Data Mining Project - Netflix challenge

Prof. Pieter Libin - Dr. Denis Steckelmacher

pieter.libin@vub.be
denis.steckelmacher@vub.be

1 Project Aim

For the Information Retrieval and Data Mining course, you are asked to write a project that performs recommendations on the Netflix challenge dataset. This dataset contains a list of movies, and a large amount of ratings, that users gave to the movies. Your task is to write a user-user recommendation system that will predict ratings for unseen movies and users.

You will work together in groups of two individuals, **please report your group composition at the start of the WPO** on Wednesday, 20 April, 2022. If you can't find a team mate, contact Prof. Pieter Libin via e-mail to find a match (`pieter.libin@vub.be`). **Working students** do not have to form groups (they can, they don't have to).

You can use any programming language that you want, but we list important constraints on algorithms and code reuse later in this assignment. Regardless of the programming language, you will be evaluated on the clarity (and documentation) of your code, in addition to its correctness. We do not directly evaluate efficiency (compute speed), but particularly efficient code will lead to bonus points.

2 The Netflix Challenge dataset

The dataset is available on Canvas. It is a zip file that uncompresses to about 2 GB of text files. A README in the zip explains what the structure of the files is. For this project, you only need one kind of files, the `combined_data_X.txt` files. Their structure is as follows:

```
3:
1711859,4,2005-05-08
1245640,3,2005-12-19
7:
808731,4,2005-10-31
337541,5,2005-03-23
```

Lines that contain a number and a colon indicate a MovieID. The following lines list, for that particular movie, UserIDs, a comma, a rating, a comma, then the date when the rating was done (for this project, we will ignore the date).

These files are therefore not CSV files! You will have to write your own parser for these files, to create a matrix of Movies $\times$ User, with the rating given by the user for the movie in each cell.

Hint: The dataset is somewhat large, and processing it takes time. In Task 1, we explain how you can truncate the dataset when debugging your project, for quicker run-times. For the final version of the project, don't hesitate to run your code on the machines in the VUB computer rooms. They have Python, NumPy and SciPy installed, and have all the computational power to successfully complete this project.

For this project, you will have to perform the 4 tasks described below.

Task 1: Loading the Dataset

The dataset contains 480K users (whose IDs can go much higher than that) and 17K movies. If you were to create a Numpy matrix of 480K by 17K floats, it would consume 30 GB of RAM! It is not practical, and for this course on scalable analytics, this kind of brute force approach is **not allowed**. Instead, you have to parse the Netflix Challenge dataset into a sparse matrix, that only uses memory for cells that have a non-zero value.

1. Load the Netflix dataset into 3 lists, one of movie IDs, one of user IDs, and one of ratings (the i-th element of each list indicates that the i-th user provided the i-th rating for the i-th movie)

2. Create two sparse matrices from this data: one that has MOVIES rows and USERS columns, with the contents of the cells being the ratings given by the users for the movies, and one that has USERS columns and MOVIES rows. The next two tasks have different requirements regarding whether the rows or columns indicate a movie.

You are allowed to use any sparse matrix library of your choice. We recommend the use of SciPy, a library built on top of Numpy (sometimes used during the practicals), that has a sparse matrix class. This task therefore becomes parsing the Netflix dataset into a Scipy sparse matrix.

Hint: When writing your project (debugging it), don't hesitate to truncate the dataset. For instance, load only the first 1000 ratings from the first file.

Task 2: DIMSUM

The dataset can be summarized by performing a Singular Value Decomposition. At the core of SVD is the computation of $A^T A$, the transpose of the sparse matrix created in Task 1 multiplied with the sparse matrix itself.

When A is large, computing $A^T A$ using a standard matrix multiplication is impossible, making the full SVD operation impossible. The DIMSUM algorithm (Lecture 4, slide 121 and onward, more details in[1]) allows to approximate $A^T A$ in a scalable way. It is a MapReduce algorithm, but you don't have to use Hadoop or anything like that. You can simply implement the Map and Reduce steps of the algorithm in pure Python (or any other programming language).

1. For this task, consider a matrix A that has the users in rows (each row is a different user), and movies in columns (each column is a movie). This is a tall (many users) skinny (few movies) matrix, on which DIMSUM is designed to operate.

2. Implement the DIMSUM algorithm, that will map your skinny A to a matrix of cosine similarities B, of size $|\text{movies}| \times |\text{movies}|$.

3. Compute an approximation of $A^T A$ from B. Slide 118 of Lecture 4 will help you.

4. Compute the exact $A^T A$ operation on the tall skinny sparse matrix. You can use any matrix library here (such as SciPy.sparse). Compare the resulting $A^T A$ matrix with the approximated one (for instance by computing the average MSE over all the entries). **Write these results in your report**, and analyse the impact of γ (DIMSUM parameter) on the performance of DIMSUM.

You have to implement the pseudocode given in the slides yourself. You can use the `random` Python library to generate random numbers, and NumPy and SciPy to access the sparse matrix elements, perform summations, and compute vector norms.

Task 3: (Stochastic) Gradient Descent with Latent Factors

Task 3 is independent from Task 2 and does not use its output (these are two separate exercises in the project). For Task 3, you have to write code that will estimate ratings for (movie, user) pairs that don't appear in the Netflix dataset.

[1] `https://arxiv.org/abs/1304.1467`

Start by considering the sparse matrix that has the **movies in rows, and users in columns**. Note that this matrix, being wide but short (instead of tall and skinny), is not a good fit for the DIMSUM algorithm of Task 2. So, in this task, do not use DIMSUM. Summarize that dataset with SVD (Singular Value Decomposition). Most programming languages have ready-made functions for computing the SVD of a sparse matrix (SciPy has one), so don't implement this yourself! With the dataset summarized with SVD, it is possible to use Latent Factors to predict the ratings for users and movies that don't have a rating. For this, you will need to implement the algorithm presented in Lecture 5, slide 92 and onward (in the slide PDFs on Canvas).

The big sparse matrix of the dataset has now been summarized in three matrices, $A = U\Sigma V^t$. These components are pre-multiplied to create $P^t = \Sigma V^t$ and $Q = U$, so that $A = QP^t$. It is expected that Q somewhat encodes information about the movies, and P^t information about the users.

However, because there are missing values in A (not every user rated every movie), P and Q are currently inexact and don't generalize well. Implement the Stochastic Gradient Descent algorithm presented in Lecture 5 to iteratively refine P and Q. You can find more details on the use of stochastic gradient descent (SGD) in the Neflix challenge in `https://pantelis.github.io/cs634/docs/common/lectures/recommenders/netflix/`. You are encouraged to read that article.

Perform the following sub-tasks:

- Implement Stochastic Gradient Descent using information from the article linked above.
- Add a (global) variable that allows to configure how many epochs of the algorithm will be performed. One epoch consists of computing and applying the gradient for all of the ratings in the Netflix dataset.
- Also implement Batch Gradient Descent, that should be only a small modification of what you have written in Point 2 above. **Compare** the accuracy obtained by Batch and Stochastic gradient descent, depending on the number of epochs and the gradient step, and write your findings in the report. **Make plots** in your report that show how the training accuracy evolves epoch after epoch, both for Stochastic and Batch Gradient Descent. Task 4 details how to measure accuracies.

Hint: A good gradient step for Batch Gradient Descent is 0.1. For Stochastic Gradient Descent, the step must be much smaller, about 1e-5 (0.00001).

Hint: The URL given above goes into detail explaining the gradients you need for both stochastic and batch gradient descent, and gives you the gradients of P and Q. The lecture slides provide a higher-level overview.

You cannot use any library that computes gradients automatically or perform gradient descent (no PyTorch, JAX, Tensorflow, SimPy, etc).

Task 4: Accuracy

With your optimized P and Q produced in Task 3, you can now consider a new matrix $M = QP^t$ that will contain predicted ratings for every movie and every user. That matrix M would also consume about 30 GB of memory, so don't compute it! Instead, feel free to multiply lines of Q and columns of P^t manually to produce individual entries of M, that you can compare to the actual values in the original sparse matrix A that you produced in Task 1. This allows to measure the accuracy on the training set.

1. Compute the RMSE (root mean-squared error) between the set of training movie-user pairs and their corresponding predictions. This is the training error.
2. Split the sparse matrix A into a separate training and testing set, with the training set used to produce P and Q, and the testing set used to compute the RMSE. This is the testing error.
3. *(bonus)* Read `probe.txt` to split the Netflix dataset into training and testing sets.

3 Submission

The deadline for the project is Sunday the 5th of June 2022, at 23:59 Brussels Time. The project is to be submitted on Canvas and must be a zip file that contains:

1. One file that implements the 4 tasks above. It can be Python, Java, Scala, C++, C#, ... Jupyter Notebooks are not allowed, but you can easily save a Jupyter Notebook as a Python file and submit it.
2. A 4-pages report that presents your results, for the different tasks, both in text and figures. For the DIMSUM γ parameters and the SVD K parameters (the number of eigenvalues), we ask you to compare at least 10 different values of these parameters. Feel free to use the rest of the pages to tell us what you want us to know about your project, for instance points that were particularly difficult to do, or a description of code that you think needs an explanation. We will have read your 4-pages report before your presentation in June at a time slot during the exam period, so use these 4 pages to "pre-brief" us on your project.

4 Evaluation

Your project will be evaluated based on the code and report you submit and the 10 minute presentation you give at a time slot during the exam period.

Your work will be evaluated based on the correctness and clarity of your code, the rigour with which you setup your experiments and the quality of your report and visualisations. You will also be evaluated on your presentation, taking into account the clarity and correctness of your presentation, and your ability to answer our questions after the presentation.

5 Plagiarism

You are allowed to look at the lecture slides, the MMDS book and use Google as much as you want for this project. However, the code that you submit, and your report, must be written by you (a member of your team). In case you take close inspiration from code found online, **put the URL in comment**. If we find two projects that have similar code (and no URL), and/or similar reports, we will assume cheating.

Do not look at the code of other students. We say that to help you, from experience. You are allowed to discuss this project among your peers, to send each other links to articles, Github repositories, etc. But do not look at each other's code. Once you have seen the code of someone else, it is very difficult to depart from it and write something original. Seeing another student's code will dramatically increase your chances of plagiarizing that student even without wanting to.

We have automated and manual methods of identifying plagiarism, that have led to true positives (and action) in the past.