Project Mile Stone 3

Tianye Song(tianyes), Udaikaran Singh(udaikars), Luís Gomes (lfgomes),
Liangwei Zhang (liangwez), Tushar Vatsa(tvatsa)

April 2022

1 Model Infrastructure

The movie recommendation system infrastructure is illustrated in the graph below.

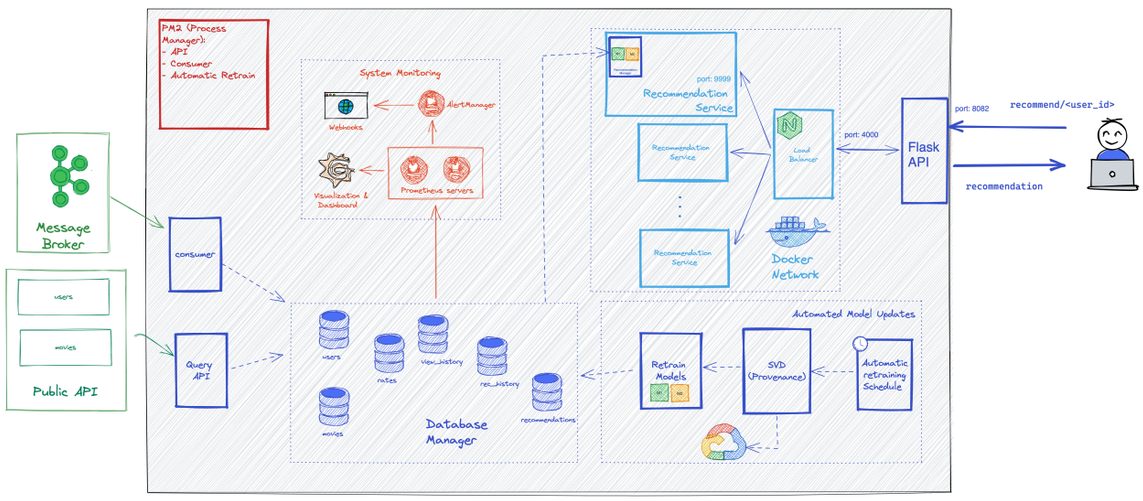

Figure 1: Movie recommendation system infrastructure.

2 Containerization

Our model inference service is containerized as presented in figure 1. In our Docker network we use two images: one for our recommendation service itself and another to the load balancer. The load balancer is based on Nginx, it receives the requests on port 4000 from the Flask API and distributes the requests load in a round-robin fashion among the multiple containers of our recommendation service. The recommendation service containers are created using the command *docker-compose scale recom_service=n*, where n is the number of instances that we want. This architecture allow us to scale the number of requests and also change between models without downtime since each instance can have access through the database to new models that are retrained.

The links for the implementation are the following:
https://github.com/cmu-seai/group-project-s22-dsu/blob/master/docker/Dockerfile
https://github.com/cmu-seai/group-project-s22-dsu/blob/master/nginx.conf
https://github.com/cmu-seai/group-project-s22-dsu/blob/master/docker-compose.yml

3 Automated Model Updates

In this milestone, we used the KNN model for periodical updates. We retrain the model for every three days, each with the latest five days' data for training. For instance, on April 8, we retrained the model by training rating data from April 4 to April 8, while the rating data from April 8 to April 12 was used for training the model on April 12.

To automate the training we use a scheduler that will rerun the training. Since the provenance script (SVD) is responsible for retraining the model and keep track of the correspondent artifacts, we call that script, that indirectly retrain the models. The link for the scheduler (see figure 1) is the following: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/automatic_retrain.py

The retrained models are automatically deployed through our production database, from where our containers read.

4 Monitoring

Prometheus and Grafana are used to set up the monitoring infrastructure for the movie recommendation system as illustrated in the infrastructure graph 1 . Prometheus is used to store metrics data from the

system and Grafana, which use Prometheus as data source, is used to visualize the metrics on dashboard. (https://github.com/cmu-seai/group-project-s22-dsu/blob/master/monitor/prometheus.yml)

The recommendation service availability 2 and model quality 3 are monitored at this stage. For recommendation service availability, the request latency and total request count are monitored. The total past request count and past 24h request count are calculated separately. The request latency of different value is presented in histogram and the percentage of request which latency less than the maximum request time for all the past request and past 24h is calculated.(https://github.com/cmu-seai/group-project-s22-dsu/blob/master/monitor/api_monitor.py)

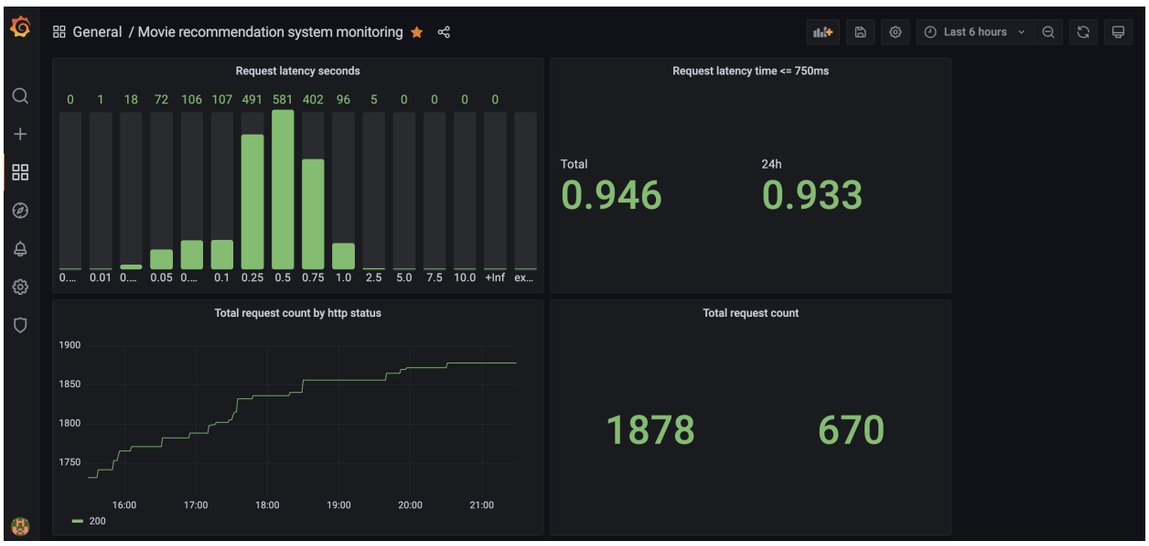

Figure 2: Screenshot for Grafana movie recommendation service monitoring.

For model quality or the quality of the prediction, the online evaluation metrics are monitored. The metrics details are in Mile Stone 2 online evaluation part. (https://github.com/cmu-seai/group-project-s22-dsu/blob/master/monitor/model_quality_monitor.py)

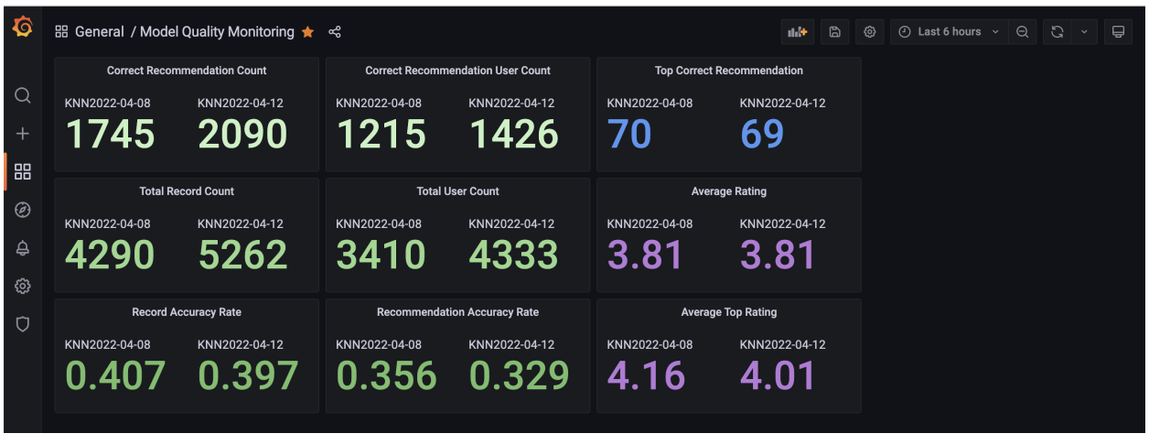

Figure 3: Screenshot for Grafana model quality monitoring.

An alert for the model request latency is set by using Alert manager of Prometheus 4and it will set alerts when the percentage of the latency of request greater than the maximum time 750ms exceeds 20% and send notification on webhooks. (https://github.com/cmu-seai/group-project-s22-dsu/blob/master/monitor/first_rule.yml)

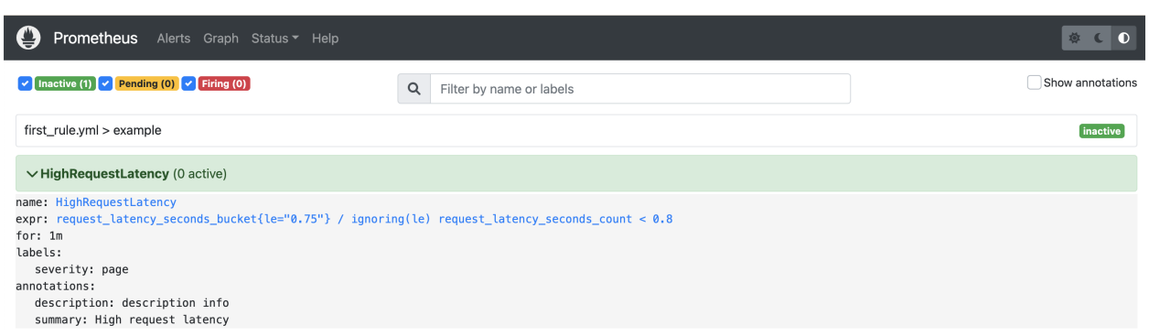

Figure 4: Screenshot for Prometheus alert manager.

To access the Prometheus service at the machine, please use the following URL using a tunnel from the

machine: http://128.2.205.106:9090/targets

To access the Grafana service at the machine, please use the following URL using a tunnel from the machine: http://128.2.205.106:3000/?orgId=1

username: admin, password: admin

5 Experimentation

In our experimentation infrastructure, we have a pipeline for AB-testing in which we look to perform an experiment allowing for requests to go towards one of two models. This experiment is performed across 5 hours and records the data by the hour.

For online experimentation, we deployed two KNN models (M1 and M2, see figure1),with the same hyper-parameters but different training data. The older version of KNN model uses the rating data from April 4 to April 8, while the new version uses the data from April 8 to April 12. We continuously record the user request since April 13, splitting the data by user ID. We define that the older model provides recommendations if the first digit of the user ID is odd and the new model provides recommendations otherwise. Since it is online evaluation, we performed the online evaluation in the real time once per hour. From 11.30 a.m to 11.30 p.m, we have obtained 15 results, each includes the user requests from the beginning of April 13 to the time we performed evaluations then. The key metrics used to evaluate the quality of the models includes record accuracy, recommendation accuracy, average ratings, and top average ratings. A link to the implementation can be found here: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/ABtesting.py.

5.1 Record Accuracy

Record accuracy is obtained by dividing the number of recommended movies users have watched by the total number of movies users have watched.

$$\text{Record Accuracy} = \frac{\text{No. recommended movies watched}}{\text{No. movies users have watched}}$$

Models	11:30a.m.	12:30p.m.	1:30p.m.	2:30p.m.	3:30p.m.	4:30p.m.
old KNN (Apr 08)	40.91%	40.46%	40.31%	40.54%	40.46%	40.73%
new KNN (Apr 12)	39.45%	39.22%	39.24%	39.44%	39.23%	39.53%

Table 1: A portion of online evaluation record accuracy on April 13

5.2 Recommendation Accuracy

Recommendation accuracy denotes the ratio of users watching the recommended movies on the total number of users requesting. recommendations.

$$\text{Recommendation Accuracy} = \frac{\text{No. users watching the recommended movies}}{\text{Total Number of users requesting. recommendations}}$$

Models	11:30a.m.	12:30p.m.	1:30p.m.	2:30p.m.	3:30p.m.	4:30p.m.
old KNN (Apr 08)	35.56%	35.12%	35.27%	35.53%	35.30%	35.53%
new KNN (Apr 12)	33.46%	33.09%	33.45%	33.55%	33.22%	33.58%

Table 2: A portion of online evaluation recommendation accuracy on April 13

5.3 Average Rating Score

We measure the quality of the recommendation by calculating the rating scores of those movies. We evaluate the average rating of those recommended movies which have been watched and rated by users.

$$\text{Average Rating Score} = \frac{\text{Total rating scores for recommended movies watched}}{\text{No. recommended movies users have watched}}$$

Models	11:30a.m.	12:30p.m.	1:30p.m.	2:30p.m.	3:30p.m.	4:30p.m.
old KNN (Apr 08)	3.8143	3.8227	3.8124	3.8050	3.8073	3.8003
new KNN (Apr 12)	3.7913	3.7956	3.8027	3.8151	3.8148	3.8084

Table 3: A portion of online evaluation Average Rating Score on April 13

5.4 Average Top Movie Rating Score

We also focus on the evaluation of our top recommendation, which is vital for users' impression on our recommendation system. Therefore, we also calculate the average rating for the top recommendations.

$$\text{Average Top Movie Rating Score} = \frac{\text{Total rating scores for the top recommended movies watched}}{\text{No. top recommended movies users have watched}}$$

Models	11:30a.m.	12:30p.m.	1:30p.m.	2:30p.m.	3:30p.m.	4:30p.m.
old KNN (Apr 08)	4.0	4.04	4.069	4.0857	4.075	4.1136
new KNN (Apr 12)	4.0	4.0571	4.027	4.0465	4.0455	4.0417

Table 4: A portion of online evaluation Average Top Movie Rating Score on April 13

We performed statistical testing on those metrics. The t-test indicates how significant the differences between different groups are and also reports whether those differences could happen by chance. The t-test we are using is a basic t-test between the two groups we test in the AB-testing. Inherently, the t-test is a test of a centrality measurement; in this case, we are testing if the mean of metrics associated with the two models are statistically different. We are also assuming that the mean of the two groups are normally distributed with equal variance; this is a necessary assumption in our experimentation infrastructure because we are not performing a pair-wise test. A pair-wise test is inappropriate for this setting, because we cannot give different recommendations to the same user at a given time. The general implementation can be found here: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/statistical_testing.py.

Also, in our experimentation infrastructure, we make a random split of users between the A and B groups (https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/ABtesting.pyL198-L212). Our model measure statistics like recommendation accuracy, the average top ratings, and number of recommendations coming into the system (https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/ABtesting.pyL22-L79).

Since it is not convincing enough to compare two models by just one evaluation result at some timestamp, we utilized the set of evaluation results in different timestamps for the online evaluation on Apr 13. We set up the null hypothesis that the mean value for those metrics on the old model are not better than the new model, where the alternative hypothesis is that the old model has a better metric than the new model. The results are shown in the following.

Attributes	Record Accuracy	Recommendation Accuracy	Average Rating	Average Top Rating
t-value	13.4104	23.0386	-0.5017	4.1399
p-value	0.0000	0.0000	0.6899	0.0002

Table 5: Statistical Test for different attributes

```
model1 data on record_accuracy [0.4091, 0.4046, 0.4031, 0.4054, 0.4046, 0.4073, 0.4051, 0.4057, 0.406, 0.4014, 0.4035, 0.4026, 0.4068, 0.4061]
model2 data on record_accuracy [0.3945, 0.3922, 0.3924, 0.3944, 0.3923, 0.3953, 0.3961, 0.395, 0.3945, 0.3948, 0.3974, 0.3974, 0.3981, 0.3968]
The mean of model1 data on record_accuracy 0.4050928571428571
The mean of model2 data on record_accuracy 0.3950857142857144
t-test 13.4103949721
p-value 0.0000000000
we reject null hypothesis that KNN2022-04-08 is not better than KNN2022-04-12 on record_accuracy
model1 data on recommendation_accuracy [0.3556, 0.3512, 0.3527, 0.3553, 0.353, 0.3553, 0.3561, 0.3526, 0.3546, 0.3539, 0.3528, 0.3515, 0.3543, 0
.3543]
model2 data on recommendation_accuracy [0.3346, 0.3309, 0.3345, 0.3355, 0.3322, 0.3358, 0.336, 0.3322, 0.3272, 0.3291, 0.3292, 0.3283, 0.3285, 0
.3275]
The mean of model1 data on recommendation_accuracy 0.35380000000000006
The mean of model2 data on recommendation_accuracy 0.33153571428571427
t-test 23.0385735417
p-value 0.0000000000
we reject null hypothesis that KNN2022-04-08 is not better than KNN2022-04-12 on recommendation_accuracy
model1 data on average_rating [3.8143, 3.8227, 3.8124, 3.805, 3.8073, 3.8003, 3.8002, 3.7947, 3.7961, 3.8038, 3.8074, 3.8106, 3.8126, 3.8135]
model2 data on average_rating [3.7913, 3.7956, 3.8027, 3.8151, 3.8148, 3.8084, 3.8129, 3.81, 3.804, 3.8123, 3.8128, 3.8175, 3.8115, 3.8125]
The mean of model1 data on average_rating 3.8072071428571435
The mean of model2 data on average_rating 3.808671428571429
t-test -0.5017340282
p-value 0.6899615180
we accept null hypothesis that KNN2022-04-08 is not better than KNN2022-04-12 on average_rating
model1 data on average_top_rating [4.0, 4.04, 4.069, 4.0857, 4.075, 4.1136, 4.125, 4.098, 4.0893, 4.1, 4.0968, 4.1111, 4.1449, 4.1571]
model2 data on average_top_rating [4.0, 4.0571, 4.027, 4.0465, 4.0455, 4.0417, 4.06, 4.0577, 4.0755, 4.0702, 4.0508, 4.0317, 4.0147, 4.0145]
The mean of model1 data on average_top_rating 4.09325
The mean of model2 data on average_top_rating 4.04235
t-test 4.1399340330
p-value 0.0001621468
we reject null hypothesis that KNN2022-04-08 is not better than KNN2022-04-12 on average_top_rating
```

Figure 5: Screenshot for T test on different attributes.

Let the confidence interval as 5%, it can be seen that for record accuracy, recommendation accuracy, and average top rating, the p value is far less than the confidence interval, we reject the null hypothesis on those metrics. For average rating score, the p value is far above the confidence interval, which indicates that we will not reject the null hypothesis that the new model has a better performance on average rating score than the old models. Above all, in the whole picture, it is not sufficient to replace the old model with the new one for better recommendation quality.

6 Provenance

In Machine Learning, is it possible to debug an issue? Let's say we want the model to behave in a certain way but for some particular input we get a bizarre output. Will it be possible to track the issue or in other words reproduce the error? That's exactly where Provenance comes into picture. There is a lot of non-determinism which arises from distributed training and stochastic processes involved in machine learning. Provenance helps to keep track of the version of the data, version of the model, the pipeline and everything that is needed to reproduce the same output for the same input.

6.1 Data Provenance

It's not possible to track the changes in the dataset using git repository because the data weighs several gigabytes. To version data files, we are using dvc. It produces versioned pointers to files instead of the files themselves. These files are finally stored in a local cache and this cache can be synchronized with a remote storage. We are using google drive linked to (andrew id : tvatsa) for actual storage of the data. The data through the Kafka stream is being stored continuously in our mongoDB database. We use K Nearest Neighbors to produce recommendations. The data pointer can be stored using the following command :

```
$ dvc run -n KNN$1 -d KNN$1/data.json -d KNN$1/ratings.csv -o
      KNN$1/model.pickle python3 KNN_recommendation.py $2
```

Before running this command, we need to make sure that we have dvc installed on our virtual machine. We also need to run the dvc init command to initialize a DVC project in the current working directory. In Fig 6, as it can be seen that the data has been stored on the google cloud and its pointer has been stored on github(data provenance link). The file is stored based on the timestamp(KNN_num_2022-04-12) and the label which will be passed as the input to the svd.sh file. We are distributing the load using a load balancer and splitting the data based on whether the user_id is less than 5000 or greater than it. So, basically two inputs will be provided to the svd.sh file from the scheduler which are : 1. The timestamp 2. Label of the model.

```
schema: '2.0'
stages:
  KNN_1_2022-04-14:
    cmd: python3 ml/KNN_recommendation.py 1
    outs:
    - path: KNN_1_2022-04-14/data.json
      md5: 827bb0c41af2f24e13d6373f09a9b3c9
      size: 1976192
    - path: KNN_1_2022-04-14/model.pickle
      md5: b3ac4444fefb67686fcdf31fd1812f05
      size: 381638976
    - path: KNN_1_2022-04-14/ratings.csv
      md5: fb73aa70dd2298bec34c5a2367295c33
      size: 1670293
```

Figure 6: The dvc yaml file stored on git

In Figure 5, the name "KNN_1_2022-04-14" denotes that the data was trained on 14th April, 2022 using the first model. The model version can be found in the requirements.txt on github commit for that particular date. Using the dvc push command, we are storing the actual data on the google drive. Here is the link to google drive

6.2 Model Provenance

We are using K Nearest Neighbors trained on different data sets as two different models. The inputs are splitted based on whether user_id is above a certain threshold. A scheduler sends the timestamp and the label of the model to the svd.sh file which further runs the training script. Once the training script is run, a .pickle file of the model is generated. It is stored in the pointer file as can be seen in Fig 6. So, it can be pulled from the google drive in case we need to reproduce the output for a particular input. Since it carries the timestamp and the model label(model1 or model2), the exact output can be reproduced easily. For example If the inference was done for a particular user on "20th-April-2022", and the user_id is greater than 5k, we can conclude that the model2 was responsible for it and the model parameters can be obtained from the folder KNN_2_2022-04-20.

6.3 Pipeline Provenance

The pipeline is stored on the git and we don't need to store it on dvc. Since we are training the model everyday, so the pipeline code, the hyper parameters and all the library versions can be tracked using the commit of that particular day on github. The Figure 7 shows the Google drive folder where all the data, outputs and models are being pushed to. google drive

6.4 Provenance Summary

Here is the example of a user_id: 16635. Since, the user_id is greater than 5000, the inference was generated by model 2. The time stamp associated with this user_id is "April 12, 2022". And the top 10 recommendations generated by the model are following :

- the+lives+of+others+2006
- battle+royale+2000
- the+crow+1994

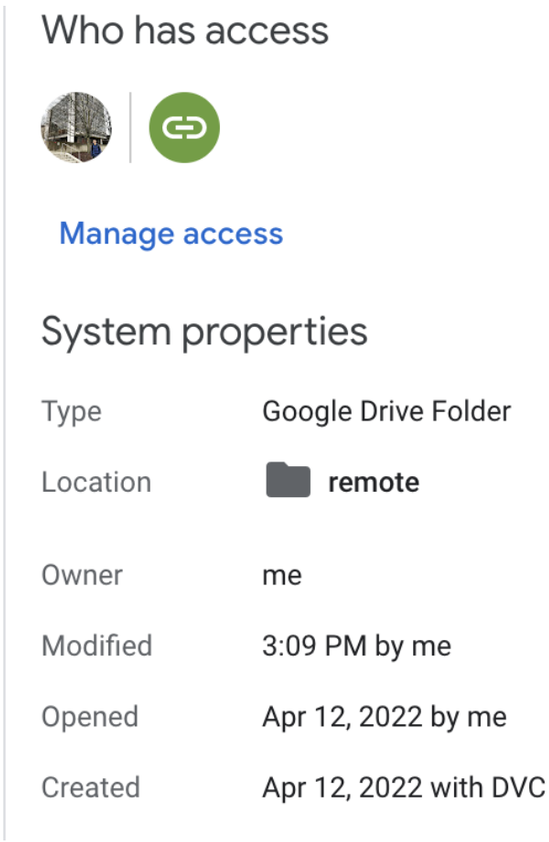

Figure 7: The actual data/model/outputs stored on google drive

- a+guide+to+recognizing+your+saints+2006
- yi+yi+2000
- metropolis+2001
- grave+of+the+fireflies+1988
- another+year+2010
- a+love+song+for+bobby+long+2004
- prayers+for+bobby+2009

The data and the model can be tracked and pulled from the folder on dvc remote storage using the timestamp and the pipeline can be found out from the git commit for that particular day.