Recommender system on the Movielens dataset

**1 Abstract**

Recommender systems have become ubiquitous in our lives. Yet, currently, they are far from optimal. In this project, we attempt to understand the different kinds of recommendation systems and compare their performance on the MovieLens dataset. We attempt to build a scalable model to perform this analysis. We start by preparing and comparing the various models on a smaller dataset of 100,000 ratings. Then, we try to scale the algorithm so that it is able to handle 20 million ratings by using Apache Spark. We find that for the smaller dataset, using user-based collaborative filtering results in the lowest Mean Squared Error on our dataset.

**2 Introduction**

A recommendation system is a type of information filtering system which attempts to predict the preferences of a user, and make suggests based on these preferences.

There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google.

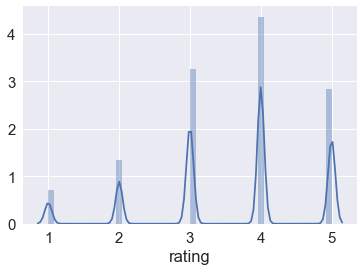
Due to the advances in recommender systems, users constantly expect good recommendations. They have a low threshold for services that are not able to make appropriate suggestions. If a music streaming app is not able to predict and play music that the user likes, then the user will simply stop using it. This has led to a high emphasis by tech companies on improving their recommendation systems. However, the problem is more complex than it seems.

Every user has different preferences and likes. In addition, even the taste of a single user can vary depending on a large number of factors, such as mood, season, or type of activity the user is doing. For example, the type of music one would like to hear while exercising differs greatly from the type of music he’d listen to when cooking dinner. Another issue that recommendation systems have to solve is the exploration vs exploitation problem. They must explore new domains to discover more about the user, while still making the most of what is already known about of the user.

In this assignment, our group built a movie recommender system to help recommend or predict different genres of movies to people.

**3 Methodology**

The dataset we applied is a one-million movielens dataset. Its number of rating record is 1,000,209, rating score is from 1 to 5, total number of users is 6040 and total number of movies is 3883. The distribution of the user ratings is as the following image.

****

In our assignment, we applied **Latent Factor Model** to predict the possible rating a user may give to a movie. Besides, we also applied **Model-Based Collaborative Filtering** and **Deep Learning/ Neural Network** to make the same prediction and the compare the MSE to prove that our performance is better than the two methods listed. The detailed introduction is the following chapters.

**3.1 K-Latent Factor Model**

A latent factor model is a [statistical model](https://en.wikipedia.org/wiki/Statistical_model) that relates a set of [observable variables](https://en.wikipedia.org/wiki/Observable_variable) (so-called *manifest variables*) to a set of [latent variables](https://en.wikipedia.org/wiki/Latent_variable).

It is assumed that the responses on the indicators or manifest variables are the result of an individual's position on the latent variable(s), and that the manifest variables have nothing in common after controlling for the latent variable ([local independence](https://en.wikipedia.org/wiki/Local_independence)).

And in our case, K-latent-factor model takes the relation between users and items into considerations. The equation is:

Where and are and matrix. is the hyper parameters, is the number of users and is the number of item.

Our objective function now becomes:

Here, we use **Stochastic Gradient Descent** **(SGD)** to optimize this objective function. The derivative function for is as follows:

In our task, wo set appropriate parameters *K*, to help decrease the MSE and therefore improve the performance.

**3.2 Collaborative Filtering**

For each user, recommender systems recommend items based on how similar users liked the item. Let’s say Alice and Bob have similar interests in video games. Alice recently played and enjoyed the game "Legend of Zelda". Bob has not played this game, but because the system has learned that Alice and Bob have similar tastes, it recommends this game to Bob. In addition to user similarity, recommender systems can also perform collaborative filtering using item similarity (“Users who liked this item also liked X”).

The detailed steps are as follows:

1. We measured the pearson similarity between users and objects and use the cosine similarity to measure the similarity between a pair of vectors.

As for measuring the similarity between different users, the pearson similarity function is following:

The is the rating by user *u* and the is the rating by user *v*. Besides, the means that this item is rated both by user *u* and the user *v.*

2.We applied model-based collaborative filtering to identify similar users or items.

Model-based Collaborative Filtering is based on matrix factorization (MF)which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. The detailed steps are as follows.

We have an   matrix consisting of the ratings of n users and m items. Each element of the matrix (i, j) represents how user i rated item j. Since we are working with movie ratings, each rating can be expected to be an integer from 1-5 (reflecting one-star ratings to five-star ratings) if user i has rated movie j, and 0 if the user has not rated that particular movie.

And then for each user, we want to recommend a set of movies that they have not seen yet (the movie rating is 0). To do this, we will effectively use an approach that is similar to weighted K-Nearest Neighbors.

For each movie j user i has not seen yet, we find the set of users **U** who are similar to user i and have seen movie j. For each similar user u, we take u‘s rating of movie j and multiply it by the cosine similarity of user i and user u. Sum up these weighted ratings, divide by the number of users in **U**, and we get a weighted average rating for the movie j.

Finally, we sort the movies by their weighted average rankings. These average rankings serve as an estimate for what the user will rate each movie. Movies with higher average rankings are more likely to be favored by the user, so we will recommend the movies with the highest average rankings to the user.

Instead of doing evaluation manually by ourselves, we used the python Suprise Library that provided various ready-to-use powerful prediction algorithms including (SVD) to evaluate its RMSE (Root Mean Squared Error) on the MovieLens dataset. It is a Python scikit building and analyzing recommender systems.

**3.3 Deep Learning/ Neural Network**

The idea of using deep learning is similar to that of Model-Based Matrix Factorization. In matrix factorization, we decompose our original sparse matrix into product of 2 low rank orthogonal matrices.

But different from the previous collaborative filtering, in deep learning implementation, we don’t need them to be orthogonal, we want our model to learn the values of embedding matrix itself. The user latent features and movie latent features are looked up from the embedding matrices for specific movie-user combination. These are the input values for further linear and non-linear layers. We can pass this input to multiple linear or sigmoid layers and learn the corresponding weights by any optimization algorithm (Adam, SGD, etc.).

The basic code is based on the approach outlined in [Alkahest's blog](https://medium.com/@james_aka_yale/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223). We built our basic model based on the code.

Then we compile the model using Mean Squared Error (MSE) as the loss function and the AdaMax learning algorithm. After that we trained the model and predicted the ratings a random user will give to a random movie. Finally we applied the freshly trained deep learning model for all the users and all the movies, using 100 dimensional embeddings for each of them.

During the training process above, we saved the model weights each time the validation loss has improved. Thus, we used that value to calculate the best validation Root Mean Square Error. By comparing our RMSE, we can achieve the conclusion that which model performs the best.