Big Data Analytics Programming
Assignment 3:
Index structures and Movie recommendations

Jessa Bekker
jessa.bekker@cs.kuleuven.be

Toon Van Craenendonck
toon.vancraenendonck@cs.kuleuven.be

Collaboration policy:
Projects are independent: no working together! You must come up with how to solve the problem independently. Do not discuss specifics of how you structure solution, etc. You cannot share solution ideas, pseudocode, code, reports, etc. You cannot use code that is available online. You cannot look up answers to the problems online. If you are unsure about the policy, ask the professor in charge or the TAs.

1 Part 1: Index structures (6 pts)

1.1 Task description

The first part of this assignment is about indexes to access databases efficiently. You will implement two types of indexes: bitmap and bitslice. For each of them you need to be able to generate the index for a feature and use them to answer some queries.

1.2 Implementation

Skeletons are provided for both indexes (`BitMap.java` and `BitSlice.java`). Make sure that you understand the code and complete it where indicated. You can add extra class variables and methods but you are not allowed to change the interface of the provided methods. The indexes need to be generated from a data table that is implemented in `DataTable.java`. This class is already complete, but make sure you understand it.

1.3 Experiments

To test your code and appreciate the efficiency of using indexes, you will run some experiments. Six queries will be executed in two ways: 1) on the database directly and 2) using the indexes. Both methods are to be timed so that you can compare the speed. The class `Experiments.java` contains the skeleton for the experiments. The queries might need a combination of operations on the different indexes. To execute the queries directly on the data, you need to specify how to select a record and, for making sums, how to extract a number from a record. These functions are implemented and passed using the interfaces

`RecordSelector.java` and `ValueSelector.java` respectively. To avoid separate files for all of the selector implementations, you can implement them as anonymous classes inside `Experiments.java`. The following is an example of how to make an anonymous `RecordSelector` class that select Trappist style beers:

```
data.select(new RecordSelector() {
    @Override
    public boolean keep(String[] record) {
        return record[style].equals("Trappist");
    }
});
```

The dataset that you will use is from an online beer store (`http://www.belgianbeerz.com/`). It contains 326 beers and it has the following features: name, price, style and volume (in cl). The file `beer.info` provides more information about the features and their attributes.

You will answer the following questions about the data:

1. Which seasonal beers are offered?
2. How many beers have a volume of 33 cl?
3. How much would does it cost to order one of each beer?
4. How many beers cost more than 10 euros?
5. What is the average price of a 25 cl Pils?
6. How many Abbey beers cost 3.00 EUR or less?

You can compile all classes with `javac *.java` and run the experiments with

```
java Experiments <pathToData> 326 \;
```

2 Part 2: Movie recommendations (18 pts)

2.1 Task description

In the second part of this assignment you will implement a simple collaborative filtering model for recommending movies. Collaborative filtering filters information by using the recommendations of other people. It is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. A person who wants to see a movie for example, might ask for recommendations from friends with a similar taste.

We will use the MovieLens dataset to train our recommendation model. There are several variations of this classical dataset, but the one that we will use contains 10^6 ratings by $69,878$ users over $10,667$ movies and is available online.[1] Download and extract the archive in the `data/` directory. More datasets and information about MovieLens can be found at `https://grouplens.org/datasets/movielens/`.

Most collaborative filtering systems apply the so called neighborhood-based technique. In the neighborhood-based approach a number of users is selected based on their similarity to the active user. This means that the recommendations of some users who have similar interests are trusted more than recommendations from others. We will use the Pearson correlation coefficient[2] to measure the similarity between users. As a first

[1] `https://people.cs.kuleuven.be/~toon.vancraenendonck/bdap/movielens.tgz`
[2] `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`

subtask you should therefore implement a method that computes this Pearson correlation coefficient for each pair of users in the MovieLens dataset and write the resulting matrix to a CSV file with the following format:

```
3
param1=value,param2=value
1.0000,-.3650,NaN
-.3650,1.0000,.0012
NaN,.0012,1.0000
```

Note that this file should contain the full matrix. The first line contains the size of the matrix, the second line is reserved for additional parameters used while computing the matrix (leave this line empty if you do not use any additional parameters) and all other lines contain the matrix values rounded to four decimal digits. The value on row i and column j should contain the Pearson correlation between the users with internal ids i and j (see `MovieHandler.java` for more information on these internal ids) or `NaN` if this correlation is not defined (e.g., no commonly rated movies) or when you decide not to compute it (e.g., not enough commonly rated movies to be meaningful).

Note that the size of this full matrix will be quite large (about 20GB) when you run your code on the full dataset. Therefore, do not use your home folder on the student lab computers to store this file. Use the `\tmp` folder instead. Also, you will have to come up with an appropriate data structure and be creative to fit this correlation matrix into the 2GB memory limit. There are several approaches possible. Make sure to discuss the approach you take in the report.

With our similarity matrix in hand, we can now predict the ratings that were not included with the data. There are many possible approaches to compute these predictions. You are free to take any feasible approach.

List of files in `assignment3/MovieRecommendation/src/`: The following extra files are provided for this part of the assignment.

- `MovieHandler.java` class (complete). Provides a set of utility functions to access the rating data.
- `MovieRunner.java` (incomplete!). Main file to run and evaluate the movie predictor. In this file you must implement the following method.
  - `predictRating(userID, movieID)`. Estimate and return the rating of `userID` for movie `movieID`. You must complete this method.
- `PearsonCorrelation.java` (incomplete!). Main file to generate the Pearson correlation matrix. In this file you must implement the following methods.
  - `correlation(List<MovieRating> xRatings, List<MovieRating> yRatings)`. Estimate and return the Pearson correlation between two lists of movie ratings. You must implement this method.
  - `main(String[] args)`. Complete this method such that it computes and saves the Pearson correlation matrix as discussed above.
- `Neighbor.java` class (complete). A simple wrapper class to store similar users.
- `MovieRating.java` class (complete). A simple wrapper class to store movie ratings.

2.2 Running and testing

The following command should compute the Pearson correlation matrix for the `ra.train` data and store the result in the format discussed above.

```
java -cp .:bin/ PearsonCorrelation -trainingFile data/ra.train -outputFile data/ra.matrix
```

Next, the following command should train your recommender on the `ra.train` data and evaluate its performance on the `ra.test` testing data.

```
java -cp .:bin/ MovieRunner -trainingFile data/ra.train -matrixFile data/ra.matrix -testFile
data/ra.test
```

You can already execute the recommender, but the predictions will not be good as the `predictRating` method always returns the same prediction (2.5/5).

You may add any optional arguments. Please include the command with the optimal parameter settings in a `README` file (we should be able to simply run this command after compiling your code).

2.3 Parameter tuning and evaluation

At this stage, your algorithm should have a variety of parameters (minimal number of commonly rated movies, aggregating method, default prediction, etc.). Understanding the impact of each parameter and finding the optimal combination of parameters for the problem at hand is an important, and difficult aspect of every data analysis process. Run experiments and find the best combination of parameters for this application. Do not forget to discuss these experiments in the report. Note that your system will be evaluated using the parameters that you provide in your `README` file.

In order to evaluate the quality of your predictions, we will use Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{\sum_i^n (\hat{y}_i - y_i)^2}{n}}$$

Where n is the number of testing examples, $\hat{y}_i$ is the predicted rating for the movie i and y_i is the actual rating for this movie. The $RMSE$ of your recommender should be compared to the baseline: predicting the average rating for this movie in the dataset.

Beware! Predicting movie ratings is inherently difficult because the liking of a movie involves a variety of human factors that are both complex and not reflected in the database. Unlike for more simple tasks, advanced techniques only improve the accuracy of the prediction by small margins. As a matter of fact, the Netflix competition awarded USD 1,000,000 to improve the baseline by only 10% and the recommender system that won the challenge was the result of several years of research and tuning. For reference, a very basic solution for this exercise outperforms the baseline by $\sim 4\%$ on RMSE. The extent to which you are able to reduce the RSME is important and will affect your grade. It is, of course, also important to discuss each improvement to your pipeline that leads to a reduction on RMSE in your report.

3 Report (6 points)

You will write a small report about your assignment. Please respect the page limits.

3.1 Indexes

For both the naive implementation and index-based implementation, run each query ten times and compute the average run time. Construct a bar chart showing the average run time and the standard deviation for each method on each query. Write a small interpretation (a maximum of half a page of text) comparing the performance of each approach.

3.2 Filtering

Write a report to describe your implementation and to present the experiments that you ran to tune parameters and to evaluate the performance of your algorithm(s). Please focus on the key decision choices in terms of the efficiency and accuracy. The report can be a maximum of 3 pages, excluding any figures or tables.

4 Important remarks

- Your final code should run on the full datasets (with the parameter settings that you give), not only on subsamples of the data. This will be part of the evaluation. Note that you can use the departmental machines to run code overnight if necessary.
- The assignment should be handed in on Toledo **before noon on Monday, April 16th, 2018**. As usual, there will be a 10% penalty per day, starting from the due day.
- The main grading criteria are the correctness of the code, the runtime efficiency, and accuracy.
- **Do not** upload any data file (i.e. the data directory must remain empty).
- **Do not** upload the computed Pearson correlation matrix.
- Do not forget the `README` file, with one command line for computing the pearson correlation matrix and one for the movie recommender. These will be used for evaluation.
- You must upload an archive (`.zip` or `.tar.gz`) containing the report (as pdf) and a folder with all the source files (See file hierarchy below.) Feel free to add as many files or directories as you need. However, if you think this extra content is valuable and deserved to be looked at, please describe why in the report.

```
assignment3/
├── report.pdf
├── IndexStructures/
│   └── src/
│       ├── BitMap.java
│       ├── BitSlice.java
│       └── Experiments.java
└── MovieRecommendation/
    ├── README
    └── src/
        ├── MovieHandler.java
        ├── MovieRunner.java
        ├── PearsonCorrelation.java
        └── ...
```

Automated grading Automated tests are used for grading. Therefore you must follow these instructions:

- Use the exact filenames as specified in the assignment.
- Do not change the provided interface/skeletons, this includes the constructor.
- Your code must run on the departmental machines, this can be done over ssh (see documents of exercise session 1).
- Never use absolute paths.

If you fail to follow these instructions, you will lose points.

Good luck!