Team 3 - Milestone 1

1 - Learning

In this first milestone, we chose to use the Matrix Factorization technique, taking into account only the scores assigned to films by users. This is a Collaborative Filtering technique, which is the process of evaluating items using other people's opinions. The input data is a 'user x item' matrix populated with the associated rating. Matrix Factorization decomposes the user-item interaction matrix into the product of two smaller rectangular matrices [1]. To predict the response of user 'u' to item 'i', we calculate the dot product between the embedding vectors and consider the result of the corresponding user and item (in other words, it predicts how well matched this user and item are).

More specifically, we used Alternating Least Square (ALS), a Matrix factorization algorithm implemented in Apache Spark ML. The term 'Alternating' refers to the fact that ALS tries to minimize two loss functions alternately. This means, for example, that it can start by keeping the user array fixed and then deriving the loss function with respect to the item vectors, and secondly, keeping the items array fixed and then deriving the loss function with respect to the item vectors. It continues alternately until convergence.[2]

These first choices led us to discard, at first, information potentially relevant to the prediction model, such as user interaction with movies (time spent by users watching a specific movie). However, the technique used is considered promising and easy to scale. Additionally, keeping the machine learning part of the milestone fairly simple allowed us to focus some of our time on better understanding the project's requirements, understanding how to interact with the data pipeline through Kafka, and setting up the infrastructure around the project: REST API and Docker containers.

The training dataset used for the model consists of 200,000 unique user-movie ratings composed of 25,528 unique movies. The model was coded and trained on Google Colab [see Colab file] and saved in our team's GitHub repo [Team-3].

2 - Inference service

Given the feedback matrix A ∈ R(m×n), where m is the number of users and n is the number of movies, the model learns, from the matrix factorization technique:

- a user embedding matrix U∈R(m×d), where row i is the embedding for user i.
- a movie embedding matrix V∈R(n×d), where row j is the embedding for movie j.

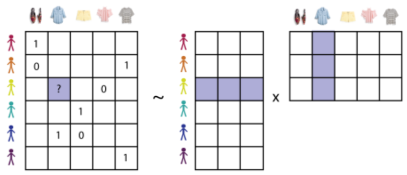

Fig 1: Matrix Factorization [3]

The number in our matrix indicates the ratings from the users predicted by the model for a given movie based on past records. The model's performance is reported by root mean square error, where we're using the predicted rating minus the actual rating, and square the result can get the RMSE. A smaller RMSE indicates that the model works as expected and vice versa.

We executed a grid search for some hyperparameters: rank, that is the number of latent factors in the model (it corresponds to the matched dimensions in the user and the item embeddings matrices), and regParam, which is a regularization parameter in ALS algorithms. The best options were 200 and 0.01 respectively.

Training the model took approximately two hours on GCP with 200,000 data from Kafka steam. The input for the service is a user id. The system makes rating predictions on all movies, from the dot product operation, and the output will be a list of the top 20 movies with the highest predicted rating for the given user. The ranking indicates which movies will be considered to have a higher score than others based on the user's previous preference, which means they are more recommended for the user to watch and give a score for it. Additionally, our current system utilizes caching by pre-computing and storing recommendations for each known user and later looking up the recommendations for a new API request. Since each new recommendation request with the current model takes over a minute to process, caching saves a lot of time and computation.

So far, we haven't implemented any solution to the 'cold start' problem, which occurs when we need to recommend items to new users, i.e., users who don't exist in the training dataset. The current model randomly recommends movies for these new users. To address these limitations of the current system, we've considered optimizing the model, and even using other ML techniques for the upcoming milestones. We did not yet use user feedback for model retraining. In the future, we will use the new ratings and user behavior to update the system and have more accurate recommendations.
Link to our Jupyter notebook for the ML model:
https://colab.research.google.com/drive/1iVHJefQsbeaO-fQPvrUnMpB2aLxcwZSY
Live API active on our virtual McGill server: http://fall2022-comp585-3.cs.mcgill.ca:8082/

3 - Team process and meeting notes:

The team was divided into two groups: (1) Software Engineering, and (2) Machine Learning. Of the 5 team members, Kua Chen and Shanzid Shaiham were assigned to Software Engineering, and Marcos Souto Jr., Zeyu Li, and Barry Li were assigned to Machine Learning.

The Machine Learning group focused primarily on deciding which ML technique to use, creating the ML model, and training/optimizing the model. The Software engineering group focused on setting up the infrastructure around the ML model, which consisted of creating pipelines to gather training data, setting up the REST API, and setting up Docker images for the project.

The team used Slack as the primary communication channel with additional tools like Google Colab and Google Docs to collaborate online. Additionally, weekly in-person team meetings helped the team share ideas and progress from each group and plan the next steps.

Meeting notes (written by Zeyu Li):

1. *September 23rd*:
   The first in-person meeting. Divide further works based on individual strengths. Discuss what algorithm should be used for training and the input data format to train the model.

   Barry Li, Marcos Souto Jr., and Zeyu Li are responsible for building the model. Two models are planned, and one of them will be selected in a future meeting to hand in for milestone 1. Shanzid Shaiham and Kua Chen are responsible for collecting the data from Kafka and clearing the data to make sure useful parts are preserved for training the model.

   Two models are built with data collected. One of them will be used further.

2. *September 28th*:
   Meeting in person. Select the ‘matrix factorization’ model for training and hand it in for the first milestone. Decide how much data we should feed into the model. Talk about the functions of the model and see if it can fulfill needs for the current stage and how to interpret the data we have for the moment.

   Collect everyone’s ideas on further actions for the feedback loop and how the model will grow when we get new data inputs. We agreed on how the model should grow; as new data are fed into the model, we separate a smaller segment that only works on the new data while the bigger model is preserved.

   The model is trained with 20,000 data inputs after the meeting.

3. *September 30th*:
   Meeting in person. Set up the software infrastructure with the Flask framework and integrate it into the recommendation model. Finalized the Docker containers and how to set them up in the cloud VMs. Discuss the remaining tasks to do for this milestone, the current limitations of the model, and how to improve the recommender model.

References

[1]Liao, Kevin. Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering. Available at: https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1

[2]Rosenthal, Ethan. Explicit Matrix Factorization: ALS, SGD, and All That Jazz. Available at: https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea

[3]Le, James; Ororbia, Alexander G. From Matrix Factorization To Deep Neural Networks: The Evolution Of Collaborative Filtering Solutions for Recommendation Systems