RazorPay data science case study report

Submitted by: Rohil Pal

Goal

The goal of this case study is to build a robust pipeline addressing data extraction, model development, and system design. The use case is Customer churn prediction which is crucial in the Fintech domain.

Time invested in this case study

Due to my full-time job and a part-time work at a startup that I do during the evenings, I couldn't spend a lot of time on this case study. In retrospect, if I had started before then I would have identified issues with Kubeflow and Airflow sooner and would have enough time to implement the custom approach in a much better way.

Date	Hrs invested	What work did I do?
12/02	1	Assignment received. Read the problem statement and explored the dataset.
13/02	2	Read about Kubeflow and Airflow
14/02	1	Successfully setup Airflow locally
15/02	0	
16/02	4	Wrote the EDA notebook and trained and evaluate 2 models
17/02	14	Tried setting up Kubeflow; finally gave up; decided to go with Airflow
18/02	14	Created a DAG in Airflow but the DAG run never completed. Debug the issue but failed. With only 8 hrs

		remaining, I finally gave up and decided to go with my custom approach
19/02	4	Wrap up, code cleanup, documentation, report and video

How did I approach this problem?

What is required to build a ML pipeline?
In order to build a ML pipeline, it is important -

1. To identify what are the components of the pipeline
2. Implement each component in a modular way so that they can be reused by multiple pipelines
3. Stitch together all the components in the order they should be executed.

Issues with Kubeflow and Airflow
As suggested in the problem statement, I am supposed to -

1. Use airflow to build the pipeline
2. Use kubeflow for managing the ML workflow on a local Kubernetes cluster

- **Kubeflow**
  - Started setting up Kubeflow using minikube
  - Installation of minikube, create a local k8 cluster was straightforward
  - Issue was with deploying kubeflow using kfctl. I believe it was some version compatibility issue between kfctl and kubeflow. I tried finding the solution on many forums including the Kubeflow documentation but couldn't figure out the real problem.
  - Learnt that setting up Kubeflow using MiniKF is simpler. Unfortunately, it requires VirtualBox and is not available for my M1 Macbook Pro. So had to abandon this approach too.
- **Airflow**
  - Setting up Airflow was very straightforward
  - Created a DAG for training and inference
  - Execution was an issue. The DAG run never finished. The error message was "Scheduler is unhealthy".
  - Tried restarting the DAG, uninstalled and installed Airflow but nothing worked
  - Initially I thought it was due to loading of dataframes in memory of the worker. So I created a dummy pipeline with just print statements, but the issue still persisted. So had to abandon this approach too.

Final approach
Since it was important to demonstrate system design understanding in this case study, I decided to create 2 services - training and inference. These services as of now, have 1 endpoint each - /train and /predict respectively.

Training services

- The /train endpoint can be invoked manually, triggered due to a job schedule or events like data drift.
- This endpoint currently accepts a .csv file containing the training data.
- The actual working of the endpoint is governed by the training pipeline.
- As of now, when I say pipeline, it refers to just a sequence of steps (Python methods) that are called in the endpoint. The steps are defined as separate methods under utils/ folder.
- The steps of the pipeline are -
    - *load_data*
    - *Process_data*
    - *Initiate_training*
    - *Logging metrics, models into MLflow*
    - *Register the model and tag it as "Staging". Later when the model performs well in staging environment, it can be promoted to "Production" using MLflow APIs.*
- **Next steps**
    - Modify the /train endpoint to invoke a script that contains end to end training code. This script can be uploaded using an endpoint by the data scientists whenever they update the training pipeline.
    - If I am able to resolve Airflow / Kubeflow issues, then the endpoint can invoke the DAG instead.

Inference services

- The /predict endpoint can be invoked manually, triggered due to a job schedule
- This endpoint currently accepts a .csv file (same schema as training data without the 'Churn' column) comprising of test data.
- The actual working of the endpoint is governed by the testing pipeline.
- Similar to training pipeline, the steps are defined as separate methods under utils/ folder
- The steps of the pipeline are -
    - *Load_data*
    - *Process_data*
    - *Load model from mlflow*
    - *Generate predictions*
- **Next steps**
    - Modify the /predict endpoint to invoke a script that contains end to end inference code. This script can be uploaded using an endpoint by the data scientists whenever they update the inference pipeline.
    - If I am able to resolve Airflow / Kubeflow issues, then the endpoint can invoke the DAG instead.
    - Incorporate CI/CD

Highlights of this approach

1. MLflow integration to manage training experiments
2. Logging of models, metrics and other artifacts like confusion matrix plots to MLflow run
3. Model serving and model training using fastAPI APIs
4. Using MLflow, multiple versions of a model can be maintained and tagged as “Staging”, “Production”. In the current version, every model will be tagged as “Staging”

Model performance evaluation

- I experimented with two basic ML models - Random Forest and Logistic Regression.
- My primary evaluation metric was f1-score. But I am also showcasing there precision and recall so that we know in what scenarios the model is good and bad at?
- I am also showcasing the metrics only for the positive class since we want to identify customers who are at a high risk of churning.

Model	**Precision (class = 1)**	**Recall (class = 1)**	**F1-score (class = 1)**
Random forest	0.51	0.85	0.74
Logistic Regression	0.52	0.84	0.75

What are the next steps to implement?

1. Incorporate Airflow / Kubeflow to build and manage the pipeline.
2. Implement the workflow on deploying a new training pipeline
3. Setup this complete architecture in cloud