Project Mile Stone 1

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

February 2022

1 Learning

Basically, the system utilizes two sources of data. The first source is from Kafka event stream which provides the user behavior of watching and rating movies, and the second Source is from the public API which provides the user and movie information. Besides, we save these data in database and call for it when training the model.

1.1 Data

On this first approach to the problem, we use only the user rating information. This information can be represented in a tabular format, where each element represents the rating that a specific user assign to a particular movie. The rating records serve as an essential part to make recommendations, since the model will obtain the neighbor users with similar tastes based on the rating records. The ratings data is formatted as a sparse matrix of (user, movie) pairs. The ratings serve as a measurement of how much a user enjoyed the movie and serves as the basis for how we make recommendations.

1.2 Models

Based on the collected data, our group implemented two machine learning models based on Collaborative Filtering, since it is the most adequate method to solve recommendation problems. One of the models use K-NN and the other was matrix factorization using Singular Value Decomposition (SVD). The models were trained by a grid-search over hyperparameters; in the case of K-NN, we used a grid-search over the K values and for matrix factorization, we searched over values for the learning rate, number of epochs, and regularization rate.

1.2.1 K Nearest Neighbors

K Nearest Neighbors (K-NN) is a machine learning algorithm to find clusters of similar users based on ratings on similar movies. Then it predicts the ratings based on the average ratings of its top-k nearest neighbor users. The approach makes the assumption that people with similar characteristics share similar taste. For instance, suppose Bob and Mike have seen many movies together already with identical ratings. Therefore it is likely that they make similar ratings for the same movies in the future.

K-NN is a non-parametric learning method without making assumptions on the distribution of the data. It applies feature similarity such as cosine similarity to calculate the "distance" between users, and then obtains the top k nearest neighbor users for movie recommendations. It serves a common baseline in recommendation system, since it does not require parameter settings but only the hyper parameter k needed.

The link to the implementation of the learning step is the following: https://github.com/cmu-seai/group-project-s22-dsu/blob/knnmodel/ml/KNN.pyL74-L84 .

1.2.2 Matrix Factorization with SVD

We used a matrix factorization model trained using SVD. In this setting, given a ratings matrix of users and movies, we can predict the ratings of the unknown pairs of (user, movie) by decomposing the ratings matrix into a lower-rank matrix approximation. Typically, this is represented as breaking the ratings matrix down into a user-matrix and movie-matrix. For example, suppose the rating matrix is of size (n,m). The user and movie matrices will be respectively (n,k) and (k,m) such that we can reconstruct an approximation of the ratings matrix in rank k.

Collaborative Filtering is suited for this task, because it allows us to make approximations of (user, movie) pairs that are missing within the original rating matrix. By learning this lower-rank approximation, the model learns the relationships between different movies and users. This allows us to estimate a ratings for an unseen (user, movie) pair which can then be used to make recommendations.

The link to the implementation of the learning step is the following: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/cf_svd.py

2 Model Comparison

We measured five qualities for this recommendation system. In this section we present those qualities, the respective metrics and the way we operationalized them. A table with the values is presented in the model selection subsection.

Prediction Accuracy

Generally, for prediction accuracy, since it is tricky to directly classify whether the recommendation is correct or not, we use Root Mean Square Error (RMSE), Mean square error (MSE), and Mean Absolute Error (MAE) to serve as metrics to evaluate the performance of the models.

The formula for RMSE serves as

$$RMSE = \sqrt{\frac{1}{|R|} \sum_{r'_{ui} \in R} (r'_{ui} - r_{ui})^2}$$

The formula for MSE serves as

$$MSE = \frac{1}{|R|} \sum_{r'_{ui} \in R} (r'_{ui} - r_{ui})^2$$

The formula for MAE serves as

$$MAE = \frac{1}{|R|} \sum_{r'_{ui} \in R} |r'_{ui} - r_{ui}|$$

We import the **surprise.accuracy.rmse**, **surprise.accuracy.mse**, and **surprise.accuracy.mae** method to calculate the metrics directly for the predicted ratings. The link to the implementation of this metric for KNN is the following: https://github.com/cmu-seai/group-project-s22-dsu/blob/knnmodel/ml/KNN.pyL80-L83 . The link to the implementation of this metric for collaborative filtering using SVD is the following: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/cf_svd.pyL79-L87 .

Training Cost

The time that the models need to train is used as a metric. This includes the model initialization and fitting on the training data. To get this metric we calculate the difference between system times acquired before and after training. The python function **time()** from **time** library provides a way to get those times. The link to the implementation of this metric for KNN and CF using SVD is respectively the the following: KNN and Collaborative Filtering using SVD.

Inference Cost

The metric that we use as inference cost is the maximum, minimum and average inference times of queries per period of time. In this case, we calculate it on a daily basis and this document reports metrics for the day *09 February 2022*. Other important metrics, such as the 90, 95 and 99 percentiles where also discussed and can be operationalized in further versions of the service. We store in our database the recommendations history (collecting also the *response_time* and *status*) and we get the metrics with direct queries (those queries can be seen here: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/mongodb/commands.md)

The inference cost would be the same for both models since we store the predicted results in memory and retrieve them to the user on demand.

Model Size

The used metric is the memory of the model in the system. We implemented by saving the trained models to files by pickle and then calculating the file size by os. For model size, it is related with the number of movies and number of users involved. It can be seen that the K-NN model takes up much more memory than SVD model. The link to the implementation of this metric for KNN is the following: https://github.com/cmu-seai/group-project-s22-dsu/blob/knnmodel/ml/KNN.pyL101 . The link to the implementation of this metric for Collaborative Filtering using SVD is the following: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/ml/cf_svd.pyL135 .

2.1 Model Choice

Based on the metrics presented on Table 1 , we would choose the SVD collaborative filtering as the better model for our recommendations. In all the metrics, we found that the SVD collaborative filtering model was able to produce lower scores for each of the error metrics. This means that the predicted ratings are closer to the ground truth in the collaborative filtering model. Moreover, K-NN method suffers from a larger training cost, and it cannot hold the increasing size of training data as well as SVD model.

Therefore, when producing rankings and recommending movies to users, it is more likely that the rankings are appropriate and produce better recommendations. A trade-off we found is that the collaborative filtering model takes more space in memory than the K-NN model. This makes sense, because we are storing 2 dense matrices of floats, while the K-NN model just requires us to store the original ratings matrix, which is a sparse matrix of integer values. This difference in memory usage seems to advantage the K-NN model; however, the memory of the collaborative filtering does not scale with increased training set size. Therefore, as we gather more data, the collaborative filtering will eventually become more optimal in terms of memory usage.

Table 1: Metrics

Quality	Measurement	Models	
		K-NN	SVD
Prediction Accuracy	Root mean square error	0.7485	0.7032
	Mean square error	0.5603	0.4985
	Mean absolute error	0.6072	0.5573
Training cost	Training time (s)	27.8330	2.3041
Inference cost	Average Inference Time	130.1877	
	Maximum Inference Time	864	
	Minimum Inference Time	16	
Model size	Memory (byte)	4086721740	35177568

3 Prediction Service

The infrastructure of the whole recommendation system is briefly described in Figure 1.

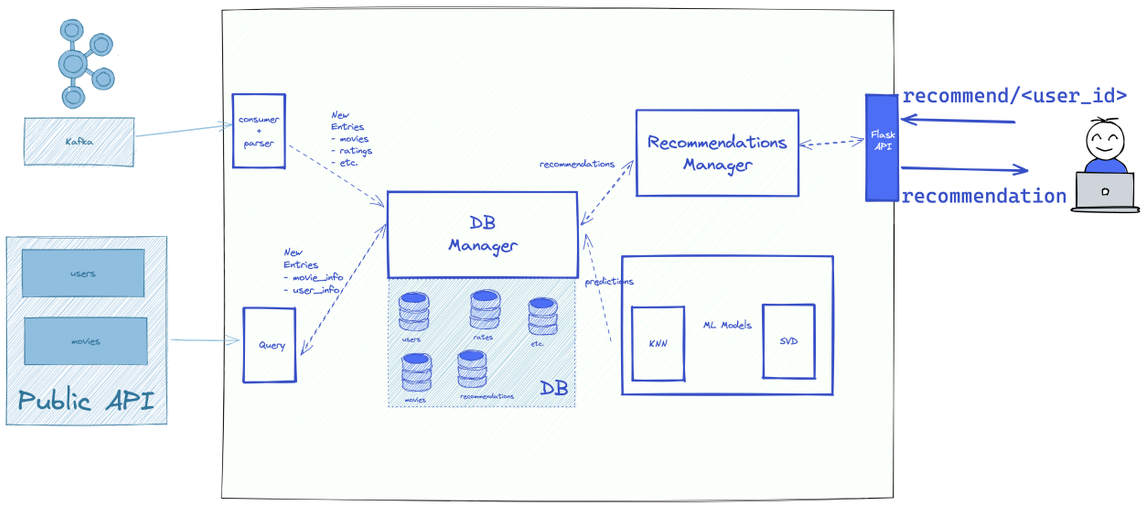

Figure 1: System Infrastructure

We build the model inference service that provides movie recommendations. In the prediction phase, for each user, we predict the ratings for all movies and then rank them by the predicted ratings. Then we choose the top 20 movies by its predicted score. If the user has never rated any movie before, which is a cold-start problem, we randomly return 20 of the top-50 popular movies. Popular movies are chosen by the vote average ratings among those most voted movies, which takes both most viewed and high quality into consideration. In detail, we first sort the movies by "vote count" to extract the top 100 popular and most viewed movies. Then among those movies, we sort by "vote average" to extract the top 50 movies with highest vote rating scores.

Regarding the prediction service implementation, we pre-compute the recommended result for all users stored, and store the result in the database. The recommendations manager is responsible for loading the model on startup and use the predictions every time a new request is made. Request are handled by the Flask API on port 8082.

To save metrics and continuously acquire data to improve the models, our system saves all past recommendation history and new users and movies (acquired from the kafka broker).

The links to the implementation of each part of the system involved on the recommendation service are the following:

- *Flask API*: https://github.com/cmu-seai/group-project-s22-dsu/blob/master/flask/api.py;
- *Recommendations Manager*: https://github.com/cmu-seai/group-project-s22-dsu/blob/knnmodel/recom_manager.py.
- *DB Manager*: https://github.com/cmu-seai/group-project-s22-dsu/blob/knnmodel/db_manager.py

4 Team process and meeting notes

Our team create a Slack work space (Figure 2) and use it as the main communication channel. Our team organization reflects the architecture of the system. We divide the work by the infrastructure of the system, assigning the team members according to their experience related to each component. Besides, we use GitHub and Trello 3 to manage and assign personal tasks. We had at least one meeting each week to track progress and record what has achieved. We use Google document for meeting summary.

By dividing the work based on the infrastructure in Figure 1, for the first week, one person is responsible for Kafka stream, one person is responsible for database establishment, one person is for flask API, and one person is for public API data acquirement. For the second week, two people are responsible for the two models, one person is responsible for the database and Kafka and one person is responsible for the report. We came up with five measurement metrics together.

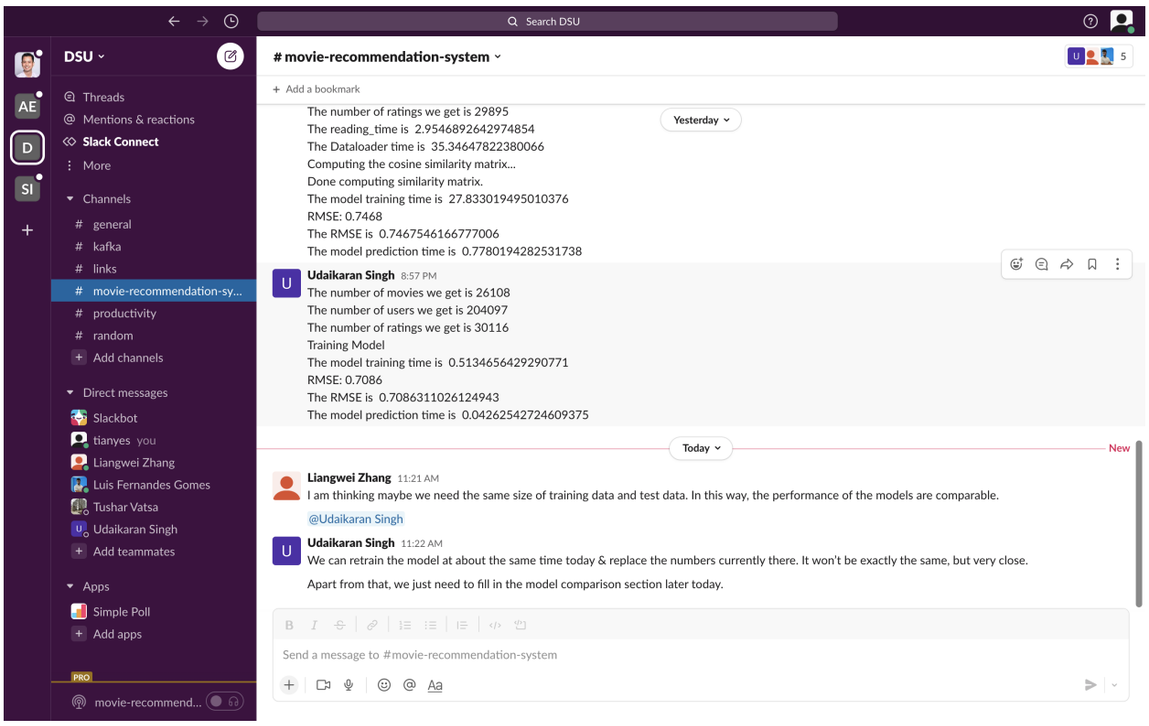

Figure 2: Slack

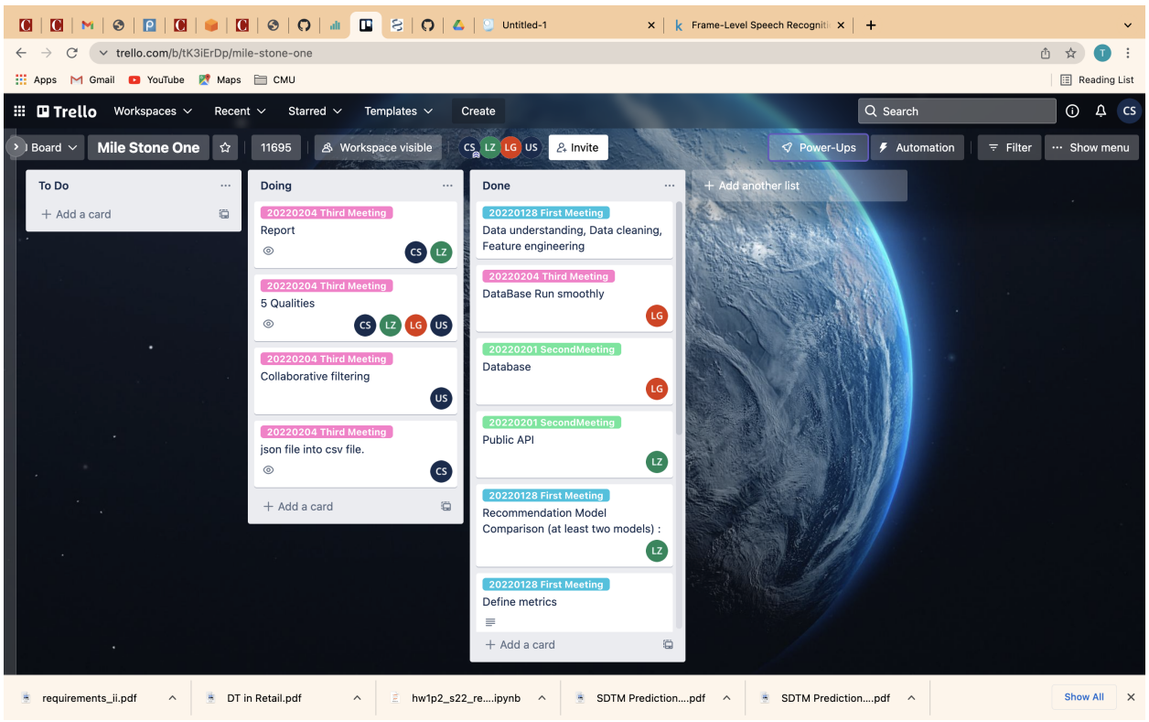

Figure 3: Work assignment