algorithm to compute SHAP values for trees and ensembles of trees.
- **DeepExplainer,** an approximate algorithm to compute SHAP values for deep learning models.
- **GradientExplainer,** an implementation of expected gradients to estimate SHAP values for deep learning models.
- **LinearExplainer,** to compute the exact SHAP values analytically for a linear model with independent features.
- **KernelExplainer,** a realization of Kernel SHAP to estimate SHAP values for any model.

Example:–

The figure 3.10 illustrates how the features contribute to push the model output from base value (0.1) to the model output (0.4). The features that push the prediction higher (+) are shown in dark-pink, and those that push the prediction lower (-) are visualized in blue. Therefore, SHAP values explain why did the model produce a certain value as an output.

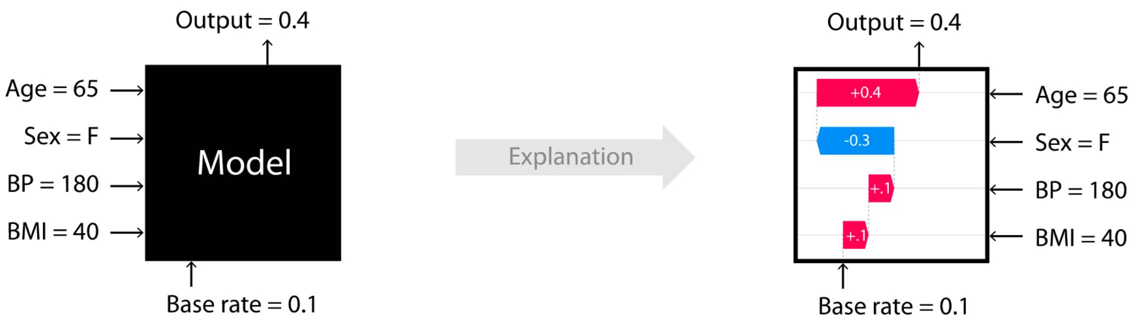


Figure 3.10 – SHAP Illustration [8]

Installation:–

To install SHAP, execute the following command in the command prompt:–

```
# Install the Library
pip install shap

# Installation with conda-forge
conda install -c conda-forge shap
```

Listing 3.9 – SHAP Installation

3.2.3.4 ICE [9]

ICE is an acronym for Individual Conditional Expectation. It displays one line per instance that depicts the change in the prediction of an instance when a feature is changed. An ICE plot visualizes the dependence of the feature's prediction separately for each instance.

Formal Definition:–

$$\begin{gathered} For\ each\ instance\ in\ \{(x_S^{(i)}, x_C^{(i)})\}_{i=1}^{N}\ , \\ the\ curve\ \hat{f}_S^{(i)}\ is\ plotted\ against\ x_S^{(i)}, \\ while\ x_C^{(i)}\ remains\ constant. \end{gathered} \tag{3.1}$$

where,

$\hat{f}$ = fitted model,

x_{S} = analyzed features, and

x_{C} = other features.

Example:–

Figure 3.11 displays four ICE plots for the California housing dataset. In the plots, a linear relationship is observed between the median income and the house price in the PD line. However, the house price remains fixed in some ranges of the median income, advocating the fact that the ICE lines show a few exceptions.

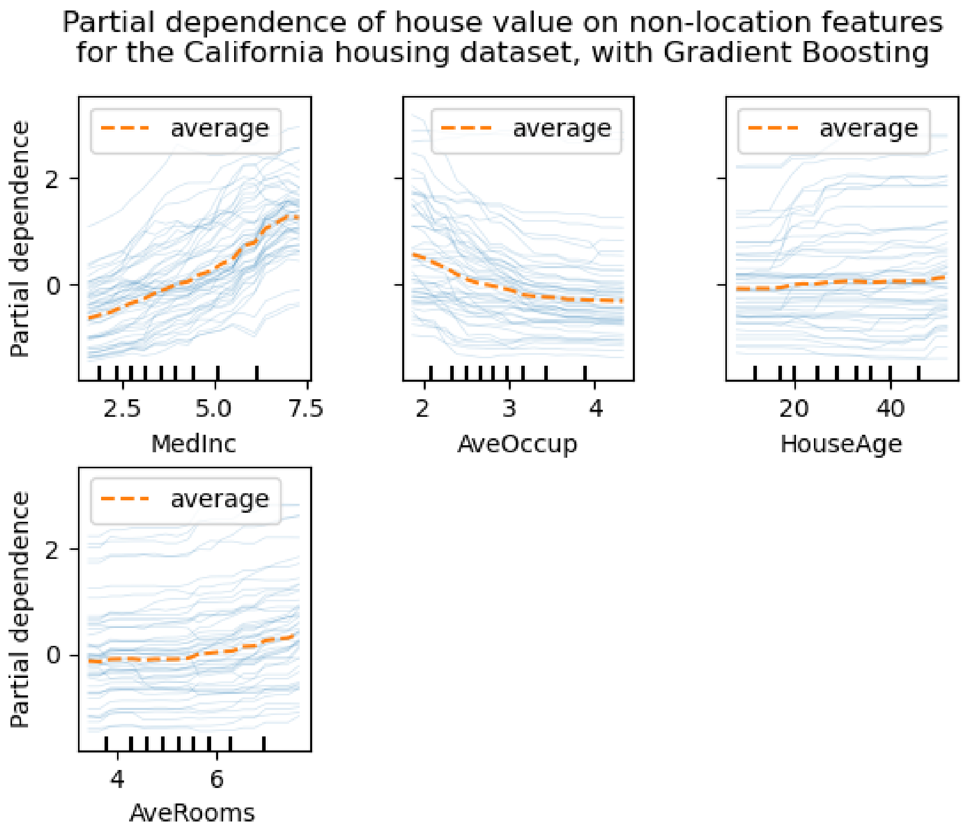


Figure 3.11 – ICE Plots [9]

3.2.3.5 PDP [9]

PDP is an acronym for Partial Dependence Plot. It shows the interaction between the target and a set of input features of interest x_S, marginalizing over the values of remaining input features x_C. Intuitively, partial dependence can be interpreted as the expected response as a function of the analyzed input features. PDP depicts the average effect of the input feature. If a studied feature is not correlated with other features, then the PDP perfectly represents the average feature influence on the prediction. When the features are correlated, PDP shows how the average prediction in the dataset changes when the j^{th} feature is changed.

Example:–

The figure 3.12 illustrates two one-way and one two-way PDPs for the California housing dataset. One-way PDPs convey the interaction between the target and input features of interest. The left plot visualizes the effect of an average occupancy on the median house price through a linear relationship between them when the average occupancy is ancillary to three persons. Similarly, in the middle plot, there is an effect of the house age on the median house price. Whereas, two-way PDPs show the interactions among two features of interest. The third plot proclaims the dependence of median house price on joint values of house age and average occupants per household. For an average occupancy > 2, the house price is almost independent of the house age, and on the contrary, there is a strong dependence on the house age.

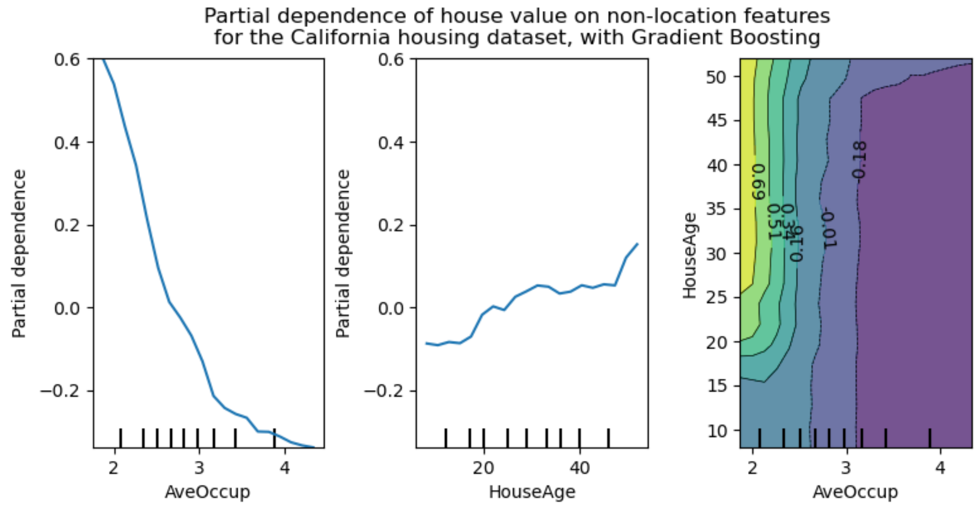


Figure 3.12 – PDP Plots [9]

The sklearn.inspection module provides a function, **from_estimator()** to create one-way and two-way partial dependence plots, as given below:–

```
    # one-way PDPs
    features = [0, 1, (0, 1)]        # dataset - make_hastie_10_2
    PartialDependenceDisplay.from_estimator(model, X, features)
    # two-way PDPs
    features = [3, 2, (3, 2)]        # dataset - load_iris
    PartialDependenceDisplay.from_estimator(model, X, features,
  target=0)
```

Listing 3.10 – PDP Usage

3.2.3.6 Global Surrogate [33]

A global surrogate model is a trained interpretable model to resemble the predictions of a black-box model. Conclusions can be drawn about the black-box model by interpreting the surrogate model. In this technique, machine learning interpretability is achieved using machine learning itself. The main aim of surrogate models is to come close to the predictions of the black-box model as accurately as possible. The prediction function f is approximated as accurate as possible to the surrogate model prediction function g. It doesn't need any information about the inner workings of the underlying model. Rather, it only needs access to the data and the prediction function. To figure out how good a surrogate model is in approximating the black-box model, R-squared measure can be used. The idea of surrogate models is also known as response surface model, meta-model, approximation model, emulator, etc.

Following are some of the advantages of this model:–

- Flexible
- Easy to implement
- Very easy to explain
- Straightforward

In simple terms, a linear regression version of a surrogate model to estimate the black-box model would be:–

$$g(x) \;=\; \beta_0 \,+ \beta_1 x_1 \,+ \beta_2 x_2 \,+\, ... \,+ \beta_p x_p \tag{3.2}$$

And, a surrogate model in the form of a decision tree would be:–

$$g(x) \;=\; \sum_{m=1}^{M} c_m I\{x \,\epsilon\, R_m\} \tag{3.3}$$

where,

$$\begin{aligned}
g(x) \;&=\; surrogate\, model\, prediction\, function,\\
x \;&=\; input\, variable(s),\\
M \;&=\; number\, of\, nodes,\\
I \;&=\; identity\, function,\\
R \;&=\; node,\\
c \;&=\; training\, instance\, in\, leaf\, node,\\
\beta_0 \;&=\; intercept,\\
\beta_1 \,..\, \beta_p \;&=\; correlation\, coefficient,\\
I\{x \,\epsilon\, R_m\} \;&=\; \begin{cases} 1 & \text{if } x \,\epsilon\, R_m \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

The table 3.1 shown below summarizes the properties of above-discussed transparency techniques:–

	Type	**Data**	**Category**
LIME	Model Agnostic	Images, Text, and Tabular Data	Local Methods
Anchor			
SHAP			
ICE		Tabular Data	
PDP		Images and Tabular Data	Global Methods
Global Surrogate		Tabular Data	

Table 3.1 – Properties of Transparency Methods

3.2.4 Monitoring

To use traditional tools like Prometheus and Grafana for alerting, monitoring, and visualization of models and applications, a kubernetes cluster or a docker network is required. Another research outcome of this thesis is a python library that overcomes this drawback by exposing application metrics to a specific endpoint. This endpoint is utilized by Prometheus to scrape generated metrics for monitoring. Moreover, Prometheus provides this data to Grafana for visualization. Therefore, without a network or an infrastructure, an application can directly be monitored. This idea is depicted in figure 3.13 below:–

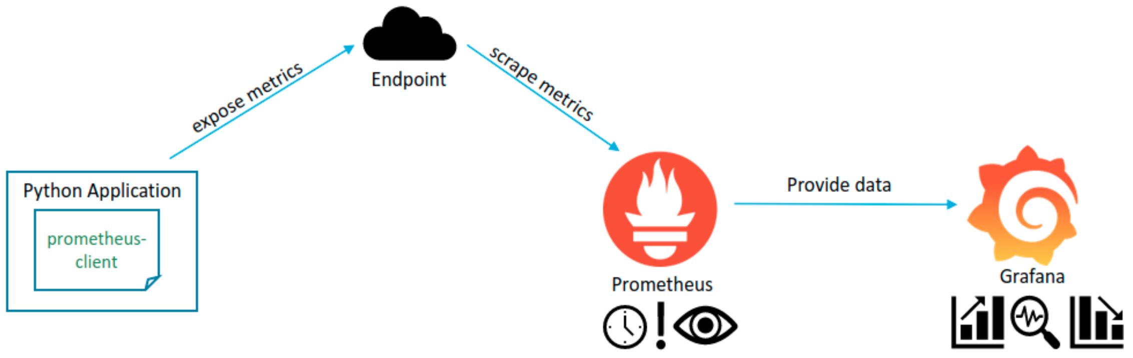


Figure 3.13 – Direct Monitoring Scheme [10] [11]

3.2.4.1 prometheus-client 0.11.0 [34]

Under **Prometheus Python Client** project, prometheus-client is a PyPI (Python Package Index) library for exposing metrics from a python application. This python library helps to expose the application logs and metrics to an endpoint, from where Prometheus can scrape the contents easily. It acts like a python client for the Prometheus monitoring system. It offers the following four types of metrics:–

- **Counter:–**

  A counter is a cumulative metric to represent a single monotonically increasing token whose value can only increase or reset to zero when restarted. It can be used to exhibit the number of served requests, completed tasks, etc. Since the value of a counter can only increase, it should not be used with a value that can decrease. For example, a counter for pending tasks can not be used, as pending tasks will be reduced upon completion. To implement a counter, execute a similar code:–

```python
from prometheus_client import Counter

c = Counter('counter_tag', 'counter_description')
c.inc()     # increment by 1
c.inc(1.5)  # increment by specified value
```

Listing 3.11 – Counter Usage

- **Gauge:–**

  A gauge is a metric to render a single numerical value that can increase and decrease arbitrarily. Gauges can be used with counts that can go up and down, and measured values such as temperature, memory usage, etc. To implement a gauge, execute similar code:–

```
    from prometheus_client import Gauge

    g = Gauge('gauge_tag', 'gauge_description')
    g.inc()         # increment by 1
    g.dec(10)       # decrement by given value
    g.set(7.86)     # set a value
```

Listing 3.12 – Gauge Usage

- **Summary:–**

  A summary samples observations, provides a total count of observations, and a sum of all observed values. It calculates configurable quantiles over a sliding time frame. A summary with a base metric name, <base_name> exposes the following time series values during a scrape:–

  – streaming quantiles of observed events

  – sum of observed values, exposed as <base_name>_sum

  – count of observed events, exposed as <base_name>_count

  To implement a summary metric, execute similar code:–

```
    from prometheus_client import Summary

    s = Summary('summary_tag', 'summary_description')
    s.observe(2.2)     # observe at 2.2 units
```

Listing 3.13 – Summary Usage

- **Histogram:–**

  A histogram is a metric for sampling observations to count them in configurable buckets. A histogram with a base metric name, <base_name> exposes the following time series values during a scrape:–

  – cumulative counters for the observation buckets, exposed as <base_name>_bucket{le="<upper_bound (inclusive)>"}

  – sum of all observed values, exposed as <base_name>_sum

  – count of observed events, exposed as <base_name>_count

  To implement a histogram, execute similar code:–

```
    from prometheus_client import Histogram

    h = Histogram('histogram_tag', 'histogram_description')
    h.observe(2.4)     # observe at 2.4 units
```

Listing 3.14 – Histogram Usage

Expose:–

As the metrics are created, there are many ways to expose these metrics to an endpoint such as HTTP, Twisted, WSGI, ASGI, and Flask. An example to expose the metrics is given below:–

```
from flask import Flask
from werkzeug.middleware.dispatcher import DispatcherMiddleware
from prometheus_client import make_wsgi_app

app = Flask(__name__)   # create a flask app

# expose metrics at '/metrics' endpoint
app.wsgi_app = DispatcherMiddleware(app.wsgi_app, {
    '/metrics': make_wsgi_app()
})
```

Listing 3.15 – Exposing Metrics

Installation:–

To install this library, execute the following command in the terminal:–

```
# Install the Library
pip install prometheus-client
```

Listing 3.16 – Installation of prometheus-client

Chapter 4

Implementation

In this chapter, the implementation results of this thesis are discussed in detail. To portray the research outcomes, an automated flask application is created containing MLOps, transparency methods, monitoring, and docker deployment. This application trains several machine learning models, but uses only the best model for production based on performance metrics. These performance metrics include accuracy, precision, recall, f1 score, and confusion matrix. The following list represents the technologies used in this application:–

- **Programming Language:-** Python, Bash/Shell
- **Libraries:–**
  - Pandas
  - Joblib
  - Numpy
  - Scikit-Learn
  - Seaborn
  - Scipy
  - Matplotlib
  - Flask Prometheus Metrics
  - Flask Monitoring Dashboard
- **Tools & Technologies:–**
  - Flask
  - ML-FLow
  - AutoML (Auto-Sklearn)
  - Lime
  - Shap
  - Anchor
  - Global Surrogate
  - Prometheus
  - Grafana
  - Docker & Docker-Compose

4.1 Dataset

To demonstrate the concepts discussed in theory, a complex dataset is purposefully chosen for the application. This application is built on a CNC Mill Tool Wear [35] dataset from Kaggle. For this dataset, machining data was collected from a CNC machine. It consists of data collected through 18 experiments with, **experiment number, material, feed-rate, and clamp pressure** as input variables and, **tool condition and passed_visual_inspection** are the output variables. In this application, this dataset is used **to classify worn and unworn cutting tools.**

Moreover, from these 18 experiments, time series data was collected with a sampling rate of 100 ms, with measurements from 4 motors in the CNC machine. Following are the features available in the machining dataset (experiments) for all of the 4 motors, X axis, Y axis, Z axis, and the spindle (S):–

- **ActualPosition** – actual x/y/z/s position of part (mm)
- **ActualVelocity** – actual x/y/z/s velocity of part (mm/s)
- **CommandPosition** – reference x/y/z/s position of part (mm)
- **CommandVelocity** – reference x/y/z/s velocity of part (mm/s)
- **ActualAcceleration** – actual x/y/z/s acceleration of part (mm/s/s)
- **CommandAcceleration** – reference x/y/z/s acceleration of part (mm/s/s)
- **OutputPower** – x/y/z/s power (kW)
- **DCBusVoltage** – x/y/z/s voltage (V)
- **OutputVoltage** – x/y/z/s voltage (V)
- **OutputCurrent** – x/y/z/s current (A)
- **CurrentFeedback** – x/y/z/s current (A)
- **S1_SystemInertia** – torque inertia (kg*m^2)
- **M1_sequence_number** – line of G-code being executed
- **M1_CURRENT_FEEDRATE** – instantaneous feed rate of spindle
- **M1_CURRENT_PROGRAM_NUMBER** – number the program is listed under on the CNC machine

Therefore, for intuitive understanding, the structure of this dataset is as follows:–

```
train.csv (18 rows X 7 columns)
|-- Material Number
|-- Material (Wax)
|-- Feed Rate
|-- Clamp Pressure
|-- Tool Condition
|-- Machining Finalized
|-- Passed Visual Inspection

experiments
|-- experiment_01.csv (100 rows X 48 columns)
|-- experiment_02.csv (100 rows X 48 columns)
      .
      .
|-- experiment_17.csv (100 rows X 48 columns)
|-- experiment_18.csv (100 rows X 48 columns)
```

****Note:– All the features are variables; however, all variables are not features.***

4.2 CNC Mill Tool Wear App

CNC Mill Tool Wear App is a python-flask application developed as a part of this thesis to render standard practices of model development, model deployment, and model operations. It is an end-to-end machine learning project involving the latest libraries and tools for different purposes. This project includes the following:–

- The latest technology AutoML, for model development
- LIME, SHAP, Anchor, and Global Surrogate as transparency methods
- ML-Flow for tracking model experiments
- Prometheus and Grafana for monitoring
- Other endpoints for data distribution, model statistics, application metrics, and retraining

4.2.1 ML Pipeline

The application also includes an optimized ML pipeline with feature engineering, exploratory data analysis, feature selection, hyperparameter tuning, model building, and visualization of the model metrics.

4.2.1.1 Data Processing

The final version of data processing for this dataset contains the following steps:–

- First, all the data and experiments are read and concatenated in a variable.
- In Feature Engineering,
  - Only active machining data is selected using the following condition:–

```
dataframe.Machining_Process == 'Layer 1 Up'
dataframe.Machining_Process == 'Layer 1 Down'
dataframe.Machining_Process == 'Layer 2 Up'
dataframe.Machining_Process == 'Layer 2 Down'
dataframe.Machining_Process == 'Layer 3 Up'
dataframe.Machining_Process == 'Layer 3 Down'
```

  - Now, new features are created using already available variables, by taking the difference between command values and actual values for X, Y, Z, and S motors. Then, the variables using which new features were created are dropped. ***For example,***

```
# Create a new feature
dataframe['X1_DiffPosition'] = dataframe['
    X1_CommandPosition'] - dataframe['X1_ActualPosition']

# Drop the old variable
dataframe = dataframe.drop(['X1_CommandPosition', '
    X1_ActualPosition'], axis=1)
```

- Next, a dummy variable for the categorical variable, **"tool_condition"** is created. Then, only newly created dummy variables are kept, while the redundant features are removed. In this step, labels are created out of the the output variable.
- Then, the features having constant behavior are dropped. For example, **'S1_SystemInertia', 'Z1_OutputVoltage', 'Z1_OutputCurrent', 'Z1_DCBusVoltage', 'Z1_CurrentFeedback'** as shown in figure 4.1:–

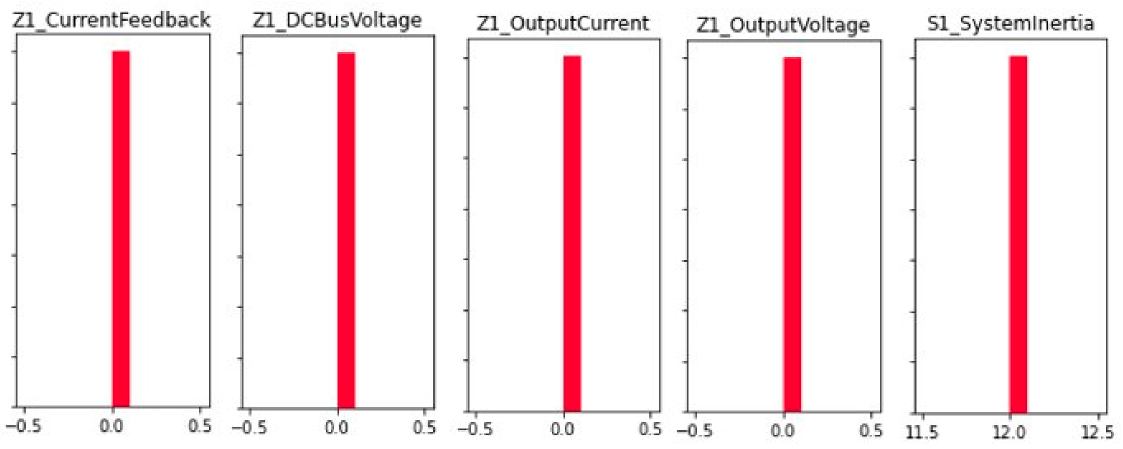


Figure 4.1 – Features with Constant Behavior

- Finally, the dataset is split into a training set and test set, with **80%** of the data for training and the remaining **20%** for the validation.

4.2.1.2 Model Building

A total of 3 models, Decision Tree, Random Forest, and Auto-Sklearn are created in the pipeline. All of these models are built with almost the same features, with slight changes in hyperparameters. The final version of model development for this dataset contains the following steps:–

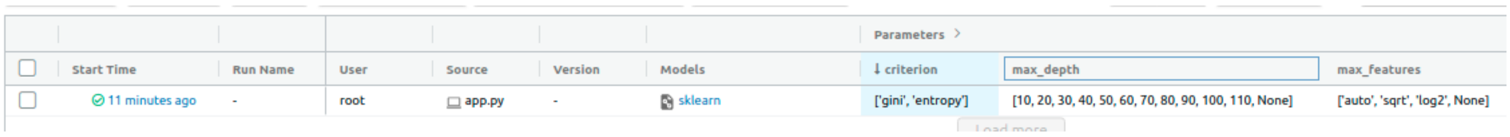
- Before building the model, feature selection is implemented based on a threshold. Limiting the variables based on their importance saves time and yields better results in most cases. In this application, feature selection is done automatically.

- **Hyperparameter Optimization:–**

  – For the decision tree model, hyperparameters are tuned using below given values:–

```
max_depth = [10, 20, .. 100, 110, None]
max_features = ['auto', 'sqrt', 'log2', None]
min_samples_split = [2, 4, 6, 8, 10]
min_samples_leaf = [1, 2, 4, 8, 12]
criterion = ['gini', 'entropy']
```

  The snapshot of hyperparameter options for a decision tree model is shown in figure 4.2:–



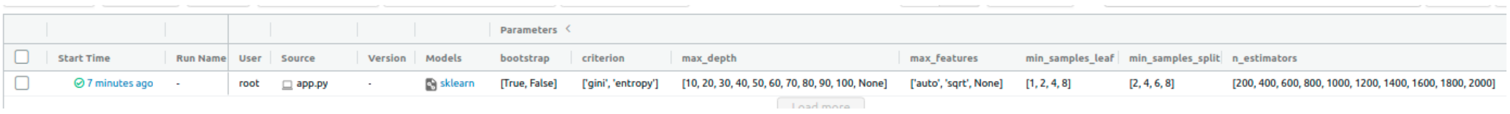
						Parameters >		
☐	Start Time	Run Name	User	Source	Version	Models	↓criterion	max_depth	max_features
☐	✓ 11 minutes ago	-	root	app.py	-	sklearn	['gini', 'entropy']	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None]	['auto', 'sqrt', 'log2', None]

Figure 4.2 – Decision Tree Hyperparameters

  – For the random forest model, hyperparameters are tuned using the following values:–

```
n_estimators = [200, 400, .. 1800, 2000]
max_depth = [10, 20, .. 90, 100, None]
max_features = ['auto', 'sqrt', None]
min_samples_split = [2, 4, 6, 8]
min_samples_leaf = [1, 2, 4, 8]
criterion = ['gini', 'entropy']
bootstrap = [True, False]
```

  The snapshot of hyperparameter options for a random forest model is shown in figure 4.3:–



							Parameters <						
☐	Start Time	Run Name	User	Source	Version	Models	bootstrap	criterion	max_depth	max_features	min_samples_leaf	min_samples_split	n_estimators
☐	✓ 7 minutes ago	-	root	app.py	-	sklearn	[True, False]	['gini', 'entropy']	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]	['auto', 'sqrt', None]	[1, 2, 4, 8]	[2, 4, 6, 8]	[200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]

Figure 4.3 – Random Forest Hyperparameters

  – And for the Auto-Sklearn model, there is no need of hyperparameter selection. It automatically selects a new configuration of hyperparameters every time depending on some constraints. These constraints can be dataset, use case, the maximum time to generate the model, time allotted for each iteration, etc.

- Finally, the models are created using scikit-learn and auto-sklearn libraries.

4.2.1.3 Visualization of Model Metrics

Visualization of the model performance and metrics is also a part of the ML pipeline in this application. Every time the model is run, its metrics are logged and the performance measures are updated.

- First, the performance metrics are evaluated for the classifier, such as accuracy, confusion matrix, precision score, recall score, f1-score, and correlation coefficient.
- Next, a precision-recall curve is rendered on updated results.
- Finally, the transparency methods are visualized for a better understanding of the underlying system.

Note:– The output variable "tool_ condition" consists of 2 classes, "worn" and "unworn", depicted as classes "1" and "0" respectively.

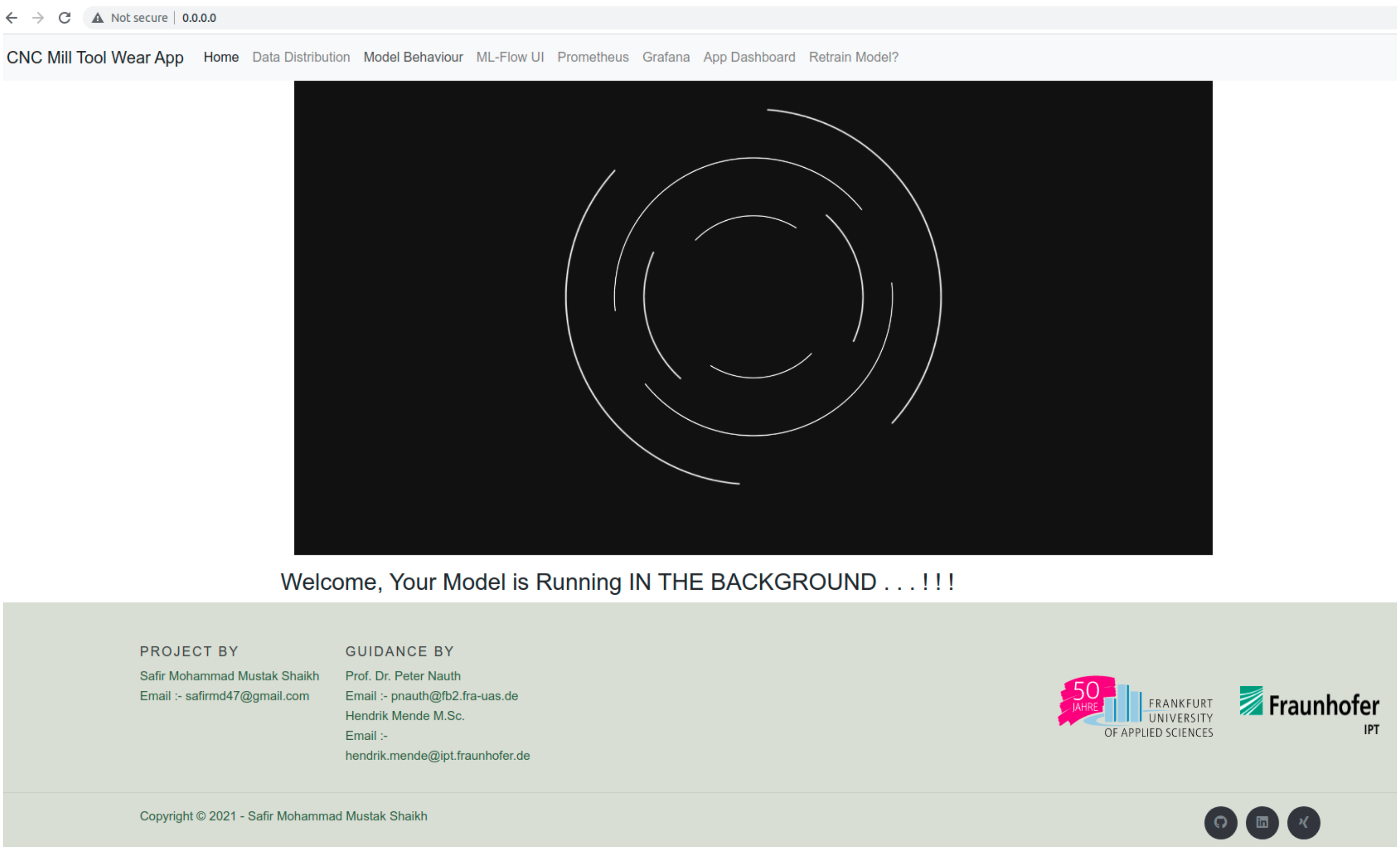
$$Prediction\,Class\; = \begin{cases} 1 & \boldsymbol{if\;\; prediction\, value\;\; > \;\; 0.5} \\ 0 & \boldsymbol{otherwise} \end{cases} \tag{4.1}$$

4.2.2 Application Endpoints and Snapshots

CNC Mill Tool Wear App is targeted at data scientists, stakeholders, and end-users. Therefore, it comprehends the following web pages:–

4.2.2.1 Home

The home page of this application is illustrated in figure 4.4. It depicts that the application has successfully loaded its processes and the model is ready for predictions.



Welcome, Your Model is Running IN THE BACKGROUND . . . ! ! !

PROJECT BY
Safir Mohammad Mustak Shaikh
Email :- safirmd47@gmail.com

GUIDANCE BY
Prof. Dr. Peter Nauth
Email :- pnauth@fb2.fra-uas.de
Hendrik Mende M.Sc.
Email :-
hendrik.mende@ipt.fraunhofer.de

Copyright © 2021 - Safir Mohammad Mustak Shaikh

Figure 4.4 – Home

4.2.2.2 Data Distribution

The data distribution API visualizes the feature distribution graphs of the processed features (final variables selected in the feature selection step). Moreover, it shows a clustered correlation matrix of the same features and data as shown in figure 4.5.

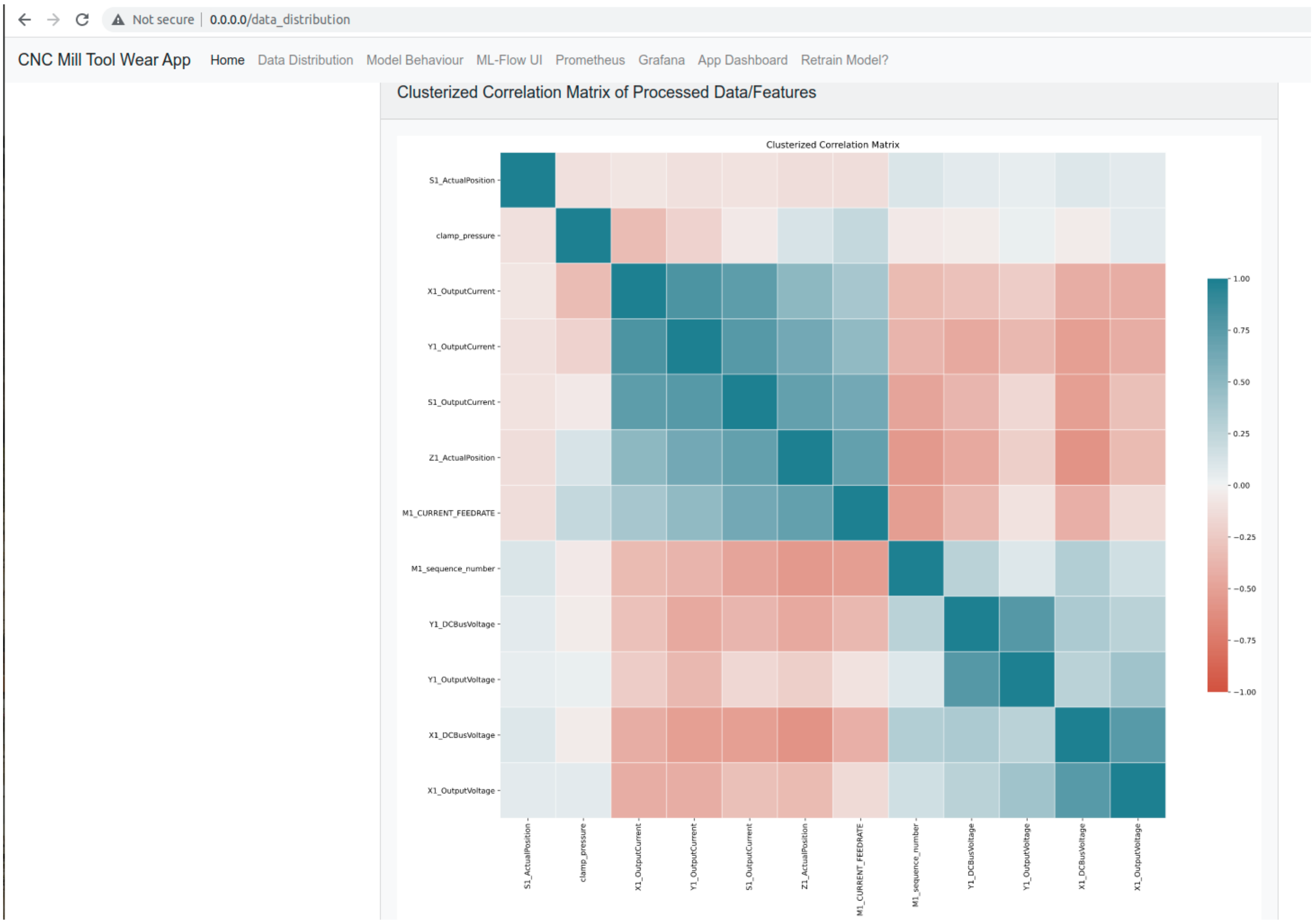
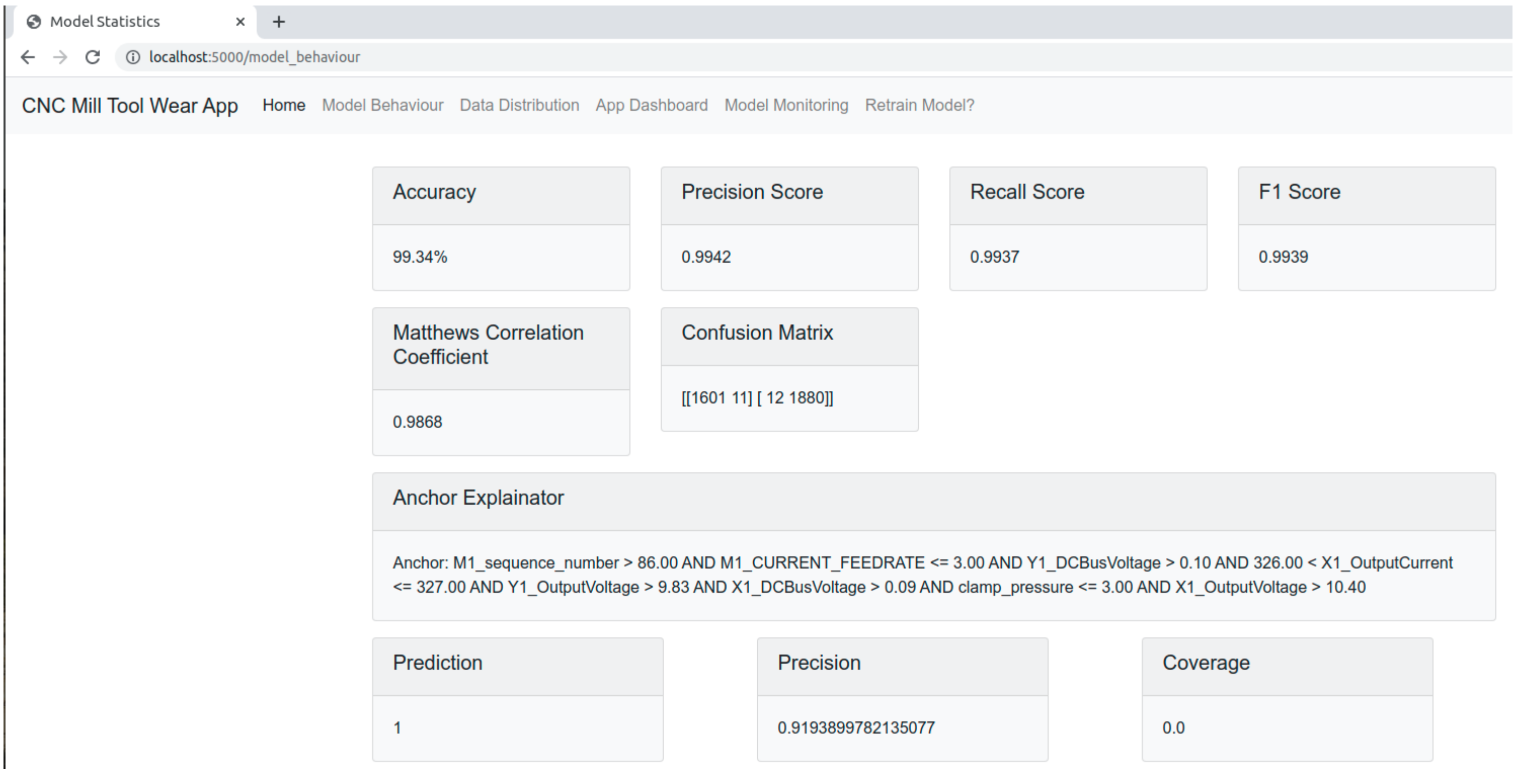


Figure 4.5 – Data Distribution

4.2.2.3 Model Behaviour

As its name indicates, this endpoint provides access to assess many properties and charts related to model behavior.

- The figure 4.6 lists the accuracy, precision score, recall score, f1-score, correlation coefficient, and confusion matrix of the model. Moreover, it denotes the anchor explanation for the model. As the anchor is a local transparency method, it represents the rules used for the prediction of first observation from "X_Test". **It has classified the observation into class "1", with the precision value of 0.9193 and 0.0 as the coverage value since there are no predictions after seeing this explanation.** It explains the conditions on different variables that caused this output, such as,
  the value of "M1_sequence_number" is greater than 86.00 in the observation, the value of "M1_CURRENT_FEEDRATE" is less than or equal to 3.00 in the observation, and so on.



Accuracy	Precision Score	Recall Score	F1 Score
99.34%	0.9942	0.9937	0.9939

Matthews Correlation Coefficient	Confusion Matrix
0.9868	[[1601 11] [12 1880]]

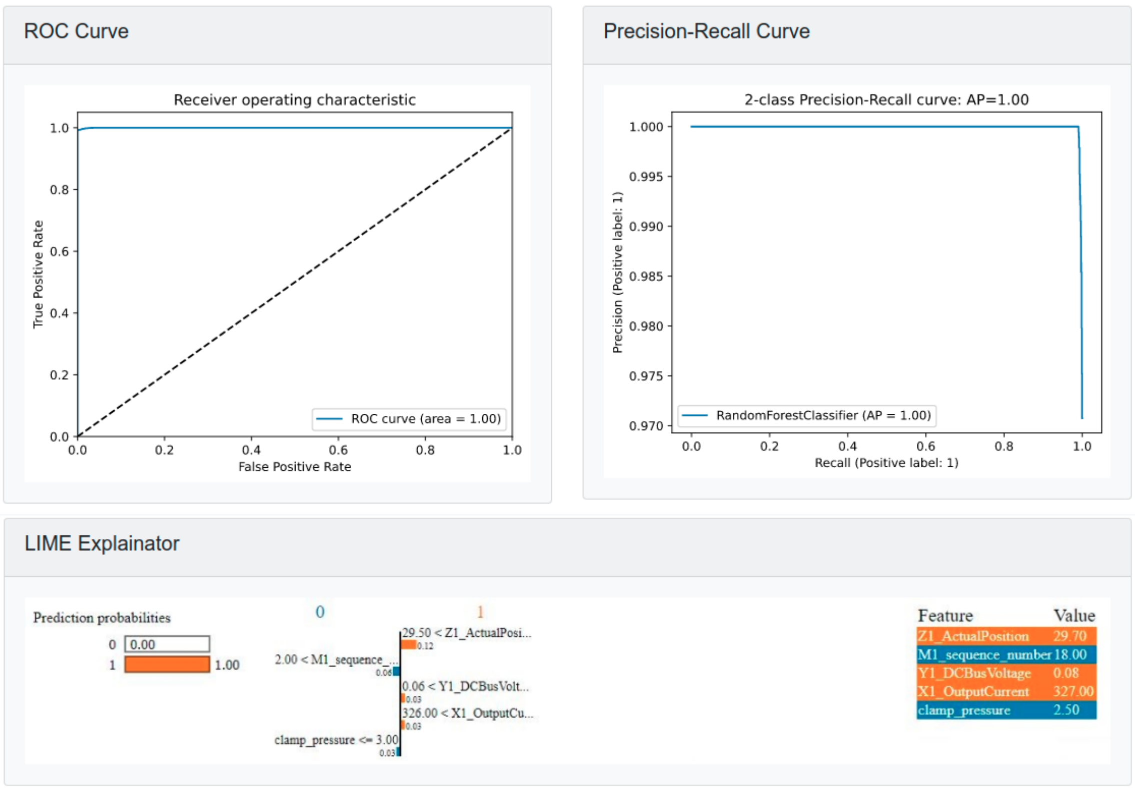
Anchor Explainator

Anchor: M1_sequence_number > 86.00 AND M1_CURRENT_FEEDRATE <= 3.00 AND Y1_DCBusVoltage > 0.10 AND 326.00 < X1_OutputCurrent <= 327.00 AND Y1_OutputVoltage > 9.83 AND X1_DCBusVoltage > 0.09 AND clamp_pressure <= 3.00 AND X1_OutputVoltage > 10.40

Prediction	Precision	Coverage
1	0.9193899782135077	0.0

Figure 4.6 – Model Behavior (1)

- The figure 4.7 delineates three graphs, ROC curve, Precision-Recall curve, and the Lime Explainator. The first two graphs advocate the above-calculated accuracy of the model, as almost 100% of the area lies under the curve. More importantly, the transparency method LIME has also predicted the class of the first observation as "1". **It classifies whether which of the features fall in class "0" and which of the features fall in class "1". From the explanation, it can be stated that the features M1_sequence_number and clamp_pressure add weightage to class "0", whereas Z1_ActualPosition, Y1_DCBusVoltage, and X1_OutputCurrent force the prediction to choose a class "1".**



Feature	Value
Z1_ActualPosition	29.70
M1_sequence_number	18.00
Y1_DCBusVoltage	0.08
X1_OutputCurrent	327.00
clamp_pressure	2.50

Figure 4.7 – Model Behavior (2)

- In figure 4.8, the SHAP explanation for the first observation from X_Test (upper graph) and X_Train (lower graph) are outlined. As shown in the figure, **the starting point of the chart is the baseline value (0.43), and the features are forces that increase or decrease this value. The dark-pink ones push the prediction close to class "1" (worn), and the light-blue ones close to class "0" (unworn). As depicted, the dark-pink ones are higher and the highest influence is done by "Z1_ActualPosition" (28.5). Therefore, the predicted class is "1" with the final SHAP value of 0.97.** In the previous graph, LIME has also considered this variable as the highest impacting feature for this prediction.

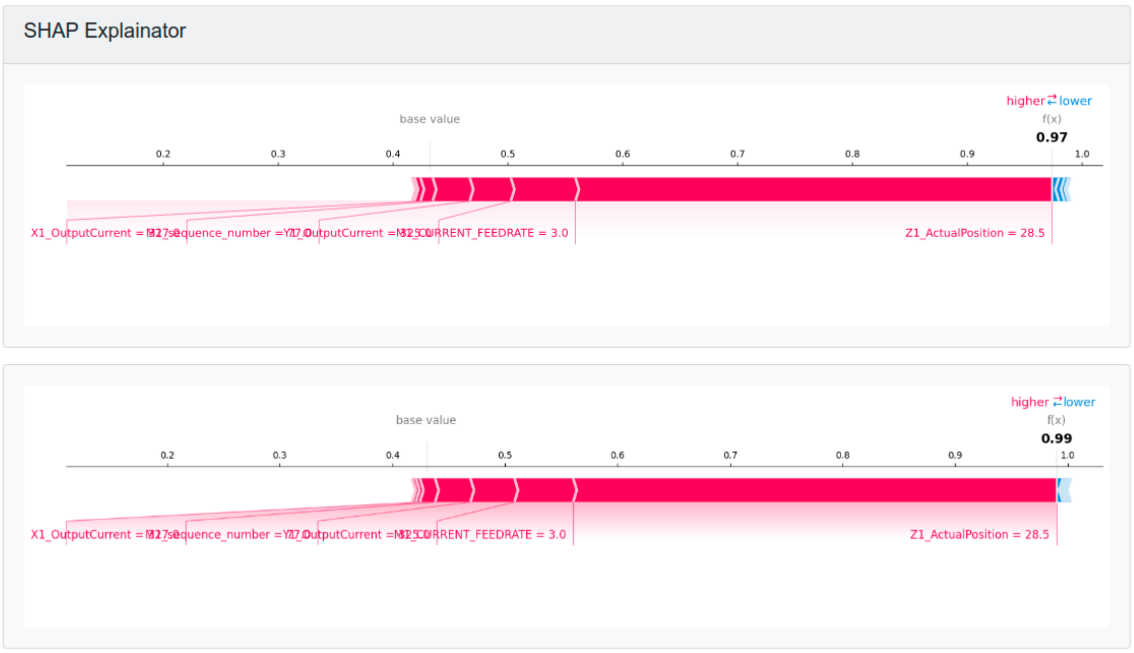


Figure 4.8 – Model Behavior (3)

For better understanding, consider another example, as shown in figure 4.9. The base value, in this case, is 0.5423, and the prediction value ended up as 0.34 (class "0"). As the light-blue ones are higher in this case, the prediction was forced to go to the lower side, with "S1_CommandPosition" and "M1_sequence_number" causing more dominance.

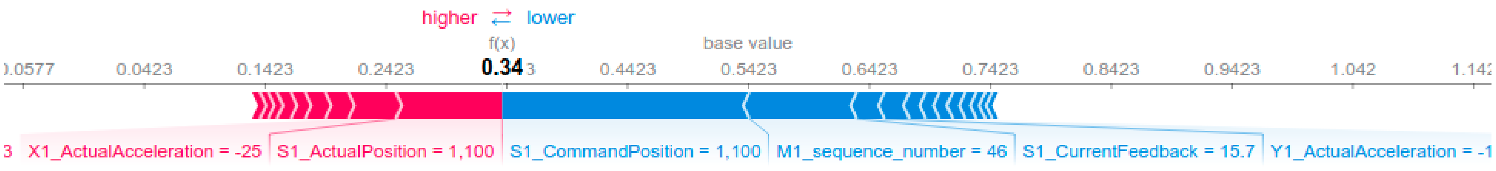


Figure 4.9 – Model Behavior (4)

- Finally, this endpoint renders a global transparency method, "Global Surrogate" as shown in figure 4.10. This method defines rules collectively for the entire dataset. **First, it collects the predictions of the black-box model. Then, the global surrogate model is trained and the model-specific methods are used for transparency to achieve the final results. Therefore, it generates a decision tree that defines different conditions of features for the output classes.**

  For example, the feature constraints for one of the use cases identified for class "1" are listed below:–

```
class "1 (Worn)"

    |-- Z1_ActualPosition <= 27.65
    |-- gini = 0.497
    |-- samples = 14016
    |-- value = [6430, 7586]
```

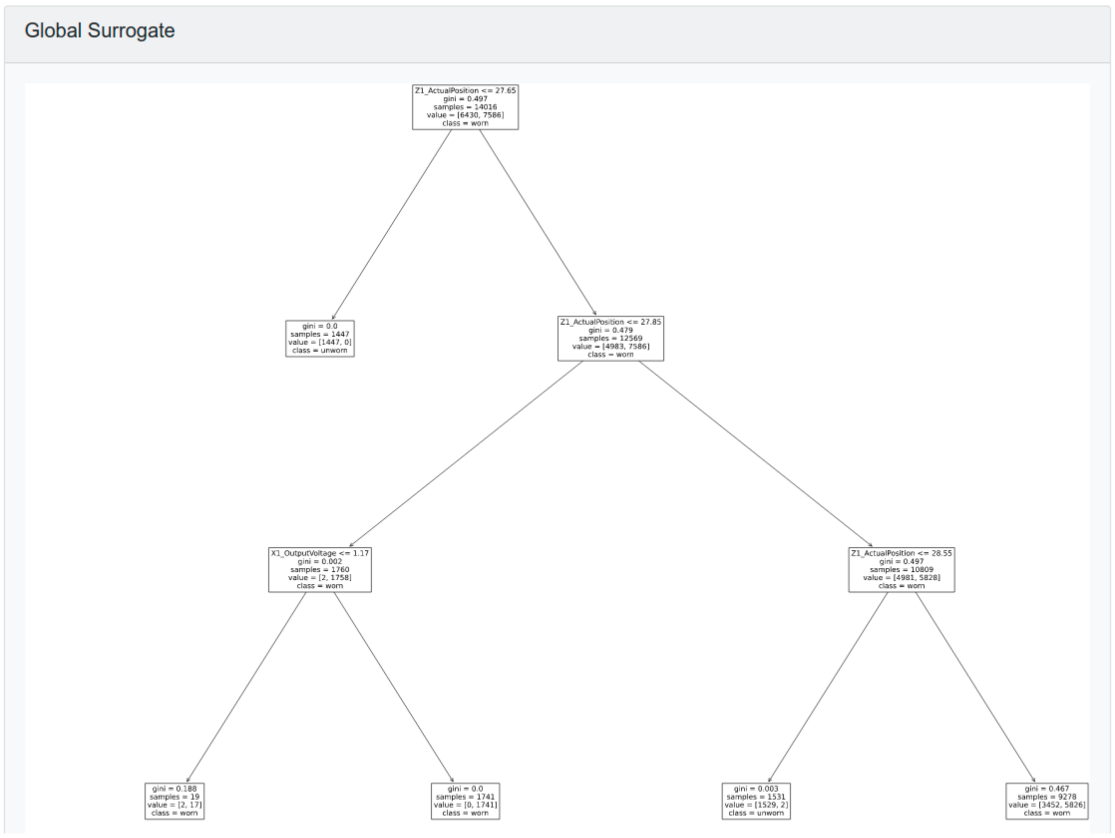

Global Surrogate
Z1_ActualPosition <= 27.65 gini = 0.497 samples = 14016 value = [6430, 7586] class = worn
gini = 0.0 samples = 1447 value = [1447, 0] class = unworn
Z1_ActualPosition <= 27.85 gini = 0.479 samples = 12569 value = [4983, 7586] class = worn
X1_OutputVoltage <= 1.17 gini = 0.002 samples = 1760 value = [2, 1758] class = worn
Z1_ActualPosition <= 28.55 gini = 0.497 samples = 10809 value = [4981, 5828] class = worn
gini = 0.188 samples = 19 value = [2, 17] class = worn
gini = 0.0 samples = 1741 value = [0, 1741] class = worn
gini = 0.003 samples = 1531 value = [1529, 2] class = unworn
gini = 0.467 samples = 9278 value = [3452, 5826] class = worn

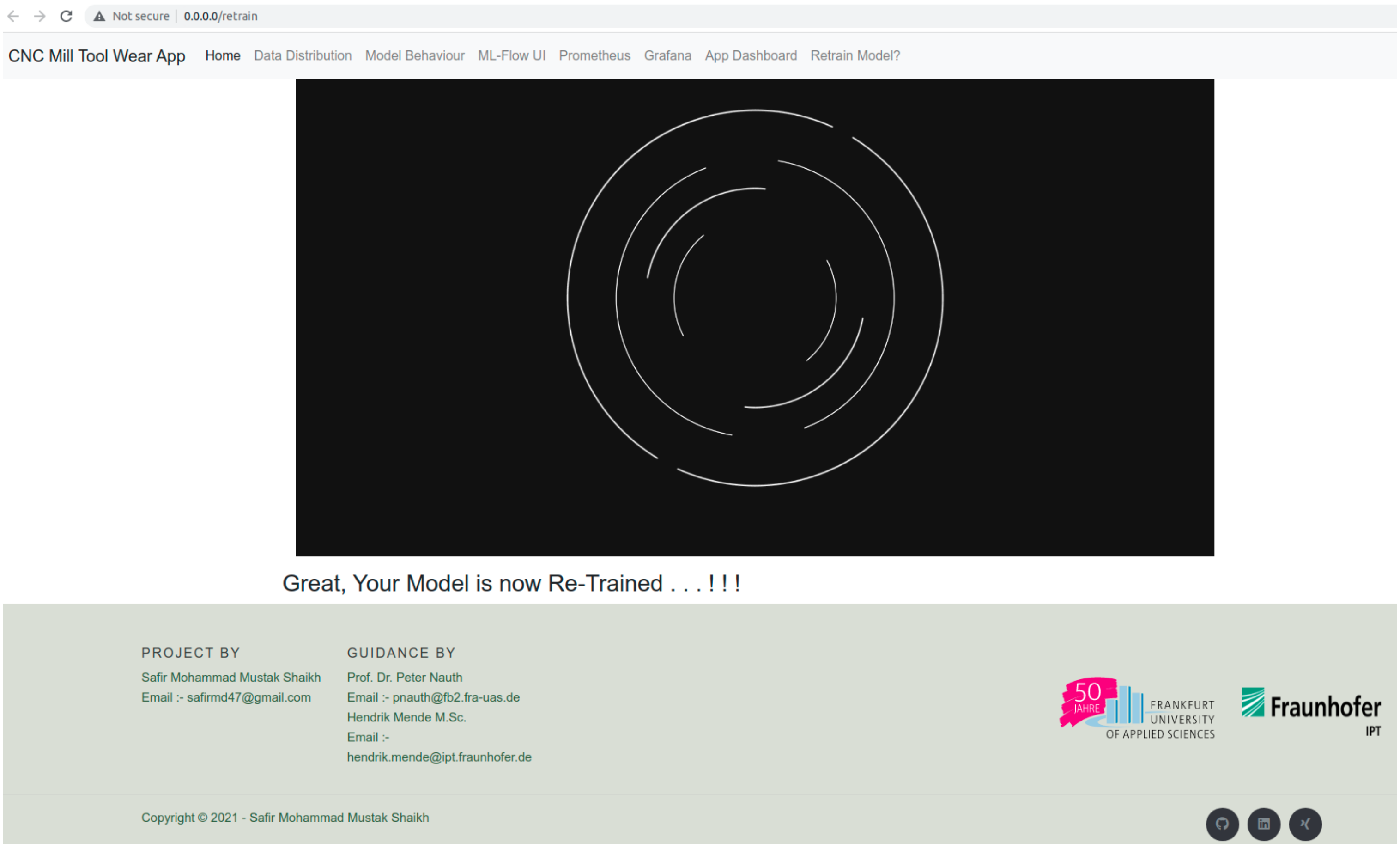
Figure 4.10 – Model Behavior (5)

4.2.2.4 Retrain

Based on the ML pipeline triggers, the retrain API automatically realizes the following things:–

- Retrain all the 3 models on new data
- Log the performance metrics
- Deploy the model with top performance
- Update the visualizations
- Following metrics are updated (used for Prometheus and Grafana):–
  - model_retrain_count_total
  - model_retrain_duration_buckets

After successful completion of above steps, the application confirms the status using a message as shown in figure 4.11:–



Great, Your Model is now Re-Trained . . . ! ! !

PROJECT BY
Safir Mohammad Mustak Shaikh
Email :- safirmd47@gmail.com

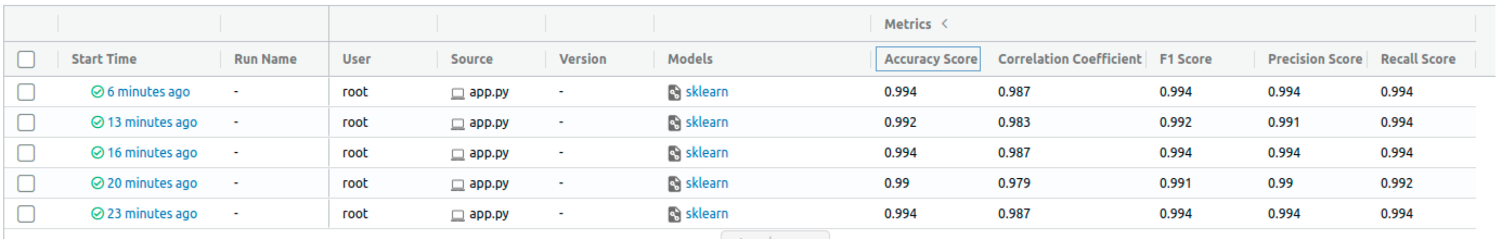
GUIDANCE BY
Prof. Dr. Peter Nauth
Email :- pnauth@fb2.fra-uas.de
Hendrik Mende M.Sc.
Email :-
hendrik.mende@ipt.fraunhofer.de

Copyright © 2021 - Safir Mohammad Mustak Shaikh

Figure 4.11 – Retrain

4.2.2.5 ML-Flow

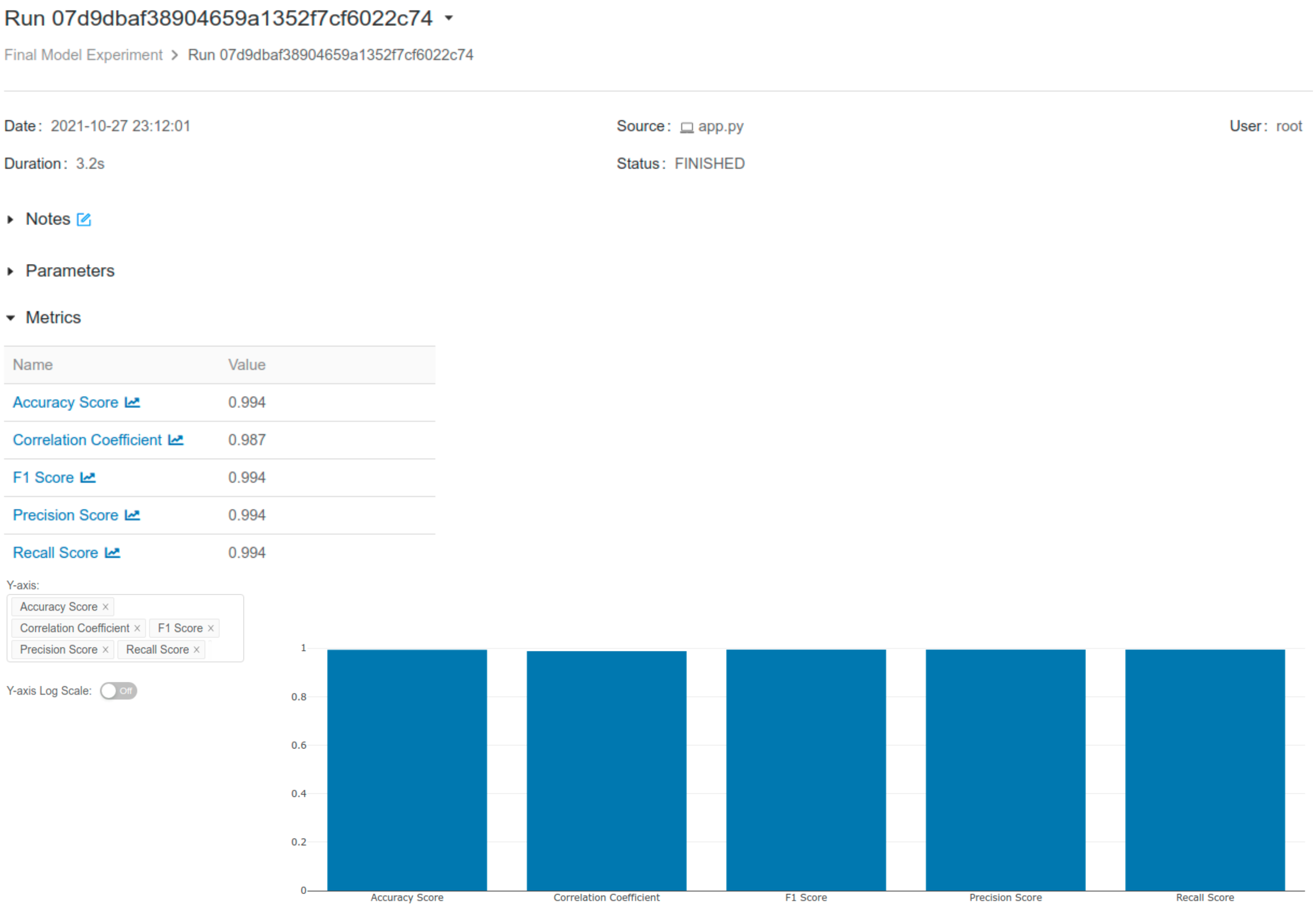
- ML-Flow is also implemented as part of this application. Figure 4.12 depicts the model runs tracked by ML-Flow. It consists of the details such as time, user, source, metrics, and parameters.



							Metrics				
☐	Start Time	Run Name	User	Source	Version	Models	Accuracy Score	Correlation Coefficient	F1 Score	Precision Score	Recall Score
☐	6 minutes ago	-	root	app.py	-	sklearn	0.994	0.987	0.994	0.994	0.994
☐	13 minutes ago	-	root	app.py	-	sklearn	0.992	0.983	0.992	0.991	0.994
☐	16 minutes ago	-	root	app.py	-	sklearn	0.994	0.987	0.994	0.994	0.994
☐	20 minutes ago	-	root	app.py	-	sklearn	0.99	0.979	0.991	0.99	0.992
☐	23 minutes ago	-	root	app.py	-	sklearn	0.994	0.987	0.994	0.994	0.994

Figure 4.12 – ML-Flow (1)

- Moreover, every individual run consists of the model-related details, like date of experiment, run id, name of the experiment, status, artifacts, and graphs for the model metrics as shown in figure 4.13:–



Run 07d9dbaf38904659a1352f7cf6022c74

Final Model Experiment > Run 07d9dbaf38904659a1352f7cf6022c74

Date: 2021-10-27 23:12:01
Source: app.py
User: root
Duration: 3.2s
Status: FINISHED

Notes

Parameters

Metrics

Name	Value
Accuracy Score	0.994
Correlation Coefficient	0.987
F1 Score	0.994
Precision Score	0.994
Recall Score	0.994

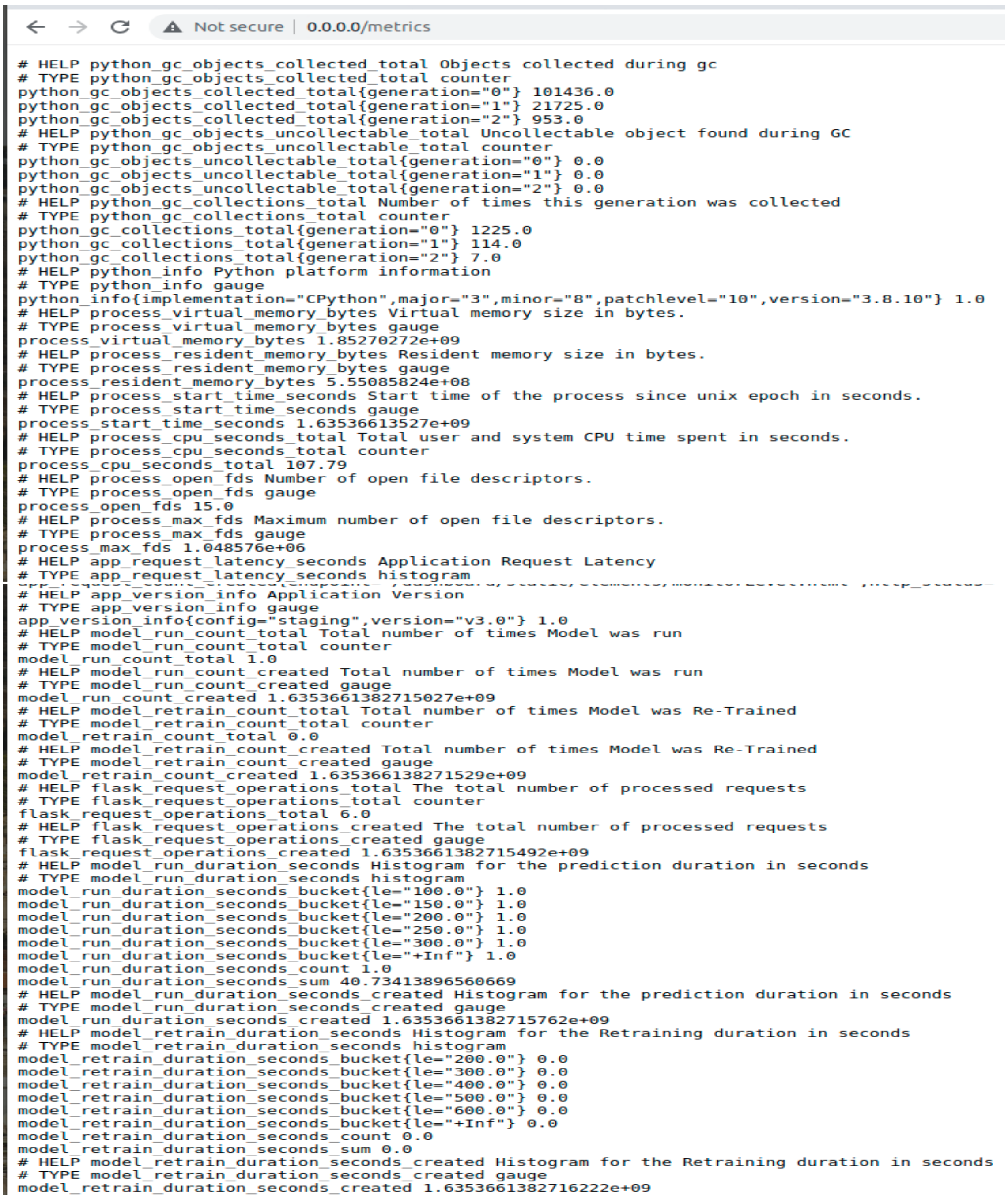
Y-axis: Accuracy Score × Correlation Coefficient × F1 Score × Precision Score × Recall Score ×

Y-axis Log Scale: Off

Figure 4.13 – ML-Flow (2)

4.2.2.6 Metrics Endpoint

The CNC Mill Tool Wear App exposes certain model and application-related metrics to an endpoint, **"0.0.0.0/metrics"**. These metrics are necessary for Prometheus and there are several kinds of metrics related to model runs, retraining, application run count, latency, CPU, memory, etc. Following are the snapshots of exposed metrics in this application:–

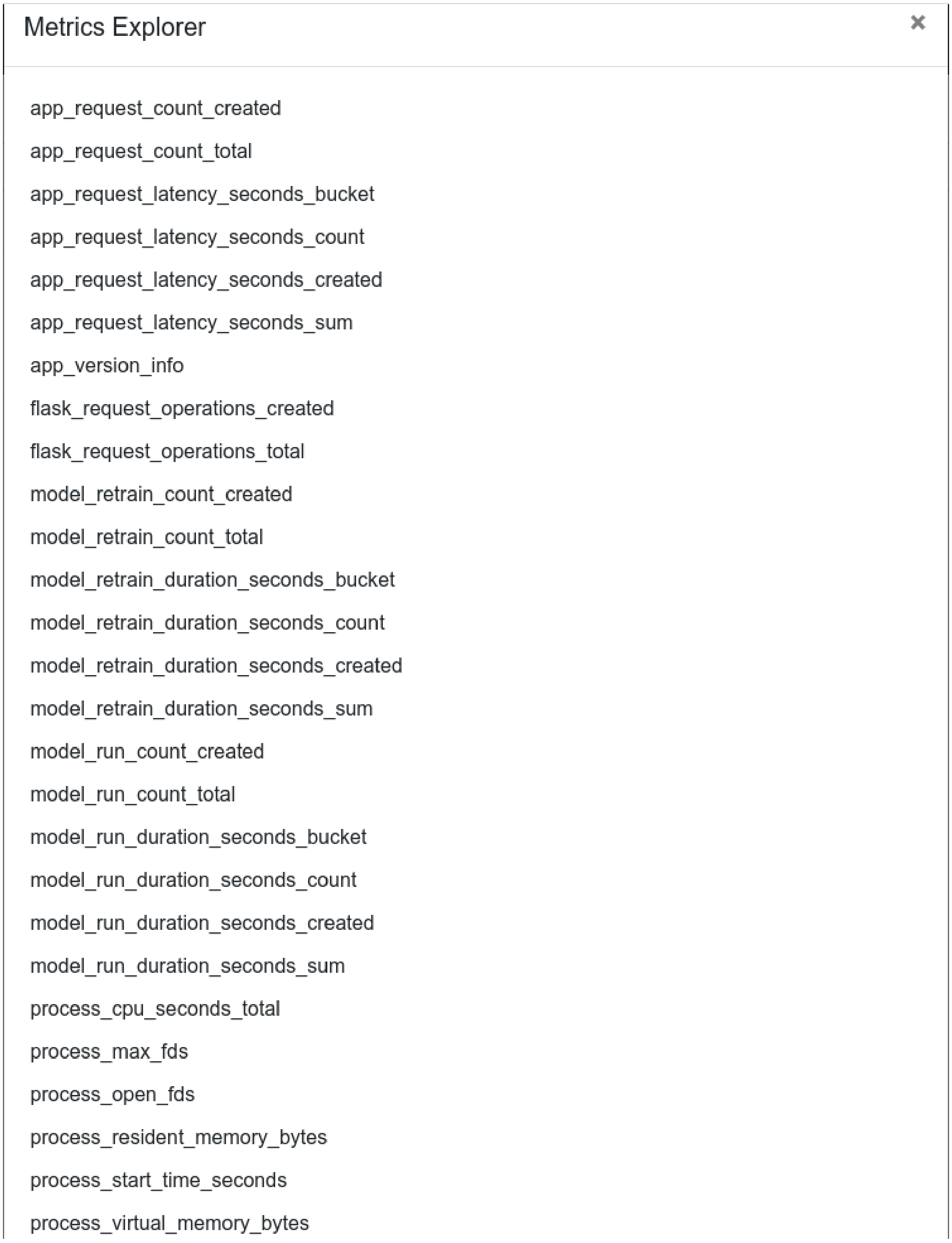


```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 101436.0
python_gc_objects_collected_total{generation="1"} 21725.0
python_gc_objects_collected_total{generation="2"} 953.0
# HELP python_gc_objects_uncollectable_total Uncollectable object found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 1225.0
python_gc_collections_total{generation="1"} 114.0
python_gc_collections_total{generation="2"} 7.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="8",patchlevel="10",version="3.8.10"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 1.85270272e+09
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 5.55085824e+08
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.63536613527e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 107.79
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 15.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP app_request_latency_seconds Application Request Latency
# TYPE app_request_latency_seconds histogram
[illegible]
# HELP app_version_info Application Version
# TYPE app_version_info gauge
app_version_info{config="staging",version="v3.0"} 1.0
# HELP model_run_count_total Total number of times Model was run
# TYPE model_run_count_total counter
model_run_count_total 1.0
# HELP model_run_count_created Total number of times Model was run
# TYPE model_run_count_created gauge
model_run_count_created 1.6353661382715027e+09
# HELP model_retrain_count_total Total number of times Model was Re-Trained
# TYPE model_retrain_count_total counter
model_retrain_count_total 0.0
# HELP model_retrain_count_created Total number of times Model was Re-Trained
# TYPE model_retrain_count_created gauge
model_retrain_count_created 1.635366138271529e+09
# HELP flask_request_operations_total The total number of processed requests
# TYPE flask_request_operations_total counter
flask_request_operations_total 6.0
# HELP flask_request_operations_created The total number of processed requests
# TYPE flask_request_operations_created gauge
flask_request_operations_created 1.6353661382715492e+09
# HELP model_run_duration_seconds Histogram for the prediction duration in seconds
# TYPE model_run_duration_seconds histogram
model_run_duration_seconds_bucket{le="100.0"} 1.0
model_run_duration_seconds_bucket{le="150.0"} 1.0
model_run_duration_seconds_bucket{le="200.0"} 1.0
model_run_duration_seconds_bucket{le="250.0"} 1.0
model_run_duration_seconds_bucket{le="300.0"} 1.0
model_run_duration_seconds_bucket{le="+Inf"} 1.0
model_run_duration_seconds_count 1.0
model_run_duration_seconds_sum 40.73413896560669
# HELP model_run_duration_seconds_created Histogram for the prediction duration in seconds
# TYPE model_run_duration_seconds_created gauge
model_run_duration_seconds_created 1.6353661382715762e+09
# HELP model_retrain_duration_seconds Histogram for the Retraining duration in seconds
# TYPE model_retrain_duration_seconds histogram
model_retrain_duration_seconds_bucket{le="200.0"} 0.0
model_retrain_duration_seconds_bucket{le="300.0"} 0.0
model_retrain_duration_seconds_bucket{le="400.0"} 0.0
model_retrain_duration_seconds_bucket{le="500.0"} 0.0
model_retrain_duration_seconds_bucket{le="600.0"} 0.0
model_retrain_duration_seconds_bucket{le="+Inf"} 0.0
model_retrain_duration_seconds_count 0.0
model_retrain_duration_seconds_sum 0.0
# HELP model_retrain_duration_seconds_created Histogram for the Retraining duration in seconds
# TYPE model_retrain_duration_seconds_created gauge
model_retrain_duration_seconds_created 1.6353661382716222e+09
```

Figure 4.14 – Metrics Endpoint

4.2.2.7 Prometheus

- For the monitoring and alerting of models and the application, Prometheus is incorporated with this application. It provides a set of queries as shown in figure 4.15, which can be used to track the performance.


Metrics Explorer

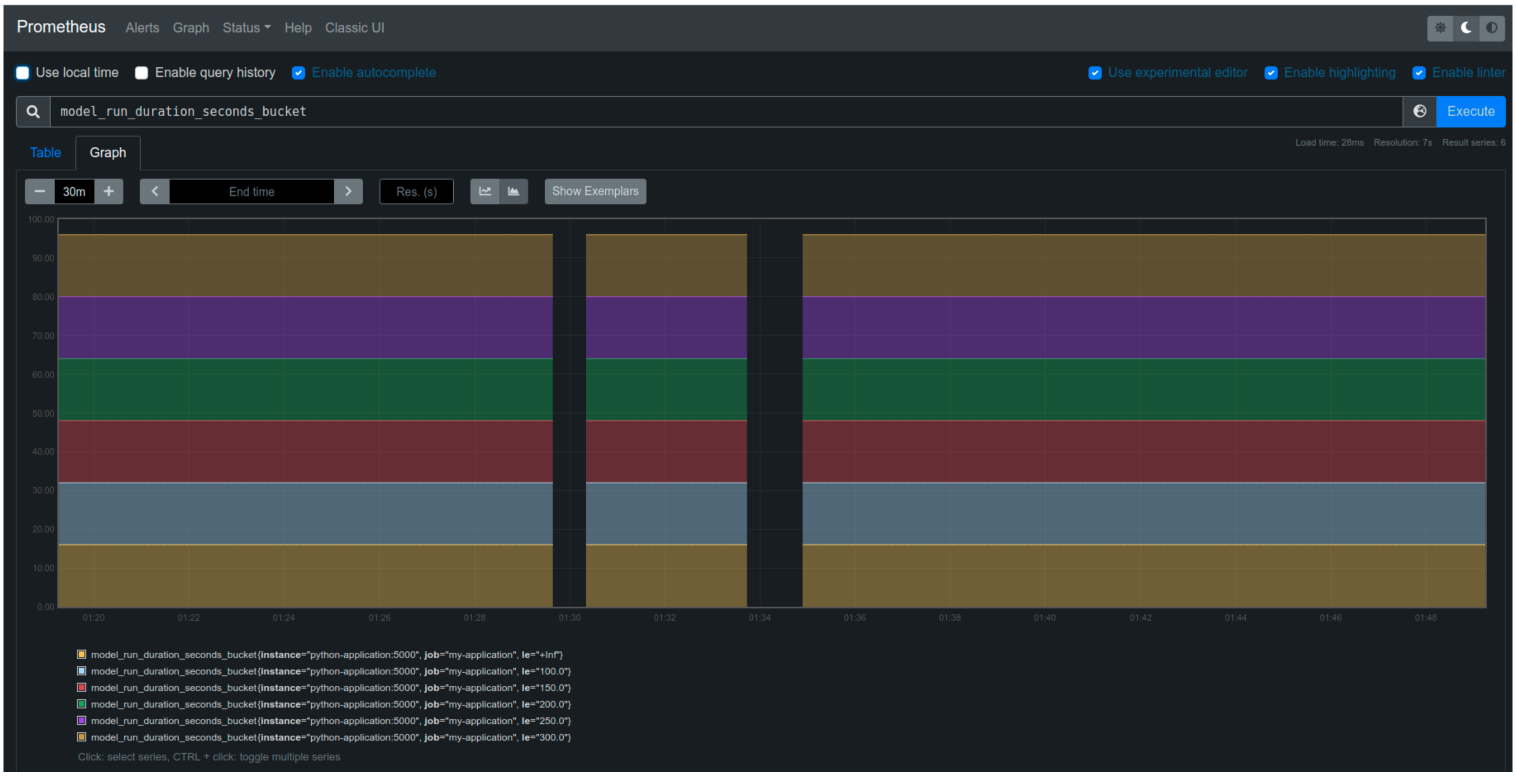
app_request_count_created
app_request_count_total
app_request_latency_seconds_bucket
app_request_latency_seconds_count
app_request_latency_seconds_created
app_request_latency_seconds_sum
app_version_info
flask_request_operations_created
flask_request_operations_total
model_retrain_count_created
model_retrain_count_total
model_retrain_duration_seconds_bucket
model_retrain_duration_seconds_count
model_retrain_duration_seconds_created
model_retrain_duration_seconds_sum
model_run_count_created
model_run_count_total
model_run_duration_seconds_bucket
model_run_duration_seconds_count
model_run_duration_seconds_created
model_run_duration_seconds_sum
process_cpu_seconds_total
process_max_fds
process_open_fds
process_resident_memory_bytes
process_start_time_seconds
process_virtual_memory_bytes

Figure 4.15 – Prometheus (1)

- Also, it offers two options for depicting the results:–
  - Table
  - Graph

  The figure 4.16 visualizes the graphical result for the query:–
  - model_run_duration_seconds_bucket

This represents a histogram for the number of seconds the model consumes for a single prediction. The X-axis represents the time in minutes, while Y-axis represents the buckets for the histogram. The model requires approximately 100 - 300 seconds for each prediction, as listed by the color codes.



Use local time · Enable query history · ✓ Enable autocomplete · ✓ Use experimental editor · ✓ Enable highlighting · ✓ Enable linter

```
model_run_duration_seconds_bucket
```

Load time: 28ms Resolution: 7s Result series: 6

```
model_run_duration_seconds_bucket{instance="python-application:5000", job="my-application", le="+Inf"}
model_run_duration_seconds_bucket{instance="python-application:5000", job="my-application", le="100.0"}
model_run_duration_seconds_bucket{instance="python-application:5000", job="my-application", le="150.0"}
model_run_duration_seconds_bucket{instance="python-application:5000", job="my-application", le="200.0"}
model_run_duration_seconds_bucket{instance="python-application:5000", job="my-application", le="250.0"}
model_run_duration_seconds_bucket{instance="python-application:5000", job="my-application", le="300.0"}
```

Click: select series, CTRL + click: toggle multiple series

Figure 4.16 – Prometheus (2)

4.2.2.8 Grafana

For live monitoring, Grafana is the preferred traditional approach, as it provides a plethora of descriptive and interactive options. Therefore, it is also included in this application. The input data source for Grafana is Prometheus. Some of the snapshots of this application rendering Grafana dashboards are illustrated in figures 4.17 and 4.18.

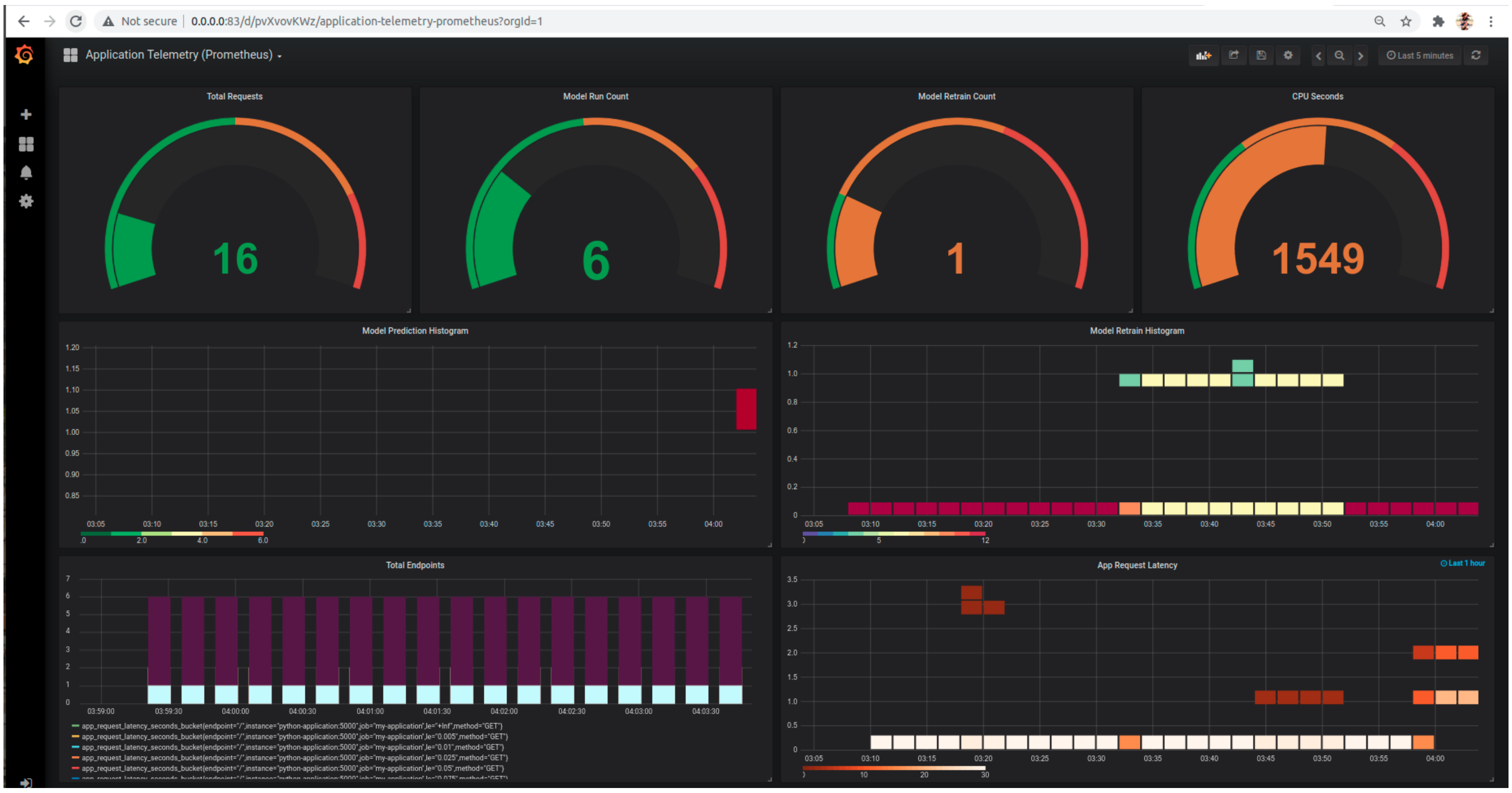
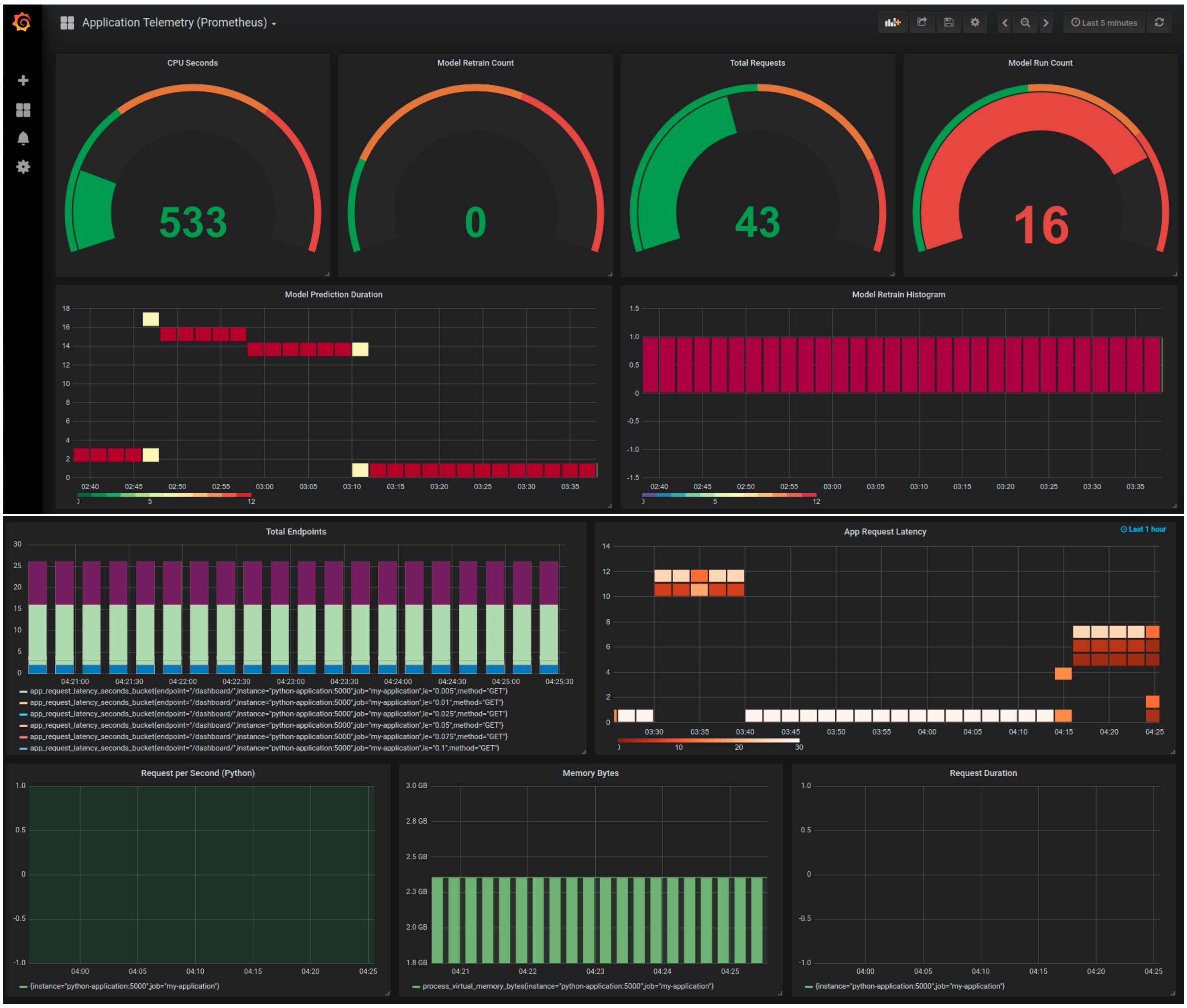


Figure 4.17 – Grafana (1)

Currently, the dashboard consists of following panels:–

- **CPU Seconds**:– The number of seconds since the application is running.
- **Total Requests**:– Total number of requests received by the application.
- **Model Run Count**:– Total number of times the model is run.
- **Model Retrain Count**:– Total number of times the model is retrained.
- **Model Prediction Duration**:– Histogram for the duration (in seconds) for model prediction.
- **Model Retrain Histogram**:– Histogram for the duration (in seconds) for retraining the model.

- **Request Duration [36]**:– Time taken by the application requests.
- **Request per Second [36]**:– Number of application requests per second.
- **Total Endpoints**:– Total number and count of the application endpoints.
- **App Request Latency**:– Histogram for the latency of application request.



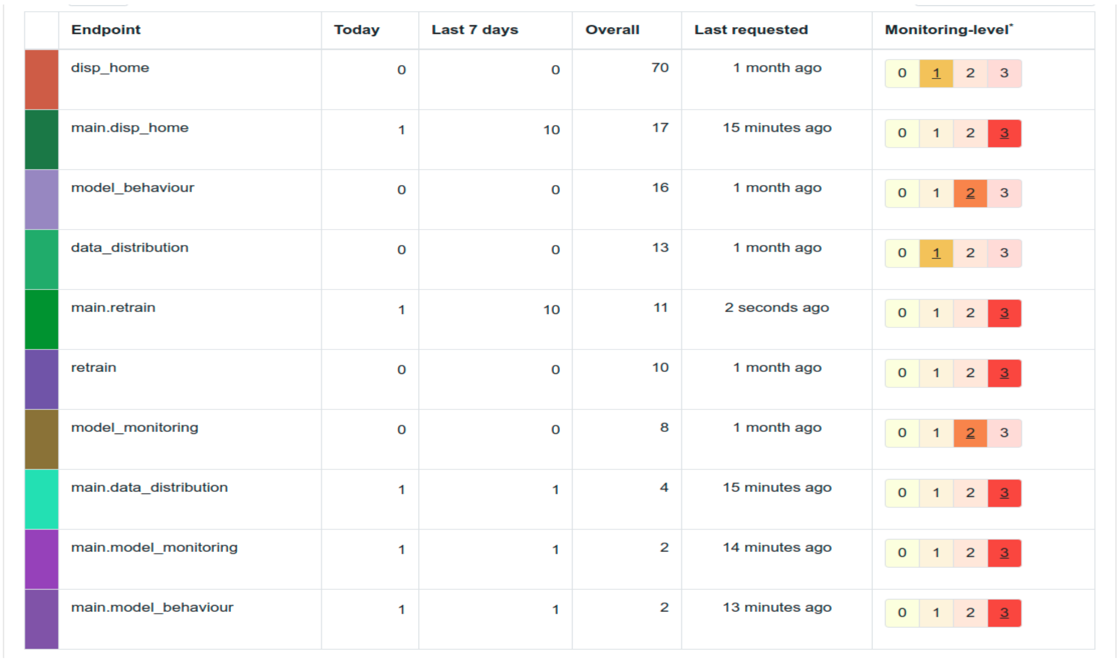
app_request_latency_seconds_bucket{endpoint="/dashboard/",instance="python-application:5000",job="my-application",le="0.005",method="GET"}
app_request_latency_seconds_bucket{endpoint="/dashboard/",instance="python-application:5000",job="my-application",le="0.01",method="GET"}
app_request_latency_seconds_bucket{endpoint="/dashboard/",instance="python-application:5000",job="my-application",le="0.025",method="GET"}
app_request_latency_seconds_bucket{endpoint="/dashboard/",instance="python-application:5000",job="my-application",le="0.05",method="GET"}
app_request_latency_seconds_bucket{endpoint="/dashboard/",instance="python-application:5000",job="my-application",le="0.075",method="GET"}
app_request_latency_seconds_bucket{endpoint="/dashboard/",instance="python-application:5000",job="my-application",le="0.1",method="GET"}

{instance="python-application:5000",job="my-application"}
process_virtual_memory_bytes{instance="python-application:5000",job="my-application"}
{instance="python-application:5000",job="my-application"}

Figure 4.18 – Grafana (2)

4.2.2.9 Flask Dashboard [37]

In this application, a free-to-use monitoring dashboard is also included. It has many features to monitor the flask application and its APIs in several ways. As shown in figure 4.19, all the application endpoints are listed with their details like overall count, count for last seven days, count for the present day, last requested, and monitoring level. The monitoring level can be set according to the severity of the API.



	Endpoint	Today	Last 7 days	Overall	Last requested	Monitoring-level*
🟥	disp_home	0	0	70	1 month ago	0 [✓ 1] 2 3
🟩	main.disp_home	1	10	17	15 minutes ago	0 1 2 [✓ 3]
🟪	model_behaviour	0	0	16	1 month ago	0 1 [✓ 2] 3
🟩	data_distribution	0	0	13	1 month ago	0 [✓ 1] 2 3
🟩	main.retrain	1	10	11	2 seconds ago	0 1 2 [✓ 3]
🟪	retrain	0	0	10	1 month ago	0 1 2 [✓ 3]
🟫	model_monitoring	0	0	8	1 month ago	0 1 [✓ 2] 3
🟩	main.data_distribution	1	1	4	15 minutes ago	0 1 2 [✓ 3]
🟪	main.model_monitoring	1	1	2	14 minutes ago	0 1 2 [✓ 3]
🟪	main.model_behaviour	1	1	2	13 minutes ago	0 1 2 [✓ 3]

Figure 4.19 – App Dashboard (1)

Moreover, in figure 4.20, a box-plot is rendered for the performance of an API, "model_behavior" based on the execution time in milliseconds.

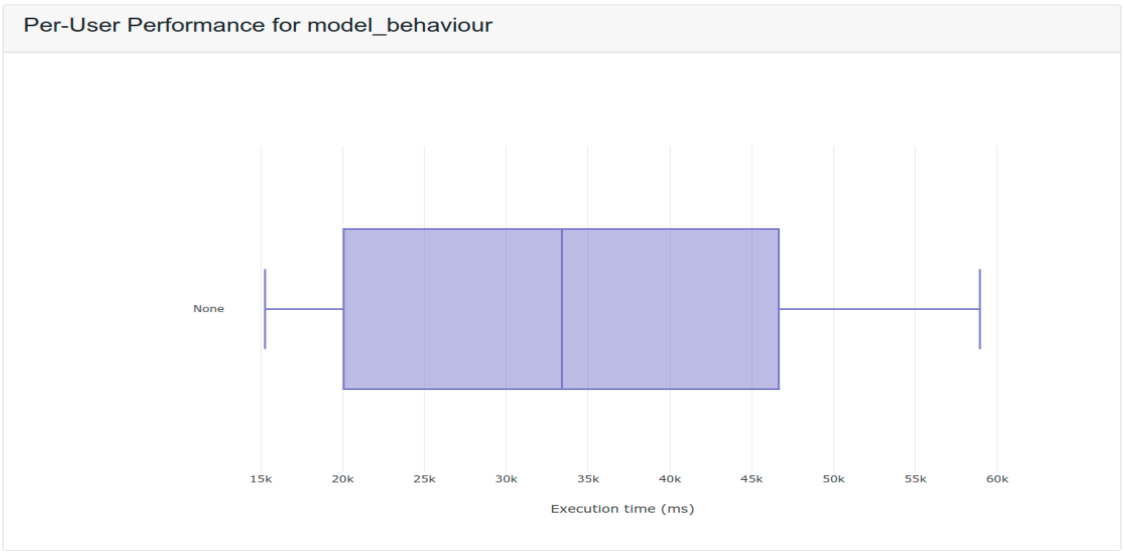


Figure 4.20 – App Dashboard (2)

4.3 How to Run?

- For using this application, following pre-requisites must be fulfilled:–

```
|-- Python
|-- Docker
|-- Docker-Compose
|-- IDE (Recommended - VS Code)
```

- Once these dependencies are installed, execute following commands to download the application:–

```
# Download the Project
git clone https://github.com/safir-md/DevOps-for-Machine-
    Learning.git

# Traverse to the application directory
cd DevOps-for-Machine-Learning/cnc-mill-app
```

Listing 4.1 – Application Installation

- Now, the application is ready to run and go live. To achieve the same, execute below commands:–

```
# start prometheus service
docker-compose up -d prometheus

# start grafana service
docker-compose up -d grafana

# run grafana dashboard
docker-compose up -d grafana-dashboards

# start the application
docker-compose up -d --build python-application

# run ml-flow ui
docker exec -it $(docker ps -qf "name=python-application") /bin
    /sh -c "mlflow ui --host 0.0.0.0 --port 5001"
```

Listing 4.2 – Running Application

- Congratulations, the application is running now in the docker containers. View following URLs in the browser to access the application:–

```
|-- Application:- 0.0.0.0:80
|-- ML-Flow:- 0.0.0.0:81
|-- Prometheus:- 0.0.0.0:82
|-- Grafana:- 0.0.0.0:83
|-- Exposed Metrics for Prometheus:- 0.0.0.0:80/metrics
```

Note:– If the application changes are not being reflected in the browser, try either Hard Reloading (Ctrl + F5) or re-try after clearing the browser cache!!!

Running in Debug Mode:–

- For running the same application in debug mode, execute the following commands to create and run a docker container locally:–

```
cd python-application

docker image build -t python-application .

docker run -it -p 80:5000 -p 81:5001 python-application
```

Listing 4.3 – Running in Debug Mode (1)

- Now, execute the below command **in a new terminal** to run the ML-Flow service:–

```
    docker exec -it $(docker ps -qf "name=python-application")
   /bin/sh -c "mlflow ui --host 0.0.0.0 --port 5001"
```

Listing 4.4 – Running in Debug Mode (2)

Troubleshooting:–

- In case of problems related to docker containers or ports, execute the below commands to clear all the local containers and images respectively:–

```
docker container rm -f $(docker container ls -a -q)
docker image rm -f $(docker image ls -q)
```

Listing 4.5 – Clear Containers and Images

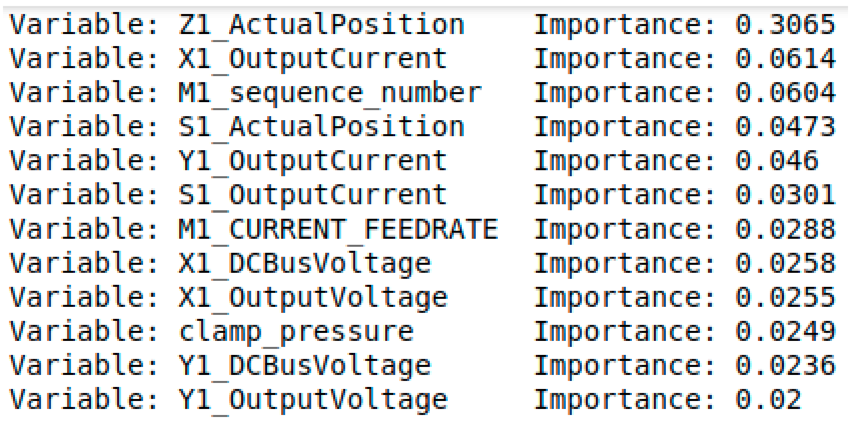
Chapter 5

Results and Discussion

While implicating the research work into the implementation part, a total of 3 models were created,

- Decision Tree,
- Random Forest, and
- Auto-Sklearn Model

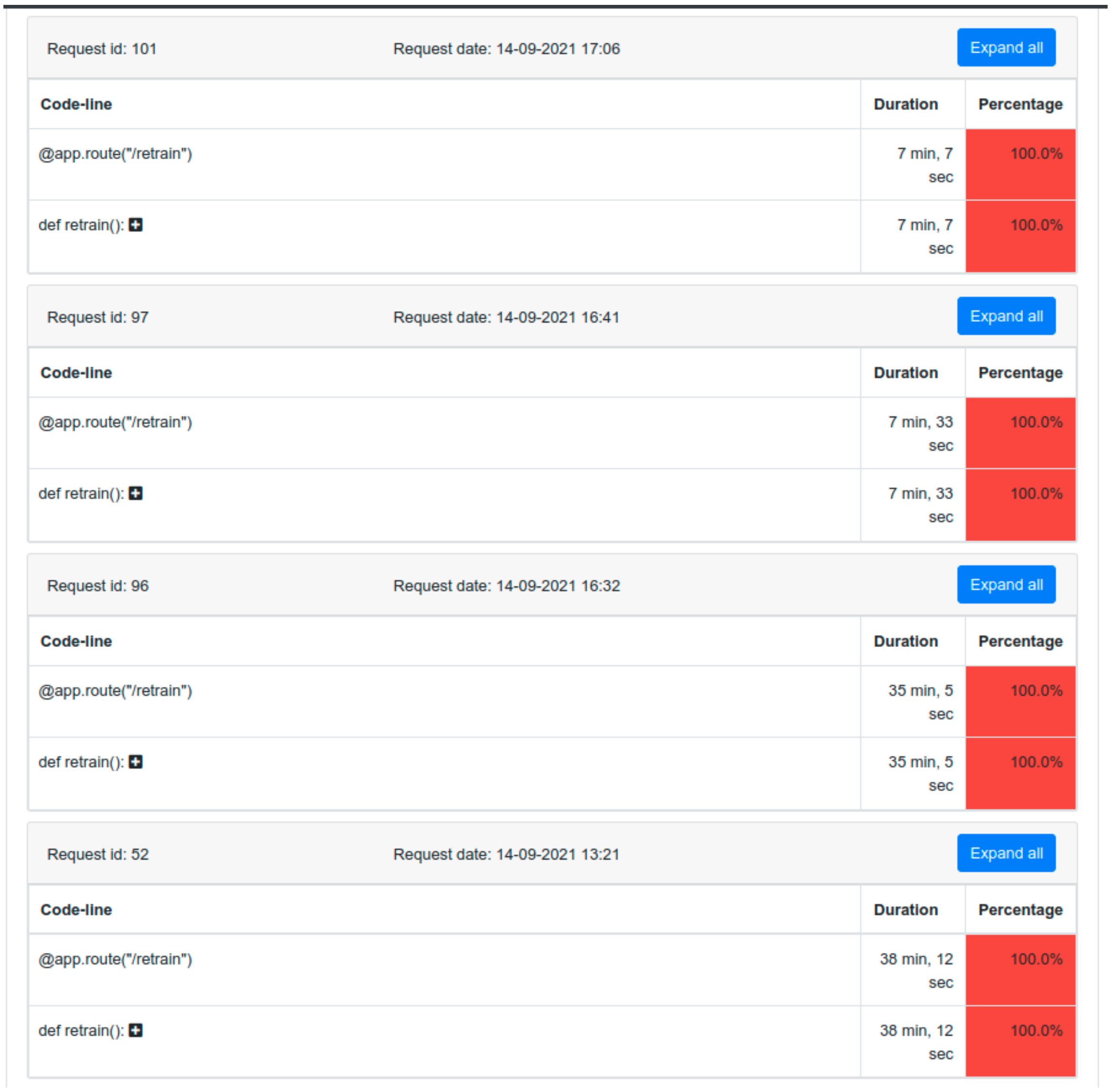
And, in each of the three models, the same kind of data-processing was performed. For all of the models, **a statistical significance threshold of 95%** was set during feature selection, i.e., only features possessing cumulative importance of 95% and covering at least 2% of the variance were selected. Therefore, a total of **12 of 48** features were finally selected to feed into the model. The figure 5.1 shows the selected features along with their importance:–



```
Variable: Z1_ActualPosition    Importance: 0.3065
Variable: X1_OutputCurrent     Importance: 0.0614
Variable: M1_sequence_number   Importance: 0.0604
Variable: S1_ActualPosition    Importance: 0.0473
Variable: Y1_OutputCurrent     Importance: 0.046
Variable: S1_OutputCurrent     Importance: 0.0301
Variable: M1_CURRENT_FEEDRATE  Importance: 0.0288
Variable: X1_DCBusVoltage      Importance: 0.0258
Variable: X1_OutputVoltage     Importance: 0.0255
Variable: clamp_pressure       Importance: 0.0249
Variable: Y1_DCBusVoltage      Importance: 0.0236
Variable: Y1_OutputVoltage     Importance: 0.02
```

Figure 5.1 – Selected Features

After including this step of feature selection, **remarkable improvement in training time** was discovered as compared to training with all the features, without affecting model performance. Earlier, it used to take about 35-40 minutes for retraining, however, now it takes **only 7-8 minutes**, as proved by the "Duration" column of "/retrain" API in figure 5.2:–



Request id: 101 Request date: 14-09-2021 17:06 Expand all

Code-line	Duration	Percentage
@app.route("/retrain")	7 min, 7 sec	100.0%
def retrain():	7 min, 7 sec	100.0%

Request id: 97 Request date: 14-09-2021 16:41 Expand all

Code-line	Duration	Percentage
@app.route("/retrain")	7 min, 33 sec	100.0%
def retrain():	7 min, 33 sec	100.0%

Request id: 96 Request date: 14-09-2021 16:32 Expand all

Code-line	Duration	Percentage
@app.route("/retrain")	35 min, 5 sec	100.0%
def retrain():	35 min, 5 sec	100.0%

Request id: 52 Request date: 14-09-2021 13:21 Expand all

Code-line	Duration	Percentage
@app.route("/retrain")	38 min, 12 sec	100.0%
def retrain():	38 min, 12 sec	100.0%

Figure 5.2 – Optimized Training Time

All the 3 models were evaluated on different performance metrics, such as accuracy score, confusion matrix, precision score, recall score, f-1 score, and correlation coefficient. The table 5.1 depicts the figures obtained for all the metrics by the models:–

	Accuracy	**Precision**	**Recall**	**F-1 Score**	**Correlation Coefficient**
Decision Tree	99.00 %	0.9900	0.9915	0.9908	0.9799
Random Forest	99.34 %	0.9942	0.9937	0.9939	0.9868
Auto-Sklearn	**99.46 %**	**0.9958**	**0.9942**	**0.9950**	**0.9891**

Table 5.1 – Performance Metrics

Moreover, the following table delineates the confusion matrix for Decision Tree, Random Forest, and Auto-Sklearn:–

	0	**1**
0	1591	21
1	16	1876

Table 5.2 – Decision Tree

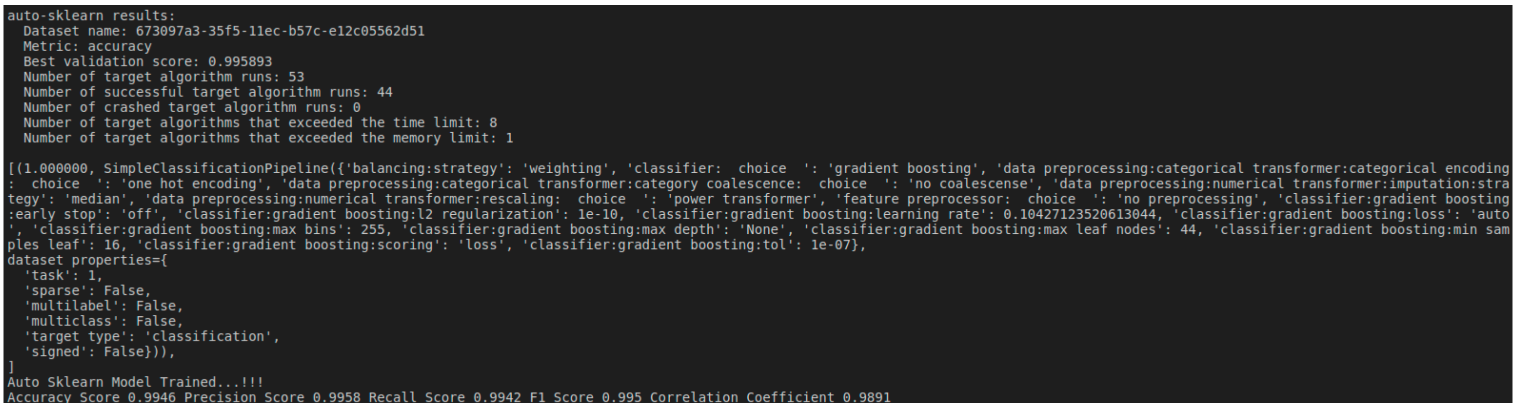
	0	**1**
0	1601	11
1	11	1881

Table 5.3 – Random Forest

	0	**1**
0	1604	10
1	9	1881

Table 5.4 – Auto-Sklearn

Therefore, the AutoML technique seemed to be promising with the highest performance among the 3 methods. Another advantage of AutoML is the validation score, which depicts that, the model would perform well on new data as well. **Hence, by defeating the best of models, Auto-Sklearn has achieved an accuracy of 99.46 % and the best validation score of 0.9959.** The figure 5.3 renders the console logs of the auto-sklearn model:–



```
auto-sklearn results:
  Dataset name: 673097a3-35f5-11ec-b57c-e12c05562d51
  Metric: accuracy
  Best validation score: 0.995893
  Number of target algorithm runs: 53
  Number of successful target algorithm runs: 44
  Number of crashed target algorithm runs: 0
  Number of target algorithms that exceeded the time limit: 8
  Number of target algorithms that exceeded the memory limit: 1

[(1.000000, SimpleClassificationPipeline({'balancing:strategy': 'weighting', 'classifier:  choice  ': 'gradient boosting', 'data preprocessing:categorical transformer:categorical encoding
:  choice  ': 'one hot encoding', 'data preprocessing:categorical transformer:category coalescence:  choice  ': 'no coalescense', 'data preprocessing:numerical transformer:imputation:stra
tegy': 'median', 'data preprocessing:numerical transformer:rescaling:  choice  ': 'power transformer', 'feature preprocessor:  choice  ': 'no preprocessing', 'classifier:gradient boosting
:early stop': 'off', 'classifier:gradient boosting:l2 regularization': 1e-10, 'classifier:gradient boosting:learning rate': 0.10427123520613044, 'classifier:gradient boosting:loss': 'auto
', 'classifier:gradient boosting:max bins': 255, 'classifier:gradient boosting:max depth': 'None', 'classifier:gradient boosting:max leaf nodes': 44, 'classifier:gradient boosting:min sam
ples leaf': 16, 'classifier:gradient boosting:scoring': 'loss', 'classifier:gradient boosting:tol': 1e-07},
dataset properties={
  'task': 1,
  'sparse': False,
  'multilabel': False,
  'multiclass': False,
  'target type': 'classification',
  'signed': False})),
]
Auto Sklearn Model Trained...!!!
Accuracy Score 0.9946 Precision Score 0.9958 Recall Score 0.9942 F1 Score 0.995 Correlation Coefficient 0.9891
```

Figure 5.3 – Auto-Sklearn Logs

As auto-sklearn runs an ensemble of different algorithms, it has figured out **"Gradient Boosting"** as the best fit for this dataset. Within the given time frame, it ran a total of 53 algorithms, and out of 53, only 44 were successful, because the remaining 9 algorithms exceeded the time/memory limit. Moreover, it automatically performs data processing (if required) and hyperparameter optimization. Since in this case, the data was already processed and the best features were already selected, it just skipped the same.

Chapter 6

Conclusion and Future Scope

- **Conclusion:–**

  In this thesis, thorough research has been conducted on the best practices of machine learning model deployment and operations to conquer the failure causes of ML systems. To demonstrate the research outcomes, an end-to-end, automated, and production-ready python application is developed to monitor, analyze and improve the system. Moreover, significant improvement is noticed after the implementation of MLOps with transparency methods, such as lessened failures, reduced time consumption, and effective model performance. To add efficiency, the application is implemented with cloud-support technologies like docker and docker-compose for faster deployment and higher-quality operations.

- **Future Scope:–**

  Even though spectacular results are achieved in this thesis, the following things can be improved **in the future**:–

  - **Python Docker Image:–** Currently, the docker image for python takes too long to build and consumes a lot of memory.
  - **Under Development Tools:–** Many tools discussed and implemented in this thesis are still under development, and some of them support only the Linux platform.

Bibliography

[1] Frank Hutter, Marius Lindauer, “AutoML | Freiburg-Hannover,” 2018, Last Accessed: 2020-10-22. [Online]. Available: https://www.automl.org/automl/

[2] Databricks, “MLflow - An open source platform for the machine learning lifecycle,” 2018. [Online]. Available: https://mlflow.org/

[3] O. SCHWARTZ, “In 2016, Microsoft’s Racist Chatbot Revealed the Dangers of Online Conversation,” Nov 2019. [Online]. Available: https://spectrum.ieee.org/in-2016-microsofts-racist-chatbot-revealed-the-dangers-of-online-conversation

[4] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, “Hidden Technical Debt in Machine Learning Systems,” in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

[5] “MLOps: Continuous delivery and automation pipelines in machine learning,” 07 2020. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[6] M. T. Ribeiro, S. Singh, and C. Guestrin, “"Why Should I Trust You?": Explaining the Predictions of Any Classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.

[7] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin, “ Anchors: High-Precision Model-Agnostic Explanations ,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[8] S. M. Lundberg and S.-I. Lee, “A Unified Approach to Interpreting Model Predictions,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*,

vol. 12, pp. 2825–2830, 2011. [Online]. Available: https://scikit-learn.org/stable/modules/partial_dependence.html

[10] Grafana Labs, "Grafana Documentation," 2018. [Online]. Available: https://grafana.com/docs/

[11] Bjorn Rabenstein and Julius Volz, "Prometheus: A Next-Generation Monitoring System (Talk)." Dublin: USENIX Association, May 2015. [Online]. Available: https://prometheus.io/docs/introduction/overview/

[12] Singh, Pramod, "Model Deployment and Challenges," in *Deploy Machine Learning Models to Production: With Flask, Streamlit, Docker, and Kubernetes on Google Cloud Platform.* Berkeley, CA: Apress, 2021, pp. 55–66. [Online]. Available: https://doi.org/10.1007/978-1-4842-6546-8_2

[13] F. Hinder, J. Jakob, and B. Hammer, "Analysis of drifting features," *CoRR*, vol. abs/2012.00499, 2020. [Online]. Available: https://arxiv.org/abs/2012.00499

[14] O'Neil, Cathy, *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*, English ed. Crown, 2016, vol. 1.

[15] R. Soden, D. Wagenaar, D. Luo, and A. Tijssen, "Taking Ethics, Fairness, and Bias Seriously in Machine Learning for Disaster Risk Management," *CoRR*, vol. abs/1912.05538, 2019. [Online]. Available: http://arxiv.org/abs/1912.05538

[16] S. Barocas, M. Hardt, and A. Narayanan, "Fairness in Machine Learning," 2020. [Online]. Available: http://cse.iitkgp.ac.in/~saptarshi/courses/ml2020s/ML-15-Fairness.pdf

[17] R. S. S. Kumar, D. R. O'Brien, K. Albert, S. Viljöen, and J. Snover, "Failure Modes in Machine Learning Systems," *CoRR*, vol. abs/1911.11034, 2019. [Online]. Available: http://arxiv.org/abs/1911.11034

[18] R. S. S. Kumar, M. Nystrom, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia, "Adversarial Machine Learning - Industry Perspectives," *SSRN*, vol. http://dx.doi.org/10.2139/ssrn.3532474, 2020. [Online]. Available: https://ssrn.com/abstract=3532474

[19] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning," *CoRR*, vol. abs/1804.00308, 2018. [Online]. Available: http://arxiv.org/abs/1804.00308

[20] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, and M. Young, "Machine Learning: The High Interest Credit Card of Technical Debt," in *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[21] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms," in *Proc. of KDD-2013*, 2013, pp. 847–855.

[22] E. Frank, M. A. Hall, and I. H. Witten, "Weka 3 - Data Mining with Open Source Machine Learning Software in Java," 2016. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/index.html

[23] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: https://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[24] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, M. Urban, M. Burkart, M. Dippel, M. Lindauer, and F. Hutter, "Towards Automatically-Tuned Deep Neural Networks," in *AutoML: Methods, Sytems, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Springer, Dec. 2018, ch. 7, pp. 141–156.

[25] L. Zimmer, M. Lindauer, and F. Hutter, "Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 3079 – 3090, 2021. [Online]. Available: https://arxiv.org/abs/2006.13799

[26] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data," *arXiv preprint arXiv:2003.06505*, 2020.

[27] E. LeDell and S. Poirier, "H2O AutoML: Scalable Automatic Machine Learning," *7th ICML Workshop on Automated Machine Learning (AutoML)*, July 2020. [Online]. Available: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

[28] A. A. D. ROMBLAY, "GitHub - AxeldeRomblay/MLBox: MLBox is a powerful Automated Machine Learning python library." 2019. [Online]. Available: https://github.com/AxeldeRomblay/MLBox

[29] K. Moore, K. F. Kan, L. McGuire, M. Tovbin, M. Ovsiankin, M. Loh, M. Weil, S. Nabar, V. Gordon, and P. Vlad, "GitHub - salesforce/TransmogrifAI: TransmogrifAI," 2018. [Online]. Available: https://github.com/salesforce/TransmogrifAI

[30] T. T. Le, W. Fu, and J. H. Moore, "Scaling tree-based automated machine learning to biomedical big data with a feature set selector," *Bioinformatics*, vol. 36, no. 1, pp. 250–256, 2020.

[31] H. Felzmann, E. Fosch-Villaronga, C. Lutz, and A. Tamò-Larrieux, "Towards Transparency by Design for Artificial Intelligence," vol. 26, no. 6, pp. 3333–3361, Nov. 2020. [Online]. Available: https://doi.org/10.1007/s11948-020-00276-4

[32] C. Molnar, "Scope of Interpretability," in *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable.* BOOKDOWN, 10 2021, pp. 57–59. [Online]. Available: https://christophm.github.io/interpretable-ml-book/scope-of-interpretability.html

[33] Christoph Molnar, "Global Surrogate," in *Interpretable Machine Learning - A Guide for Making Black Box Models Explainable*. BOOKDOWN, 10 2021, pp. 221–226. [Online]. Available: https://christophm.github.io/interpretable-ml-book/global.html

[34] Brian Brazil, "Prometheus Python Client - prometheus-client 0.11.0," in *Python Package Index*, 06 2021. [Online]. Available: https://pypi.org/project/prometheus-client/

[35] Sharon Sun, "CNC Mill Tool Wear - Version 4," in *Kaggle*, 03 2018. [Online]. Available: https://www.kaggle.com/shasun/tool-wear-detection-in-cnc-mill

[36] Marcel Dempers, "docker-development-youtube-series/monitoring/prometheus," in *GitHub*, 02 2020. [Online]. Available: https://github.com/marcel-dempers/docker-development-youtube-series/tree/master/monitoring/prometheus

[37] Patrick Vogel, Bogdan Petre, Thijs Klooster, "Flask Monitoring Dashboard - Version 3.0.7," in *GitHub*, 02 2020. [Online]. Available: https://github.com/flask-dashboard/Flask-MonitoringDashboard

****Note:– All the references mentioned above were accessed on October 22, 2021.***

Appendix A - Definitions

- **Performance Degradation:**– In performance degradation, the predictive performance of many models decreases over time as it is tested on new data within rapidly evolving environments.

- **Label Drifting:**– A label drift is a change in the probability of a label p(Y), where, Y is referred to as the true label population.

- **Prediction Drifting:**– Prediction drift is a change in the distribution of a predicted label, i.e., a change in the relationship between the model input and the model output.

- **Research Environment:**– It is the development environment in which data scientists conduct experiments to create an optimized model.

- **Production Environment:**– After deployment, the model is served in the stakeholder's production environment, exposed to the customers, and receives completely new data for prediction. It is new and usually different from the training/research environment where the model was built.

- **Intentional Failure:**– It is the failure caused by an active contender attempting to destabilize the system, either to misclassify the result or to steal the underlying algorithm. ***For example,*** poisoning attack, model stealing, model inversion, perturbation attack, etc.

- **Unintentional Failure:**– It is the failure caused when an ML system produces a result, which is formally correct but completely unsafe. ***For example,*** side effects, reward hacking, distributional shifts, etc.

- **White Box Attack:**– In white-box attacks, the attacker is supposed to know the training data, feature values, learning algorithm, and the trained parameters.

- **Black Box Attack:**– In black-box attacks, the attacker doesn't hold any knowledge about the training set but can manage a substitute data set. Moreover, the learning algorithm and the feature set are familiar, while the trained parameters are not known. However, the trained parameters can be estimated by optimizing the algorithm on the substitute dataset.

- **Training-Serving Skew:**– It is the contrast between model performance during training and model performance during serving. It can be caused due to change in data from training to serving, feedback loops, or handling data differently during training and serving.

- **Baseline Value:**– The base value is the output of an average model trained on the training dataset.

- **Coverage Value:**– The coverage value in an anchor explanation depicts the fraction of predicted instances after seeing the explanations. If there is only one prediction, the coverage value is 0.

Appendix B - Formulae

- **Equal Width Binning :–**

$$Width = \frac{Max - Min}{N}$$

$$where,$$
$$N = Number\,of\,Intervals \tag{6.1}$$

- **Min-Max Normalization (0 to 1) :–**

$$x' = \frac{x - Min}{Max - Min} \tag{6.2}$$

- **Z-Score Normalization :–**

$$x' = \frac{x - mean}{\sigma}$$

$$where,$$
$$\sigma = Standard\,Deviation \tag{6.3}$$

- **Decimal Scaling Normalization :–**

$$x' = \frac{x}{10^{\,j}} \tag{6.4}$$

- **Entropy Based Discretization :–**

$$E(S,\,T) = \frac{|S_1|}{S}\,E(S_1) + \frac{|S_2|}{S}\,E(S_2)$$

$$\begin{aligned} &where, \\ E &= Entropy \\ S &= Dataset \\ T &= Boundary \\ S_1\,\&\,S_2 &= Intervals \end{aligned} \tag{6.5}$$

- **SSE :–**

$$\begin{aligned} SSE \;&=\; \Sigma_i\,(y_i \;-\; \hat{y}_i)^2 \\ & \\ &where, \\ SSE \;&=\; Sum\,of\,Squared\,Errors \\ y_i \;&=\; Actual\,Target \\ \hat{y}_i \;&=\; Predicted\,Output \end{aligned} \tag{6.6}$$

- **AIC & SBC:–**

$$\begin{aligned} AIC \;&=\; n.\,log\,(\frac{SSE}{n}) \;+\; 2k \\ SBC \;&=\; n.\,log\,(\frac{SSE}{n}) \;+\; k.\,log(n) \\ & \\ &where, \\ n \;&=\; Number\,of\,Observations \\ k \;&=\; Number\,of\,Independent\,Variables \end{aligned} \tag{6.7}$$

- **R-Squared Measure:–**

$$\begin{aligned} R^2 \;&=\; 1 \;-\; \frac{SSE}{SST} \\ & \\ &where, \\ R^2 \;&=\; Coefficient\,of\,Determination \\ SST\;(Total\;sum\;of\;Squares) \;&=\; \Sigma_i\,(y_i \;-\; y_{avg})^2 \end{aligned} \tag{6.8}$$

- **Accuracy, Precision, & Recall :–**

$$\begin{aligned} Recall \;&=\; \frac{\#TP}{\#TP \;+\; \#FN} \\ Precision \;&=\; \frac{\#TP}{\#TP \;+\; \#FP} \\ Accuracy \;&=\; \frac{\#TP \;+\; \#TN}{\#TP \;+\; \#FN \;+\; \#FP \;+\; \#TN} \\ & \\ &where, \\ \# \;&=\; Count \\ TP \;&=\; True\,Positive \\ TN \;&=\; True\,Negative \\ FP \;&=\; False\,Positive \\ FN \;&=\; False\,Negative \end{aligned} \tag{6.9}$$

Appendix C - Public Access

All of the research and implementation work discussed above can be found at the GitHub repository :–

- **GitHub Link :–**
  - `https://github.com/safir-md/DevOps-for-Machine-Learning`
- **Repository Structure :–**
  - The Documentation folder contains the thesis proposal, report, and the LaTeX code of this report.
  - The cnc-mill-app directory contains the code for an automated application, implemented to exhibit the research work.

```
DevOps-for-Machine-Learning
    |-- Documentation
    |-- cnc-mill-app
    |-- readme.md
```