Enabling Continuous Improvement in Machine Learning Workflows through Data Logging and Monitoring

Swaraj Shaw

Dublin Business School

This dissertation is submitted for the degree of
Master of Science

January 2023

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Swaraj Shaw
January 2023

Acknowledgements

I am grateful to my supervisor, Professor Dr. Assem Abdelhak, for his guidance and support throughout this project. His expertise in the field of machine learning and data science was invaluable in helping me to develop and refine my research ideas. He provided valuable guidance and direction to my project, helping me to organize our work in a logical and coherent manner. He ensured providing regular feedback and support, and made sure that I was making progress towards our goals and deadlines to high standards.

I am grateful to my previous manager Mr. Rahul Gupta, I would like to thank him for his support and guidance in brainstorming the topic of my thesis. His insights and suggestions were invaluable in helping me to refine my ideas and focus on a topic that was both interesting and relevant. His guidance and support, has been instrumental in my progress and success in the field of Machine Learning and AI. I am grateful to have had such a knowledgeable and supportive mentor.

I would also like to thank my family and friends for their unwavering support and encouragement throughout this process. I am deeply grateful to my sister for her selfless sacrifice and support. Her determination and perseverance enabled me to pursue my dreams and achieve my goals. Their love and understanding made it possible for me to focus on my work and achieve this significant milestone.

Finally, I would like to acknowledge the support of the wider research community, whose contributions to the field of machine learning have made this project possible. I am grateful to have been able to build upon the work of others, and hope that my own contributions will be of value to future researchers. Thank you all for your help and support. This thesis would not have been possible without you.

Abstract

Recent advances in machine learning (ML) have made it possible to build complex, data-intensive models that can be deployed in real-world applications. However, deploying these models in production environments presents significant challenges, including the need for effective MLOps (machine learning operations) practices to manage the end-to-end lifecycle of ML models. In this thesis, we focus on the importance of data logging and monitoring in MLOps, and present a comprehensive solution based on open-source tools such as MLFlow and WhyLogs. Our solution is designed to be production-ready and can be easily integrated into existing enterprise environments, such as Microsoft Azure with Docker containers. We also address the issue of model drift, which can arise when the distribution of data used for training differs from the distribution of data used for inference. Our solution includes techniques for detecting and mitigating drift, ensuring that ML models remain accurate and reliable over time. Overall, this thesis demonstrates the importance of effective data logging and monitoring in MLOps, and presents a practical solution that can be used in real-world production environments.

Keywords : *MLOps, Data Logging, Data Monitoring, MLFlow, WhyLogs, Open Source, Enterprise, Production Ready, Machine Learning Workflows, Microsoft Azure, Docker, Drift, Minio, MySQL.*

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 The Evolution of Machine Learning and Its Adoption in Business

Machine learning has come a long way since its inception in the 1950s. Initially, machine learning algorithms were based on simple statistical models that could be trained on small datasets to make predictions. However, as data availability and computing power increased, more complex algorithms were developed, such as neural networks, which are based on the structure of the human brain. These algorithms, known as deep learning models, have achieved state-of-the-art performance in a wide range of tasks, such as image and speech recognition.

The adoption of machine learning in businesses has also increased significantly in recent years. Many companies are using machine learning to automate and optimize various business processes, such as customer service, fraud detection, and supply chain management. However, the deployment of machine learning models to production can be a time-consuming and complex process, as it requires specialized infrastructure and expertise.

To address this challenge, many companies are turning to cloud computing platforms, which provide scalable and flexible infrastructure for machine learning. These platforms allow companies to easily train and deploy machine learning models, without the need to invest in expensive hardware and software. This has greatly accelerated the adoption of machine learning in businesses, and has enabled companies to quickly and easily leverage the power of deep learning to improve their operations.

"Mastering others is strength. Mastering yourself is true power." - Lao Tzu

1.2 Advantages of Open Source Technology for Machine Learning

Open source technology refers to software that is freely available to use, modify, and distribute. In contrast, enterprise software is proprietary and typically requires a license or subscription fee. There are several benefits of using open source technology for machine learning production pipelines. First, open source software is typically freely available, which can greatly reduce the cost of using machine learning for businesses, especially small and medium-sized businesses that may not have the resources to invest in expensive proprietary software. Second, open source software is often developed and maintained by a large and active community of users and developers, which can provide access to a wealth of knowledge and expertise. This can be particularly beneficial for businesses looking to use machine learning and deep learning, as there is a wealth of open source libraries and frameworks that can be easily integrated into production pipelines. Third, open source software is typically highly customizable and flexible, which allows businesses to easily adapt it to their specific needs and requirements. This can be particularly useful for machine learning production pipelines, as they often require specialized algorithms and processes that may not be available in proprietary software. Overall, the use of open source technology can greatly benefit businesses looking to use machine learning and deep learning to improve their operations and grow their revenue. By leveraging the power of open source software, businesses can build robust recommendation systems and other machine learning applications, and improve their efficiency and competitiveness in the market.

1.3 Key Strategies adopted by Enterprises in building a robust ML Pipeline

There are several key strategies that companies use to improve the performance of deep learning models. These strategies include:

1. Data collection and pre-processing: The quality and quantity of the data used to train a deep learning model is critical to its performance. Companies often use large and diverse datasets to train their models, and carefully pre-process the data to ensure that it is clean, consistent, and representative of the task at hand.

2. Model architecture and hyperparameter optimization: The architecture and configuration of a deep learning model can greatly impact its performance. Companies

often experiment with different architectures and hyperparameters to find the best configuration for their specific use case. This can be done using techniques such as grid search, random search, and Bayesian optimization.

3. Transfer learning and model ensembling: Transfer learning involves using a pre-trained model as a starting point for training a new model, which can greatly reduce the amount of data and computation required. Model ensembling involves combining the predictions of multiple models to improve performance, which can be particularly effective for complex tasks.

4. Hardware and software optimization: The performance of deep learning models can also be improved by using specialized hardware and software designed specifically for deep learning. This can include GPUs, TPUs, and other hardware accelerators, as well as specialized deep learning frameworks and libraries. Overall, these strategies can help companies improve the performance of their deep learning models and achieve state-of-the-art results on a wide range of tasks. By carefully selecting and pre-processing data, optimizing model architecture and hyperparameters, and leveraging specialized hardware and software, companies can build highly effective deep learning models that can drive business value and growth.

1.4 Ensuring Continuous Maintenance of Deep Learning Models

MLOps (Machine Learning Operations) is a set of practices and processes that aim to improve the collaboration, automation, and integration of data science and machine learning into the software development lifecycle. It is an essential part of the development and deployment of deep learning models. Most companies often overlook the maintenance of deep learning models after deployment, which can lead to a decline in model performance over time. This can happen due to various reasons such as changes in the data distribution, new data, or changes in the business requirements. This phenomenon is known as data or concept drift, and it can significantly impact the performance of the model. In such scenarios, retraining the model with the updated data and features is crucial to maintain the performance of the model. This process is called continuous learning or continuous training, and it is an essential part of MLOps. It helps companies to keep their deep learning models up-to-date and aligned with the changing business requirements. By implementing MLOps practices, companies can ensure that their deep learning models are continuously monitored, maintained, and

retrained in production. This helps to avoid the impact of data or concept drift and maintain the performance of the model over time.

1.5 PyTorch vs TensorFlow: Choosing a Deep Learning Framework

PyTorch and TensorFlow are two popular deep learning frameworks that are used for both research and production purposes. Both frameworks have their own strengths and weaknesses, and the choice of which framework to use depends on the specific requirements of the project. In terms of production, PyTorch is often considered more suitable for deploying models in production environments because of its flexibility and ability to perform dynamic graph computations [Paszke *et al.* (2017)]. It also offers a more intuitive API and allows for easier debugging. On the other hand, TensorFlow is often considered more suitable for large-scale production environments because of its robust and scalable architecture and ability to perform efficient parallel computations [Abadi *et al.* (2016)]. In terms of research, PyTorch is often considered more suitable because of its flexibility and ability to perform dynamic graph computations, which allows for easier experimentation and prototyping. It also offers a more intuitive API and allows for easier debugging. On the other hand, TensorFlow is often considered more suitable for research because of its extensive support for various platforms and hardware, including GPUs, TPUs [Jouppi *et al.* (2017)], and distributed training. Overall, PyTorch is often considered a better choice for research and experimentation, while TensorFlow is a better choice for large-scale production environments.

Pros of PyTorch	**Cons of PyTorch**
Flexible and dynamic graph computations	Less robust and scalable compared to TensorFlow
Intuitive API and easy debugging	Limited support for distributed training
Suitable for research and experimentation	
Pros of TensorFlow	**Cons of TensorFlow**
Robust and scalable architecture	More complex API compared to PyTorch
Extensive support for various platforms and hardware	Not as flexible and dynamic as PyTorch
Suitable for large-scale production environments	Less suitable for research and experimentation

Table 1.1 Pros and Cons of PyTorch and TensorFlow Frameworks

1.6 AI and Deep Learning for News Classification

News classification is an important task for several reasons. First, it allows for the efficient organization and distribution of news content by automatically assigning each news article

to one or more categories or topics [Wang *et al.* (2020)]. This makes it easier for readers to find articles of interest and for news organizations to distribute content to the appropriate channels. Second, news classification can be used to identify and filter out hate speech and other forms of offensive or inappropriate content [Ali *et al.* (2022)]. By training a news classification model to recognize hate speech, it can be used to automatically flag and remove such content, helping to create a safer and more inclusive online environment. Third, news classification can be used to identify and track trending news topics in real-time [Vyas (2022)]. By continuously classifying news articles and tracking the frequency of each category, it is possible to identify which topics are currently receiving the most attention and highlight them to readers. Finally, news classification can be used to automatically tag entities in news articles, such as people, places, and organizations [Hung and Huy (2022)]. This can be useful for a variety of applications, such as automatically generating links to relevant information about the entities mentioned in an article or for conducting entity-level analysis of news content. Overall, news classification using AI and deep learning models like BERT can provide many benefits for both news organizations and readers. It can help to organize and distribute news content, identify and filter out hate speech, track trending topics, and automatically tag entities in articles.

1.7 Benefits of Cloud-Based Machine Learning Infrastructure

One of the main drawbacks of on-premises machine learning infrastructure is the high upfront cost of purchasing and maintaining the hardware and software required for training and deploying machine learning models. This can be a significant barrier for companies, especially small and medium-sized businesses, that may not have the resources to invest in such infrastructure.

In addition, on-premises infrastructure can be inflexible and difficult to scale, which can hinder the development and deployment of machine learning models. As data volumes and complexity increase, companies may need to constantly upgrade and expand their on-premises infrastructure, which can be time-consuming and expensive.

In contrast, cloud-based machine learning platforms provide a scalable and flexible infrastructure for training and deploying machine learning models. These platforms allow companies to easily access and use the latest hardware and software for machine learning, without the need to invest in and maintain their own infrastructure. This can greatly reduce

the cost and complexity of using machine learning, and allows companies to quickly and easily scale their machine learning capabilities.

Moreover, cloud-based platforms provide access to large and diverse datasets, which are essential for training effective machine learning models. This allows companies to leverage the latest advances in machine learning and deep learning, without having to spend time and resources collecting and curating their own data.

In summary, while on-premises infrastructure can provide some control and security, it can also be costly and inflexible. In contrast, cloud-based platforms offer a more cost-effective and scalable solution for companies looking to use machine learning and deep learning to improve their operations.

1.8 Factors that Contribute to Deep Learning Model Failure in Production

There are several reasons why deep learning models can fail in production over time. One important factor to consider is **concept drift**, which refers to the phenomenon where the statistical properties of the data that a model was trained on change over time. This can occur due to various external factors such as changes in the underlying data distribution, shifts in the business environment, or changes in the way the data is collected or processed. When concept drift occurs, the model's performance may degrade because it is no longer able to accurately predict outcomes based on the changed data distribution.

Another factor that can contribute to model failure is **overfitting**, which occurs when a model is overly complex and has too many parameters relative to the amount of training data. Overfitted models tend to perform well on the training data but may fail to generalize to unseen data, leading to poor performance in production.

Another common reason for model failure is a lack of **robustness**, which refers to the ability of a model to perform well in the presence of various types of noise or perturbations in the data. Models that are not robust may perform poorly when exposed to such variations, leading to poor performance in production.

There are also various external factors that can contribute to the failure of deep learning models in production. For example, if the data used to train the model is of **poor quality** or is biased in some way, the model's performance may be compromised. Additionally, if the model is not properly integrated into the production environment or is not properly maintained over time, it may become less effective as changes occur in the system. Finally, it is important to note that even well-designed and trained models may fail in production

due to various unforeseen circumstances. For example, a model may perform poorly if it is exposed to data that it was not designed to handle, or if it is deployed in an environment with unexpected constraints or requirements.

In conclusion, *there are several factors that can contribute to the failure of deep learning models in production over time, including concept drift, overfitting, lack of robustness, poor quality or biased data, improper integration or maintenance, and unforeseen circumstances.*

To mitigate these risks and ensure the long-term success of deep learning models in production, it is important to carefully design and train the models, pay attention to the quality and diversity of the training data, regularly monitor and maintain the models in production, and be prepared to adapt to changes in the environment as needed.

1.9 Data Logging and Monitoring

Data logging and drift monitoring are important practices for ensuring the performance and stability of machine learning models in production. Data logging refers to the process of collecting and storing data about the model's performance, input data, and the environment in which it is running. This data can be used to monitor the model's performance over time, detect changes in the data or environment that may affect the model's performance, and identify issues or errors that may occur during model operation. Drift monitoring, on the other hand, refers specifically to the process of detecting and tracking changes in the statistical properties of the data that a model was trained on. This is important because such changes, known as concept drift, can lead to a decline in the model's performance over time. By regularly monitoring for drift, organizations can identify when a model's performance may be compromised due to changes in the data distribution, and take appropriate corrective action to address the issue. According to [Webb *et al.* (2016)] there are several different types of **concept drift** that can occur, including:

1. **Sudden drift:** This type of drift refers to a sudden and significant shift in the data distribution that occurs over a relatively short period of time. This type of drift can be difficult to detect and can have a significant impact on model performance if not addressed promptly.

2. **Incremental drift:** This type of drift refers to a gradual and ongoing shift in the data distribution that occurs over a longer period of time. This type of drift can be easier to detect than sudden drift, but can still have a significant impact on model performance if left unmitigated.

3. **Covariate shift:** This type of drift refers to a change in the distribution of the input features of the data, rather than the output labels. This can occur when the data collection process changes, or when the data is collected from different sources or populations.

4. **Prior probability shift:** This type of drift refers to a change in the relative frequency of different classes in the data. For example, if a model was trained on data where the positive class occurred 80

5. **Non-stationary drift:** This type of drift refers to a dynamic or unpredictable change in the data distribution, such as a change in the underlying business or environmental factors that the model is being used to predict.

Effective drift monitoring requires a ***combination of data logging and appropriate drift detection algorithms***. There are several different approaches to detecting drift, including statistical tests, distance-based methods, and change point detection algorithms. The choice of approach will depend on the specific characteristics of the data and the needs of the organization. In conclusion, data logging and drift monitoring are important practices for ensuring the performance and stability of machine learning models in production. There are several different types of concept drift that can occur, including sudden drift, incremental drift, covariate shift, prior probability shift, and non-stationary drift. Effective drift monitoring requires a combination of data logging and appropriate drift detection algorithms to identify and address changes in the data distribution in a timely manner.

Chapter 2

Literature Review

2.1 Introduction

Machine learning (ML) has become a widely used approach for building complex models that can be deployed in real-world applications. However, deploying these models in production environments presents significant challenges, including the need for effective MLOps (machine learning operations) practices to manage the end-to-end lifecycle of ML models. In this literature review, we will explore the importance of data logging and monitoring in MLOps, and present a comprehensive solution based on open-source tools such as MLFlow and WhyLogs. We will also address the issue of model drift, which can arise when the distribution of data used for training differs from the distribution of data used for inference. Our solution will be designed to be production-ready and can be easily integrated into existing enterprise environments, such as Microsoft Azure with Docker containers.

2.2 Data Logging and Monitoring in MLOps

Effective data logging and monitoring is crucial for MLOps, as it enables organizations to track and optimize the performance of their ML models over time. Data logging refers to the process of collecting and storing data related to the training, testing, and deployment of ML models, while data monitoring involves analyzing and interpreting this data to identify any issues or trends that may impact the performance of the model [Renggli *et al.* (2021)]. There are several best practices and tools that organizations can use to implement effective data logging and monitoring for their ML systems.

2.3 MLFlow and WhyLogs

MLFlow is an open-source platform for managing the end-to-end ML lifecycle, including experimentation, reproducibility, deployment, and management [Zaharia *et al.* (2018)]. It provides a centralized platform for tracking and organizing ML experiments, as well as tools for deploying and managing ML models in production environments. WhyLogs is another open-source platform for data monitoring and debugging of ML systems, designed to scale to large and complex ML pipelines [Whylabs (2019)]. It provides tools for collecting, storing, and analyzing data related to ML pipelines, as well as features for detecting and mitigating issues that may impact the performance of the model.

2.4 Enterprise Machine Learning Workflows

There are several challenges and best practices that organizations should consider when building and deploying ML models in the enterprise. These include issues related to scalability, reproducibility, and governance. Scalability refers to the ability of the ML system to handle increasing amounts of data and traffic, while reproducibility refers to the ability to replicate the results of an ML experiment or deployment. Governance involves the policies and procedures that organizations put in place to ensure the ethical and responsible use of ML models.

2.5 Production-Ready Machine Learning

Building and deploying ML models in production environments requires a thorough understanding of the approaches and techniques that are available. A survey of approaches and techniques for production-ready ML was conducted, which identified several key considerations, including model selection, deployment strategies, and monitoring and maintenance [Ruf *et al.* (2021)]. Model selection involves choosing the most appropriate ML model for a given task, while deployment strategies involve deciding how and where to deploy the model, such as on-premises or in the cloud. Monitoring and maintenance involves tracking the performance of the model over time and taking corrective action if necessary.

2.6 Microsoft Azure Machine Learning

Microsoft Azure Machine Learning is a cloud-based platform that provides a comprehensive set of tools and services for building, deploying, and managing ML models at scale. It

offers a range of capabilities, including data preparation, model training and evaluation, and deployment, as well as tools for monitoring and managing ML models in production environments.

2.7 Docker for Machine Learning

Docker is a containerization platform that allows organizations to package and deploy applications and services in a standardized and portable way. It can be particularly useful for ML development and deployment, as it allows organizations to easily set up and manage reproducible ML environments. Best practices for using Docker for ML include considerations for containerization, scalability, and reproducibility [Xu *et al.* (2017) & Mao *et al.* (2020)].

2.8 Drift Detection in Machine Learning Models

Model drift refers to the phenomenon of changes in the distribution of the data used to train an ML model, which can impact the accuracy and reliability of the model over time. Drift detection refers to the process of detecting these changes and taking corrective action to mitigate their impact on the model. A survey of drift detection techniques was conducted, which identified several approaches and techniques that organizations can use to detect and mitigate model drift.

2.9 Minio

Minio is an open-source object storage server designed for ML workflows (Minio Inc.,). It provides features such as data durability, scalability, and security, which are important considerations for ML systems that handle large amounts of data [Minio (2016)]. MySQL Databases in Machine Learning Workflows: MySQL is a widely used database management system that can be used in ML workflows (MariaDB Foundation, n.d.). It offers features such as performance, scalability, and data management, which can be useful for ML systems that require fast access to large amounts of data. However, there are also challenges that organizations should consider when using MySQL in ML workflows, including issues related to data management, performance, and scalability.

2.10 Open Source Machine Learning

Open source tools and frameworks offer several benefits for ML development and deployment, including access to a wide range of resources, flexibility, and cost-effectiveness [Nguyen *et al.* (2019)]. However, there are also challenges that organizations should consider when adopting open source ML solutions, including issues related to adoption, integration, and support.

2.11 MLOps in the Cloud

Implementing MLOps in the cloud can provide organizations with several benefits, including scalability, security, and cost-effectiveness. However, there are also challenges that organizations should consider when implementing MLOps in the cloud, including issues related to resource management, security, and compliance.

2.12 Conclusion

In this literature review, we have explored the importance of data logging and monitoring in MLOps, and presented a comprehensive solution based on open-source tools such as MLFlow and WhyLogs. We have also addressed the issue of model drift and discussed the considerations and challenges involved in building and deploying ML models in the enterprise and in production environments. Overall, this review demonstrates the importance of effective data logging and monitoring in MLOps, and presents a practical solution that can be used in real-world production environments.

Chapter 3

State of the Art

3.1 Self-Attention in Transformers: A Deep Learning Revolution

Transformers are a type of deep learning architecture that uses self-attention mechanisms to model the interactions between different input tokens. They were introduced in the paper "Attention is All You Need" Vaswani *et al.* (2017) and have become widely used in natural language processing tasks, such as machine translation and text classification. In a transformer model, the input tokens are first passed through an embedding layer, where each token is represented as a dense vector. These embedded tokens are then fed into the transformer architecture, which consists of multiple layers of self-attention and feed-forward networks. The self-attention layers allow the model to attend to different parts of the input sequence, based on the relationships between the tokens. This enables the model to capture the dependencies between the tokens, and encode them into the contextualized token representations. For example, in a machine translation task, a transformer model may use self-attention to identify the subject and the verb in a sentence, and encode their relationship into the contextualized token representations. This information can then be used by the feed-forward layers to generate the translated sentence. Transformers are a powerful deep learning architecture that uses self-attention mechanisms to model the interactions between different input tokens, and enable the model to capture the dependencies between them.

3.2 Understanding BERT: A State-of-the-Art Natural Language Processing Model

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing model developed by Google Research in 2018. It is designed to pre-train deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers. BERT has been widely used for various natural language processing tasks such as text classification, question answering, and language translation, and has achieved state-of-the-art performance on many benchmarks.

The BERT algorithm is trained on large amounts of unlabelled text data and uses a self-supervised learning approach to learn the underlying structure of the language. It achieves this by using a series of masked language modelling and next sentence prediction tasks, where a portion of the input text is masked and the model must predict the missing words based on the surrounding context. Mathematically, the BERT algorithm can be represented by the following formula:

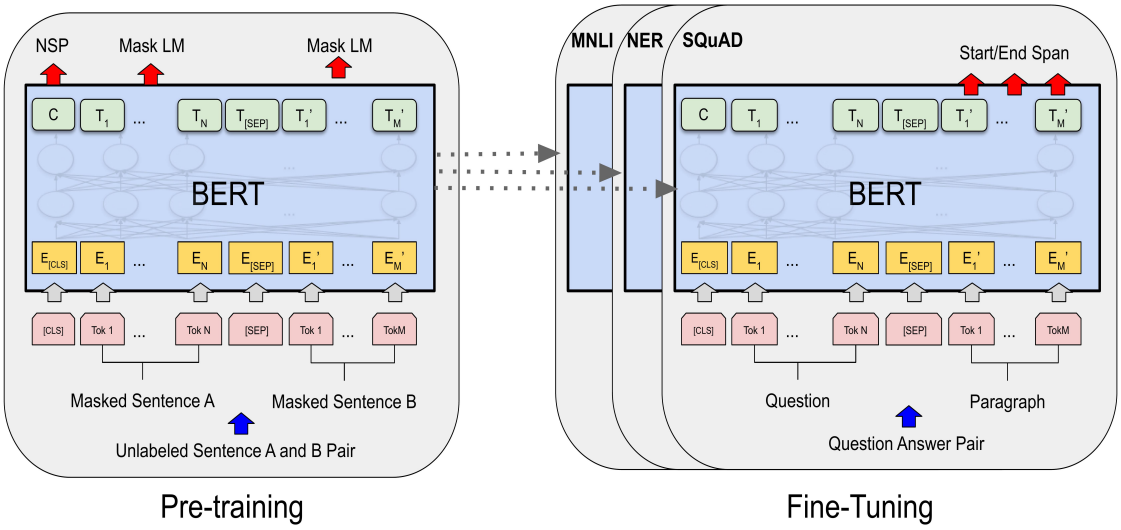
$$\textbf{BERT}(x) = \textbf{Transformer}(x, W) \tag{3.1}$$

Where x is the input text, W is the set of model parameters (e.g. weights and biases), and **Transformer** is the transformer-based architecture used by BERT. The transformer architecture consists of a series of encoder and decoder layers, each of which contains multiple attention heads. The encoder layers are used to encode the input text and the decoder layers are used to predict the masked words or the next sentence in the input text. The attention mechanism in BERT is a key component of the algorithm, allowing it to consider the context of a given input word by attending to different parts of the input text. This is represented mathematically by the following formula:

$$\textbf{Attention}(Q, K, V) = \textbf{softmax}(QK^T / \sqrt{d_k})V \tag{3.2}$$

Where Q, K, and V are the query, key, and value matrices, respectively, and d_k is the dimension of the keys. BERT algorithm uses a transformer-based architecture and attention mechanisms to learn the underlying structure of the language in a self-supervised manner, allowing it to perform well on a variety of natural language processing tasks.

3.3 BERT Architecture



Devlin *et al.* (2018)

Fig. 3.1 Understanding BERT: A State-of-the-Art Natural Language Processing Model

1. The model uses a transformer architecture, which is composed of multiple layers of self-attention and feed-forward neural networks. The input to the model is a sequence of tokens, such as words or sub-words in a sentence. These tokens are passed through an embedding layer, where each token is represented as a dense vector.

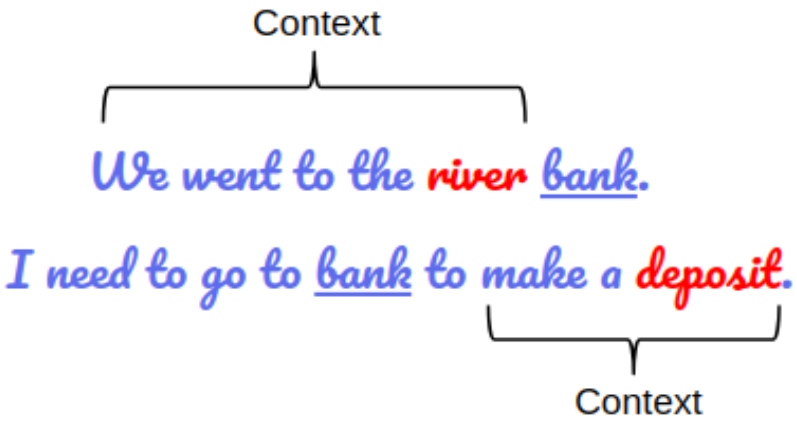
2. The embedded tokens are then fed into the transformer architecture, which consists of multiple layers of self-attention and feed-forward networks. The self-attention layers allow the model to attend to different parts of the input sequence, while the feed-forward layers enable the model to capture more complex relationships between the input tokens.

3. Within each transformer layer, the model calculates self-attention weights for each token, which determine how much attention the model should pay to that token. These weights are then used to calculate a weighted sum of the input tokens, which is used as the input to the feed-forward network.

4. The output of the transformer layers is a sequence of contextualized token representations, which capture the meaning of each token within the context of the entire input sequence.

5. Finally, the contextualized token representations are passed through a classification layer, which uses them to make predictions about the input sequence, such as identifying the named entities or the sentiment of a sentence.

Overall, the BERT architecture allows the model to capture the relationships between different tokens in the input sequence, which enables it to perform natural language processing tasks with high accuracy.



Devlin *et al.* (2018)

Fig. 3.2 BERT utilizing bidirectional context in NLP

3.4 Comparing BERT Uncased and Cased Models

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing model developed by Google Research. BERT has two versions: BERT uncased and BERT cased. BERT uncased refers to the version of BERT that has been pre-trained on lowercase text only. This means that the model has only been exposed to text where all the words are in lowercase, and it has not been trained on text where some words are in uppercase. As a result, BERT uncased may perform worse on tasks that require the model to differentiate between uppercase and lowercase words, such as named entity recognition or part-of-speech tagging. BERT cased, on the other hand, refers to the version of BERT that has been pre-trained on both lowercase and uppercase text. This means that the model has been exposed to text where some words are in uppercase and some are in lowercase, and it has been trained to differentiate between these cases. As a result, BERT

cased may perform better on tasks that require this ability, such as named entity recognition or part-of-speech tagging. Overall, the choice between BERT uncased and BERT cased depends on the specific natural language processing task at hand. If the task does not require the model to differentiate between uppercase and lowercase words, then BERT uncased may be a good choice. However, if the task does require this ability, then BERT cased may be a better choice.

3.5 Fine-tuning BERT for News Classification

News classification is the task of assigning a news article to one or more categories or topics based on its content. For example, a news article about a new scientific discovery could be classified as science, technology, or health news. BERT can be used for news classification by fine-tuning the pre-trained model on a labelled dataset of news articles and their corresponding categories [Jing and Bailong (2021), Shishah (2021) & Kiela *et al.* (2020)]. To fine-tune BERT for news classification, the first step is to prepare the dataset. This involves extracting the text of each news article and its corresponding categories or topics. The dataset should also be divided into training and testing sets, with the training set used to fine-tune the pre-trained BERT model and the test set used to evaluate the performance of the fine-tuned model. Next, the fine-tuned BERT model can be trained on the training set by adjusting the model's parameters to better predict the categories of the news articles. This is done by feeding the articles and their corresponding categories into the model and using an optimization algorithm, such as stochastic gradient descent, to adjust the model's parameters to minimize the classification error on the training set. Once the fine-tuned BERT model is trained, it can be used to predict the categories of new, unseen news articles. This is done by feeding the text of the news article into the model and using the trained model to predict the most likely categories for the article. The performance of the model can be evaluated on the test set by comparing the predicted categories with the true categories and calculating a performance metric, such as accuracy or F1 score. Overall, BERT can be a powerful tool for news classification tasks, allowing for accurate and efficient classification of news articles into their corresponding categories.

3.6 Managing Machine Learning Lifecycle with MLFlow

MLFlow is an open-source platform for managing the end-to-end machine learning lifecycle. It is designed to help data scientists and machine learning engineers track and manage

their machine learning experiments, model training and deployment [Zaharia *et al.* (2018)]. MLFlow consists of three main components:

1. **MLFlow Tracking:** This component allows data scientists to track and log their machine learning experiments, including the hyperparameters, metrics and artifacts generated during training. This enables them to compare and reproduce different experiments, and monitor the performance of their models over time.

2. **MLFlow Projects:** This component enables data scientists to package and share their machine learning code and dependencies as reproducible projects. This makes it easier to collaborate and reproduce experiments across different teams and environments.

3. **MLFlow Models:** This component allows data scientists to package their trained machine learning models as artifacts, which can be deployed to a variety of platforms, such as cloud services or on-premises servers. It also provides an API for making predictions using the deployed models.

3.7 Tracking and Managing ML Experiments with MLFlow

In MLFlow, a **run** is a single execution of a machine learning **experiment**, which can be tracked and logged using the MLFlow Tracking component. A run typically consists of multiple stages, such as data pre-processing, model training and evaluation, and can be initiated either manually or through an MLFlow Project.

At each stage of a run, data scientists can log various types of data, such as metrics, hyperparameters, and artifacts, using the **MLFlow API**. This data is then stored in a centralized tracking server, which enables data scientists to compare and reproduce different runs, and monitor the performance of their models over time.

For example, a data scientist may log the hyperparameters used for training a machine learning model, such as the learning rate and the number of hidden layers, along with the evaluation metrics, such as the **accuracy** and the **f1-score**. This data can then be used to compare the performance of different model configurations and select the best one for deployment.

The ability to log data at each stage of a run in MLFlow helps data scientists to track and manage their machine learning experiments, and enables them to make more informed decisions about their model training and deployment.

3.8 Data Management with PyTorch's Lightning Data Module

The Lightning data module in PyTorch is a high-level API that simplifies the process of loading, transforming, and training machine learning models with PyTorch [LightningAI *et al.* (2018)]. It provides a set of classes and functions that enable data scientists to easily define and manage their machine learning data pipelines, and focus on building and training their models. The Lightning data module provides several key features, such as:

1. **Data loading and pre-processing:** The Lightning data module provides classes and functions for loading and pre-processing data from various sources, such as CSV files, image directories, or PyTorch datasets. It also enables data scientists to define custom data transformations, such as image augmentation or text tokenization.

2. **Data loaders and samplers:** The Lightning data module provides classes for creating PyTorch data loaders, which are used to feed data into the model during training. It also provides samplers, which allow data scientists to control how data is sampled from the dataset, such as using a random or balanced sampling strategy.

3. **Batching and padding:** The Lightning data module provides classes and functions for batching and padding data, which are necessary for training deep learning models with variable-length inputs. It also enables data scientists to define custom padding strategies, such as using the maximum or average length of a batch.

Lightning data module in PyTorch simplifies the process of managing and training machine learning models with PyTorch, and enables data scientists to focus on the core aspects of their model development.

Chapter 4

Problem Statement

4.1 Aim & Objective

It is not uncommon for companies to deploy deep learning models without adequately considering the maintenance and optimization of these models over time. This can result in model degradation, which can ultimately lead to decreased accuracy and effectiveness of the model. There are several factors that can contribute to model degradation, such as changes in the data distribution, concept drift, or changes in the business context.

The objective of this project is to develop a production-ready solution for the optimization and performance of deep learning models in the enterprise environment. The solution aims to address the problem of model degradation over time due to various internal and external factors such as model drift, data drift, concept drift, or changes in business scope.

To achieve this objective, the project involves implementing a data logging and monitoring system using open source tools such as MLFlow and WhyLogs. The data logging system will ensure proper logging of data parameters and metrics, while the monitoring system will analyze the logged data for anomalies or drifts that may occur over time.

In addition to data logging and monitoring, the project also involves deploying the solution in a Docker container hosted on a Microsoft Azure virtual machine. This will enable the solution to be easily integrated into the production environment, making it more accessible to enterprise users.

Overall, the goal of this project is to improve the optimization and performance of deep learning models in the enterprise environment, by enabling data logging, monitoring, and retraining of the models as needed. By utilizing open source tools and a Docker-based deployment on Azure, the solution is designed to be both scalable and flexible, making it suitable for a wide range of machine learning workflows in the enterprise environment.

4.2 Scope & Limitations

4.2.1 Scope of the Research

Clarifying the specific benefits and outcomes that the project aims to achieve for a well defined scope and purpose of the project. In the context of this project, some potential benefits or outcomes might include:

- **Improved model accuracy:** By implementing data logging and monitoring, and regularly retraining the model as needed, the solution aims to improve the accuracy of the deep learning models in the enterprise environment. This could potentially result in better decision-making and data-driven recommendations, leading to increased revenue for the company.
- **Reduced time spent on model maintenance:** By automating the process of data logging, monitoring, and retraining, the solution aims to reduce the time and resources spent on manual model maintenance. This could free up data scientists to focus on other tasks, such as developing new models or exploring new data sources.
- **Enhanced scalability and flexibility:** By deploying the solution in a Docker container on Azure, the solution aims to be scalable and flexible, able to adapt to changing business needs and requirements. This could allow the solution to be used in a wide range of machine learning workflows, improving the efficiency and effectiveness of deep learning models in the enterprise environment.

4.2.2 Limitations

1. The solution is limited in its applicability to certain machine learning and deep learning models that have been tested within the constraints of the project. It may not be universally applicable to all types of machine learning or deep learning models.
2. The solution is dependent on the accuracy and reliability of the data logged and monitored by MLFlow and WhyLogs. If the data is not properly logged or is unreliable, the solution may not be effective.
3. The solution is only applicable in a monitored academic/enterprise environment, and may not be suitable for certain types of organizations or environments.
4. The solution is dependent on the scalability and flexibility of the Docker container and Azure virtual machine. If these components are not able to handle the workload

or adapt to changing needs, the solution may be limited in its ability to optimize and improve deep learning models.

5. The solution may not be able to address all potential causes of model degradation, such as changes in business strategy or unforeseen external events.

6. The solution may require significant resources and expertise to implement and maintain, which may not be feasible for all companies.

7. The solution may be subject to the limitations and constraints of the open source tools used, such as MLFlow and WhyLogs.

8. The solution may be affected by external factors such as network connectivity or hardware failures, which could disrupt the data logging and monitoring process.

9. The solution may not be able to fully automate the process of model maintenance, and may require some manual intervention or oversight from data scientists.

10. The solution may not be able to guarantee improved model accuracy or performance, as this will depend on various factors such as the quality and diversity of the data used to train the model.

11. The solution is intended to provide important insights based on data logging and monitoring metrics, but it should not be taken as a replacement for model drift algorithms or the role of a data scientist in model development. It is meant to be used in addition to these tools and processes, rather than as a replacement.

12. The solution may be limited to the specific versions of tools and libraries used in the project. Any updates or changes to these tools and libraries may affect the compatibility and functionality of the solution.

Chapter 5

Implementation

5.1 Architecture Diagram

The architecture described in the block diagram outlines the process of solving a business usecase using machine learning. It involves defining success criteria, collecting and preparing data, training a model using BERT, and monitoring progress through data logging and a dashboard.

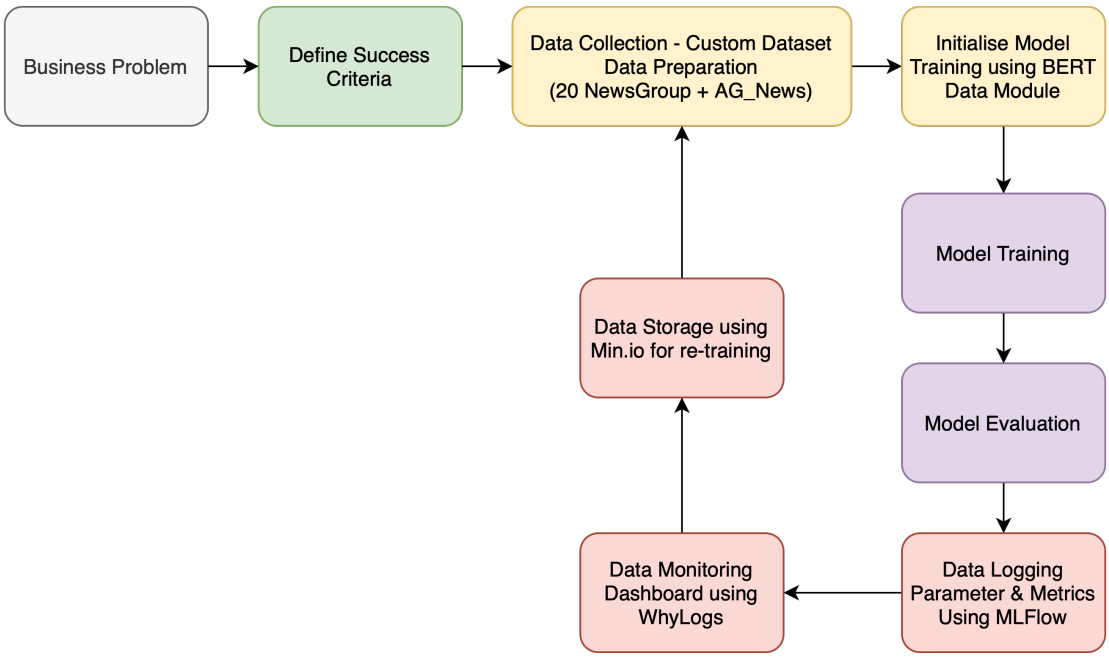


Fig. 5.1 Block diagram of the ML Workflow for solving a business problem through data collection, preparation, and training of a machine learning model using BERT and monitoring with a dashboard.

5.2 Pipeline Explanation

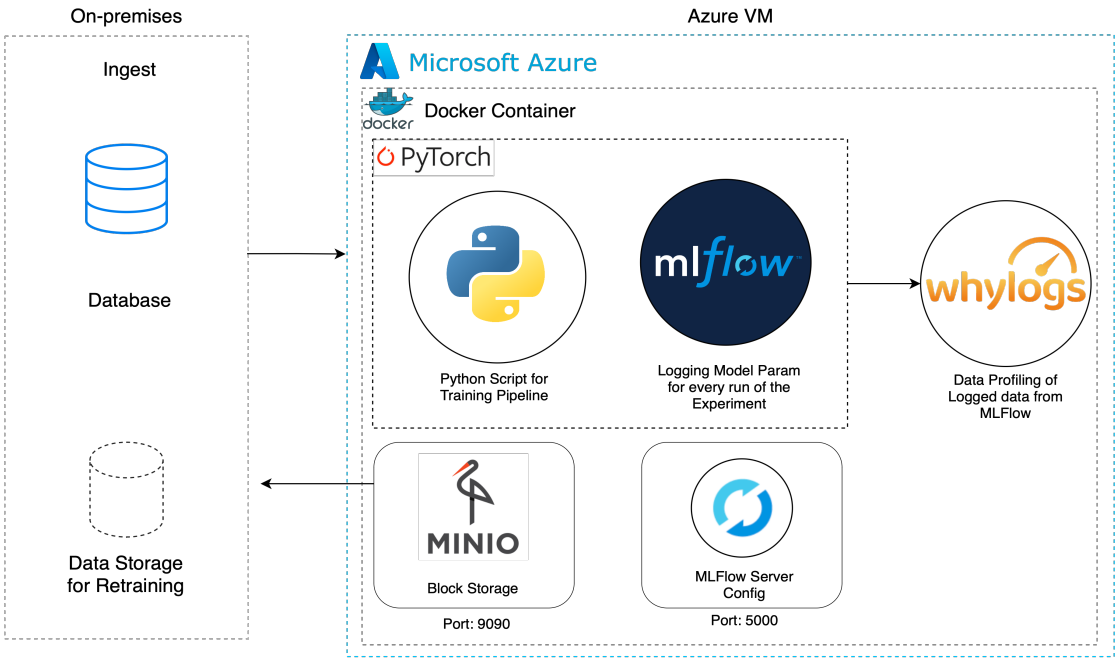


Fig. 5.2 Proposed pipeline architecture with deployment in Docker Container on Azure VM

The code is an example of a natural language processing (NLP) pipeline that uses PyTorch Lightning to train a BERT model on two different datasets: 20 Newsgroups and AG News. The pipeline has the following steps:

1. Import the required libraries, including PyTorch Lightning, BERT, and PyTorch.
2. Define a function **get_20newsgroups()** that fetches the **20 Newsgroups** dataset, selects a subset of categories, and returns a Pandas DataFrame with the text data and labels.
3. Define a function **get_ag_news()** that reads the AG News dataset from a file and returns a Pandas DataFrame with the text data and labels.
4. Define a class **NewsDataset** that extends the IterDataPipe class from the **torchdata** library. This class defines an iterator that tokenizes the input text data, pads it to a specified maximum length, and converts it to PyTorch tensors.
5. Define a PyTorch Lightning Module class **TextClassification** that represents the BERT model. This class contains a **forward()** method that applies the BERT model to the input data, followed by a linear layer and a softmax function to predict the class probabilities.

6. Define a function train() that sets up the PyTorch Lightning trainer, specifies the training and validation datasets, and trains the BERT model.

7. Define a function main() that parses the command-line arguments, sets up the BERT tokenizer, creates the training and validation datasets, and trains the BERT model.

8. To summarize, the code defines a pipeline for training a BERT model on text classification tasks using PyTorch Lightning.

5.3 Logging & Monitoring using MLFlow and WhyLogs

1. Install the MLFlow and WhyLogs packages using pip:

   - **pip install MLFlow**
   - **pip install WhyLogs**

2. Import the MLFlow and WhyLogs modules in the code:

   - **import mlflow.pytorch**
   - **import whylogs as why**

3. Initialize a new mlflow experiment:

   - **mlflow.create_experiment("<experiment_name>")**

4. Start a new mlflow run:

   - **mlflow.start_run()**

5. Log the hyperparameters used for training the model:

   - **mlflow.log_param("<parameter_name>", "<parameter_value>")**

6. Use whylogs to capture the training data and log it in mlflow:

   - Initialize a new whylogs context: **whyctx = why.Context()**
   - Create a new whylogs dataset: **ds = why.Dataset(data=<training_data>)**
   - Add the dataset to the whylogs context: **whyctx.add_dataset(ds)**
   - Generate a summary of the training data: **summary = whyctx.summary()**
   - Log the summary in mlflow: **mlflow.log_metrics(summary.metrics)**

7. Use mlflow to log the model artifacts (e.g. trained model weights, training logs, etc.):
    - **mlflow.pytorch.log_model(model, "<model_name>")**
8. End the mlflow run:
    - **mlflow.end_run()**

This methodology allows for efficient tracking and logging of model information using mlflow, which can be accessed and analyzed later for model evaluation and improvement.

5.4 Dataset

The code uses two datasets for training the model - **20newsgroups** and **ag_news**.

5.4.1 20NewsGroups

The 20newsgroups dataset is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The dataset was originally collected and distributed by [Ken Lang (1995)], a computer scientist at ATT Bell Laboratories, in the early 1990s as a way to help researchers study machine learning algorithms. The newsgroups included in the dataset cover a wide range of topics, from computer science and politics to religion and sports. Each document in the dataset is labelled with the newsgroup it belongs to, making it a useful dataset for training and testing classification algorithms. The 20newsgroups dataset has been widely used in the field of natural language processing, particularly in the development of text classification algorithms. It has also been used in other areas of machine learning research, such as topic modelling and information retrieval. In addition to the text of the newsgroup documents, the 20newsgroups dataset also includes metadata such as the date the document was posted and the author's name. This metadata can be useful for further analysis, such as studying the evolution of newsgroup discussions over time. One of the unique characteristics of the 20newsgroups dataset is that it contains a mix of both long and short documents, as well as a mix of both well-written and poorly-written texts. This makes it a challenging dataset for natural language processing algorithms, as it requires the ability to handle a wide range of text types and styles. The 20newsgroups dataset has been used in numerous academic research papers and is considered a benchmark dataset in the field of natural language processing. It continues to be a valuable resource for researchers studying text classification and other related topics.

5.4.2 AG News

The AG News dataset is a collection of over one million news articles in the domain of sports, business, science, and entertainment [Zhang *et al.* (2015)]. It is commonly used for text classification tasks, such as identifying the topic or sentiment of a given article. The dataset is split into four distinct classes, each corresponding to a specific topic: sports, business, science, and entertainment. Within each class, the articles are further divided into training and test sets, with the training set containing 750,000 articles and the test set containing 20,000 articles. Each article in the dataset is represented by a title, a description, and a set of labels indicating the article's topic and sentiment. The labels are binary, with 1 indicating a positive sentiment and 0 indicating a negative sentiment. The dataset is widely used in the field of natural language processing, particularly in the area of text classification. Researchers have used the AG News dataset to train and evaluate machine learning models for tasks such as sentiment analysis, topic classification, and language modelling. The AG News dataset is well-suited for these tasks due to its large size and well-defined structure. The inclusion of both a title and a description for each article allows for a more comprehensive representation of the content, and the use of binary labels for sentiment allows for straightforward evaluation of model performance. AG News dataset is a valuable resource for researchers studying text classification and natural language processing. Its large size and well-defined structure make it an ideal dataset for training and evaluating machine learning models in these domains.

- 20newsgroups is a dataset of newsgroup posts on a variety of topics, including atheism and religion. It is provided by the sklearn library, and can be downloaded using the **fetch_20newsgroups** function. The dataset contains text data and corresponding labels for each newsgroup post.

- **ag_news** is a dataset of news articles from the AG's Corpus of news articles on the web. It is provided by the torchtext library, and can be downloaded using the **AG_NEWS** function. The dataset contains text data and corresponding labels for each news article. The labels are in the range of 1-4, representing the four categories in the dataset - **World, Sports, Business, and Sci/Tech**.

Both datasets are used to train the model for text classification, with the 20newsgroups dataset being used by default. The user can specify the dataset type in the dataset parameter of the NewsDataset class.

5.5 Azure Docker Deployment

5.5.1 Azure Virtual Machine Configuration

Here are the steps to create a virtual machine (VM) in Azure:

1. Go to the Azure portal (https://portal.azure.com/) and sign in with your Azure account.
2. Click the "Create a resource" button in the top left corner.

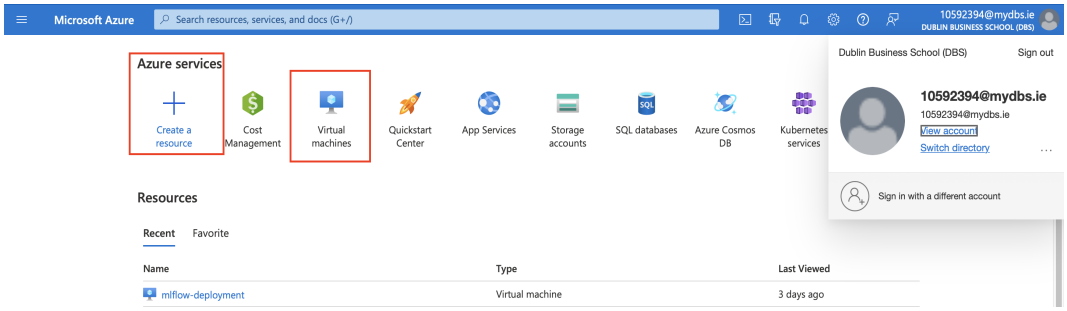


Fig. 5.3 Create a resource on Azure VM

3. In the "Search the Marketplace" field, type "Ubuntu Server" and select the first result.

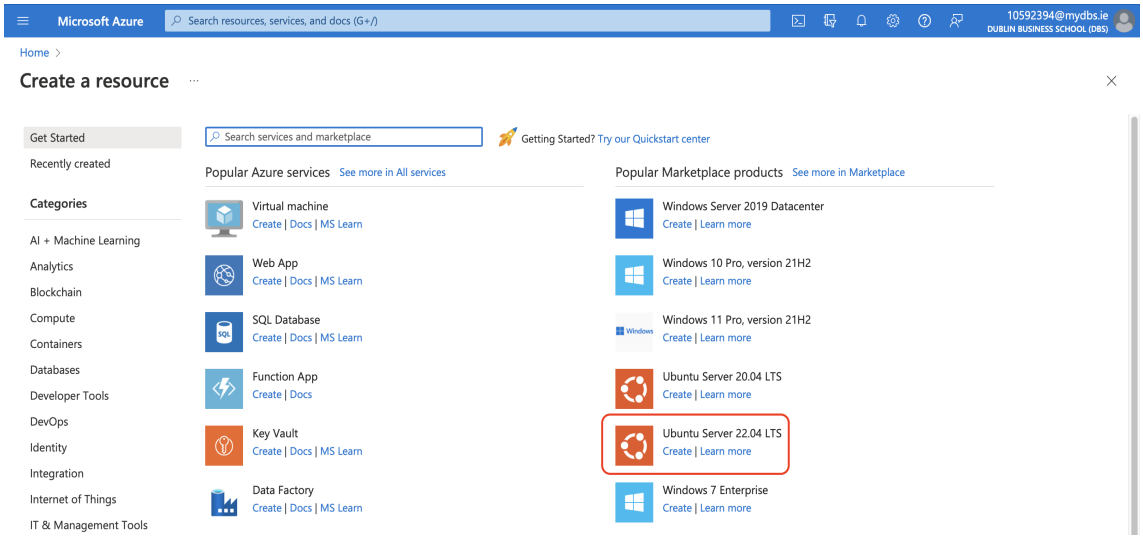


Fig. 5.4 Create a Ubuntu Server

4. On the "Create a virtual machine" page, select the options for the VM you want to create, such as the region, size, and resource group.

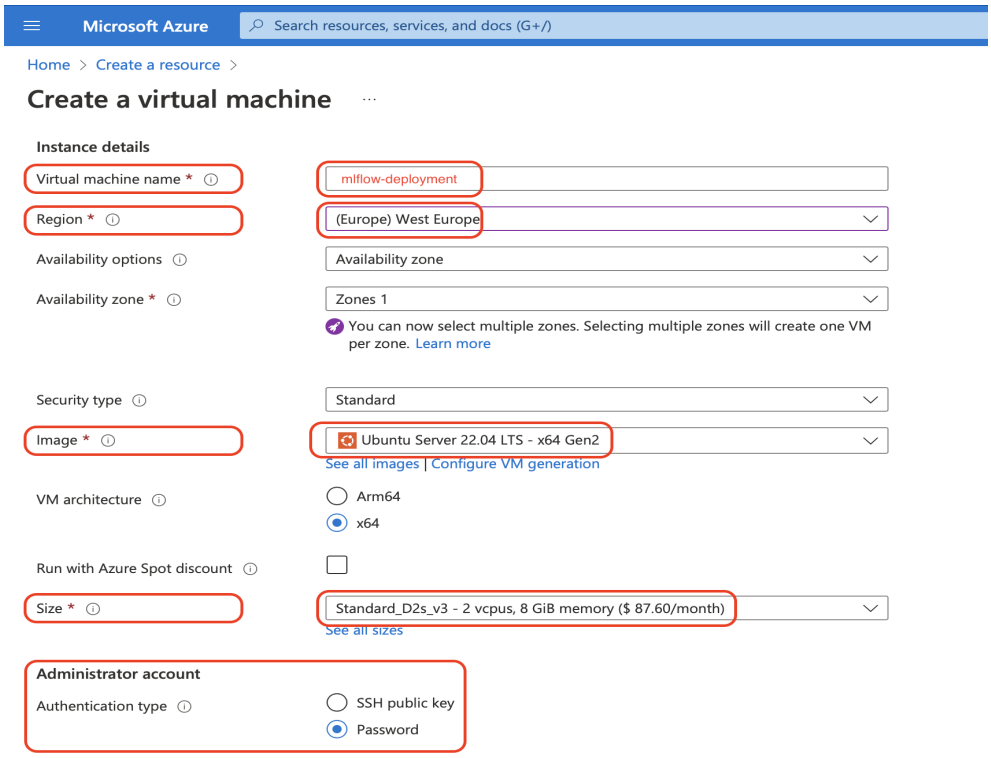


Fig. 5.5 Virtual Machine Configuration Details

5. On the "Settings" page, configure the options for the VM, such as the administrator username and password, which will be used to login later in the terminal session.

6. On the "Summary" page, review the options you have selected and click the "Create" button to create the VM.

7. Once the VM has been created, click the "Go to resource" button to open the VM's page.

8. On the VM's page, click the "Add inbound port rule" button at the top of the page.

9. In the "Add inbound port rule" blade that appears, enter the name and description for the rule.

10. Select the protocol (TCP or UDP) and the port range for the rule (TCP in our use case). You can specify a single port or a range of ports.

11. Specify the destination port range specifically for **S3_Bucket_Minio, MLFlow_UI**, **AllowAnyCustom9090Inbound** as per the screenshot below.

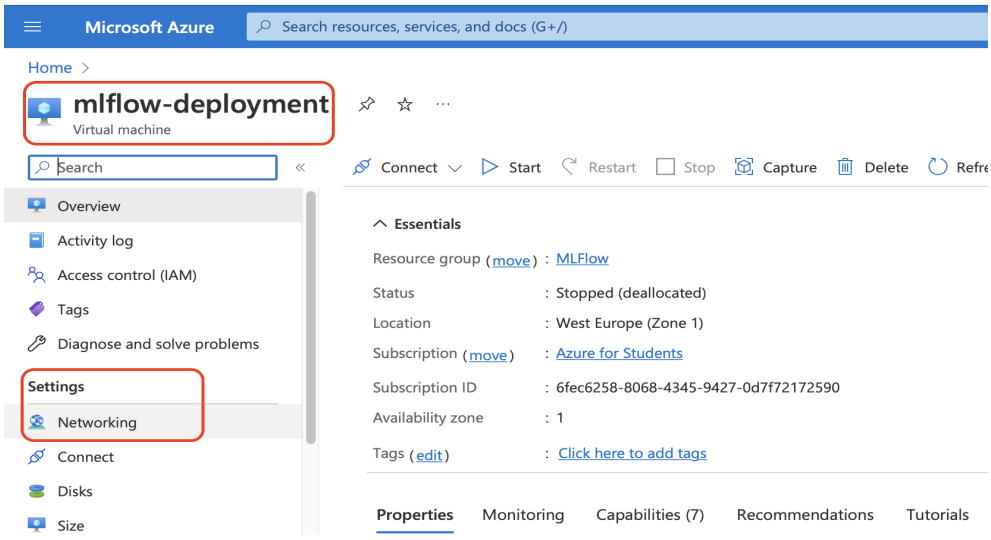


Fig. 5.6 Network Configuration of VM

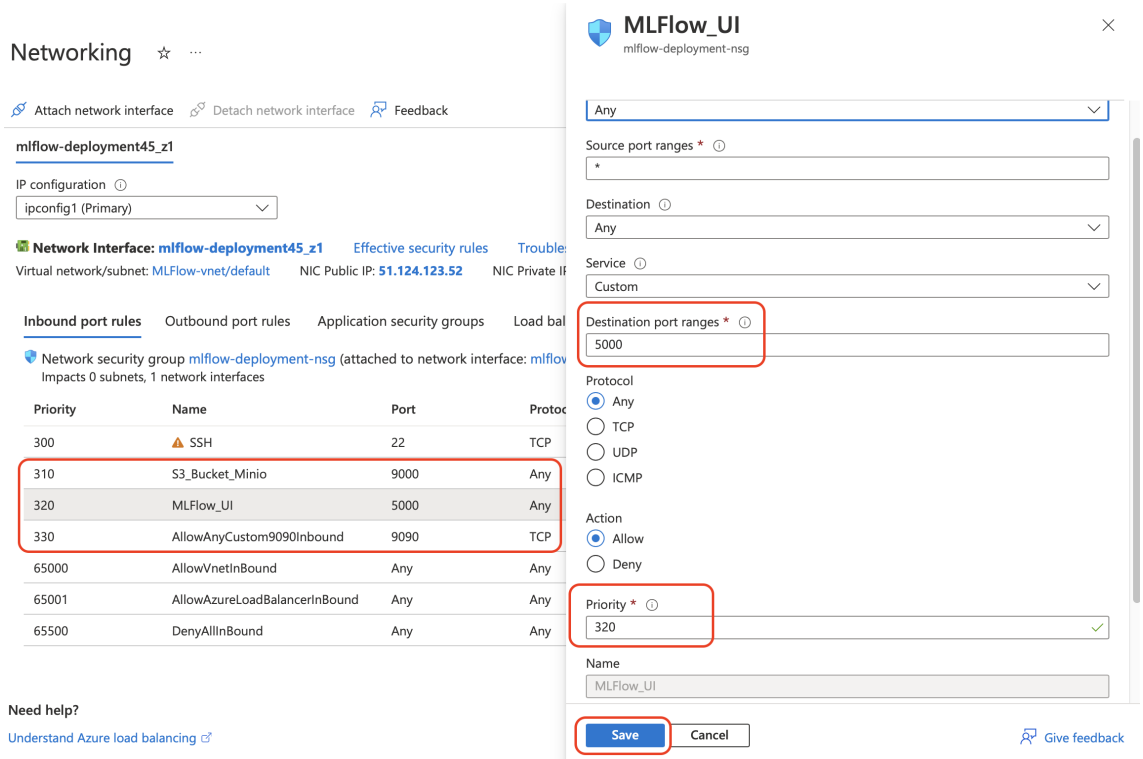


Fig. 5.7 Allowing traffic to destination port

12. Specify the source for the rule. You can choose to allow traffic from any source, or you can specify a specific IP address or range of addresses.
13. Click the "Add" button to create the rule.
14. The rule will be added to the list of inbound port rules for the virtual machine. You can view, edit, or delete the rule by clicking on it in the list.
15. Open a terminal window on your local machine. At the terminal prompt, enter the following command: **ssh <username>@<vm-ip-address>**, replacing **<username>** with the username of the account you want to use to log in to the VM, and **<vm-ip-address>** with the IP address of the VM.
16. Press "Enter" to initiate the SSH connection.
17. Enter the password for the user account you specified in step 1. Type the password and press "Enter".
18. If the login is successful, you will see a command prompt for the VM.

5.6 Docker Deployment

Next we install Docker Engine and Docker Compose on our Ubuntu VM, follow these steps:

1. Update the package manager index:

Fig. 5.8 Update Package Manager's package Index

2. Install packages to allow apt to use a repository over HTTPS:
3. Add the official Docker GPG key:
4. Add the Docker stable repository to APT sources:
5. Install Docker Engine:
6. Verify that Docker Engine is installed and running:

```
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Fig. 5.9 Docker Installation - 1

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
  dearmor -o /etc/apt/keyrings/docker.gpg
```

Fig. 5.10 Docker Installation - 2

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-
  by=/etc/apt/keyrings/docker.gpg]
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee
  /etc/apt/sources.list.d/docker.list > /dev/null
```

Fig. 5.11 Docker Installation - 3

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-
  compose-plugin
```

Fig. 5.12 Docker Installation - 4

```
sudo docker run hello-world
```

Fig. 5.13 Test Docker Installation: Docker Installation - 5

```
git clone https://github.com/swarajshaw/MLFlow-Docker-Deployment.git
```

Fig. 5.14 Clone git repo and fetch codebase

7. Test the installation: **docker-compose –version**

8. Next use **git clone** command followed by the repository URL to load required config and docker compose files.

The **.env** file contains environment variables that are used in the **bucket.py** script and **docker-compose.yml** file. These environment variables include:

- **MINIO_ROOT_USER:** the root user for the Minio S3 service
- **MINIO_ROOT_PASSWORD:** the password for the root user of the Minio S3 service
- **MLFLOW_BUCKET_NAME:** the name of the bucket to be used by the MLFlow server
- **MYSQL_DATABASE:** the name of the MySQL database used by MLFlow
- **MYSQL_USER:** the username for the MySQL database used by MLFlow
- **MYSQL_PASSWORD:** the password for the MySQL user
- **MYSQL_ROOT_PASSWORD:** the root password for the MySQL database
- **MLFLOW_S3_ENDPOINT_URL:** the endpoint URL for the Minio S3 service
- **MLFLOW_TRACKING_URI:** the URI for the MLFlow server

The **bucket.py** script reads these environment variables and checks that they are not empty. If any of the environment variables are empty, it will print an error message and exit with a status code of 1.
The **docker-compose.yml** file specifies how to build and run the Docker containers for the Minio S3 service, MySQL database, and MLFlow server. It sets the environment variables for these containers using the **environment** key, and specifies the values for these variables using **$VARIABLE_NAME** syntax, which tells Docker to substitute the value of the specified environment variable at runtime.
The environment variables specified in the **.env** file can be loaded into the current shell session by running the command **source .env**. This will make the environment variables available to any script or command run within the current shell session.

Next run, **docker-compose up -d** command on the terminal:

The **docker-compose** up command is used to start and run Docker containers based on the **docker-compose.yml** configuration file. The **-d** flag stands for "detached mode", which

```
1 sudo docker-compose up -d
```

Fig. 5.15 Start and run docker container

means that the containers will be started in the background and you will be returned to the command prompt.

The **sudo docker ps** command is used to list the running Docker containers along with their container IDs, names, image names, and other information on the VM.

The **execute_bucket.sh** script is a shell script that installs the **Minio** Python library and then runs the **bucket.py** script. The script starts by using the **source** command to load the environment variables from the **.env** file into the current shell session. The **set -o allexport** command enables the export of all variables to the environment of any subsequently executed command, and the **set +o allexport** command disables this behaviour. Next, the script installs the **Minio** library using **pip3**, and then runs the **bucket.py** script using **python3.** To run the **execute_bucket.sh** script, you can use the **bash** command followed by the name of the script, like this:

```
1 bash execute_bucket.sh
```

Fig. 5.16 Run the deployment pipeline

This will execute the commands in the script and create a bucket with the name specified in the **MLFLOW_BUCKET_NAME** environment variable, using the Minio S3 service running at the endpoint specified in the **MLFLOW_S3_ENDPOINT_URL** environment variable.

Copy the public IP address of the VM and run it with port 5000 and 9090 respectively.
MLFlow UI Server Deployment: http://51.124.123.52:5000
Minio S3 service: http://51.124.123.52:9090

5.7 Data Monitoring using Whylogs

Whylabs is a data profiling and monitoring platform that can be used to track the quality and behaviour of data over time. It integrates with the open-source machine learning platform MLFlow, allowing users to monitor data profiling from already logged data in MLFlow. One of the benefits of using WhyLabs is that it makes it easier to track and monitor data quality and behaviour. Data profiling is the process of examining and summarizing the characteristics of a dataset, and it can help identify issues such as missing values, inconsistencies, or outliers in the data. By tracking data profiling over time, users can identify changes in the data that may affect the performance of their machine learning models. WhyLabs offers a feature called "drift monitors" that can be used to track changes in data over time. Drift monitors allow users to set up thresholds for specific data attributes, and the platform will alert users if the data exceeds these thresholds. This can be useful for detecting changes in the data that may affect the performance of a machine learning model, such as a sudden increase in the number of missing values or a change in the distribution of a particular attribute.
In addition to drift monitors, WhyLabs also offers "anomaly windows" which allow users to specify a time period during which data is expected to be consistent. If the data exhibits unexpected behaviour outside of this window, the platform will alert the user. This can be useful for identifying unusual patterns or changes in the data that may not be immediately apparent.
One of the strengths of WhyLabs is its visualization capabilities. The platform offers a range of interactive charts and graphs that can be used to visualize data profiling results. These visualizations can be helpful for identifying patterns and trends in the data and for comparing data from different sources or time periods.
Whylabs is an open-source platform, which means that its source code is freely available for users to access and modify. This can be beneficial for users who want to customize the platform to meet their specific needs, or for those who want to contribute to the development of the platform.
In conclusion, WhyLabs is a powerful data profiling and monitoring platform that can be used to track the quality and behaviour of data over time. It offers features such as drift monitors and anomaly windows that can be used to detect changes in the data, and its visualization capabilities make it easy to identify patterns and trends in the data. As an open-source platform, WhyLabs is also highly customizable, making it a flexible and powerful tool for data professionals.
To set up a WhyLabs account and create a new model, follow these steps:

1. Go to the WhyLabs website https://hub.whylabsapp.com/ and click on the "Sign Up" button in the top right corner. Fill in the required information to create a new account.
2. Once you have created your account and logged in, you will be taken to the WhyLabs dashboard. From here, you can create a new model by clicking on the "Create New" button in the top right corner.

To create a new project in the WhyLabs dashboard and generate an access token for it, follow these steps:

1. Go to the WhyLabs dashboard and click on the "Projects" tab in the top menu.
2. On the Projects page, click the "Create New" button in the top right corner.
3. Fill in the required information to create a new project. Give your project a name (e.g., "mlflow-whylogs") and select an organization (if applicable).
4. Once your project is created, go to the "Integrations" tab in the top menu.
5. On the Integrations page, click the "Setup" button next to the "MLFlow" integration.
6. In the Setup dialog, select your newly created project from the dropdown menu.
7. Click the "Generate Access Token" button to generate a new access token for the selected project.
8. Copy the generated access token and use it in your integration with MLFlow

5.8 Whylabs Data Profiling

1. Installs the required Python packages using **pip**. The packages being installed are: **whylogs[mlflow], scikit-learn, matplotlib, pandas, mlflow-skinny**, and **whylabs-client**.
2. Imports the **os** module and sets some environment variables. The environment variables are:
    - **WHYLABS_DEFAULT_ORG_ID**: a case-sensitive organization ID for the WhyLabs platform.
    - **WHYLABS_API_KEY**: an API key for accessing the WhyLabs platform.

- **WHYLABS_DEFAULT_DATASET_ID**: a dataset ID for a selected model project in the WhyLabs platform.

3. Imports the **pandas** module and the **fetch_20newsgroups** function from the **sklearn.datasets module**.

4. Defines a function **get_20newsgroups** that retrieves a subset of the 20 Newsgroups dataset from the **sklearn.datasets** module and returns it as a Pandas dataframe. The subset consists of four categories: "alt.atheism", "talk.religion.misc", "comp.graphics", and "sci.space".

5. Calls the **get_20newsgroups** function to retrieve the subset of the 20 Newsgroups dataset and assigns the result to a variable called **df**.

6. Replaces the numeric labels in the df dataframe with the corresponding category names.

7. Assigns the **df** dataframe to a variable called **dataset**.

8. Shuffles the rows of the **dataset** dataframe and divides it into seven equal-sized batches.

9. Generate a batch of datasets with different levels of drift (i.e., changes to the data) from the original 20 Newsgroups dataset. We set a list of percentages that will be used to delete or change data points in the original dataset.

10. Then we create a loop where it creates copies of the original dataset, converts them to NumPy arrays, and modifies them according to the percentage of data points to delete or change. Specifically, it randomly selects a certain percentage of data points and either deletes them or changes their labels randomly. The modified datasets are then converted back to Pandas dataframes.

11. Adds a "date" column to each batch, with the date being set to the current date plus a number of days (0 to 6) depending on the batch.

12. Concatenates all the batches into a single dataframe called **dataset**.

13. Imports the **whylogs** module and uses its **log** function to perform data profiling on the **dataset** dataframe. The result of the data profiling is stored in a variable called **profile_result**.

14. Uses the **view** function of the **profile_result** to create a visual summary of the data profiling results.

15. Writes the data profiling results to the WhyLabs platform using the **writer** function and the **whylabs** writer.

16. Converts the data profiling results to a Pandas dataframe using the **to_pandas** function and displays the results.

5.9 Minio: Distributed Object Storage

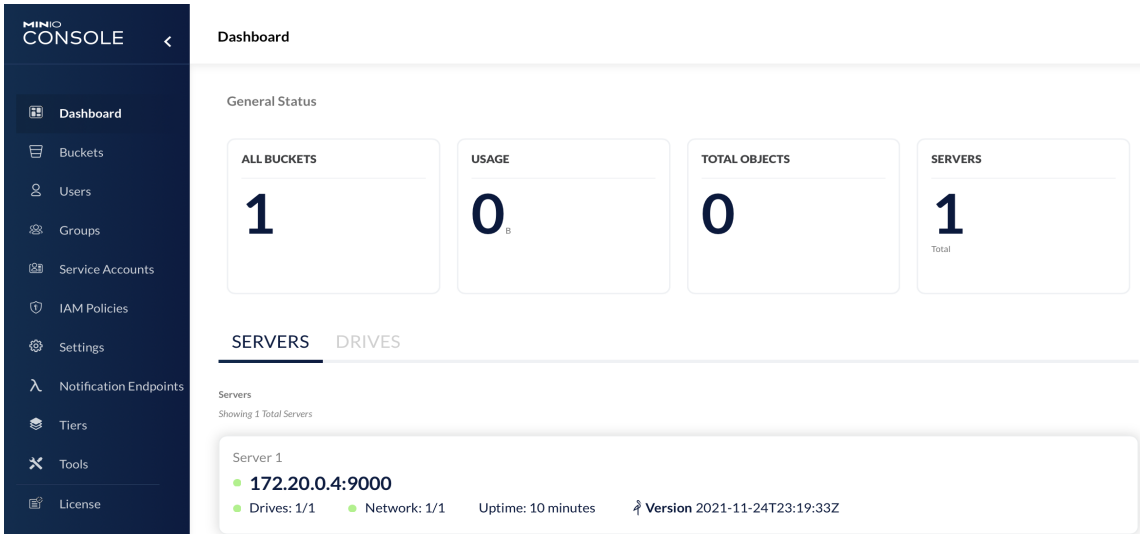


Fig. 5.17 Minio Storage

1. Scalability: Minio is a distributed object storage system that is designed to scale horizontally, which means you can add more servers to the cluster as your storage needs grow. This can be useful for large-scale machine learning projects where you need to store and access large amounts of data.

2. Compatibility: MLFlow natively supports Minio as a storage backend, so you can easily use it to store your MLFlow artifacts, such as models, parameters, and metrics.

3. Security: Minio provides secure, encrypted data storage and transmission using SSL/TLS certificates, which can be important for sensitive machine learning projects.

4. Performance: Minio is optimized for high performance, with support for parallelism and multi-threading. This can help improve the speed and efficiency of your machine learning pipelines.

5. Cost-effectiveness: Minio is open source and can be deployed on-premises or in the cloud, which can help you save on storage costs compared to proprietary solutions.

The above deployment is routed on Azure VM public IP address and serves as a block storage backend to the data logging solution. Then, when we run the machine learning pipeline and log the model artifacts and other relevant information using the MLFlow API, the information will be stored directly on Minio. Having a storage layer allows different layers of the stack to be developed, tested, and deployed independently, which can improve the maintainability and scalability of the system. It also allows different storage solutions to be used at the storage layer, depending on the needs of the application. For example, you might use Minio for storing large amounts of unstructured data, such as images or video, or for storing data for machine learning pipelines.

Chapter 6

Result & Analysis

6.1 Evaluation

6.1.1 Model Comparison

Different model records can help to fine-tune a model in several ways:

1. Model records can help you compare the performance of different models. By comparing the metrics and parameters recorded for each model, you can identify which models perform better on your data and choose the one that is most suitable for your task.

2. Model records can help you identify patterns and trends in the data. By looking at the various metrics recorded for each model, you can identify which parameters or hyperparameters are most important for your task and fine-tune your model accordingly.

3. Model records can help you identify areas for improvement. By comparing the performance of different models, you can identify areas where your models are underperforming and work on improving them.

4. Model records can help you choose the best model for your task. By comparing the performance of different models, you can choose the one that performs the best on your data and deploy it for your task.

Based on the comparison with the new run 10, it seems that the model's performance is generally better when the number of epochs is lower, the learning rate is lower, and the number of samples is higher. For example, in run 10, the model achieved an average test accuracy of 0.758 with 20 epochs, a learning rate of 0.001, and a large number of samples

						Metrics			Parameters				
	Run Name	Created	Duration	User	Models	↓ avg_test_acc	train_loss	val_loss	batch_size	epochs	learning_rate	num_samples	num_workers
	big-ray-752	3 hours ago	3.8h	swarajshaw	pytorch	0.758	0.333	0.542	32	20	0.001	20000	4
	salty-donkey-508	15 hours ago	2.2h	swarajshaw	pytorch	0.739	0.659	0.565	64	10	0.001	20000	4
	funny-turtle-72	11 days ago	54.8min	swarajshaw	pytorch	0.68	0.964	0.937	32	20	0.001	2000	4
	suave-sheep-110	20 days ago	29.8min	swarajshaw	pytorch	0.644	0.972	0.948	32	10	0.001	2000	4
	marvelous-hawk-658	29 days ago	13.4min	swarajshaw	pytorch	0.585	1.063	1.112	64	5	0.001	2000	4
	polite-skunk-834	6 days ago	13.1min	swarajshaw	pytorch	0.564	1.113	1.112	64	5	0.001	2000	4
	gregarious-mouse-848	26 days ago	24.6min	swarajshaw	pytorch	0.552	0.833	1.097	32	10	0.01	2000	4
	shivering-bat-317	16 hours ago	20.9min	swarajshaw	pytorch	0.55	0.886	0.997	64	1	0.001	20000	4
	intrigued-grub-683	27 days ago	15.9min	swarajshaw	pytorch	0.533	1.019	1.117	64	5	0.001	2000	6
	dazzling-wolf-605	22 days ago	6.8min	swarajshaw	pytorch	0.367	1.287	Infinity	32	10	0.01	2000	4
	stately-deer-895	4 hours ago	22.1min	swarajshaw	-	-	-	-	32	20	0.001	20000	6
	lyrical-newt-231	16 hours ago	4.0s	swarajshaw	-	-	-	-	64	-	0.001	20000	4
	welcoming-seal-647	16 hours ago	11.3s	swarajshaw	-	-	-	-	64	-	0.001	2000	4
	unique-roo-219	16 hours ago	4.8s	swarajshaw	-	-	-	-	64	-	0.001	2000	4
	fun-wren-154	17 hours ago	10.1s	swarajshaw	-	-	-	-	64	-	0.001	2000	4
	overjoyed-kite-317	11 days ago	13.0s	swarajshaw	-	-	-	-	32	-	0.001	2000	4

Fig. 6.1 Models runs

Run Name	epochs	learning_rate	max_epochs	num_samples	avg_test_acc	train_loss	val_loss	Duration	Status	batch_size	num_workers	optimizer_name
run 10	20	0.001	20	20000	0.758277535	0.333182037	0.542199731	3.8h	FINISHED	32	4	AdamW
run 9	10	0.001	10	20000	0.739172876	0.658686936	0.564948916	2.2h	FINISHED	64	4	AdamW
run 8	20	0.001	20	2000	0.680397749	0.963944077	0.937088728	54.8min	FINISHED	32	4	AdamW
run 7	10	0.001	10	2000	0.644176126	0.971881628	0.948357522	29.8min	FINISHED	32	4	AdamW
run 6	5	0.001	5	2000	0.584517062	1.063081145	1.111751795	13.4min	FINISHED	64	4	AdamW
run 5	5	0.001	5	2000	0.563742876	1.112571239	1.11247015	13.1min	FINISHED	64	4	AdamW
run 4	10	0.01	10	2000	0.551846564	0.833009481	1.09726429	24.6min	FAILED	32	4	AdamW
run 3	1	0.001	1	20000	0.550243437	0.885872245	0.997453392	20.9min	FINISHED	64	4	AdamW
run 2	5	0.001	5	2000	0.533333361	1.018695474	1.11707592	15.9min	FAILED	64	6	AdamW
run 1	10	0.01	10	2000	0.366714001	1.286780357	Infinity	6.8min	FINISHED	32	4	AdamW

Table 6.1 Model Runs with different parameters sorted by highest Avg Test Accuracy

(20000). On the other hand, in run 1, the model only achieved an average test accuracy of 0.367 with 10 epochs and a learning rate of 0.01, and a smaller number of samples (2000). Additionally, it appears that the model's performance is related to the duration of training. In general, the model performed better when the training duration was longer. For example, in run 10, the model achieved an average test accuracy of 0.758 with a training duration of 3.8 hours, while in run 1, the model only achieved an average test accuracy of 0.367 with a training duration of 6.8 minutes.

It's also worth noting that the optimizer used in all of the runs is AdamW. This is a variant of the Adam optimizer that includes weight decay, which can help prevent overfitting by adding a regularization term to the loss function.

In terms of the model's overall performance, it seems that the model is achieving decent results, with average test accuracies ranging from 0.367 to 0.758. However, there is room for improvement, as some runs had failed or had higher validation losses, indicating that the model may be overfitting or not generalizing well to the validation data. Tuning the

hyperparameters and increasing the number of samples may help improve the model's performance.

6.1.2 Data Profiling

To simulate drift in the 20newsgroup dataset, batches of streams were used. This means that the data was divided into smaller chunks, or streams, and processed in batches. This allows for more frequent updates to the model, as the data is continuously being refreshed with new streams.

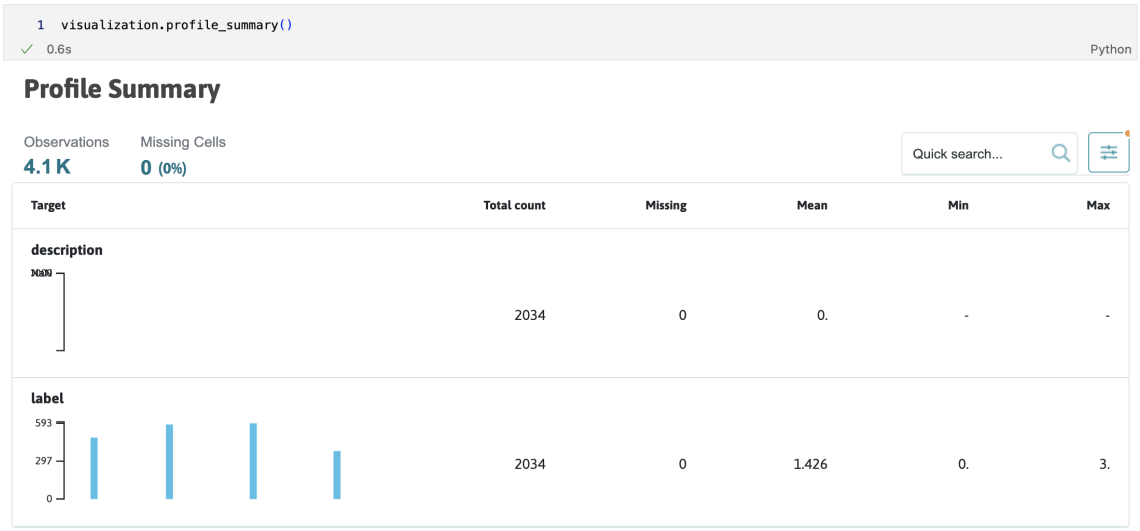


Fig. 6.2 Profiling Summary

The above function generates a summary of the profiling data for our dataset. This summary can include statistical measures such as mean, minimum, and maximum values for various metrics, as well as plots and charts that help you understand the distribution and trends in the data.

The *summary_drift_report()* function generates a report that helps you understand the magnitude and direction of the drift in a dataset, as well as the potential causes of the drift. The report includes statistical measures such as mean, median, minimum, and maximum values for various metrics, as well as plots and charts that help you visualize the drift over time.

Based on the simulation results, there was drift detected in one of the two features in the dataset, which was the "description" feature. The drift was in the range of **0.00 to 0.05**. There was no evidence of drift in the other feature, "label". The target features for the drift analysis were **"description"** and **"label"**, and the reference feature used for comparison was *"profile_view_drift"*.

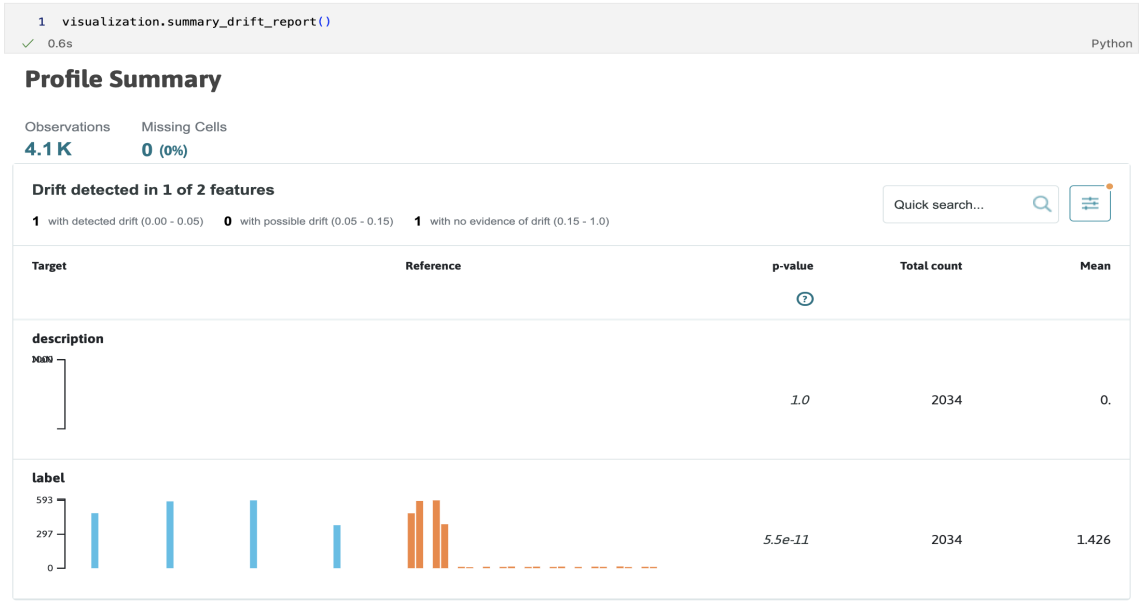


Fig. 6.3 Drift Profiling

The p-value for the "description" feature was **1.0**, which suggests that there is no statistically significant difference between the reference feature and the target feature. On the other hand, the p-value for the "label" feature was **5.5e-11**, which indicates that there is a very strong statistical significance between the reference and target features. This suggests that there has been a significant change in the statistical properties of the "label" feature over time.

The total count of observations in the dataset was **4.1K**, with **2034** observations for both the "description" and "label" features. The mean value for the "description" feature was **0**, while the mean value for the "label" feature was **1.426**.

Based on these results, it appears that there has been a significant change in the statistical properties of the "label" feature over time, as indicated by the low p-value and the high mean value. It is possible that this change is due to some external factor, such as a change in the data generating process or a change in the data collection or processing system.

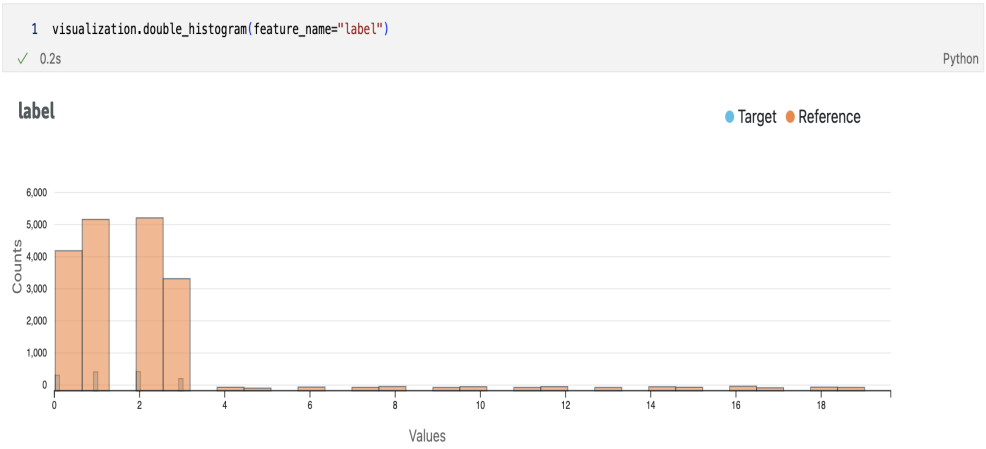


Fig. 6.4 Drift detected in the logged data

A double histogram plot is a type of chart that compares the distribution of two sets of data by displaying two histograms side by side. In the figure 6.4, there is a comparison between two profiles of data logged into Whylogs profiler namely, Target and Reference. We can draw conclusion that the target data is very insignificant overall compared to the reference drift simulated dataset.

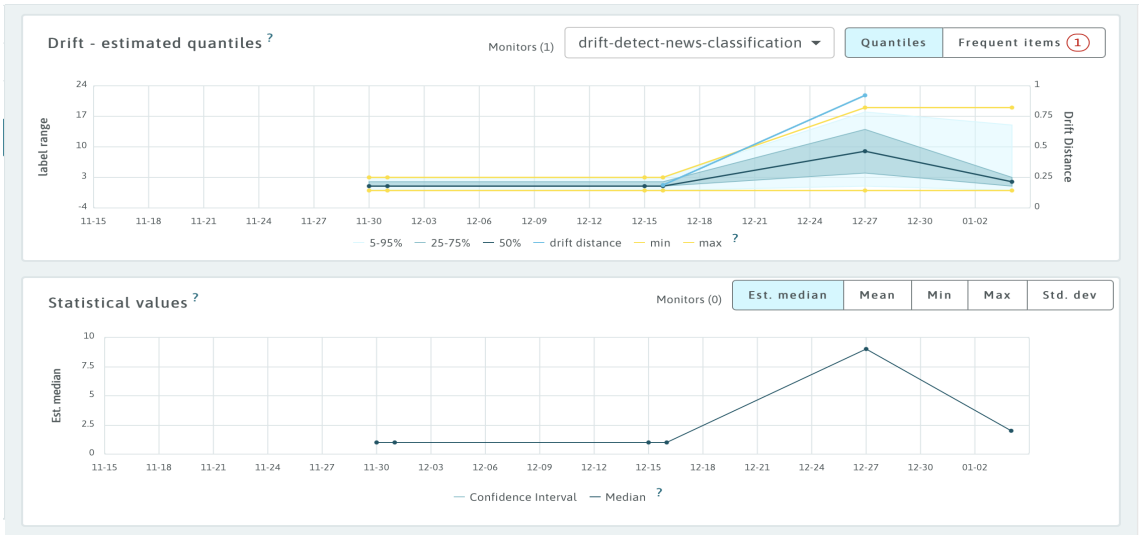


Fig. 6.5 Drift detected in the logged data at Whylogs Platform

We were successful in capturing drift in the dataset over a period of time using the WhyLogs platform. By using the **visualization.summary_drift_report()** and **visualization.double_histogram()** functions, we were able to identify changes in the statistical properties of the data and understand the magnitude and direction of the drift.

The solution is going to help improve pipelines by data logging and monitoring because it allows us to track the flow of data through the system, visualize the dependencies between different components, and identify potential issues or bottlenecks. By understanding how the system is behaving in real-time and being able to troubleshoot problems as they arise, we can ensure the reliability and performance of the system over time.

By logging and monitoring the data, we can also detect drift in the dataset and take appropriate measures to address it. This can help prevent issues caused by changes in the data generating process, data collection process, or data storage or processing systems, and can help us maintain the quality and integrity of the data over time.

6.2 Conclusion

In conclusion, the goal of this project was to investigate the importance of data logging and monitoring in machine learning operations (MLOps) and to present a comprehensive solution based on open-source tools such as MLFlow and WhyLogs. The solution was designed to be production-ready and easily integratable into existing enterprise environments, such as Microsoft Azure with Docker containers.

To evaluate the effectiveness of the proposed solution, a series of experiments were conducted using a variety of machine learning models on different datasets. The results of these experiments were then compared and analyzed to understand the impact of data logging and monitoring on model performance.

The results of the experiments demonstrated the importance of effective data logging and monitoring in MLOps. By logging and monitoring key metrics, such as training and validation loss, and monitoring the performance of models over time, it is possible to identify issues and take corrective action before they become major problems.

One key finding from the experiments was that the use of open-source tools such as MLFlow and WhyLogs can significantly improve the effectiveness of data logging and monitoring in MLOps. These tools provide a range of features, including the ability to track and visualize metrics, detect and alert on anomalies, and provide insights into model performance. By leveraging these tools, organizations can gain a deeper understanding of their models and take more informed decisions about how to optimize and deploy them. Another important finding was the need to address the issue of model drift, which can arise when the distribution of data used for training differs from the distribution of data used for inference. By implementing techniques for detecting and mitigating drift, organizations can ensure that their models remain accurate and reliable over time.

The project has successfully demonstrated that effective data logging and monitoring is

critical to the success of MLOps. By implementing the right tools and practices, organizations can improve the performance and reliability of their machine learning models, and ensure they are well-suited for deployment in real-world production environments.

6.3 Future Scope

There are several potential areas for future work and development in this project.
One potential direction is to expand the scope of the data logging and monitoring solution to include additional open-source tools and technologies. For example, tools such as Prometheus Prometheus (2012) and Grafana Labs (2018) could be integrated into the solution to provide more advanced monitoring and visualization capabilities. Additionally, there may be opportunities to integrate the solution with other machine learning platforms and frameworks, such as TensorFlow or PyTorch, to provide even greater flexibility and scalability.
Another area for future work is to further explore the issue of model drift and develop more advanced techniques for detecting and mitigating it. This could involve building more sophisticated drift detectors that can identify subtle changes in the data distribution and trigger corrective action accordingly. Additionally, there may be opportunities to develop techniques for actively avoiding drift by adjusting the training data or modifying the model architecture.
A third potential direction for future work is to explore the use of the data logging and monitoring solution in real-world production environments. This could involve deploying the solution in enterprise environments, such as Microsoft Azure with Docker containers, and testing its performance and reliability under real-world conditions. This work could also involve gathering feedback from users and incorporating their insights and suggestions into future versions of the solution.
Finally, it may be interesting to study the impact of data logging and monitoring on the overall performance and reliability of machine learning models in different types of applications. For example, the solution could be tested on models used for image classification, natural language processing, or time series forecasting, to understand how it affects the accuracy and reliability of these models in different contexts.
There are many exciting opportunities for future work and development in this project, and it is likely that the data logging and monitoring solution will continue to evolve and improve over time as new tools and technologies emerge.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P.A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. and Zhang, X. (2016) Tensorflow: A system for large-scale machine learning *CoRR* **abs/1605.08695**

Ali, R., Farooq, U., Arshad, U., Shahzad, W. and Beg, M.O. (2022) Hate speech detection on twitter using transfer learning *Computer Speech & Language* **74**, p. 101365

Devlin, J., Chang, M., Lee, K. and Toutanova, K. (2018) BERT: pre-training of deep bidirectional transformers for language understanding *CoRR* **abs/1810.04805**

Hung, B.T. and Huy, T.Q. (2022) Named entity recognition based on combining pretrained transformer model and deep learning in: *Artificial Intelligence and Sustainable Computing* pp. 311–320 Springer

Jing, W. and Bailong, Y. (2021) News text classification and recommendation technology based on wide & deep-bert model in: *2021 IEEE International Conference on Information Communication and Software Engineering (ICICSE)* pp. 209–216 IEEE

Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A. *et al.* (2017) In-datacenter performance analysis of a tensor processing unit in: *Proceedings of the 44th annual international symposium on computer architecture* pp. 1–12

Ken Lang, L. (1995) Newsweeder: Learning to filter netnews in: *Proceedings of the Twelfth International Conference on Machine Learning* pp. 331–339

Kiela, D., Firooz, H., Mohan, A., Goswami, V., Singh, A., Ringshia, P. and Testuggine, D. (2020) The hateful memes challenge: Detecting hate speech in multimodal memes *Advances in Neural Information Processing Systems* **33**, pp. 2611–2624

Labs, G. (2018) Grafana documentation

LightningAI *et al.* (2018) Lightning ai

Mao, Y., Yan, W., Song, Y., Zeng, Y., Chen, M., Cheng, L. and Liu, Q. (2020) Differentiate quality of experience scheduling for deep learning applications with docker containers in the cloud *arXiv preprint arXiv:2010.12728*

Minio (2016) Minio high performance object storage

Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., López García, Á., Heredia, I., Malík, P. and Hluchỳ, L. (2019) Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey *Artificial Intelligence Review* **52**(1), pp. 77–124

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. (2017) Automatic differentiation in pytorch in: *NIPS 2017 Workshop on Autodiff*

Prometheus (2012) Prometheus - monitoring system time series database

Renggli, C., Rimanic, L., Gürel, N.M., Karlaš, B., Wu, W. and Zhang, C. (2021) A data quality-driven view of mlops *arXiv preprint arXiv:2102.07750*

Ruf, P., Madan, M., Reich, C. and Ould-Abdeslam, D. (2021) Demystifying mlops and presenting a recipe for the selection of open-source tools *Applied Sciences* **11**(19), p. 8861

Shishah, W. (2021) Fake news detection using bert model with joint learning *Arabian Journal for Science and Engineering* **46**(9), pp. 9115–9127

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I. (2017) Attention is all you need *CoRR* **abs/1706.03762**

Vyas, P. (2022) Fake news detection on the web: A deep learning based approach

Wang, Y., Yang, W., Ma, F., Xu, J., Zhong, B., Deng, Q. and Gao, J. (2020) Weak supervision for fake news detection via reinforcement learning in: *Proceedings of the AAAI conference on artificial intelligence* vol. 34 pp. 516–523

Webb, G.I., Hyde, R., Cao, H., Nguyen, H.L. and Petitjean, F. (2016) Characterizing concept drift *Data Mining and Knowledge Discovery* **30**(4), pp. 964–994

Whylabs (2019) Whylogs overview: Whylabs documentation

Xu, P., Shi, S. and Chu, X. (2017) Performance evaluation of deep learning tools in docker containers in: *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)* pp. 395–403 IEEE

Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S.A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M. *et al.* (2018) Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.* **41**(4), pp. 39–45

Zhang, X., Zhao, J. and LeCun, Y. (2015) Character-level convolutional networks for text classification *Advances in neural information processing systems* **28**

Appendix A

Installation

A.1 Conda Requirement

```
1. channels:
2. conda-forge
3. dependencies:
4. python=3.8
5. pip
6. pip:
7. mlflow
8. sklearn
9. cloudpickle==1.6.0
10. boto3
11. transformers>=4.0.0
12. pandas
13. torch>=1.11.0
14. torchdata
15. torchtext==0.12.0
16. pytorch-lightning==1.7.6
17. protobuf<4.0.0
```

A.2 Python Environment Requirement

```
1. python: "3.8"
2. build\_dependencies:
```

```
3. pip
4. dependencies:
5. mlflow
6. sklearn
7. cloudpickle==1.6.0
8. boto3
9. transformers>=4.0.0
10. pandas
11. torch>=1.13.0
12. torchdata
13. torchtext==0.12.0
14. pytorch-lightning==1.7.6
15. protobuf<4.0.0
```

A.3 Requirement

```
1. numpy
2. mlflow
3. sklearn
4. cloudpickle==1.6.0
5. boto3
6. transformers>=4.0.0
7. pandas
8. torch>=1.13.0
9. torchdata
10. torchtext==0.12.0
11. pytorch-lightning==1.7.6
12. protobuf<4.0.0
```

A.4 Docker Compose Requirement

```
1. version: "3.9"
2. services:
3. minio:
4. image: minio/minio:latest
5. restart: unless-stopped
```

```
6. container_name: mlflow_s3
7. ports:
8. "9000:9000"
9. "9090:9090"
10. environment:
11. MINIO_ROOT_USER=${MINIO_ROOT_USER}
12. MINIO_ROOT_PASSWORD=${MINIO_ROOT_PASSWORD}
13. command: server /data --console-address ":9090"
14. networks:
15. internal
16. public
17. volumes:
18. minio_data:/data
19. db:
20. image: mysql/mysql-server:5.7.40
21. restart: unless-stopped
22. container_name: mlflow_db
23. expose:
24. "3306"
25. environment:
26. MYSQL_DATABASE=${MYSQL_DATABASE}
27. MYSQL_USER=${MYSQL_USER}
28. MYSQL_PASSWORD=${MYSQL_PASSWORD}
29. MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
30. volumes:
31. db_data:/var/lib/mysql
32. networks:
33. internal
34. mlflow:
35. container_name: mlflow_server
36. image: mlflow_server
37. restart: unless-stopped
38. build:
39. context: ./mlflow
40. dockerfile: Dockerfile
41. ports:
```

```
42. "5000:5000"
43. environment:
44. MINIO_ROOT_USER=${MINIO_ROOT_USER}
45. MINIO_ROOT_PASSWORD=${MINIO_ROOT_PASSWORD}
46. MLFLOW_S3_ENDPOINT_URL=http://minio:9000
47. networks:
48. public
49. internal
50. entrypoint: mlflow server --backend-store-uri
mysql+pymysql://${MYSQL_USER}:${MYSQL_PASSWORD}
@db:3306/${MYSQL_DATABASE} --default-artifact-root
s3://${MLFLOW_BUCKET_NAME}/ --artifacts-destination
s3://${MLFLOW_BUCKET_NAME}/ -h 0.0.0.0
51. depends_on:
52. wait-for-db:
53. condition: service_completed_successfully
54. create_s3_buckets:
55. image: minio/mc
56. depends_on:
57. "minio"
58. entrypoint: >
59. /bin/sh -c "
60. until (/usr/bin/mc alias set minio http://minio:9090
'${MINIO_ROOT_USER}' '${MINIO_ROOT_PASSWORD}')
do echo '...waiting...' && sleep 1; done;
61. /usr/bin/mc mb minio/${MLFLOW_BUCKET_NAME};
62. exit 0;
63. "
64. networks:
65. internal
66. wait-for-db:
67. image: atkrad/wait4x
68. depends_on:
69. db
70. command: tcp db:3306 -t 90s -i 250ms
71. networks:
```

```
72. internal
73. networks:
74. internal:
75. public:
76. driver: bridge
77. volumes:
78. db_data:
79. minio_data:
```

Appendix B

Code

```
1. #Importing all the libraries that are needed for the code to run.
2. import math
3. import os
4. from argparse import ArgumentParser
5. import logging
6. import numpy as np
7. import pandas as pd
8. import pytorch_lightning as pl
9. import torch
10. import torchtext
11. import torch.nn.functional as F
12. import torchtext.datasets as td
13. from pytorch_lightning.callbacks import ( EarlyStopping,
                                              ModelCheckpoint,
                                              LearningRateMonitor,)
14. from sklearn.datasets import fetch_20newsgroups
15. from sklearn.metrics import accuracy_score
16. from sklearn.model_selection import train_test_split
17. from torch import nn
18. from torch.utils.data import Dataset, DataLoader
19. from torch.utils.data.dataset import random_split
20. import torchdata
21. from torchdata.datapipes.iter import IterDataPipe
22. from torchtext.data import functional
23. from torchtext.data.functional import to_map_style_dataset
```

```
24. from torchtext.datasets import AG_NEWS
25. from transformers import BertModel, BertTokenizer, AdamW
26. import mlflow.pytorch
27. import mlflow.pytorch
```

```
1. def get_20newsgroups(num_samples):
2. categories = ["alt.atheism", "talk.religion.misc",
                 "comp.graphics", "sci.space"]
3. X, y = fetch_20newsgroups(subset="train",
        categories=categories, return_X_y=True)
4. return pd.DataFrame(data=X, columns
           ["description"]).assign(label=y)
            .sample(n=num_samples)
```

```
1. def get_ag_news(num_samples):
2. # reading the input
3. td.AG_NEWS(root="data", split=("train", "test"))
4. train_csv_path = "data/AG_NEWS/train.csv"
5. return (pd.read_csv(train_csv_path, usecols=[0, 2],
      names=["label", "description"]).assign(
      label=lambda df: df["label"] - 1)
6. #make labels
7. zero-based.sample(n=num_samples))
```

```
1. class NewsDataset(IterDataPipe):
2. def init(self, tokenizer, source, max_length, num_samples, dataset="20newsgroups")
3. """
Custom Dataset - Converts the input text and label to tensor
:param tokenizer: bert tokenizer
:param source: data source - Either a dataframe or DataPipe
:param max_length: maximum length of the news text
:param num_samples: number of samples to load
:param dataset: Dataset type - 20newsgroups or ag_news
5. """
6. super().init()
7. self.source = source
```

```
8. self.start = 0
9. self.tokenizer = tokenizer
10. self.max_length = max_length
11. self.dataset = dataset
12. self.end = num_samples
13. def __iter__(self):
14. worker_info = torch.utils.data.get_worker_info()
15. if worker_info is None:
    iter_start = self.start
    iter_end = self.end
16. else:
    per_worker = int(
    math.ceil((self.end - self.start) / float(worker_info.num_workers)))
17. worker_id = worker_info.id
18. iter_start = self.start + worker_id * per_worker
19. iter_end = min(iter_start + per_worker, self.end)
20. for idx in range(iter_start, iter_end):
21. if self.dataset == "20newsgroups":
22. review = str(self.source["description"].iloc[idx])
23. target = int(self.source["label"].iloc[idx])
24. else:
    target, review = self.source[idx]
    target -= 1
25. encoding = self.tokenizer.encode_plus(
    review,
    add_special_tokens=True,
    max_length=self.max_length,
    return_token_type_ids=False,
    padding="max_length",
    return_attention_mask=True,
    return_tensors="pt",
    truncation=True,)
26. yield {"review_text": review,
    "input_ids": encoding["input_ids"].flatten(),
    "attention_mask": encoding["attention_mask"].flatten(),
    "targets": torch.tensor(target, dtype=torch.long),}
```

Appendix C

Deployment Code

```
1. import os
2. from minio import Minio
3. from minio.error import InvalidResponseError
4. accessID = os.environ.get('MINIO_ROOT_USER')
5. accessSecret = os.environ.get('MINIO_ROOT_PASSWORD')
6. minioUrl = os.environ.get('MLFLOW_S3_ENDPOINT_URL')
7. bucketName = os.environ.get('MLFLOW_BUCKET_NAME')
8. if accessID == None:
    print('[!] AWS_ACCESS_KEY_ID environment
    variable is empty! run \'source .env\' to
    load it from the .env file')
    exit(1)
9. if accessSecret == None:
    print('[!] AWS_SECRET_ACCESS_KEY environment
    variable is empty! run \'source .env\' to
    load it from the .env file')
    exit(1)
10. if minioUrl == None:
    print('[!] MLFLOW_S3_ENDPOINT_URL environment
    variable is empty! run \'source .env\' to
    load it from the .env file')
    exit(1)
11. if bucketName == None:
    print('[!] AWS_BUCKET_NAME environment variable
    is empty! run \'source .env\' to load it
```

```
    from the .env file')
    exit(1)
12. minioUrlHostWithPort = minioUrl.split('//')[1]
13. print('[*] minio url: ', minioUrlHostWithPort)
14. s3Client = Minio(
    minioUrlHostWithPort,
    access_key=accessID,
    secret_key=accessSecret,
    secure=False)
15. #Check if the bucket already exists
16. if s3Client.bucket_exists(bucketName):
17. #Delete all objects in the bucket
18. objects = s3Client.list_objects(bucketName, recursive=True)
19. for obj in objects:
20. s3Client.remove_object(bucketName, obj.object_name)
21. #Delete the bucket
22. s3Client.remove_bucket(bucketName)
23. #Create a new bucket with the same name
24. s3Client.make_bucket(bucketName)
```

```
1. # Description: Minio Docker image
2. # docker run \
3. # -p 9000:9000 \
4. # -p 9090:9090 \
5. #--name minio-dock \
6. #-v ~/minio/data:/data \
7. #-e "MINIO_ROOT_USER=ROOTNAME" \
8. #-e "MINIO_ROOT_PASSWORD=CHANGEME123" \
9. #quay.io/minio/minio server /data --console-address ":9090"
```

```
1. #!/bin/bash
2. docker-compose up -d --build
3. #!/bin/bash
4. docker-compose down
```

Appendix D

Data Profiling using WhyLabs

```
1. %pip install -q 'whylogs[mlflow]'
2. %pip install -q scikit-learn matplotlib pandas mlflow-skinny
3. !pip install whylabs-client
```

```
1 import os
2 os.environ["WHYLABS_DEFAULT_ORG_ID"] = "org-ZXsw2C"
3 # ORG-ID is case sensistive
4 os.environ["WHYLABS_API_KEY"] \
    = "HbKymCA1m3.QtWbFxfCdGAAjAhT51SiBrHqHyLPWFNzCGngIxaBMh9x64Khe7DcP"
6 os.environ["WHYLABS_DEFAULT_DATASET_ID"] = "model-6"
7 The selected model project "My Model
    (model-1)" is "model-1"
8 import pandas as pd
9 from sklearn.datasets import fetch_20newsgroups
10 def get_20newsgroups() -> pd.DataFrame:
11 categories = ["alt.atheism", "talk.religion.misc",
12 "comp.graphics", "sci.space"]
13 X, y = fetch_20newsgroups(
14 subset="train", categories=categories, return_X_y=True)
15 dataframe = pd.DataFrame(data=X,
    columns=["description"]).assign(label=y)
16 return dataframe
17 # phrase label 0 as alt.atheism and label 1 as talk.religion.misc \
18 and label 2 as comp.graphics and label 3 as sci.space
19 df = get_20newsgroups()
```

```
df["label"] = df["label"].replace(
{0: "alt.atheism",
     1: "talk.religion.misc",
     2: "comp.graphics",
     3: "sci.space"})
 df.head()
dataset = df
dataset.head()
```

```
from datetime import datetime, timezone, timedelta
#shuffle and divide dataset into 7 different dates and
load into a list called daily_batches
daily_batches = []
for i in range(7):
dataset = dataset.sample(frac=1)
daily_batches.append(dataset.iloc[:int(len(dataset)/7)])
#add a date column to each batch
for i in range(7):
daily_batches[i]["date"] = datetime.now(timezone.utc) + timedelta(days=i)
#concatenate all batches into one dataframe
dataset = pd.concat(daily_batches)
dataset.head()
#show full dataset sort by all date
dataset.sort_values(by="date")
dataset
```

```
import subprocess  # <-- add subprocess library
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
import time  # <-- add time library

# Load the 20 Newsgroups dataset
data = fetch_20newsgroups()
X = dataset['description']
```

```
10 y = dataset['label']

11 # Set the percentage of data points to delete or change in
    each modified dataset
12 percentages = [0, 10, 20, 30, 40, 50]
13
14 dataset = pd.DataFrame()  # <-- define dataset variable
15 # Create a batch of drift datasets
16 while True:
17    for percentage in percentages:
18        # Create copies of the original dataset
19        X_modified = X.copy()
20        y_modified = y.copy()  # <-- create copy of labels array
21
22        # Convert to NumPy arrays
23        X_modified = X_modified.values  # <-- convert to NumPy array
24        y_modified = y_modified.values  # <-- convert to NumPy array
25
26. # Delete a percentage of the data points
27. num_to_delete = int(len(X_modified) * percentage / 100)
28. indices_to_delete = np.random.choice(len(X_modified),
     num_to_delete, replace=False) #<-- set replace=False
29. X_modified = np.delete(X_modified, indices_to_delete, axis=0)
30. y_modified = np.delete(y_modified,
    indices_to_delete, axis=0) # <-- delete corresponding labels
31. # Change the labels of a percentage of the data points
32. num_to_change = int(len(X_modified) * percentage / 100)
33. indices_to_change = np.random.choice(len(X_modified),
    num_to_change, replace=False)  # <-- set replace=False
34. y_modified[indices_to_change] = np.random.randint(0, 20, num_to_change)
# <-- change labels
35. print(f'X_modified.shape = {X_modified.shape},
    y_modified.shape = {y_modified.shape}')
    # <-- print shapes
36. # Convert the modified dataset to a Pandas dataframe
37. drift_dataset = pd.DataFrame({'description': X_modified,
```

```
   'label': y_modified})
38. # Append the new batch of data to the dataset
39. dataset = dataset.append(drift_dataset, ignore_index=True)
40. # Log the new batch of data
41. profile_result = why.log(drift_dataset)
42. profile_view_drift = profile_result.view()
43. profile_result.writer("whylabs").write()
44. # Wait 5 seconds before generating the next batch of data
45. time.sleep(5)
```

```
1. drift_dataset.head()
2. # datatype of the dataset
3. dataset.dtypes
4. import whylogs as why
5. profile_result = why.log(dataset)
6. profile_view = profile_result.view()
7. profile_result.writer("whylabs").write()
8. import whylogs as why
9. profile_result = why.log(drift_dataset)
10. profile_view2 = profile_result.view()
11. profile_result.writer("whylabs").write()
12. profile_view.to_pandas()
13. profile_view2.to_pandas()
```

```
1. # Data Drift with whylogs
2. from whylogs.viz import NotebookProfileVisualizer
3. visualization = NotebookProfileVisualizer()
4. visualization.set_profiles
    (target_profile_view=profile_view,
    reference_profile_view=profile_view_drift)
5. visualization.profile_summary()
6. visualization.summary_drift_report()
7. visualization.double_histogram(feature_name="label")
```