MLOps and data versioning in machine learning project

– Industrial Internship –

submitted by

Yizhen Zhao

Yizhen Zhao
yizhenzhao@hotmail.com
Student ID: 2658811

Supervisor

1st: Adam Belloum
2nd: Thilo Kielmann

Universiteit van Amsterdam
Vrije Universiteit Amsterdam

Contents

1 Introduction

As machine learning and AI propagate in technical Enterprise and become core capabilities for solving complex real-word problems. If they are going to be the essential part of an Enterprise, Enterprises should establish their machine learning development process and tools to test, deploy, manage and monitor ML models in production. The word MLOps is generated by applying DevOps principles to ML system. Practicing MLOps means automatically monitor all steps in ML system, such as integration, testing, deployment and infrastructure management. Data versioning also plays an important role in ML projects. Not only the code itself needs version control, but the models and datasets. More and more data is being generated every day, thus the way to manage different versions and a large volume of dataset matters.

Why more and more companies would like to include machine learning techniques? Why data versioning matters? There are lots of reasons behind it. One of the advantages of machine learning is it has the concept of pattern recognition and through some observations, it allows computers to learn without being programmed and perform certain tasks. It helps us create an approximation of decisions logic where the logic is unknown to us, but we have observations through abundant data [1]. As the code and datasets might be improved over time, it is important to record information about how the data was processed and what artefacts or results were generated, which allows us reproducing any artefacts or results in the future[2].

The main objective of this report is to conduct an industrial implementation of data versioning and a basic ML lifecycle of a machine learning project and reflect on the characteristics and constraints of industrial practice, in comparison with study contents at university.

I will elaborate my project report step by step from the following aspects. First, background information is here in ***Introduction***. Second, I will list ***Research Questions*** and the motivations for choosing them. In the following, in ***Discussion of Literature and Blog***, I will classify the literature, online websites and conduct a detailed analysis to answer the research questions. Then, the actual industrial implementation of our project will be described in ***Project Implementation***. A reflection on study contents being applied in an industrial context and further research will be displayed in ***Reflection*** and ***Further research***. Finally, extra information about the company, data pipeline, example of data format and terminologies mentioned in this paper or names generated by our data team are explained in ***Appendix***.

2 Research Questions

One of the tasks we encounter right now is that we will use a machine learning approach for one of the features in flags library, flag_combo, which tells an outlet it sells combo or not. If an outlet sells combo, then set the value of flag_combo as 1 for that outlet. But we haven't established a framework or rules for our data,

model version control, or identify our MLOps lifecycle[1] yet. Since our code, data and models might be improved over time and it is necessary to have a good methodology in terms of version control for all of them and keep using them in production. Moreover, as machine learning approaches are also used in our other libraries, and they will take more places in the future. It is a good chance to do some researches in the field of data versioning and apply it to our pipelines. There are already lots of discussions and researches about data versioning[2] and the important role it plays in MLOps lifecycle[3].

Therefore, the research questions I defined and the rational motive for the stated research questions are:

- Q1: What is the current state of data versioning in the marketplace?
  - Before actually implementing the data versioning into our pipeline, it's better to do some research and planning first. Study the state of MLOps lifecycle with existent use cases. Identify the tools or mechanisms used for data versioning.
- Q2: What kind of requirements need to be considered when considering apply data versioning into the project pipeline?
  - Various requirements need to be considered when applying data version control in a company. For examole, we need to search for tools that support data or model version control. As Git[4] is widely used for collaborating and tracking changes in code. Maybe there are some tools for versioning dataset or model that can be integrated with Git.
  - We need to identify a framework for data versioning and MLOps lifecycle for our project that we could easily integrate with our current code architecture, figure out the process to manage it and align with our technology stack. The fact is we do not have too much time (e.g. 6 months) to implement this feature and we need to deliver our result in time.
  - We need to define our rules for version control. For example, what kind of changes on a dataset can be regarded as a new version.
- Q3: What kind of approaches or frameworks can be used for building an integrated ML system and to continuously operate it in production?
  - We want to identify the approaches in the marketplace for building an integrated ML system and to continuously operate it in production, and what is the final decision for our data versioning mechanism. Showing the discussion and plans we had for it and the reasons for choosing or not choosing certain approaches.

[1] https://blogs.nvidia.com/blog/2020/09/03/what-is-mlops/
[2] https://medium.com/acing-ai/ml-ops-data-science-version-control-5935c49d1b76
[3] https://blog.datatron.com/version-control-for-ml-models-with-code-algorithms-and-training-data-sets/
[4] https://git-scm.com/

3 Discussion of Literature and Blog

Relevant papers or blogs will be analyzed and discussed in this section to answer the research questions listed above. The selected literature and online websites can be grouped into the following topics: ***Definitions***, ***Popularity and importance of data versioning***, ***Possible issues and requirements of data versioning***, ***Tools and mechanisms for data versioning*** and ***MLOps framework and lifecycle***. The discussions of relevant findings of the literature and blogs are under each topic.

3.1 Definitions

Data versioning and MLOps are the two main topics discussed in this report. The first step is to understand the actual meaning of these two terms.

Data versioning, according to the definition in Stanford libraries[3], versioning means when you make changes on data, a new version and new copies of data have been generated, so that you can easily rollback and retrieve specific versions of data. A blog[4] explained what do we mean by 'data versioning'. A version is when something changed and different from an earlier form. We could consider it as a new version when there is a change in the content or architecture of the source.

MLOps is a relatively new field, but we are familiar with the term DevOps, In Lucy's research[5], she mentioned DevOps contains 2 words, Developers and Operations. It connects the development and operations, enables them to work and collaborate efficiently, automatically deploy, monitor and deliver software into production. According to the blogs[6][7], MLOps follows the similar principle, modeled on the existing discipline of DevOps, but involved one more module, machine learning part which contains not only the machine learning model code but also data. Hence, MLOps combines machine learning part, DevOps and data engineering, which aims to deploy, maintain and deliver the ML system in production reliably and efficiently.

3.2 Popularity and importance of data versioning

As we defined the definition of data versioning. Why is it important to introduce this term? Why it matters? The same blog[4] established on ANDS[5] mentioned researchers are required to correctly identify the dataset they used in their research so that the reproducibility can be guaranteed. By using the unique version number that follows the standardized rules, enables researchers to know whether and how the data has changed over time and choose certain version of a dataset to working with.

A use case established on Research Data Alliance[6], submitted by Jens[8] related to the data revision and version for data products said it is true that data versioning is necessary for ensuring the reproducibility of research. Modern software

[5]https://www.ands.org.au
[6]https://rd-alliance.org

systems evolve rapidly and may have many variants. Each variant may address different requirements. For example, in ML projects, models might be improved overtime and we might use a different combination of hyperparameters that conduct different results. The same for data, datasets involved in ML projects may also change overtime. Thus, versioning enables us to reproduce certain steps in ML projects, which means we could repeatedly run ML algorithms on a certain version of datasets and obtain the same or similar results on a particular project. In any continuous integration or continuous delivery cycle, reproducibility is an important characteristic.

3.3 Possible issues and requirements of data versioning

Some possible issues exist when considering data versioning. Currently, there is no agreement or standardized rules among data communities on how and when data should be versioned[4]. For example, what kind of change on a dataset can be regarded as a new version? How to define a major or minor change? Currently, the most commonly adopted approach is semantic versioning[7]. But the situation may differ from use cases and companies' software settings.

In Thorsten's research[9], they mentioned in a ML project, it is common that data scientists may test different ML algorithms, hyperparameters and so on, aiming to find out the best result for ML tasks. However, they need to manually log the version numbers so that they could easily track the process, compare results among different versions of ML models, reproduce certain steps in a ML project. But manually logging is inefficient and error-prone. Besides, the ML project is not only about code, but also data. Currently, the most common version control techniques used in software engineer systems, such as Git, could not fulfill the requirements for ML projects because of the scale and formats of data.

3.4 Tools and mechanisms for data versioning

As the possible issues and requirements for data versioning are discussed in the previous section. In this section, tools and mechanisms for implementing data versioning will be discussed. In the same research of Jens'[8], they summarized 38 use cases from 33 organizations for data versioning, such as tools they used, workflows and 'best practice' for versioning.

Regrading to what kind of changes on dataset should be considered as a new version, W3C[10] described some scenarios that most publishers agreed. A new feature is added or an existing feature is removed from a dataset, this can be a major version. While for the minor version, an error was identified in one of the existing features in the dataset and this error must be corrected. The important thing here is that avoid making changes without incrementing the version number and the versioning information of a dataset should be consistent and informative so that this dataset is trustworthy.

[7] https://semver.org

There are some similarities between software versioning and dataset versioning. For instance, some data projects release major/minor releases. We could use the principle for software versioning and apply it to dataset versioning. Jennifer[11] described the use of a distributed version control system like Git and the hosting site GitHub for data analysis and workflow. For a large project that may release frequently, the generic Git workflow[8] is, a master branch is always used for release. Feature branch is used to develop new functionalities separately and only merged with develop branch. The develop branch is created next to master branch, after successfully tested the functionalities on develop branch, it merges to master, which then can be released. Basically, the Git workflow may differ from projects. But it often makes sense to use a branch strategy described here and make sure to have a consistent naming strategy for branches and versioning.

Another tool for data versioning mentioned in this research[8] is DVC (Data Version Control)[9]. DVC is built to make ML projects shareable and reproducible. It uses so-called DVC-file[10] which contains a description of a file that can be used for versioning, to manage the large dataset, ML models and can be integrated with Git for managing code as well. DVC allows storing large data files, ML models with Git, without checking the file contents into Git. It has an easy-to-use command-line operation similar to Git and it can be used with any cloud provider (S3, Google Cloud).

3.5 ML lifecycle and MLOps framework

Data versioning is only part of the machine learning projects. Machine learning project itself also needs an end-to-end workflow, so-called lifecycle. A case study established by Saleema[12], they studied how various Microsoft software teams build their software applications with AI features and described a nine-stage workflow for developing machine learning projects, shown in Figure 1. In the first four stages (*Model requirements*, *Data collection*, *Data cleaning*, *Data labelling*), designers decide which features are needed, look for available dataset or collect their own dataset, remove outliers in the dataset and assign the true labels to each record. *Feature engineering* executes operations such as an extract or select informative features for training models, and during *Model training*, we train our model based on the clean, feature generated, labels assigned dataset. Then in *Model evaluation*, the engineers evaluate the output model on validation tests. Finally, in *Model deployment and Model monitoring*, deploy the model on the target devices and continuously monitor the model in production.

One of the main challenges of MLOps is how to integrate the ML lifecycle or workflow into the typical CI/CD process. Microsoft established a case study for designing the MLOps framework on their official website[13]. They have a similar

[8] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
[9] https://dvc.org/
[10] https://dvc.org/doc/user-guide/dvc-files-and-directories

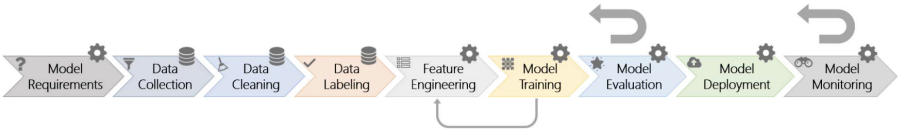

Figure 1: Nine stages of the machine learning workflow

ML lifecycle as the one described above (i.e. Saleema's[12] case). The basic idea for the MLOps framework is shown in Firgure2.

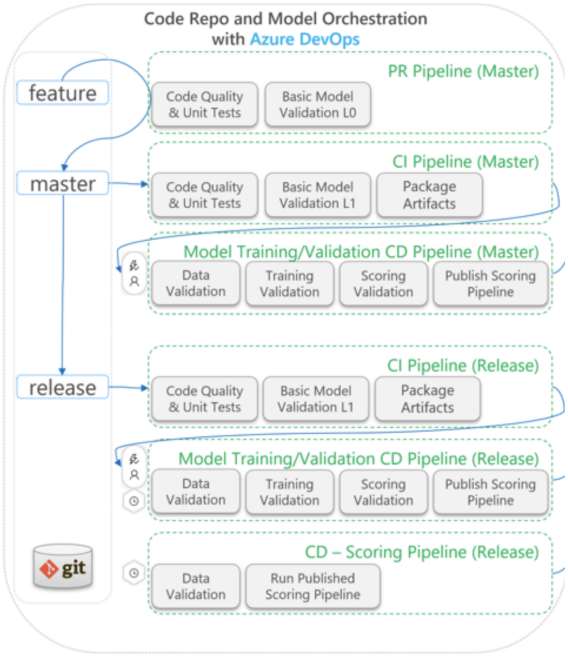

Figure 2: MLOps process flow

It follows Gitflow workflow[11], *feature* branch uses to explore or develop new features in a project, once the feature branch is done, it merges to *master* branch (or you may also have *develop* branch to record the history of a project). Once the milestone for that state is created, we create *release* branch for it. The MLOps process flow is described below.

- When a pull request is opened from a feature branch, the pipeline runs tests that test the code quality (e.g. unit test), as well as model validation test, which test model quality on the mocked data.
- When a pull request merged to master branch, the CI pipeline runs the same tests mentioned above, and it packages the whole project as the artifacts.
- After the artifacts are available, a CD pipeline will be triggered, which validate the end-to-end process on the development machine learning environment, a scoring result (e.g. accuracy result of the ML model) will also be published.

[11]https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

- Once the milestone for that specific state is created, it merges to the release branch, and the same CI/CD pipeline described above will be executed again.

4 Project Implementation

In this section, the project implementation will be described. After identified the tools and requirements for data versioning and the generic lifecycle for the ML project, we will take them into consideration when implementing our approach. First, I will elaborate on the research and general operation we explored for data versioning using DVC. Then, discuss the basic ML workflow or lifecycle, together with data versioning we decided to implement on one of our ML projects.

I will use our ML project as an example. We create a Git repository for training and validating ML model, named *dash-ml-flag-combo*. Another one named *dash-docker-flag-combo* for deploying the model.

4.1 Data versioning - DVC

Version control for ML projects, we need to consider not only the code, but also the data and ML model (i.e. ML artefacts). DVC[12] is an easy to use tool that it works on the top of Git, as we are familiar with Git, hence we decided to explore this tool for data versioning. An example diagram of the project structure is shown below to help better explaining the data versioning. A complete diagram will be shown in section 4.2. The *dvc* part will be generated when implementing data versioning, which will be described in the following bullet points.

During the implementation, I made a demo video to present the general operation of data versioning to my team, which includes the following operations. This demo can be found here.

First you need to install[13] the DVC and initialise it in your ML project Git repository. The scenarios for data versioning we considered and implementation details are listed below.

```
/
└── dash-ml-flag-combo
    ├── data
    │   └── training
    │       ├── training.csv
    │       └── training.csv.dvc ---> dvc file for training.csv
    └── .dvc
        ├── cache
        │   └── a3 ---> a copy of training.csv (stored in dvc cache)
        │       └── 04afb96060aad90176268345e10355
        └── gitignore
```

[12] https://dvc.org/doc
[13] https://dvc.org/doc/install

```
└── config ---> store information about remote storage
```

- **Versioning ML artefacts**: DVC uses a so-called *.dvc file which contains a unique md5 hash[14] to link the dataset to the project. DVC stores the copy of this data file into DVC cache, using the first two letters of the hash as the folder name, shown in the diagram above (i.e. a3). Then use the Git command (e.g. git commit) to record this stage.

```
# Using dvc add to start tracking files and generate .dvc file
dvc add data/training/training.csv
# Using git commit to version the .dvc file
git add data/training.csv.dvc data/.gitignore
git commit -m "Add training data"
```

- **Storing versioned ML artefacts**: DVC supports various cloud storage that enables us to store our dataset or models in remote cloud storage (e.g. S3). In order to do this, first, we need to setup the remote storage using DVC, then upload the ML artefacts to this remote storage. By doing so, we do not need to keep the large dataset in our Git repository, the lightweight, human-readable *.dvc file which contains the link to our real dataset will be kept in the Git repository.

```
# Setup remote storage using dvc
dvc remote add -d [storage_name] s3://[bucket]/[dvc_storage]
# Upload dataset to remote storage S3
dvc push
```

  After executing this command, a remote will be added into DVC config, which looks like:

```
['remote "storage_name"']
url = s3://[bucket]/[dvc_storage]
[core]
remote = storage_name
```

- **Retrieving ML artefacts**: Having DVC-tracked data stored remotely, it can be downloaded to Git repository when needed. As the DVC file stored in Git repository contains the hash to uniquely identify the data and remote information is stored in DVC config, it knows where to find the data on remote storage and download the data to the local project.

```
# Using dvc pull to download data
dvc pull
```

[14] https://en.wikipedia.org/wiki/MD5

- **Making changes on dataset**: When making changes to dataset locally, the DVC add command allows us to track the latest version of the dataset. It will update the md5 hash inside the DVC file and then the new version of dataset has been linked to the project.

```
# Using dvc add to track the new version of dataset
dvc add data/training/training.csv
# Push the new version to remote storage
dvc push
# Record the stage for the new version of dataset
git commit -m "training set updated"
```

- **Switch between versions**: When we want to rollback to a certain version of dataset, git checkout will help us checkout a commit or a revision of DVC file, and then using dvc checkout to synchronize data.

```
# Using git checkout to checkout a commit you want
git checkout <...>
# Using dvc checkout to sync data
dvc checkout
```

4.2 MLOps and lifecycle

Now our ML artefacts are managed by DVC, together with code, they are carefully crafted by Git. We try to establish a basic workflow for our ML project, which enables us to version control the data and model, automated training and validation stages, reproduce the ML lifecycle easily, which leads us to achieve a basic level of MLOps.

4.2.1 Automatically training and validation

Based on the discussion between our team, we revised the project structure, which is shown in the diagram below.

```
/
└── dash-ml-flag-combo
    ├── assets
    │   ├── data
    │   │   ├── training.csv
    │   │   └── validation.csv
    │   ├── data.dvc ---> dvc file for both training.csv and validation.csv
    │   ├── metrics
    │   └── model
    │       └── flag_combo.pkl ---> output of training.py
    └── src
        └── ml_pipeline
```

```
├── train.py
├── validate.py
└── select_feature.py
```

DVC also provides a *dvc.yaml*[15] describes the pipeline of a ML project and can be generated manually or by DVC command and *dvc.lock*[16] file that similar to *.dvc* file contains md5 hash information related to ML artefacts. This enable us easily run or rerun the whole pipeline or certain stages of a ML project automatically. The workflow of establishing the ML pipeline we defined is described below.

- **DVC initialise**: For a new ML project, the DVC needs to be installed and initialized. The remote storage for keeping the ML artefacts should also be initialized in this part.

```
# Install DVC
pip install dvc
# Initialize DVC (from inside of the Git repository)
dvc init
# Setup DVC remote for storing ML artefacts
dvc remote add -d [storage_name] s3://[bucket]/[dvc_storage]
```

- **DVC flow**: The strategy and workflow for generating the ML pipeline will be described here. Dataset which will be used as input of a ML project should be version controlled by DVC, the strategies of using DVC achieving data versioning is described in section 4.1. The output of the ML pipeline will be tracked by DVC automatically (i.e. the trained ML model, flag_combo.pkl). The details about the DVC command and parameters for creating ML pipeline can be found on their official documentation.

```
# 1. Using dvc add to keep track of the initial dataset
dvc add assets/data
# 2. Create dvc pipiline that runs each stage of ML project.
# Stage train. Specify the stage name (train), input (-d),
# output (-o) and followed by a command
dvc run -n train \
     -d src/ml_pipeline/train.py -d assets/data/training.csv \
     -o assets/model/flag_combo.pkl \
     python src/ml_pipeline/train.py
# Stage validate. Output format for metrics and plots is different
dvc run -n validate \
  -d src/ml_pipeline/validate.py -d assets/data/validation.csv \
  -M assets/metrics/accuracy.json \
  --plots-no-cache assets/metrics/matrix.png \
  python src/ml_pipeline/validate.py
# 3. Run git add to include all relevant files to git staging,
```

[15] https://dvc.org/doc/user-guide/dvc-files-and-directories#dvcyaml-file
[16] https://dvc.org/doc/user-guide/how-to/merge-conflicts#dvclock

```
# this will include *.dvc, dvc.yaml, dvc.lock, these dvc artefacts
# will be source controlled by git
git add [use relevant options]
# 4. Git commit, followed by dvc commit to record this stage
git commit -m "MESSAGE HERE"
dvc commit
#5. Git push and dvc push to push code to Git repository and
# ML artefacts to remote storage
git push [revelant options]
dvc push
```

At this point, we have committed to Git and DVC that are linked together. Git commit has all the information we need to reproduce the state of data, model and pipeline related to this commit.

- **Reproduce a state in a point of time**: In the previous stage, we created the ML pipeline and had the git commit that contains everything needed to reproduce the pipeline. Here we will describe how to easily rerun the whole pipeline or a certain stage in the ML project. For example, if dataset changed or the ML algorithm used changes, certain stages can be reproduced easily.

```
# Get a state for the git repository for the certain commit
git checkout -b [branch/commit]
git pull
# Get a state of assets related to this commit by using dvc pull
dvc pull
# If necessary, you could rerun the whole pipeline or certain stage
dvc repro # rerun whole pipeline based on dvc.yaml
dvc repro -p [stage_name] # rerun certain stage
# If necessary, you could compare the metrics by using following command
dvc metrics show
```

4.2.2 Model deployment

After obtained the trained model, we plan to deploy it on AWS Sagemaker and use it in production. From one of the colleagues' previous experience, we plan to use the batch transform[17] of Sagemaker to deploy our trained model. It only supports CSV file and JSON file as input file. The idea of batch transform is that by using simple API, you can run predictions on large or small batch datasets easily, there is no need to break down the datasets into multiple chunks or run prediction in real-time which could be expensive. There is a parameter that allows you to customize the payload size per mini-batch, which means it will load as much as records in the dataset it can to perform the prediction.

[17] https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-batch-transform.html#ex1-batch-transform-api-low-level

We followed an example[18] of how to deploy a custom model on Sagemaker. Our Git repository *dash-docker-flag-combo* follows the same structure. Basically, it starts a Flask service and holds your application and execute prediction job on input data. It also provides a Dockerfile that allows you bundle everything in a Docker image and upload to ECR, then in Sagemaker, you can create a model[19] based on that Docker image from ECR and start a batch transform job based on the model you created where has the prediction logic.

The key parameters[20] we set for Sagemaker to execute the batch transform are listed below in table 1. By doing so, we are able to run prediction on our input data, a JSON file (1.8GB) in 20 minutes.

Key	Value	Meaning
Job name	[your-job-name]	The name of your batch transform job
Model name	[your-model-name]	The name of the model, which loads the image pushed to ECR before with prediction logic in it
Instance type	ml.m5.2xlarge ($0.538 per hour[21])	EC2 instance type you want to use for batch transform job
Instance count	1	The number of instance
Max payload size	5MB	Maximum size allowed for a mini-batch.
Batch strategy	MultiRecord	Specifies the number of records to include in a mini-batch for an HTTP inference request.
S3 data type	S3Prefix	Input data configuration
Split type	Line	The method to use to split the transform job's data files into smaller batches.
S3 URI	s3://[bucket]/[path]	The path for your input data
Content type	application/json	The data format
S3 output path	s3://[bucket]/[path]	The path for storing the output data
Assemble with	Line	Defines how to assemble the results of the transform job as a single S3 object.

Table 1: Key parameters for launching batch transform

5 Reflection

During this project, we are trying to achieve a basic level of MLOps and apply version control to a machine learning project. This is a new experience for me and also for the company. I used to work with machine learning in one of the courses at university but never touched the version control or MLOps for a machine learning project. Before implementing this project, the company used their own ways to manage the machine learning model, which could be inefficient and need a

[18] https://github.com/ritchie46/sagemaker-custom-model
[19] https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateModel.html
[20] https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateTransformJob.html

lot of labor work. In university, courses like Data Mining[22] and Machine Learning for the Quantified Self[23] taught by Professor Mark Hoogendoorn[24], we learned the knowledge to build the machine learning model from end-to-end, which allows me to understand better about the lifecycle of machine learning project. But due to the time limit of a course, we are unable to keep working on the maintenance or go through a lot of iterations of a machine learning project. While in a company, they need to manage the machine learning model and keep improving the result of it in a long term, this is why we want to introduce data versioning and MLOps, which can help us better manage and maintain the machine learning project.

Apart from that, I also learned new skills during this project. DVC which we used for data versioning is a new tool that we have never worked with. My team gave me enough freedom to do research and explore how to use this tool and what is the best practice for versioning a machine learning project. Then I made a demo to show what I have found and created a proposal about the basic workflow of our ML project. Meanwhile, we had several discussions about this within our team to exchange our thoughts and finally decide the best practice that satisfies our needs for versioning our ML project. Sagemaker is also a new tool that we introduced in our team for deploying the ML model. One of my teammates who has rich experience with AWS and I worked together to explore the Sagemaker. As mentioned in section 4.2.2, this involves different technique skills, I have learned Docker and Flask in one of the courses (Web Services and Cloud-Based Systems[25]) taught by Professor Adam Belloum[26] and have basic hands-on experience on them. Through this project, I was able to apply what I have learned at university and combine it with a new skill (i.e. Sagemaker).

We worked as a team during this project. We would have our own tasks but meanwhile, we would work together, for example, at the beginning of this project, we had a brainstorm to define the tasks we need to do in order to achieve the final goal of this MLOps project. Bring our own ideas about the steps we need to follow and then discuss together the priority of each task. As we have a different background, senior data engineer, software engineer and we have different project experiences, this allows us to provide our own opinions and find a compromise way that everyone is satisfied with. I benefit a lot from their rich working or project experience. For example, when defining the standard architecture for our ML project and the structure for saving ML artefacts on remote storage, I did not consider the long-term management of this project and the real situation of it. In theory, the structure and mechanism I provided are clear and easy to manage. But in the real situation, it might not be suitable for long-term management and overcomplicate the way we manage it. Years of working experience and maintaining projects in long run help them know what might happen in the future and what might be the best solution when dealing with problems in a project.

[22] https://studiegids.vu.nl/en/2020-2021/courses/X_400108
[23] https://studiegids.vu.nl/en/2020-2021/courses/XM_40012
[24] https://www.cs.vu.nl/$\sim$ *mhoogen/*
[25] https://studiegids.uva.nl/xmlpages/page/2020-2021-en/search-course/course/79525
[26] https://www.uva.nl/profiel/b/e/a.s.z.belloum/a.s.z.belloum.html

One of the differences I noticed between university and company is that in the company, the project we are working on will be used in production and provides commercial value. Therefore, before actually implementing a project, we need proof of concept and validate the plan with the team leader or even the project lead. Apart from the project itself, during implementation, from the companies' perspective, we need to take cost into consideration. For example, when exploring Sagemaker, we learned from one of the colleagues' experience, Sagemaker provides another way to deploy the model but that is way more expensive and he did not notice that when he uses Sagemaker. So in our solution, we used batch transform which is cheaper and you only pay for how much you use. While in university, there are some cases that we might fail in projects. But failure is the mother of success, as long as we could learn from our failure, it is still a valuable experience.

6 Further research

We applied the data versioning and model deployment on a simple machine learning project. We will try to apply what we have developed so far to other ML projects as well. In the meantime, improve the mechanism for data versioning and try to find the best practices of MLOps that satisfies all our ML projects. As we just introduced these technologies into our company for managing ML projects, other ML projects need time to adapt to the mechanism we defined.

Furthermore, Microsoft established a maturity model[14] that helps users to clarify the MLOps principles and practices. It contains four levels, which shown in table 6 below. Now we are in-between level 2 and 3. We cannot satisfy all characteristics of MLOps requires in such a short time (i.e. two months), now we utilized DVC to build a simple pipeline for training and validating the ML model, which also allows us easily reproduce the ML project. For model deployment, Sagemaker will automatically load the input data, apply the prediction function on it and output the result data. In order to achieve further levels, there is still a lot of work to do. For example, one of the characteristics of level 4 is to automate the entire ML pipeline or lifecycle. Right now we still need manually release the model and deploy that model in production. Besides, as mentioned in section 3.5, there is a CI/CD pipeline needed for ML project that enables us to perform continuous training of the model and lead to continuous delivery of prediction service.

Level	Description
0	No MLOps
1	DevOps but no MLOps
2	Automated training
3	Automated Model Deployment
4	Full MLOps Automated Operations

Table 2: MLOps maturity model from Microsoft (For more details please see here[14])

References

[1] Why Machine Learning? *Https://towardsdatascience.com/why-machine-learning-303e6bdaa29d*.

[2] Creating reproducible data science workflows with DVC., *Https://medium.com/y-data-stories/creating-reproducible-data-science-workflows-with-dvc-3bf058e9797b*.

[3] Data versioning - Stanford Libraries, *Https://library.stanford.edu/research/data-management-services/data-best-practices/data-versioning*.

[4] Data versioning - ANDS, *Https://www.ands.org.au/working-with-data/data-management/data-versioning*.

[5] M. O. Lucy Ellen Lwakatare Pasi Kuvaja, "Dimensions of devops," *Agile Processes in Software Engineering and Extreme Programming*, vol. 212, pp. 212–217, 2015.

[6] What Is MLOps? *Https://blogs.nvidia.com/blog/2020/09/03/what-is-mlops/*.

[7] ML Ops: Machine Learning as an Engineering Discipline, *Https://towardsdatascience.com/ml-ops-machine-learning-as-an-engineering-discipline-b86ca4874a3f*.

[8] J. Klump, L. Wyborn, M. Wu, R. Downs, A. Asmi, G. Ryder, and J. Martin, *Research Data Alliance Data Versioning Working Group Compilation of Data Versioning Use Cases*, Jan. 2020. DOI: 10.15497/RDA00041. [Online]. Available: https://doi.org/10.15497/RDA00041.

[9] T. Berger, M. Chechik, T. Kehrer, and M. Wimmer, "Software Evolution in Time and Space: Unifying Version and Variability Management (Dagstuhl Seminar 19191)," *Dagstuhl Reports*, vol. 9, no. 5, T. Berger, M. Chechik, T. Kehrer, and M. Wimmer, Eds., pp. 1–30, 2019, ISSN: 2192-5283. DOI: 10.4230/DagRep.9.5.1. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2019/11379.

[10] Data on the Web Best Practices, *Https://www.w3.org/tr/dwbp/dataversioning*.

[11] J. Bryan, *Excuse Me, Do You Have a Moment to Talk About Version Control?* Nov. 2017. DOI: 10.1080/00031305.2017.1399928.

[12] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track*, ICSE 2019 Best Paper Award, IEEE Computer Society, May 2019. [Online]. Available: https://www.microsoft.com/en-us/research/publication/software-engineering-for-machine-learning-a-case-study/.

[13] Design a machine learning operations (MLOps) framework to upscale an Azure Machine Learning lifecycle, *Https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-technical-paper*.

[14] Machine Learning Operations maturity model, *Https://docs.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model*.

A Glossary

AI Artificial Intelligence. 1, 5

API An application programming interface is a computing interface that defines interactions between multiple software intermediaries.. 11

AWS Amazon Web Services. 21

CD Continuous delivery or continuous deployment. 6

CI Continuous integration. 6

CI/CD CI/CD generally refers to the combined practices of continuous integration and either continuous delivery or continuous deployment. 5

combo A "combo" is usually two regular products/food items sold together as a combination or "combo" because they taste good together. 1

CSV Stands for 'comma-separated values'. It is a simple file format used to store tabular data, such as a spreadsheet or database. 11

Deliveroo Online food ordering and delivery platform. 19

DevOps DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity. 1, 3

Docker image A read-only template that contains a set of instructions for creating a container that can run on the Docker platform.. 12

Dockerfile A text document that contains all the commands a user could call on the command line to assemble an image.. 12

ECR Elastic Container Registry. 12, 21

ECS Elastic Container Service. 21

flag_combo One of the features generated in flags library, which tells an outlet it sells combo or not (e.g. A burger with fries is a combo). 1

Git repository A repository on Git. This repository tracks all changes made to files in your project, building a history over time.. 7, 8, 12

IAM Identity and Access Management. 21

JSON JavaScript Object Notation is a lightweight data-interchange format. 11, 12

ML Machine learning. 1–7, 9–11

MLOps A compound of "machine learning" and "operations". 1, 2

online platform The online food ordering and delivery platform such as Ubereats and Deliveroo, where we could find the information related to the outlet and menu information. 19

outlet A shop or restaurant that sells the food or drinks on those online platform. 1, 19

S3 Simple Storage Service. 5, 8, 21

sourcing data In our case we call it sourcing data[27], it means raw data scraped from website, describes information about the outlets that exist on the online platform. There are two types of raw data, one contains information related to an outlet (e.g. name, address, country) and another one contains the menu information of the outlets (e.g. category, brand, volume). 19

Ubereats Online food ordering and delivery platform. 19

B DVC related file format

- ***.dvc file**: Here is an example of *training.csv.dvc*.

```
outs:
- md5: a304afb96060aad90176268345e10355
  path: training.csv
```

- **dvc.yaml file**: A yaml file generated based on the command we use in section 4.2.1 which contains two stages: train and validate.

```
stages:
train:
  cmd: python src/dash_ml_flag_combo/ml_pipeline/train.py
  deps:
  - assets/data/outlet_training_set_23102020.csv
  - src/dash_ml_flag_combo/ml_pipeline/train.py
  outs:
  - assets/model/flag_combo.pkl
validate:
  cmd: python src/dash_ml_flag_combo/ml_pipeline/validate.py
  deps:
  - assets/data/outlet_validation_set_30102020.csv
  - src/dash_ml_flag_combo/ml_pipeline/validate.py
```

[27] https://www.talend.com/resources/data-source/

```
metrics:
- assets/metrics/scoring.json:
    cache: false
plots:
- assets/metrics/confusion_matrix.png:
    cache: false
```

- **dvc.lock file**: A dock file generated automatically based on the command we used in section 4.2.1 and the yaml file.

```
train:
cmd: python src/dash_ml_flag_combo/ml_pipeline/train.py
deps:
- path: assets/data/outlet_training_set_23102020.csv
  md5: a3c254dd0904baacc93968bd79faf8e0
  size: 782343
- path: src/dash_ml_flag_combo/ml_pipeline/train.py
  md5: 14b1578f0a9be8169ce85a5e8dfc7d49
  size: 4474
outs:
- path: assets/model/flag_combo.pkl
  md5: c7b5ac139090143fa35b0c19198f879f
  size: 792590
validate:
cmd: python src/dash_ml_flag_combo/ml_pipeline/validate.py
deps:
- path: assets/data/outlet_validation_set_30102020.csv
  md5: 912d6576b001c09633704b0403b446db
  size: 423668
- path: src/dash_ml_flag_combo/ml_pipeline/validate.py
  md5: a8b5a44546e75c9666ca50be5e2a1508
  size: 4127
outs:
- path: assets/metrics/confusion_matrix.png
  md5: e07078d0ffd2959cb64bd81831d17144
  size: 15820
- path: assets/metrics/scoring.json
  md5: 91ecf5bc6583e5a83c933939aa943760
  size: 126
```

C General information

The name of the company is **Dashmote**[28], *Rokin 86, 1012 KX Amsterdam* and I work there as a Data Engineer Intern. Dashmote is a company that mainly focuses on big data analysis to provide services to companies which are in food or beverage market. Some of our clients are Heineken, Coca-cola, etc.
We use data from the Internet to help our clients to understand and predict consumption trends in the current market based on when, where, and what customers post on online platform. The end to end processes of our services is shown in Figure 3. During `Data Acquisition`, sourcing data is collected from various online platform such as Ubereats. The sourcing data contains information about the outlet that exists on that online platform. For example, the name, address, website URL, meal menu, etc. During `Data Services`, according to the requirements from the clients, we developed a customized pipeline that processes the data and head to `Business Intelligence department`, where the result will be visualized in a way that clients can have a better understanding of their brands, sales, and clients. These reports will help our clients to explore the market, control the cash flow and make future business decisions.

Figure 3: End to end process (Figure from Dashmote)

D Data pipeline

The pipeline that our data team is working on is shown in Figure 4. 5 main customized libraries in this figure are: pre-processing, matching-ml, portfolio-*, merging and flags. It shows how the sourcing data will be processed from the beginning to the end. The function of each library will be described in this section.

- **pre-processing**: The sourcing data, which contains information about the outlets, will be pre-processed. Operations like fill the missing value for address (longitude and latitude) will be executed here.
- **matching-ml**: It matches the same outlet that exists on different platforms. For example, two restaurants, through the address or postcode that we know they are the same restaurant, might exist on both Ubereats and Deliveroo.

[28] https://dashmote.com

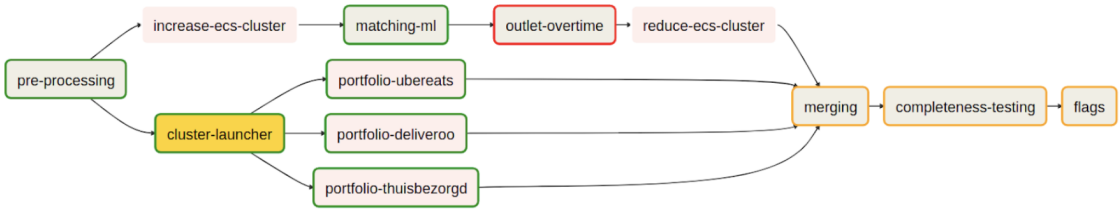

Figure 4: Data pipeline (Figure from Dashmote data team documentation)

- **portfolio-***[29]: It is a library that deals with the data at the drink level. After pre-processed the sourcing data, in the portfolio step, it extracts drinks related information (e.g. drink brand, volume) for each outlet.
- **merging**: It merges all the data mentioned in previous steps and generates the final data product. The final data product contains several tables that describe different domains of the outlet. For instance, one table describes the information related to the outlets (e.g. address), one describes the drinks information and etc.
- **flags**: Flags library is created for providing extra information to clients. The output from the flags is displayed in the visualization part. The idea of this flags library is to assign a flag (0 or 1) to an outlet in a certain domain based on the information we got during the `Merging` step. For example, if an outlet sells alcoholic drinks, then the flag for this outlet in this domain will be 1.

E Format of sourcing data

The example of the outlet related information (sourcing data) is shown in table 3.

address	country	cuisine	...	name	postal_code	source
85 Endeavour, AU	AU	Sandwiches	...	Yachties	2500	deliveroo
Corrimal, Wollongong,AU	AU	Tacos; Wraps	...	Tres Jefes	2500	deliveroo
19 corral,Wollongong,AU	AU	Pizza	...	Blue river	2500	deliveroo

Table 3: Outlet information table format

The example of the portfolio drinks table (generated in `Portfolio` step) is shown in table 4.

F Technologies and services

Table 5 shows the main technologies and services that we are using:

[29]The name after portfolio- means online platform (e.g. ubereats)

alcohol	category	id_source	...	price	source	brand	volume
False	Drinks	96f9c3-34da	...	15.00	deliveroo	Mojo	unknown
False	Congee	0651bd-04ee	...	4.500	deliveroo	Coral	600
False	Modifiers	7d4480-2f54	...	16.80	deliveroo	Heineken	330

Table 4: Drinks information table format

Technologies/ Services	Usage
Docker	It is a set of platform as a service products that use virtualization to deliver software in packages called containers. Our libraries will be run remotely in a Docker container.
AWS S3	A storage service offered by Amazon, which is used to store all the data we have.
AWS ECS	A container orchestration service, which allows us to deploy docker images on it.
AWS ECR	A Docker container registry which is used to store our Docker images.
AWS DynamoDB	A database used to store our data.
AWS IAM	A service that helps us manage access to Amazon services.
Airflow	A workflow management platform, which allows us to view and check the status of each step in our pipeline.
Jenkins	An automation service that automatically tests the code we implement to make sure the quality is good.
AWS Sagemaker	A platform that help developers to prepare, build, train and deploy machine learning models.
flask	A web framework, which provides you with tools, libraries and technologies that allow you to build a web application.
git	A distributed version-control system for tracking changes in any set of files. We use git for managing our code.

Table 5: Technologies used in Dashmote