Heidelberg University
Winter semester 2024/25

Group Artificial Intelligence for Programming (AIP)
Prof. Dr. Artur Andrzejak, Kim Tuyen Le

# Problem Set 1 for lecture Mining Massive Datasets

Due October 28, 2024, 23:59 CET

---

In this problem set you should use as computational libraries only NumPy, SciPy, Pandas, or Polars (whatever is necessary). Do not use JAX, PyTorch, TensorFlow etc. (except for Exercise #4, details are provided below). Of course, you should implement the main code yourself, i.e. calling a function from a library for recommendation systems to 'solve' an exercise is not allowed. Submit the complete source code including run setups and logs of the output from unit tests/test runs. It is recommended to use a git repository for the development (but submit only the final source files, not the repository).

## Exercise 1 (1 point)

Complete the blanks in the file `cf_algorithms_to_complete.py` located on the GitHub repository. These blanks are indicated by the function named `complete_code`. Check the correctness of your code by executing the provided Colab script.

## Exercise 2 (3 points)

Implement centered cosine distance for sparse matrices and sparse vectors.

**a)** Implement functions to calculate centered cosine similarity of vector-vector pairs and matrix-vector pairs which accept sparse data as inputs. The functions should be named `centered_cosine_sim` and `fast_centered_cosine_sim` (latter for a sparse matrix and vector). You might use the library `scipy.sparse` or other.

**b)** Write unit tests (one or more) for each function. In particular, test the function `centered_cosine_sim` using the following input data

- vector_x = $[x_0, x_1, ..., x_i, ..., x_{k-1}]$
- vector_y = $[y_0, y_1, ..., y_j, ..., y_{k-1}]$
- $y_j = x_i$ with $i + j = k - 1$

Test the function with

**b.1)** $k = 100, x_i = i + 1$

**b.2)** $k = 100$

If $i \in [c, c+10, c+20, c+30, ..., c+90]$ with $c = [2, 3, 4, 5, 6]$, then $x_i$ is NaN; otherwise, $x_i = i + 1$.

## Exercise 3 (3 points)

Rewrite the code in the file `cf_algorithms_to_complete.py` (after filling blanks) to support sparse utility matrices (UMs), using the functions implemented in **Exercise #2**. This means that the function `rate_all_items` in the file `cf_algorithms_to_complete.py` should accept a sparse UM after rewriting. Consider that you need to preprocess data differently, in particular you should keep only a sparse version of the UM `um_movielens` in memory. Let your code report the size of this matrix (similarly as in the provided routine `read_movielens_file_and_convert_to_um`). Implement a unit test which compares the results against the dense original version.

## Exercise 4 (5 points)

Write code which processes the MovieLens 25M data set and creates data structures (e.g. Python shelves) `rated_by[]` and `user_col[]` described in Lecture 2. Your implementation should keep the usage of RAM reasonably low (in particular, do not load all input data at once). Note that the contents of the data structure `user_col[]` should be sparse vectors (use data structures compatible with those in **Exercise #2**).

In this exercise, you can use the routine `load_movielens_tf` provided in `data_util.py` in the GitHub repository to load the dataset (and so as an exception, you may use the library TensorFlow Datasets).

## Exercise 5 (4 points)

Perform the following tasks:

**a)** Implement a function for estimating rating of user $x$ on item $i$ via collaborative filtering, using functions implemented in **Exercises #3** and **#4**.

**b)** Run the function on the following user-item pairs and report the results:

| **Pair no.** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| `userID` | 828 | 2400 | 3765 | 4299 | 5526 | 6063 | 7045 | 8160 | 9682 | 10277 |
| `movieID` | 11 | 4725 | 1270 | 4020 | 2432 | 4525 | 4100 | 6300 | 1212 | 7355 |

**c)** Compute and report the maximum memory usage of your rating function (to simplify, you might use the size of the whole Python process) for each of the 6 first user-item pairs from the above table (restart your script each time).