

Project Report :

Milestone 2:
Model &
Infrastructure Quality

Github Link :

<https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML>

Ameya Morbale
Ankit Kumar
Prakhar Pradeep
Praveen Ramesh
Kedi Xu

Offline Evaluation

Offline evaluation of our recommendation model involves the following strategy - define an evaluation metric, split data into appropriate train-test-validation splits train the model on train dataset using an appropriate algorithm, validate and test our model using evaluation metric defined, implement steps in place to ensure that common pitfalls in Machine Learning such as overfitting etc. are avoided.

Step 1: Choose Appropriate Metric

RMSE (Root mean squared error)

- i) Measure - predictive accuracy of the model w.r.t user ratings
- ii) Data Collection - collect actual values of user ratings from the test/validation dataset and along with it collect predicted values of user ratings per movie per user for this same test/validation dataset using pretrained model
- iii) Operationalization - we first calculate the difference between the predicted and actual values of user ratings for each item in the test/validation dataset. You then square each of these differences, take the average of the squared differences, and then take the square root of that average

Step 2: Split Data

The data is split into training and testing data, where train data is 75% of the total dataset and test data is 25% of the total dataset (75/25 split). For validation strategy, check details within k-fold cross validation part below

Step 3: Train SVD model on Train dataset

Step 4: Model Evaluation

https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/OfflineEvaluation/Offline_Accuracy.ipynb

Evaluate the model on the test data set using the metric RMSE as described in Step1. However, there might be some common pitfalls that could be encountered while evaluating our model - such as overfitting or bias towards one particular user category. We decided to combat these pitfalls using the following strategy.

- i) K-Fold Cross Validation - we split our data into 5 folds and implemented k-fold cross validation. We found that model results are very consistent across folds suggesting there is no overfitting (last block of code in notebook)
- ii) Evaluation across gender splices - we split test data into male/female and evaluated our model on both testsets. We found RMSE to be fairly consistent across genders (2nd & 3rd last block)

Online Evaluation

Online accuracy is a way of evaluating the quality of recommendations that our model delivers when it is being used in real time, while deployed in production

Step 1 : Define Accuracy Metric

The accuracy metric that we engineered captures the % of users, that were served our recommendations upon request, who actually ended up watching at least one of the recommended movies

Step 2 : Storing model predictions

Our recommendation model receives an incoming stream of recommendation requests. We serve these requests by providing top 20 movie recommendations to every user. We keep the service running for 1800s (30 mins) and while storing our provided recommendations per user served. We shut down the service after 1800s (30 mins) and store these predictions in a csv file. We then execute a sleep command for another 1800s (30 mins)

https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Pipeline/model_online_evaluation.py#L1-L72

Step 3 : Telemetry Data

Post sleep command, we initialize a kafka consumer to collect the latest telemetry data through incoming kafka stream which consists of details about the user such as user ids, movies watched, and time stamp of the movies. We read this incoming stream of data for 1800s (30 mins) and then store it in a csv files

https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Pipeline/telemetry_collection.py

Step 4 : Operationalization (Reporting Online Accuracy)

We combine the csv files from step 2 and 3 (inner join on user ids) to extract details about users served with recommendation requests, movie recommendations provided and movies that were actually watched by the user. The formula for calculating online accuracy then is as follows:

Online Accuracy: $\frac{\text{no. of users who watched at least 1 movie out of 20 given recommendations}}{\text{total number of users (intersection of user_ids in prediction \& telemetry data)}}$

https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Pipeline/model_online_evaluation.py#L90-L121

Results :

<https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/OnlineEvaluation/OnlineEvaluation.txt>

Data Quality

To ensure data quality, we need to make sure that the data schema and types are correct, and we should also check for data drift. First, we should enforce a data schema when loading the dataset from a csv file, after this, we detect potential data drift by computing the statistics of current data.

Step 1 : Enforce Data Schema

We enforce the data schema in the `read_data_from_csv` function, where data types for each feature is specified.

https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Pipeline/data_schema.py#L9

Step 2 : Detect Data Drift

We use statistical methods such as mean and standard deviation to detect data drift. In the function `detect_data_drift`, we compare the statistics of current data with those of the original data, and if the difference is greater than 5%, a warning for data drift will be triggered.

https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/blob/main/Pipeline/data_schema.py#L35

Pipeline Implementation & Testing

Pipeline description

An end-to-end pipeline was created to test and retrain the system with ease. Each part of the pipeline serves a specific purpose and the code is broken down into atomic functions, making it very easy to test out the code. The pipeline consists of the following:

1. Data collection: This file contains the code to read data from the source and saving the data
2. Data processing: This file contains the code for processing the data i.e. getting the top rated movie list as well as test and train data
3. Data schema: This file contains the code for enforcing data schema and checking data drift
4. Model training: This file contains the code for creating, training, and saving the model
5. Model offline evaluation: This file contains the code for calculating RMSE on test data
6. Model serving: This file contains the code for serving the model and converting predictions into the desired format
7. Telemetry collection: This file contains the code for getting the user watch history (telemetry data) from the kafka stream
8. Model online evaluation: This file contains the code for comparing the predictions made by the service and the telemetry data to calculate the metric (Percentage of users who watched at least one recommended movie)

LINK: <https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/tree/main/Pipeline>

Test Suite

Python's unittest module was used to conduct tests for each part of the pipeline. The tests are centralized in a folder called "Tests" which is in the pipeline directory. As mentioned above, each part of the pipeline was broken down into simplest functions (like `save_model`, `read_csv`, `init_kafka_consumer`) and these were tested independently. Each file in the pipeline also had a function called "`PIPELINE_FILE_NAME_pipeline`" which called the atomic functions to serve the purpose of the pipeline. We used the `tempfile` package to create temporary files for testing. We also used the `unittest.mock` to mock the behavior of `KafkaConsumer`. This allowed us to create isolated and controlled test scenarios to ensure the correctness of individual functions.

LINK: <https://github.com/cmu-seai/group-project-s23-The-hangover-Part-ML/tree/main/Pipeline/Tests>

We believe the testing is adequate because it covers the essential functionality of the pipeline and ensures the correctness of the code. By testing each function in isolation, we can identify and fix any issues more easily. Furthermore, by mocking the KafkaConsumer behavior, we can control the input data and focus on testing the processing logic. However, it's essential to note that the tests can always be further improved by adding edge cases, testing for unexpected input values, and considering more complex scenarios

Coverage Report

Name	Stmts	Miss	Cover	Missing

test_data_collection.py	32	1	97%	65
test_data_processing.py	46	1	98%	82
test_data_schema.py	52	8	85%	45-52
test_model_offline_evaluation.py	45	1	98%	73
test_model_online_evaluation.py	58	1	98%	100
test_model_serving.py	39	1	97%	67
test_model_training.py	39	1	97%	71
test_telemetry_collection.py	46	1	98%	85

TOTAL	357	15	96%	

Our testing efforts have resulted in a high test coverage of 96%. This impressive coverage demonstrates that we have thoroughly tested various aspects of the codebase, including edge cases and potential failure points. The extensive testing helps ensure that the implemented features are reliable and the codebase is less prone to bugs or unexpected behavior. Hence, we believe that our testing efforts are adequate

Continuous Integration

For Milestone 2, we have used Github Actions for the CI/CD process. It is an integrated tool within GitHub, to facilitate continuous integration and continuous delivery (CI/CD). We set up workflows for automating the build, test, and deployment of our applications using GitHub Actions. The steps are:








1. Create a new repository ".github/workflows" for our infrastructure code and added a GitHub Actions workflow file to the repository.
2. Define the workflow file to run infrastructure code and execute our tests.
3. Configure GitHub Actions to trigger the workflow file on a specific event, such as a pull request or a push to the main branch. It can be done by adding an "on" section to the workflow file, specifying the event that should trigger the workflow.
4. Test the workflow by making a change to the repository and committing it to the main branch. This should trigger the workflow, which will execute the defined tests.

GitHub Actions is simple to integrate with current development workflows because it supports a wide variety of programming languages, platforms, and cloud services. Additionally, it offers a comprehensive array of tools for managing and keeping track of workflows, such as logs, status checks, and notifications.

Pull Requests

In our team, we follow a structured process for code reviews and pull requests to ensure code quality and maintainability. Here is a brief overview of our process:

1. A developer creates a new branch from the main branch for each feature or bug fix.
2. The developer writes code, runs tests, and commits changes to the new branch.
3. When the feature or bug fix is complete, the developer creates a pull request (PR) to merge the new branch into the main branch.
4. Another team member reviews the PR, checking for code quality, adherence to coding standards, and potential issues. They may also run tests and provide feedback or request changes.
5. The original developer addresses the feedback and makes any necessary changes, and the reviewer checks the changes.
6. Once the PR is approved, it is merged into the main branch, and the new branch is deleted.

<input type="checkbox"/>		0 Open	<input checked="" type="checkbox"/>	6 Closed
<input type="checkbox"/>		ML pipeline	#6 by sudo-prakhar was merged 1 hour ago	
<input type="checkbox"/>		ML pipeline	#5 by sudo-prakhar was merged 3 hours ago	
<input type="checkbox"/>		Added test suites for the pipeline	#4 by sudo-prakhar was merged 3 days ago	
<input type="checkbox"/>		Refactored Pipeline module	#3 by sudo-prakhar was merged 3 days ago	
<input type="checkbox"/>		ML pipeline	#2 by sudo-prakhar was merged 5 days ago	
<input type="checkbox"/>		Refactored Directories	#1 by sudo-prakhar was merged last week	