DSCI 553: Foundations and Applications of Data Mining

Fall 2020

Assignment 2

Deadline: 10/04/2020 11:59 PM PDT

1. Assignment Overview

The goal in this assignment is to let you be familiar with A-Prior, MinHash, Locality Sensitive Hashing (LSH), and different types of recommendation systems. You will keep using MovieLens dataset that has been used in Lab sessions.

2. Requirements

2.1 Programming Environment

Python 3.6. You could use spark, numpy, scipy, pandas, sklearn, pytorch, tensorflow library.

If you need to use additional libraries, you must have approval from TA and declare it in requirements.txt

There will be a 20% penalty if we cannot run your code due to Python version inconsistency or undeclared 3rd party library.

2.2 Write your own code

Do not share code with other students!

For this assignment to be an effective learning experience, you must write your own code! We emphasize this point because you will be able to find Python implementations of some of the required functions on the web. Please do not look for or at any such code!

TAs will combine all the code we can find from the web (e.g., Github) as well as other students' code from this and other (previous) sections for plagiarism detection. We will report all detected plagiarism.

2.3 What you need to turn in

The submission format will be the the same as homework 1. Your submission must be a zip file with name: **firstname_lastname_hw2.zip** (all lowercase). You need to pack the following files in the zip:

a. Three (four if you do the bonus task) Python scripts, named: (all lowercase)

`firstname_lastname_task1.py, firstname_lastname_task2.py, firstname_lastname_task3.py, firstname_lastname_task4.py(optional)`

You could have additional files, but these 3 Python scripts are required.

b. You don't need to include your results. We will grade on your code with our testing data (data will be in the same format).

3. MovieLens Dataset

Dataset homepage: https://grouplens.org/datasets/movielens/

You could use the small dataset for development. A copy of the dataset is included, along with split training and testing data.

4. Tasks

4.1 Task 1: Find interesting association rules

Users' ratings to movies are stored in *ratings.csv*. Recall market-based model. The set of movies that a user gives 5.0 rating could be viewed as a basket.

Task: Find association rules $\{i_1, i_2, \cdots, i_k\} \rightarrow j$ in these baskets, such that interest $\geq I$, and support $\geq$ S. i and j are *movieId*.

Note:

1. Only consider 5.0 rating in this task.
2. You should use an efficient algorithm like A-Prior method.
3. Although interest could be positive or negative, only consider positive interest here
4. To simplify the computation, the threshold of support is applied to $I \cup j$. In the textbook, the support is applied to I only. j must be a single element.

Grading:

1. 20 points in total
2. This is a deterministic algorithm. Your answer should be exactly the same as the solution.
3. We will test your implementation on multiple datasets (same format). If you get them all correct, you get full points.
4. If your answer doesn't match the solution, 5 points deducted. The remaining points will be given based on an analysis of your implementation.
5. The reference implementation takes less than 10 seconds on the *ratings.csv*, with I=0.5, S =10. If your implementation takes more than 5 minutes, likely it's not implemented correctly and will deduct points.

Input format: (We will use the following command to execute your code)

python firstname_lastname_task1.py <input_filename> <output_filename> <I> <S>

Params:

input_filename: Path to input file, e.g.: *ml-latest-small/ratings.csv*

output_filename: Path to output file

I: Threshold of the of interest, e.g: 0.5. Use the definition in week 2, slide 19. Only consider positive interest here.

S: Threshold of support, e.g.: 10. Use the definition in week 2, slide 14. Support is an integer, not a percentage. The threshold is applied to $I \cup j$.

Output format:

A JSON file serialized from a nested list of association rules. Indent doesn't matter. You need to output every association rule $\{i_1, i_2, \cdots, i_k\} \rightarrow j$, along with interest and support for that rule. Below is an illustration of the format.

[

$[[i_1, i_2, i_3], j_1, \text{interest}_1, support_1],$

$[[i_1, i_2, i_4], j_2, \text{interest}_2, \text{support}_2],$

…

]

The reference output on *ratings_test_truth.csv* with I=0.2, S=2 is attached as *task1_sample.json*

Sorting Order:

1. In each association rule, the list of i are sorted in ascending order
2. Between association rules, rules are first sorted by interest, descending order
3. Then sorted by support, descending order
4. Then sorted by the list of i, ascending order. Python has sorting order defined on two lists.
5. Then sorted by j, ascending order

Please do the sorting correctly. Otherwise your answer will not match the solution and lose points. Step 2-5 could be done by *sorted* function with argument:

```
key=lambda x: (-x[2], -x[3], x[0], x[1])
```

You don't need to worry about rounding errors. There is a tolerance in grading script.

Optional: Convert *movieId* to *title*. Are there any findings from the association rules?

4.2 Task 2: Content-based recommendation system with LSH

In the Lab session, we have built a family of memory based recommendation system: content-based, user-based, and item-based. But they are not scalable to massive

datasets. For every recommendation made, the entire dataset has to be visited once. This could be optimized by using LSH algorithm to find similar items, instead of linear search.

Task: Build an efficient content-based recommendation system, that using LSH to find similar items.

Note:

1. Items (movies) are represented by a set of genres, so you could skip the n-grams step, but still need to do minhashing and LSH step. Their similarity could be measured by Jaccard similarity.
2. You should implement LSH and content-based recommendation yourself. Using spark.mllib or similar library is not allowed.
3. There are many hyper-parameters like hash size, bucket size, similarity threshold, prime number. You need to decide these hyper-parameters, as long as your MSE is below a threshold, you get full points.
4. Don't forget to clip the rating to [0.5, 5.0]
5. The submitted program shouldn't read ground truth values. To calculate MSE, do it in another program.

Grading:

1. 20 points in total
2. 0 point if your LSH or recommendation system is provided by a library
3. This algorithm has randomness in nature. We will run your program multiple times and take the average. As long as your $MSE \leq 0.85$, you get full points.
4. If your MSE is higher than the threshold, 5 points deducted. The remaining points will be given based on an analysis of your implementation.
5. We will test your implementation on multiple datasets (same format). These datasets come from the same distribution and should yield similar MSE. If your model gets consistent MSE among different datasets, we will use the MSE on provided dataset for grading. Otherwise, we will analyze your implementation to see if it's memorizing test data and deduct points if found.
6. The reference implementation takes about 10 seconds. If your implementation takes more than 5 minutes, likely it's not implemented correctly and will deduct points.
7. The running time of LSH based method is similar to linear search method on the provided dataset. That's because in this dataset, on average, each user only rated a small number of movies. The rating matrix is very sparse. The LSH based method could show its advantage when the rating matrix is denser.

Input format:

python firstname_lastname_task2.py <movies_filename> <rating_train_filename> <rating_test_filename> <output_filename>

movies_filename: Path to movies file, e.g.: *ml-latest-small/movies.csv*

rating_train_filename: Path to train rating file, e.g.: *ml-latest-small/ratings_train.csv*

rating_test_filename: Path to test rating file, e.g.: *ml-latest-small/ratings_test.csv*

output_filename: Path to output file

In *ratings_test.csv*, column *rating* is filled with 0. If you want to evaluate your model's performance, you need to compare your prediction with *rating_test_truth.csv*

Output format:

Similar to `rating_test.csv`, it's column *rating* is replaced with your model's prediction.

4.3 Task 3: Model based recommendation system

In contrast to memory based method, model based method doesn't need to store or look up the massive training dataset during inference, which makes it scalable and widely used in the industry.

Task: Build a model-based recommendation system based on matrix factorization.

Note:

1. You are free to use any matrix factorization algorithm, and any library implementing them (e.g.: numpy, sklearn), or you could implement it yourself
    a. Singular value decomposition
    b. Neural network based matrix factorization
2. You should only use these libraries to do matrix factorization. Using a recommendation system API from 3rd party library is not allowed
3. There are many hyper-parameters like number of components, regularization coefficient. You need to decide on your own, maybe with grid search. As long as your MSE is below a threshold, you get full points.
4. In order to reach required MSE, it's recommended to include overall/user/movie bias terms.

Grading:

1. 20 points in total
2. 0 point if your recommendation system is provided by a library
3. This algorithm has randomness in nature. We will run your program multiple times and take the average. As long as your $MSE \leq 0.785$, you get full points.

4. The reference implementation takes less than 10 seconds. If your implementation takes more than 5 minutes, likely it's not implemented correctly and will deduct points.
5. Same as Task 2, Grading 4

Input and output format: Same as task 2. For consistency, although this method doesn't need *movies.csv*, it's still in the list of arguments.

4.4 Bonus Task: Further improve the recommendation system

Task: Build a recommendation system that gets the lowest MSE as you can.

Note:

1. You are free to use any method in slides/textbook/online. But that must be your own implementation, not a functionality of an existing library.
2. Make sure your model performs well on the test set, not training set. We will evaluate your model in multiple datasets.
3. Possible ideas:
    a. The BellKor Solution to the Netflix Grand Prize
    b. Hybrid model of Item based and User based RS, may combined with LSH.

Grading:

1. For every 0.01 MSE reduced over baseline $MSE = 0.75$, get 1 bonus point. 20 bonus point max.
2. 0 point if your method is provided by a library, or copy-paste from GitHub.
3. Same as Task 2, Grading 3,4,5

Input and output format: Same as task 2.

5. Grading Criteria

The perfect score for this assignment is 60 points + 20 bonus point.

Assignment Submission Policy

Homework assignments are due at 11:59 pm on the due date and should be submitted in Blackboard. Every student has ***FIVE free late days*** for the homework assignments. You can use these five days for any reason separately or together to avoid the late penalty. There will be no other extensions for any reason. You cannot use the free late days after the last day of the class. You can submit homework up to one week late, but you will ***lose 20% of the possible points*** for the assignment. After one week, the assignment cannot be submitted.

(% penalty = % penalty of possible points you get)

1. You can use your free 5-day extension separately or together.
2. If we cannot run your programs with the command we specified, there will be 80% penalty.
3. If your program cannot run with the required Python/Spark versions, there will be 20% penalty.
4. If our grading program cannot find a specified tag, there will be no point for this question.
5. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
6. If the header of the output file is missing, there will be 10% penalty.
7. We can regrade on your assignments within seven days once the scores are released. No argue after one week. There will be 20% penalty if our grading is correct.
8. There will be 20% penalty for late submission within a week and no point after a week.
9. There will be no point if the total execution time exceeds 15 minutes.