Milestone 3:

Data Acquisition and Preparation

CSC 5382 - SPRING 2025

By:

Khadija Salih Alj

Supervised by: Dr. Asmar Mourhir

Introduction

Developing a robust and reproducible Machine Learning pipeline is a critical step in transitioning from a monolithic approach to a modular, scalable, and maintainable system. This milestone demonstrates the development of an ML pipeline, focusing on reproducibility, pipeline components, and the challenges faced during implementation. The pipeline was built using ZenML, DVC/Git LFS, Feast, and MongoDB, and run in Google Colab. While the final implementation is functional, the journey was fraught with challenges, particularly with tool compatibility and runtime constraints...

I. Project Modularity

Initially, I intended to include a flowchart and a dashboard of ZenML to visually represent the pipeline's workflow and its components. Unfortunately, these features are not available in the community version of ZenML. Access to the flowchart and dashboard features requires a paid subscription, which costs $99 per month (a price that is not affordable for a student☹).

Despite this limitation, the pipeline was still designed with modularity in mind, ensuring that each component; data ingestion, preprocessing, validation, feature engineering, and versioning, could be developed and tested independently. ZenML was used to orchestrate the pipeline, providing a clear structure and separation of concerns. This modular approach not only improved code readability but also made it easier to debug the pipeline and maintain each stage in isolation.

II. Code Versioning

For code versioning, Git was used to ensure that all changes to the pipeline were tracked. However, since I was working in Google Colab, I kept creating and deleting the repository multiple times while trying to integrate DVC (a lot of trial and error). The pipeline was eventually pushed to a remote GitHub repository, enabling seamless integration with other tools like DVC and ZenML, despite the challenges with the repository setup during the DVC experimentation phase.

III. Data Versioning

Data versioning was implemented using DVC (Data Version Control) to ensure that each processed dataset was saved with a timestamped filename and versioned accordingly. This allowed for each iteration of the dataset to be uniquely identifiable and easily reproducible.

Initially, I attempted to use Google Drive as a remote storage provider for DVC, as it offers an easy/familiar and cost-effective solution for storing datasets. So, I created a project in Google Cloud, enabled the necessary APIs, and set up service accounts with appropriate permissions for DVC to access Google Drive. Additionally, I installed the DVC and DVC-GDrive extensions and configured them to use Google Drive as the remote storage location.

However, I encountered a significant challenge when integrating DVC with Google Drive. Specifically, the gdrive_service_account_json configuration key was not recognized in the DVC configuration file, causing persistent errors when trying to push data to the remote storage. This issue prevented me from successfully completing the integration, as DVC was unable to authenticate with Google Drive. After extensive troubleshooting, I resolved the issue by using the DVC_GDRIVE_SERVICE_ACCOUNT_JSON environment variable instead of the gdrive_service_account_json key in the DVC configuration file. By setting the environment variable to the path of the service account JSON file, I was able to bypass the issue and successfully authenticate DVC with Google Drive.

IV. Experiment Tracking

In the ingestion part of the pipeline, the dataset was loaded into MongoDB for easy retrieval later. This approach ensured that the raw data could be accessed efficiently, providing a centralized location for all the data within the pipeline. MongoDB was chosen due to its flexibility in handling semi-structured data and its ability to scale with the potentially growing volume of my dataset. The data was ingested and stored with a timestamp, ensuring that it was properly versioned and traceable throughout the pipeline. The pipeline was orchestrated using ZenML, which provided a framework for defining, running, and tracking ML pipelines. Each step of the pipeline; data ingestion, preprocessing, validation, feature engineering, and versioning, was encapsulated in a ZenML step, ensuring a clear and reproducible workflow.

Data Ingestion

The data_ingestion step reads data from a CSV file and stores it in MongoDB. Only new entries are added to avoid duplicates, ensuring that the dataset remains up-to-date without redundancy.

Data Preprocessing

The data_preprocessing step cleans and transforms the data, preparing it for downstream processing. During this step, I implemented many transformations to ensure the data was in a consistent and uniform format; First, empty rows were dropped to eliminate any incomplete or irrelevant data, ensuring that only valid records were used in subsequent steps. Additionally, text fields, such as the ingredient names, descriptions, and categories, were converted to lowercase to maintain consistency and avoid discrepancies in future data processing tasks.

The Substitutes column, which contained nested structures, was parsed and transformed to ensure the substitutes were in a structured format. This included extracting relevant details such as the substitute name, equivalent, and any notes associated with the substitutes. These transformations helped standardize the data, making it easier to work with in later stages of the pipeline, such as validation, feature engineering, and storage.

Data Validation

The data_validation step ensures data quality by checking for missing values, validating column types, and detecting anomalies. Custom validation logic was implemented to handle specific requirements, such as ensuring that the Substitutes column is properly formatted.

Feature Engineering

The feature_store_integration step stores processed features in Feast, a feature store designed for ML applications. In addition to features such as the Category, which was stored for future use in model training, I also added embeddings for the Ingredient name and Description. These embeddings generated using the Sentence Transformers model, help capture the semantic meaning of each ingredient, making it easier to perform similarity searches or clustering tasks later on.

Data Versioning

The data_versioning step is designed to save the processed dataset and version it using DVC. The function first generates a timestamped filename for the dataset, saves it as a CSV file, and verifies its existence before proceeding. Once the file is confirmed to exist, DVC is used to version the dataset by adding it, committing the change to Git, and pushing it to the remote storage.

Logically, the code should work as expected. However, when I connected Google Drive as the remote storage provider for DVC, the dvc push command started taking an unusually long time to complete (More than 1 hour). Before linking Google Drive, the DVC versioning worked fine and uploaded the dataset to GitHub, but once the Google Drive connection was established, the process has been significantly delayed, leading to prolonged upload times. This suggests that the issue might be related to the configuration or syncing between DVC and Google Drive.

Unfortunately, I could not pinpoint the exact cause of this delay, but it highlights a challenge with integrating cloud storage as a remote provider in this pipeline ☹.

In addition to using DVC with Google Drive, I also attempted using Git Large File Storage (LFS) to handle the dataset versioning. However, this approach also did not work as expected…

V. Challenges Faced 😥

Tool Compatibility

One of the biggest challenges in developing the pipeline was tool compatibility. Initially, I aimed to use TensorFlow Transform (TFT) and TensorFlow Extended (TFX) for preprocessing and pipeline development. However, I quickly encountered multiple issues. I spent an entire day -literally one full day- trying to make these tools work.

At first, the process took about an hour to download, only to crash afterward. I spent countless hours troubleshooting, trying different approaches to get everything running. After some investigation, I discovered that the issue stemmed from using Python 3.11, which was not supported by TensorFlow Transform and TFX🙃. So, I downgraded to Python 3.9, hoping that would resolve the issue, but unfortunately, the problem persisted.

I did not give up. I tried using Docker, thinking it would help isolate the environment, but still, no luck! Every approach resulted in the same issue. The entire day was spent just trying to get the environment set up properly. As you can imagine, this was incredibly frustrating, and after many failed attempts, I decided to take a different approach.

After all the time spent troubleshooting, I realized that some tools like ZenML, which I had initially hoped to use for the pipeline orchestration, were not supported on Windows. That led me to pivot to Google Colab for running the pipeline. ZenML worked flawlessly in Colab, which allowed me to move forward with the development.

This experience taught me the importance of tool compatibility and the challenges that can arise when working with rapidly evolving ML tools. While it was a significant setback, it also helped me make the decision to shift to a more compatible environment, ultimately allowing me to continue developing the pipeline😊.

Pipeline Runtime

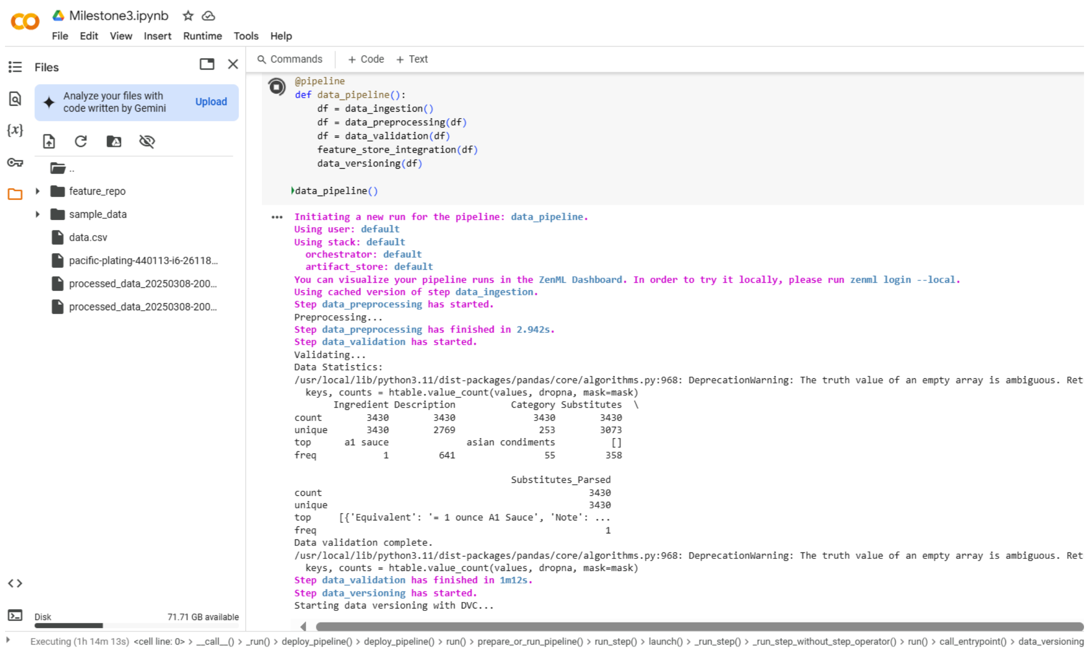
The pipeline's runtime proved to be one of the most frustrating challenges. The overall time to execute the entire pipeline varied significantly depending on the complexity of each step. On average, the pipeline took about 40mn to 2h to complete, with certain steps, such as data versioning and feature store integration, taking longer due to the added complexities of handling embeddings (see latest try in the screenshot below, where it did not fully work after 1h).

In conclusion, the development of this ML pipeline was a challenging experience. Despite the pipeline not functioning as flawlessly as anticipated, key milestones were still accomplished, and valuable lessons were learned along the way. The final implementation is still modular, reproducible, and scalable, meeting the requirements of the milestone. Key achievements include:

- **Modular pipeline design**: Each component was developed and tested independently, ensuring clarity and reusability.
- **Reproducibility**: Git and DVC were used for code and data versioning, ensuring that every iteration of the pipeline could be reproduced.
- **Data quality**: Custom validation logic ensured that the dataset was clean and consistent.
- **Feature engineering**: Features were stored in Feast for future use in model training.

While the pipeline's performance did not align with my initial expectations, the knowledge gained from this experience will be invaluable for future iterations…

(Please, feel free to consult the two notebooks to see the output of my work).


Milestone3.ipynb
File Edit View Insert Runtime Tools Help
Files
Analyze your files with code written by Gemini
Upload
Commands
+ Code + Text
..
feature_repo
sample_data
data.csv
pacific-plating-440113-i6-26118…
processed_data_20250308-200…
processed_data_20250308-200…
Disk
71.71 GB available
Executing (1h 14m 13s) <cell line: 0> > __call__() > _run() > deploy_pipeline() > deploy_pipeline() > run() > prepare_or_run_pipeline() > run_step() > launch() > _run_step() > _run_step_without_step_operator() > run() > call_entrypoint() > data_versioning

```python
@pipeline
def data_pipeline():
    df = data_ingestion()
    df = data_preprocessing(df)
    df = data_validation(df)
    feature_store_integration(df)
    data_versioning(df)

data_pipeline()
```

```
Initiating a new run for the pipeline: data_pipeline.
Using user: default
Using stack: default
  orchestrator: default
  artifact_store: default
You can visualize your pipeline runs in the ZenML Dashboard. In order to try it locally, please run zenml login --local.
Using cached version of step data_ingestion.
Step data_preprocessing has started.
Preprocessing...
Step data_preprocessing has finished in 2.942s.
Step data_validation has started.
Validating...
Data Statistics:
/usr/local/lib/python3.11/dist-packages/pandas/core/algorithms.py:968: DeprecationWarning: The truth value of an empty array is ambiguous. Ret
  keys, counts = htable.value_count(values, dropna, mask=mask)
       Ingredient Description          Category Substitutes  \
count        3430        3430              3430        3430
unique       3430        2769               253        3073
top      a1 sauce              asian condiments          []
freq            1         641                55         358

                                   Substitutes_Parsed
count                                            3430
unique                                           3430
top     [{'Equivalent': '= 1 ounce A1 Sauce', 'Note': ...
freq                                                1
Data validation complete.
/usr/local/lib/python3.11/dist-packages/pandas/core/algorithms.py:968: DeprecationWarning: The truth value of an empty array is ambiguous. Ret
  keys, counts = htable.value_count(values, dropna, mask=mask)
Step data_validation has finished in 1m12s.
Step data_versioning has started.
Starting data versioning with DVC...
```