This file is a brief presentation of the implementation of the recommendation techniques that are proposed in the paper “Optimizing parallel Collaborative Filtering approaches for improving recommendation systems performance” published in 26 April 2019 on *Information* 10, no. 5.

Table of Contents

Introduction

The problem

In this project we are trying to implement and execute the SVD++ recommendation algorithm on the epinions dataset. This dataset is consisted of four columns, which represent the userIDs, the itemIDs, the ratings and the timestamp. Only the three columns are used, since the timestamp column is dropped. So, every line of the dataset states that a specific user rated a specific item giving a rating, thus we have a rating tuple (userID, itemID, rating). All the epinions dataset contains 13.668.320 lines-rating tuples. Obviously, this a large dataset, so the SVD++ algorithm is executed on smaller input samples (1000, 5000, 20000, 50000, 100000, 300000, 1000000, 2000000, 3000000, 4000000, 8000000 lines). The main purpose of this project is to improve the prediction accuracy of SVD++ and secondary to reduce the total time of execution. For this reason we implement 7 alternative “modes”.

General Approach

In general, the implementation of SVD++ follows these stages:

1. Input sample preprocessing -> 2. Train and Test set creation -> 3. Model Training -> 4. Model Evaluation

At the first stage the input sample is used unchanged or it is splitted to 2 subsamples and maybe some extra ratings are added, depending on the “mode”. At the next stage the input sample (or subsamples) are separated in order to create train and test set. Then a SVD++ model is trained using the train set. Finally, the model is evaluated by comparing the ratings that were predicted by the SVD++ to the actual ratings of the test set. The stages 2, 3 and 4 are repeated 10 times for every sample (or subsample) because we follow the 10 fold cross validation procedure.

Core concepts

The exceptions - fallback problem

During the stage of model training, the SVD++ model creates a graph that contains all the userIDs and itemIDs included in train set. This graph is used in order to evaluate the model. During the evaluation, we ask from the SVD++ model to predict a rating for the userIDs and itemIDs of the test set. The model can make prediction only when the given userID and itemID are both included in the graph. Otherwise, an exception is thrown.
There are **three ways** to deal with this problem:
- when both userID and itemID are not included in the graph, then we “fall back” and put as predicted rating the mean of the predicted ratings that were calculated normally, without the SVD++ to throw an exception.

- when the userID is included in the graph but the itemID is not, then we “fall back” and put as predicted rating the mean of the normally predicted ratings of this particular user.
- when the itemID is included in the graph but the userID is not, then we “fall back” and put as predicted rating the mean of the normally predicted ratings of this particular item.

Also, we follow a **fourth approach** to the “exceptions - fallback problem”. In mode “3” (see below), during the stage of train and test set creation, we ensure that the train set will contain every distinct userID and every itemID at least one time. Thus, it’s impossible to have an exception thrown when we predict a rating because every userID and itemID is included in the SVD++ graph. Of course, this way of creating the train set violates the classic 10 fold cross validation, since the input sample is not splitted into 10 equal parts and then every time one of them is chosen as a test set and the remaining nine parts are used as train set. However, we manage to deal the problem, even if we don’t implement strictly the cross validation.

Most Active Users, Selection, Least Rated Items and Top Users

Most Active Users -> We consider as “most active users” (MAU) the users who have given the most ratings. The “modes” 4, 5, and 6 (see “Explanation of the “modes””) are based in the use of MAU. The main idea is that if we replicate the ratings of MAU to the two subsamples, then the overall prediction of ratings will be improved because more information about ratings is provided to the SVD++ model.

Selection -> It’s a term that is connected to MAU and the “modes” that use MAU (“modes” 4,5,6). When the “selection” is chosen in our program, then we replicate only the MAU ratings for items that already exist in each target subsample. For example, if a MAU found in the first subsample has rated the items i1, i2 and i3 and in the second subsample exists only the i2, then only MAU’s rating for i2 is replicated.

Least Rated Items -> In “mode” 7 (see “Explanation of the “modes””) we use the term “least rated items” (LRI) for the items that have received the least ratings. We believe that increasing the ratings of these LRI inside the two subsamples will provide more information for these items and thus our SVD++ model will make better predictions on their ratings.

Top Users -> Also, in “mode” 7 we use the term “top users”. It refers to the users who have rated the most the LRI and tells us how many ratings for these items we will finally replicate to the two subsamples. For example, if we have 10 “top users”, this means that we will use the ratings of the user who has given the most ratings to LRI, the second user who has done this, and then the third, the fourth until the tenth user. We could say this is a kind of “selection” of ratings like it happens in “modes” 4, 5 and 6 where we can select some of the ratings of the MAU.

Explanation of the “modes”

There are 7 “modes”:

1 -> The input sample is separated to train and test set. Then the train test is used for the model training and the test is used for the evaluation of the model. This procedure is repeated 10 times (for every fold of the cross validation).

3 -> The procedure is the same with mode ‘1’ but it differs in the way it deals the “exceptions - fallback problem”, as it is stated in the previous section.

The next “modes” preprocess the input sample by splitting it randomly in the half and creating two subsamples with equal size. Then every “mode” treats the two subsamples differently:

2 -> During this mode, the procedure is the same with mode ‘1’ but instead of using as input the total sample, we use the two subsamples. The stages of mode ‘1’ are executed for the first subsample and then are executed for the second subsample.

4 -> In this mode after the input split into two subsamples, we add rating tuples (userID, itemID, rating) to the subsamples by finding the “most active users - MAU”. First, we search the MAU in the total sample and then we find their ratings in the two subsamples. We copy the ratings found in the first subsample to the second subsample and vice versa.

5 -> In this mode we add the ratings of MAU to the subsamples but the procedure is a little different. We find the MAU of the first subsample (MAU1) and the MAU of the second one (MAU2). Then we search for the MAU1 ratings in the first subsample and we copy them to the second subsample. Also, we search for the MAU2 ratings in the second subsample and we copy them to the first one.

6 -> This mode is similar to mode ‘5’. After finding MAU1, we search for MAU1 ratings in second subsample and we copy them to the first subsample. Correspondingly, we search for MAU2 ratings in first subsample and we copy them to the second subsample.

In modes 4, 5 and 6 we can apply “selection” of ratings meaning that we replicate only the MAU ratings for items that already exist in each target subsample. Thus, we have the **simple** and the **“selected” 4, 5 and 6 “modes”**.

7 -> Contrary to modes ‘4’, ‘5’ and ‘6’, in this mode we focus on items and specially on the least rated items (LRI). First, we find the least rated items in the first subsample (LRI1) and we do the same for the second subsample (LRI2). Then we search for ratings of LRI1 into the second subsample and we copy them to the first one. Finally, we search for ratings of LRI2 into the first subsample and we copy them to the second one.
It’s important to point out that we don’t copy all the ratings of the LRI1 and LRI2. As it has previously been mentioned in “Most Active Users, Selection, Least Rated Items and Top Users”, we use the concept of “top users” meaning that we select some of these ratings based on the ranking of the users by number of ratings. So, we select the ratings of the users who have given the most ratings.

Project and the published paper

In our paper “Optimizing parallel Collaborative Filtering approaches for improving recommendation systems performance” three approaches are presented: Active Users All Ratings (AUAR), Active Users Selected Ratings (AUSR) and Informative Users Selected Ratings (IUSR). Based on the section “Explanation of the “modes””, we can see that AUAR is the “mode” 5, AUSR is also the “mode” 5 but with selection and IUSR is the “mode” 7. Obviously, these “modes” have the best performance in terms of RMSE and this is the reason that the other “modes” are not included in the paper.

Comments on the code

Terminology

- Influencers - Most Active Users -> In the code of our program, the MAU are defined as “influencers”. This happens because initially, during the development of the code, we called the MAU as “influencers” and we adopted the term “MAU” at the end. So, the two terms refer to the same thing.

- Exceptions - Fallbacks -> There are some cases that the SVD++ can’t make a prediction and throws an exception and then we “fall back” (see “The exceptions - fallback problem”). It’s obvious that the exceptions and fallbacks are related strongly. For this reason, we use exclusively the term “exceptions” for these cases in our code. The term “fallback” is used only in our result text files.

Configuration Parameters

hw_architecture -> Our program is designed to run on 3 different hardware architectures that we call “localhost”, “singlenode”, “cluster”.
first_sample -> It’s the sample that is used first.
last_sample -> It’s the sample that is used last.
folds -> It’s the number of folds in 10-fold cross validation. If we choose a number less than 10, then the 10 fold CV will be executed only for the times indicated by this number and will end uncompleted.
input_size -> Represents the sizes of the input samples.
dataset -> The dataset name (in this case is epinions).
kFactors -> The number of latent factors in SVD++.
k_inf -> The number of “influencers”.
first_mode -> The mode that is used first.
last_mode -> The mode that is used last.
create_subsamples -> It defines if new subsamples will be created (1) or not (0).
charactInput -> It’s a characterization for input file.
charactOutput -> It’s a characterization for output file.

selection -> Selection or not - replicating only the ratings for items that already exist in each subsample (1) or not (0).
top_users -> number of the users who rated the least rated items - if we choose 6, then we'll take the user who has rated the most of the least rated items, the 2nd "top user" who rated these items, the 3rd "top user"... until the 6th "top user"
top_least_rating -> It’s used as limit in function "findLeastRatedItems". It defines the rating range for the LRI (least rated items), starting from 1 rating until this limit - [1,top_least_rating]

Result files

The program creates **three result files**:
- a general result file (variable “pw”) which includes all the information that was produced by the program (RMSEs, Execution time, Input size, M.A.U. added, exceptions thrown e.t.c.)
- an exception file (variable “pw_excep”) that contains information about the exceptions which are thrown by SVD++.
- a plot information (variable “pw_plot”) which stores only the useful data for our plots.