INF553 Foundations and Applications of Data Mining

Summer 2019

Assignment 3

Deadline: June. 20th 11:59 PM PST

1. Overview of the Assignment

In Assignment 3, you will complete two tasks. The goal is to let you be familiar with MinHash, Locality Sensitive Hashing (LSH), and different types of collaborative-filtering recommendation systems. The dataset you are going to play with is a subset from the Yelp dataset (https://www.yelp.com/dataset) used in the previous assignments.

2. Assignment Requirements

2.1 Programming Language and Library 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). There will be a **10% bonus** for each task (or case) if you also submit a Scala implementation and both your Python and Scala implementations are correct.

b. You are required to only use the Spark RDD to understand Spark operations. You will not receive any point if you use Spark DataFrame or DataSet.

2.2 Programming Environment

We will use **Python 3.6, Scala 2.11, and Spark 2.3.3** to test your code. There will be a 20% penalty if we cannot run your code due to the library version inconsistency.

2.3 Write your own code

Do not share your code with other students!!

We 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 the detected plagiarism.

2.4 What you need to turn in

Your submission must be a **zip file** with the naming convention: **firstname_lastname_hw3.zip** (all lowercase, e.g., yijun_lin_hw3.zip). You should pack the following required (and optional) files in a folder named **firstname_lastname_hw3** (all lowercase, e.g., yijun_lin_hw3) in the zip file (Figure 1, only the files in the red boxes are required to submit):

a. [REQUIRED] two Python scripts containing the main function, named:

firstname_lastname_task1.py, firstname_lastname_task2.py

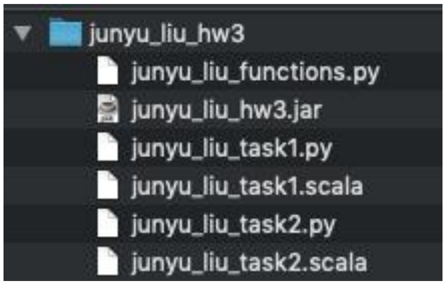
b1. [OPTIONAL] two Scala scripts containing the main function, named:

firstname_lastname_task1.scala, firstname_lastname_task2.scala

b2. [OPTIONAL] one Jar package, named:

firstname_lastname_hw3.jar

c. [OPTIONAL] You can include other scripts to support your programs (e.g., callable functions), but you need to make sure after unzipping, they are all in the same folder "firstname_lastname_hw3".



junyu_liu_hw3
junyu_liu_functions.py
junyu_liu_hw3.jar
junyu_liu_task1.py
junyu_liu_task1.scala
junyu_liu_task2.py
junyu_liu_task2.scala

d. You don't need to include any result. We will grade your code using our testing data. Our testing data will be in the same format as the validation dataset.

Figure 1: The folder structure after your submission file is unzipped.

3. Yelp Data

We generated the following two datasets from the original Yelp review dataset with some filters such as the condition: "state" == "CA". We randomly took 60% of the data as the training dataset, 20% of the data as the validation dataset, and 20% of the data as the testing dataset.

a. yelp_train.csv: the training data, which only include the columns: user_id, business_id, and stars.

b. yelp_val.csv: the validation data, which are in the same format as training data.

c. We do not share the testing dataset.

4. Tasks

4.1 Task1: Jaccard based LSH (4 points)

In this task, you will implement the Locality Sensitive Hashing algorithm with Jaccard similarity using **yelp_train.csv**. You can refer to sections 3.3 – 3.5 of the Mining of Massive Datasets book.

In this task, we focus on the "0 or 1" ratings rather than the actual ratings/stars from the users. Specifically, if a user has rated a business, the user's contribution in the characteristic matrix is 1. If the user hasn't rated the business, the contribution is 0. **You need to identify similar businesses whose similarity >= 0.5.**

You can define any collection of hash functions that you think would result in a consistent permutation of the row entries of the characteristic matrix. Some potential hash functions are:

f(x)=(ax+b)%m or **f(x) = ((ax + b) % p) % m**

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

After you have defined all the hashing functions, you will build the signature matrix. Then you will divide the matrix into *b* bands with *r* rows each, where *b x r = n (n is the number of hash functions)*. **You should carefully select a good combination of *b* and *r* in your implementation**. Remember that two items are a candidate pair if their signatures are identical in at least one band.

Your final results will be the candidate pairs whose original Jaccard similarity is **>=0.5**. You need to write the final results into a CSV file according to the output format below.

Example of Jaccard Similarity:

	user1	user2	user3	user4
business1	0	1	1	1
business2	0	1	0	0
Jaccard Similarity (business1, business2) = #intersection / #union = 1/3				

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

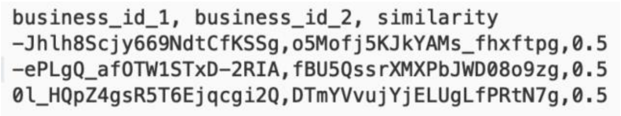
```
Python: $ ./bin/spark-submit firstname_lastname_task1.py <input_file_name> <output_file_name>

Scala:  $ ./bin/spark-submit --class firstname_lastname_task1 firstname_lastname_hw3.jar <input_file_name> <output_file_name>
```

Param: input_file_name: the name of the input file (e.g., yelp_train.csv), including the file path. Param: output_file_name: the name of the output CSV file, including the file path.

Output format:

IMPORTANT: Please strictly follow the output format since your code will be graded automatically. We will not regrade on formatting issues.

a. The output file is a CSV file, containing all business pairs you have found. The header is “business_id_1, business_id_2, similarity”. Each pair itself must be in the alphabetical order. The entire file also needs to be in the alphabetical order. There is no requirement for the number of decimals for the similarity value. Please refer to the format in Figure 2.



```
business_id_1, business_id_2, similarity
-Jhlh8Scjy669NdtCfKSSg,o5Mofj5KJkYAMs_fhxftpg,0.5
-ePLgQ_afOTW1STxD-2RIA,fBU5QssrXMXPbJWD08o9zg,0.5
0l_HQpZ4gsR5T6Ejqcgi2Q,DTmYVvujYjELUgLfPRtN7g,0.5
```

Figure 2: a CSV output example for task1

Grading:

We will compare your output file against the ground truth file using the **precision and recall metrics**.

Precision = true positives / (true positives + false positives)

Recall=truepositives/(truepositives+falsenegatives)

The ground truth file has been provided in the Google drive, named as “pure_jaccard_similarity.csv”. You can use this file to compare your results to the ground truth as well.

The ground truth dataset only contains the business pairs (from the yelp_train.csv) whose Jaccard similarity **>=0.5**. The business pair itself is sorted in the alphabetical order, so each pair only appears once in the file (i.e., if pair (a, b) is in the dataset, (b, a) will not be there).

In order to get full credit for this task you should have **precision=1 and recall>=0.98**. If not, then you will get 80% partial credit with 0.95 <= **recall** < 0.98:

Your runtime should be **less than 120 seconds** with 4G driver memory and 4G executor memory. This is the environment of the grading machine. If your runtime is more than or equal to 120 seconds, you will not receive any point for this task.

4.2 Task2: Recommendation System (8.5 points)

In task 2, you are going to build different types of recommendation systems using the **yelp_train.csv** to predict for the ratings/stars for given user ids and business ids. You can make any improvement to your recommendation system in terms of the **speed** and **accuracy**. You can use the validation dataset (yelp_val.csv) to evaluate the accuracy of your recommendation systems.

There are two options to evaluate your recommendation systems. You can compare your results to the correspond ground truth and compute the absolute differences. You can divide the absolute differences into 5 levels and count the number for each level as following:

>=0 and <1: 12345
>=1 and <2: 123
>=2 and <3: 1234
>=3 and <4: 1234
>=4: 12

This means that there are 12345 predictions with < 1 difference from the ground truth. This way you will be able to know the error distribution of your predictions and to improve the performance of your recommendation systems.

Additionally, you can compute the RMSE (Root Mean Squared Error) by using following formula:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i}(Pred_i - Rate_i)^2}$$

Where $Pred_i$ is the prediction for business *i* and $Rate_i$ is the true rating for business *i*. n is the total number of the business you are predicting.

In this task, you are required to implement:

Case 1: Model-based CF recommendation system with Spark MLlib (2.5 point)
Case 2: User-based CF recommendation system (3 points)
Case 3: Item-based CF recommendation system (3 points)

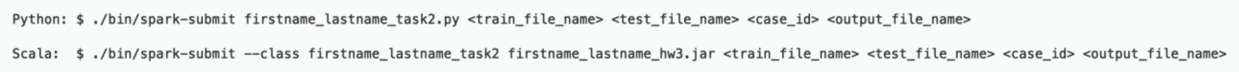
4.2.1. Model-based CF recommendation system

You will use Spark MLlib to implement this task. You can learn more about Spark MLlib by this link: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html.

4.2.2. User-basedCFrecommendationsystem

4.2.3. Item-based CF recommendation system

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



```
Python: $ ./bin/spark-submit firstname_lastname_task2.py <train_file_name> <test_file_name> <case_id> <output_file_name>

Scala:  $ ./bin/spark-submit --class firstname_lastname_task2 firstname_lastname_hw3.jar <train_file_name> <test_file_name> <case_id> <output_file_name>
```

Param: train_file_name: the name of the training file (e.g., yelp_train.csv), including the file path Param:

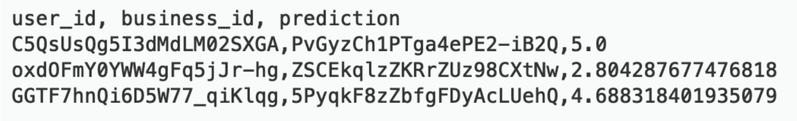
test_file_name: thenameofthetestingfile (e.g., yelp_val.csv), includingthefilepath Param: case_id: the case

number (i.e., 1, 2, or3)

Param: output_file_name: the name of the prediction result file, including the file path

Output format:

a. The output file is a CSV file, containing all the prediction results for **each user and business pair** in the validation/testing data. The header is "user_id, business_id, prediction". There is no requirement for the order in this task. There is no requirement for the number of decimals for the similarity values. Please refer to the format in Figure 3.



```
user_id, business_id, prediction
C5QsUsQg5I3dMdLM02SXGA,PvGyzCh1PTga4ePE2-iB2Q,5.0
oxdOFmY0YWW4gFq5jJr-hg,ZSCEkqlzZKRrZUz98CXtNw,2.804287677476818
GGTF7hnQi6D5W77_qiKlqg,5PyqkF8zZbfgFDyAcLUehQ,4.688318401935079
```

Figure 2: Output example in CSV for task2

Grading:

We will compare your prediction results against the ground truth. We will grade on all the cases in Task2 based on your accuracy using RMSE. For your reference, the table below shows the RMSE baselines and suggested running time for predicting the validation data.

RMSE baseline and suggested running time for predicting the validation data			
	Case 1	Case 2	Case 3
RMSE	1.31	1.12	1.10
Running Time	40s	150s	100s

For grading, we will use **the testing data** to evaluate your recommendation systems. If you can pass the RMSE baselines in the above table, you should be able to pass the RMSE baselines for the testing data. However, if your recommendation system only passes the RMSE baselines for the validation data, you will receive 50% of the points for each case.

In case 2 and 3, if your RMSE can not pass the baseline, you will receive partial credit:

case2: 100% RMSE <= 1.12,   80% 1.12 < RMSE <= 1.18 else 0%
case3: 100% RMSE <= 1.10,   80% 1.10 < RMSE <= 1.17 else 0%

5. Grading Criteria

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

1. You can use your free 5-day extension separately or together.
2. There will be 10% bonus if you use both Scala and Python.
3. If we cannot run your programs with the command we specified, there will be an 80% penalty.
4. If your program cannot run with the required Scala/Python/Spark versions, there will be a 20% penalty.
5. If our grading program cannot find a specified tag, there will be no point for this question.
6. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
7. If the header of the output file is missing, there will be 10% penalty.
8. We can regrade on your assignments within seven days once the scores are released. No argue after one week. The regrading will have a 20% penalty if our grading is correct.
9. There will be a 20% penalty for late submissions within a week and no point after a week.
10. There will be no point if the total execution time for all tasks exceeds 20 minutes.
11. Only when your results from Python are correct, the bonus of using Scala will be calculated. There is no partially point for Scala. See the examples below:

Example situations

Task	Score for Python	Score for Scala (10% of previous column if correct)	Total
Task1	Correct: 4points	Correct: 4 * 10%	4.4
Task1	Wrong: 0 point	Correct: 0 * 10%	0.0
Task1	Partially correct: 2 points	Correct: 2 * 10%	2.2
Task1	Partially correct: 2 points	Wrong: 0	2.0