CSCI 5523 Introduction to Data Mining

Spring 2023

Assignment 3 Part I (3 points)

Deadline: Apr. 2nd 11:59 PM CDT

1. Overview of the Assignment

In Assignment 3, you will complete three tasks. In Part I, you will first implement Min-Hash and Locality Sensitive Hashing (LSH) to find similar businesses efficiently. In Part II, you will implement various types of recommendation systems.

2. Requirements

2.1 Programming Requirements

a. You must use **Python** to implement all tasks. You can only use standard python libraries (i.e., external libraries like numpy or pandas are not allowed).

b. **You are required to only use Spark RDD**, i.e. no point if using Spark DataFrame or DataSet.

2.2 Submission Platform

We will use a submission platform to automatically run and grade your submission. We highly recommend that you first test your scripts on your local machine before submitting your solutions. **We will Gradescope to grade your submissions**

2.3 Programming Environment

Python 3.9.12, and Spark 3.2.1

2.4 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 to the university.

3. Yelp Data

For this assignment, we have generated sample review data from the original Yelp review dataset using some filters, such as the condition: “state” == “CA”. We randomly took 80% of sampled reviews for training, 10% for validation, and 10% as the test dataset. (We do not share the test dataset.) You can access and download the data on Google Drive:

a. train_review.json (we will also use this dataset for Part 2)

b. We will release the val_review.json for Part 2 on Mar. 15th

4. Tasks

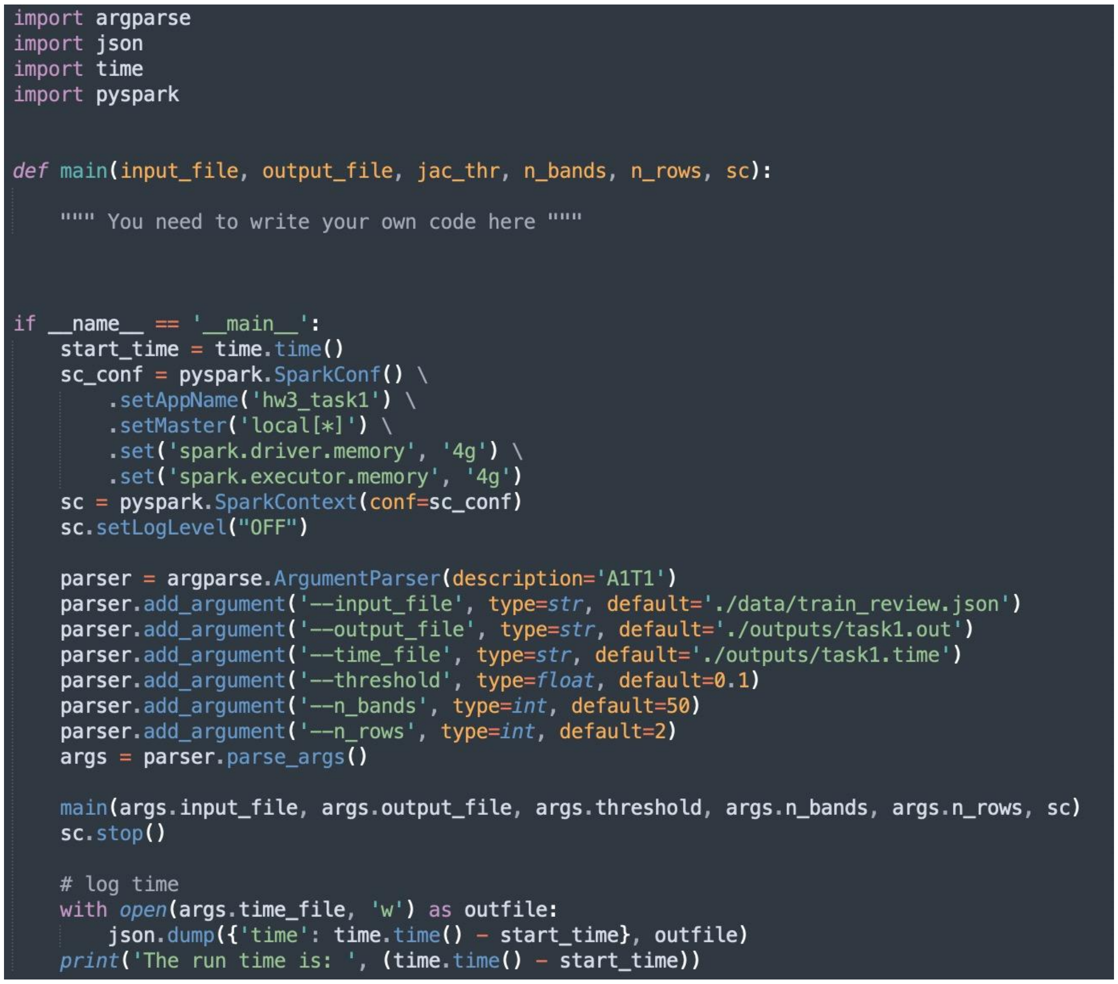
You need to submit the following files on Gradescope: (all in lowercase)

a. Python scripts: task1.py

The main function has been provided to you. See the screenshot below. Please do not change the content in the screenshot in your submission. You need to implement the **main()** function.

b. Result files: task1.res

c. [OPTIONAL] You can include other scripts to support your programs (e.g., callable functions).



```python
import argparse
import json
import time
import pyspark


def main(input_file, output_file, jac_thr, n_bands, n_rows, sc):

    """ You need to write your own code here """



if __name__ == '__main__':
    start_time = time.time()
    sc_conf = pyspark.SparkConf() \
        .setAppName('hw3_task1') \
        .setMaster('local[*]') \
        .set('spark.driver.memory', '4g') \
        .set('spark.executor.memory', '4g')
    sc = pyspark.SparkContext(conf=sc_conf)
    sc.setLogLevel("OFF")

    parser = argparse.ArgumentParser(description='A1T1')
    parser.add_argument('--input_file', type=str, default='./data/train_review.json')
    parser.add_argument('--output_file', type=str, default='./outputs/task1.out')
    parser.add_argument('--time_file', type=str, default='./outputs/task1.time')
    parser.add_argument('--threshold', type=float, default=0.1)
    parser.add_argument('--n_bands', type=int, default=50)
    parser.add_argument('--n_rows', type=int, default=2)
    args = parser.parse_args()

    main(args.input_file, args.output_file, args.threshold, args.n_bands, args.n_rows, sc)
    sc.stop()

    # log time
    with open(args.time_file, 'w') as outfile:
        json.dump({'time': time.time() - start_time}, outfile)
    print('The run time is: ', (time.time() - start_time))
```

4.1 Task1: Min-Hash + LSH (3pts)

4.1.1 Task description

In this task, you will implement the Min-Hash and Locality Sensitive Hashing algorithms with Jaccard similarity to find similar business pairs in the *train_review.json*. We focus on **0/1 ratings** rather than the actual rating values in the reviews. In other words, if a user has rated a business, the user's contribution in the characteristic matrix is 1; otherwise, the contribution is 0 (Table 1). **Your task is to identify business pairs whose Jaccard similarity is above the given threshold (e.g., 0.1).**

	user1	user2
business1	3	-
business2	-	3
business3	-	4
business4	5	4

	user1	user2
business1	1	0
business2	0	1
business3	0	1
business4	1	1

Table 1: The left table shows the original ratings. The right table shows the converted 0 and 1 ratings.

You can define any collection of hash functions to permute the row entries of the characteristic matrix to generate Min-Hash signatures. Some potential hash functions are:

$$f(x) = (ax + b) \% m$$
$$f(x) = ((ax + b) \% p) \% m$$

where p is any prime number; m is the number of bins. You can define any combination for the parameters (a, b, p, or m) in your implementation.

After you have defined all hash functions, you will build the signature matrix using Min-Hash. Then you will divide the matrix into b bands with r rows each. You can try different combinations of b and r to balance the number of candidates and the computational cost. Two businesses become a candidate pair if their signatures are identical in at least one band.

Lastly, you need to verify the candidate pairs using their original Jaccard similarity. Table 1 shows an example of calculating the Jaccard similarity between two businesses. Your final outputs will be the business pairs whose Jaccard similarity is >= 0.1.

	user1	user2	user3	user4
business1	0	1	1	1
business2	0	1	0	0

Table 2: Jaccard similarity (business1, business2) = #intersection / #union = 1/3

4.1.2 Input Format (Please make sure you use exactly the same input parameters names!)

We use the "**argparse**" module to parse the following arguments. We provide the code in task1.py:

1. Input file path (**--input_file**): The path to the input file including path, file name and extension.

2. Output file path (**--output_file**): The path to the output file including path, file name and extension.

3. Time file path (**--time_file**): The path to the time file including path, file name and extension.

4. Threshold (**--thre**): The threshold for the Jaccard similarity

5. The number of bands (**--n_bands**): The number of bands

6. The number of rows (**--n_rows**): The number of rows

Execution example:

$ python task1.py --input_file <input_file> --output_file <output_file> --time_file <time_file> --thre <thre> --n_bands <n_bands> --n_rows <n_rows>

Example: $ python task1.py --input_file train_reviews.json --output_file task1.out --time_file task1.time --thre 0.1 --n_bands 100 --n_rows 2

4.1.3 Output format

You must write a business pair and its similarity in the JSON format using **exactly the same tags like the example in Figure 1**. Each line represents a business pair, i.e., "b1" and "b2". For each business pair "b1" and "b2", you do not need to generate the output for "b2" and "b1" since the similarity value is the same as "b1" and "b2". You do not need to truncate decimals for the 'sim' values.

```
{"b1": "cYwJAgA6I12KNsd2rtXd5g", "b2": "Fid2ruy5s600SX4tvnrFgA", "sim": 0.032448377581120944}
{"b1": "7zecrDCEugcx8bgFn9LbLQ", "b2": "1VvxsddAoINg8TJXOZgEfg", "sim": 0.018867924528301886}
```

Figure 1: An example output for Task 1 in the JSON format

4.1.4 Grading

Your task 1 outputs will be graded by precision and recall metrics defined below.

Precision = # true positives / # output pairs

Recall = # true positives / # ground truth pairs

Your precision, recall, and execution time should meet the below requirement with different settings. For example, for Setting 1, your precision should be **>= 0.8 (1pt**), and recall should be **>= 0.5 (1pt).** The execution time on Gradescope should be **<= 70 seconds (1pt**).

Setting ID	Settings	Precision	Recall	Time
1	thre=0.1, n_bands=50, n_rows=2	0.8	0.5	70s
2	thre=0.1, n_bands=100, n_rows=2	0.8	0.8	100s

To evaluate the implementation, you can generate the ground truth that contains all the business pairs from the *train_review.json* file whose Jaccard similarity is >=0.1 and calculate precision and recall by yourself. You do not need to submit the files related to generating ground truth.

5. Grading Criteria

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

1. You can use your free 5-day extension separately or together. During grading, TAs will count the number of late days **based on the submission time.** For example, if the due date is 2023/09/10

23:59:59 CDT and your submission is 2023/09/11 05:23:54 CDT, the number of late days would be 1. **TAs will record grace days by default (NO separate Emails)**. However, if you want to save it next time and treat your assignment as late submission (with some penalties), **please leave a comment in your submission.**

2. There will be no point if your submission cannot be executed on the grading platform. **We will provide more information about the submission platform early next week.**
3. There is no regrading. Once the grade is posted on Canvas, we will only regrade your assignments if there is a grading error. No exceptions.
4. Homework assignments are due at 11:59 pm CT on the due date and should be submitted on Canvas. Late submissions within 24 hours of the due date will receive a 30% penalty. Late submissions after 24 hours of the due date will receive a 70% penalty. Every student has FIVE free late days for homework assignments. You can use these free late 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.**

6. Submission

The Gradescope submission will be available on **Mar 10th**.