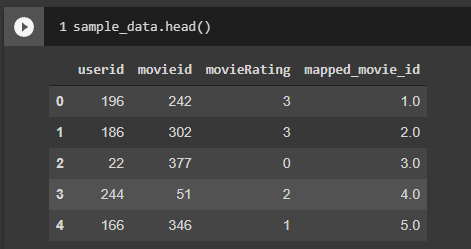
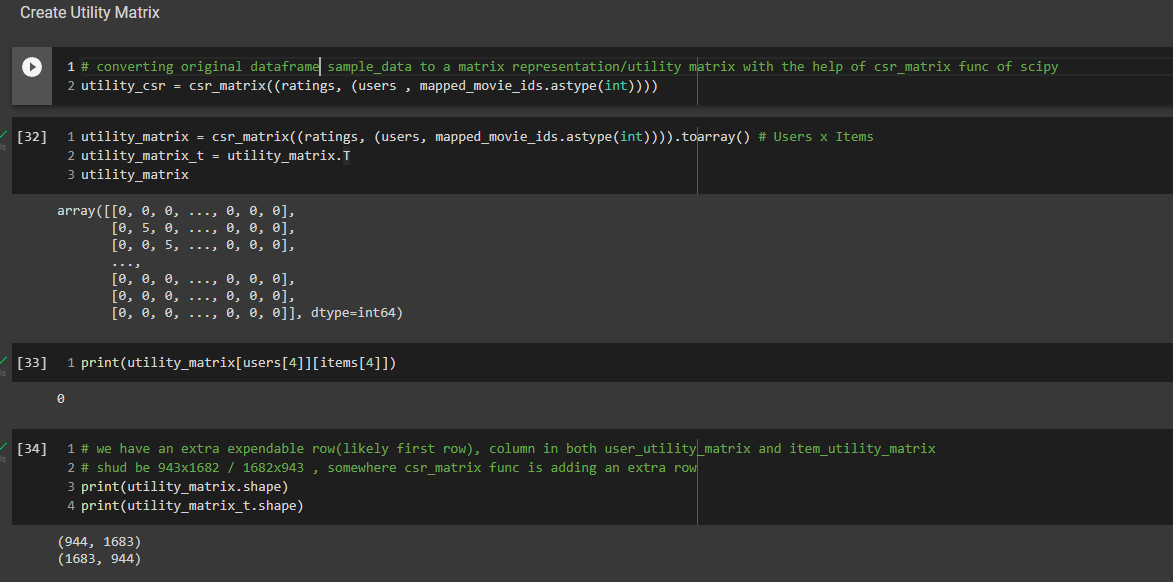
**Setup env:**

1. Install IntelliJ
2. Download code from V2 folder at <https://github.com/gguzunsjsu/CF-BSI> , and (with Root dir at this path: Recommendation-System-BSI.iml) launch intelliJ
3. Download commons-lang3-3.0.1/commons-lang3-3.0.1.jar jar file if necessary and import it through IntelliJ ( available in git repo)
4. Update relative paths in Main.java (Increase heap space in settings if necessary for large datasets)

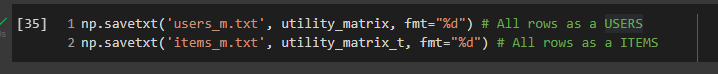
Start running .ipynb notebooks in colab to extract .txt files to be used in IntelliJ java env ( switch between envs for analysis)

**High-Level steps:**

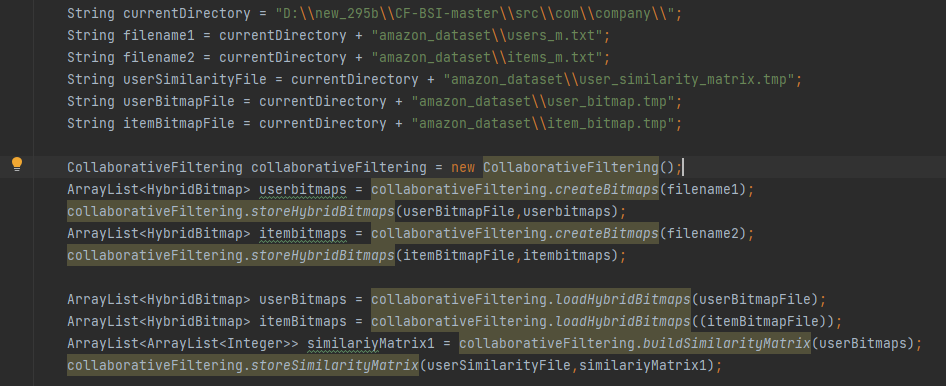
1. 
2. Preprocessing
3. We took the {movielens} dataset above and stored them as a sparse utility matrix



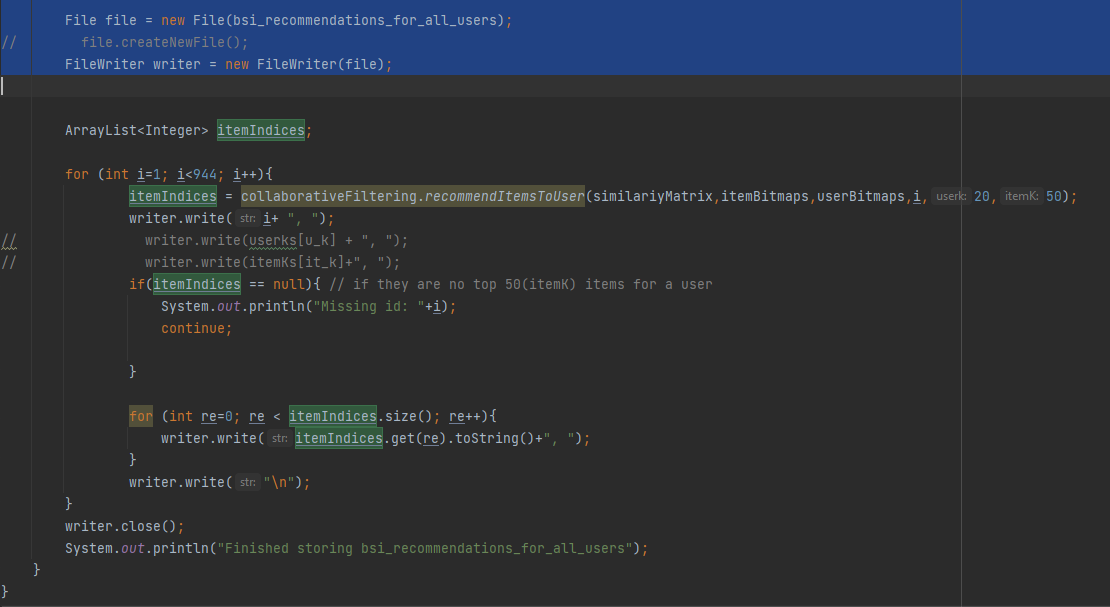
1. did further preprocessing which provides us the below two files which are used in javacode.



1. Then we created user bitmaps and item bitmaps from the above two files with {compressed bit vectors} and also created user-user similarity matrix(bitmaps) by passing userbitmaps to it.



1. Now the user-user similarity matrix bitmaps,userbitmaps,itembitmaps , userK(can be tuned , top K nearest users), itemK(can be tuned, top k ranked items in the nearest user set) is used to find recommendations.



1. Load the java code output text files into notebook and parse them. Later compare the recommendations with als (same dataset) baseline recommendations and calculate metrics like f1 score, recall. And runtime

**Precision: (cbv, als)**

Movielens100k:

5rec 0.135563, 0.056790

10rec 0.119366, 0.053439

Movielens1M:

5rec 0.135616,

10rec 0.117814

Bookscrossing:

5rec

10rec

**Recall**:

Movielens100k:

5rec 0.049196, 0.023977

10rec 0.082985, 0.045113

Movielens1M:

5rec 0.036198

10rec 0.061715

Bookscrossing:

5rec

10rec

**F1score**:

Movielens100k:

5rec 0.062842,0.027490

10rec 0.086344,0.042245

Movielens1M:

5rec 0.050316

10rec 0.070318

Bookscrossing:

5rec

10rec

**Precision: (cbv, als)**

Movielens100k:

5rec 0.135563,0.059643

10rec 0.119366,0.057679

25rec 0.098662,0.057000

50rec 0.079930,0.053679

100rec 0.061866,0.047839

**Recall**:

Movielens100k:

5rec 0.049196, 0.025373

10rec 0.082985,0.047135

25rec 0.163445,0.114265

50rec 0.265683,0.206189

100rec 0.401279,0.344057

**F1score**:

Movielens100k:

5rec 0.062842,0.029301

10rec 0.086344,0.045170

25rec 0.111076,0.068479

50rec 0.113770,0.079115

100rec 0.102104,0.080057

@ 5 recommendations

| Dataset | Precision - CBV | Precision - ALS |
| --- | --- | --- |
| Movielens 100k | X | X |
| Movielens 1million | X | X |
| Book crossings | X | X |

@ 10 recommendations

| Dataset | Precision- CBV | Precision - ALS |
| --- | --- | --- |
| Movielens 100k | X | X |
| Movielens 1million | 0.117814 | 0.061715 |
| Book crossings | X | X |

Runtime:

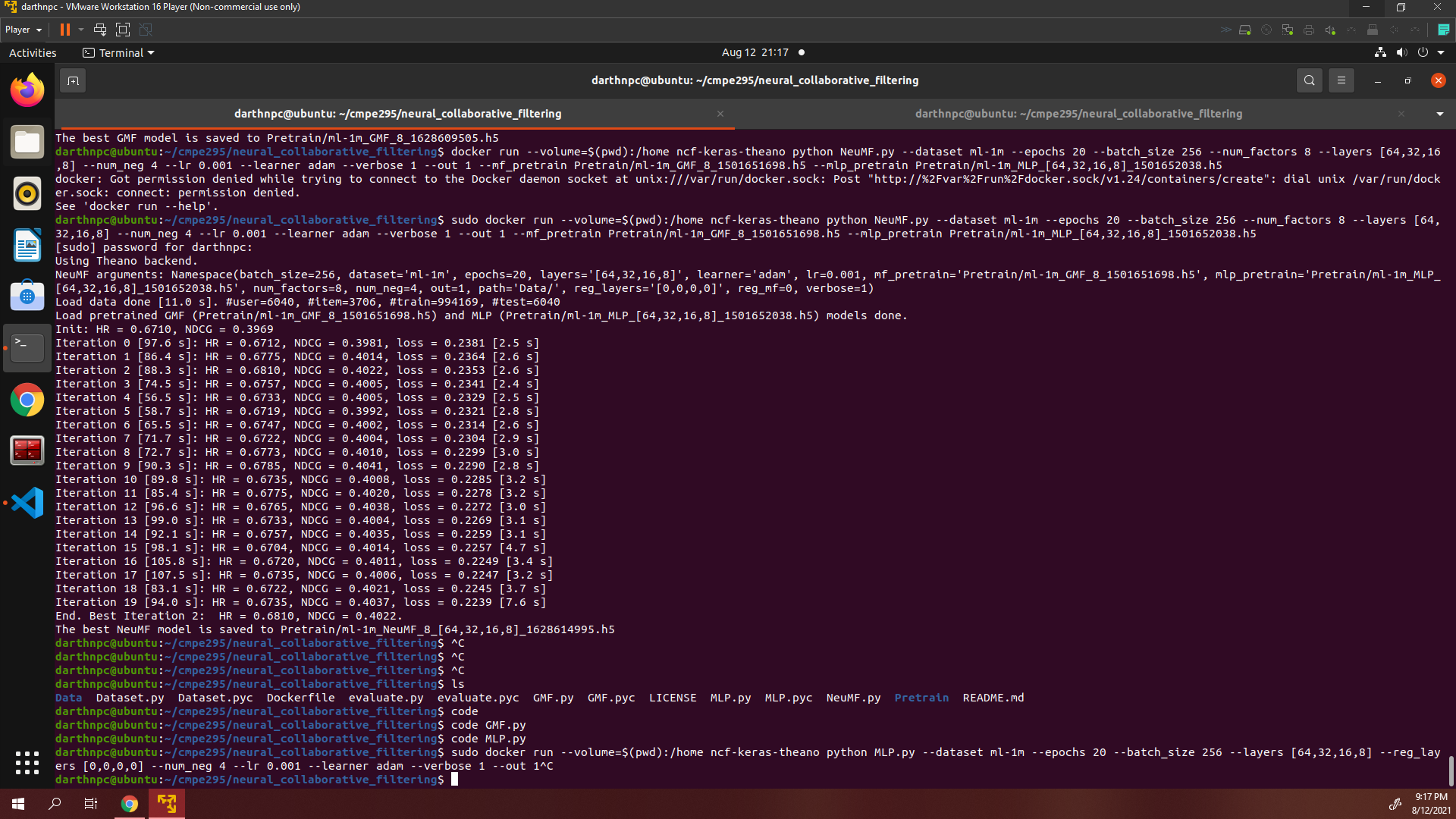
| Dataset | Number of unique users in test data | ALS runtime | CBV runtime |
| --- | --- | --- | --- |
| Movielens 100k | ~560 | 1642000 ms | 16025 ms |
| Movielens 1Million | ~4200 | 86400000 ms | 231161 ms |
| Book\_crossing1Million | ~3100(reduced utility matrices after preprocessing for memory/speed trade-off) | 64042000 ms | 851052 ms |

Neural cf:

1. **Neural Collaborative Filtering (MLP Multi Layer Perceptron): ( HR Hit Ratio, NDCG Normalized Discounted Cumulative Gain , Runtime)**

Setup: Ubuntu local VM as host(Model ran inside docker image)

<https://github.com/hexiangnan/neural_collaborative_filtering>



Hit ratio (**HR**) intuitively measures whether the test item is present on the top-10 list, and the **NDCG** Normalized Discounted Cumulative Gain accounts for the position of the hit by assigning higher scores to hits at top ranks. Both metrics were calculated for each

test user and average score is used in the end.

MLP deep learning model was ran for 20 epochs to get best Hit Ratio and NDCG.

With a simple python snippet, **runtime** of MLP model recommendation is calculated by running below code as a wrapper when metrics like HR Ratio are calculated for each iteration.

( without including the time it took to load dataset Just training time, prediction time and evaluation metric comparison time is included)

import time

start = time.time()

print("hello")

end = time.time()

print(end - start)

Took around ~**20mins** for Movielens1m to output results inside a container.

