INF 553 – Fall 2019
Assignment 3 LSH & Recommendation System

Deadline: 11/01 2019 11:59 PM PST

Assignment Overview

This assignment contains two parts. First, you will implement an LSH algorithm, using both Cosine and Jaccard similarity measurement, to find similar products. Second, you will implement a collaborative-filtering recommendation system. The dataset you are going to use is Yelp dataset (https://www.yelp.com/dataset) . The task sections below explain the assignment instructions in detail. The goal of the assignment is to make you understand how different types of recommendation systems work and more importantly, try to find a way to improve the accuracy of the recommendation system yourself. This assignment is the same as our Data Mining Competition that will end on December 17th. You can continue to improve your recommendation accuracy and submit your scores to compete.

Environment Requirements

Python 3.6, Scala 2.11 and Spark 2.3.3

IMPORTANT: We will use these versions to compile and test your code. If you use other versions, there will be a 20% penalty since we will not be able to grade it automatically.

You can only use Spark RDD.

Write your own code!

For this assignment to be an effective learning experience, you must write your own code! I emphasize this point because you will be able to find Python implementations of most or perhaps even all of the required functions on the web. Please do not look for or at any such code! **Do not share code with other students in the class!!**

Submission Details

For this assignment you will need to turn in a Python, Java, or Scala program depending on your language of preference.

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".

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.

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.

Task1: LSH (50%)

LSH Algorithm

In this task, you will need to develop the LSH technique using the yelp_train.csv file. The goal of this task is to find similar products according to the ratings of the users. In order to solve this problem, you will need to read carefully through the sections 3.3 – 3.5 from Chapter 3 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.**

In task 1, you're required to implement two approach using different similarity measurements:

1. Jaccard based LSH (30%)

Implementation Guidelines - Approach

The original characteristic matrix must be of size [users] x [products]. Each cell contains a 0 or 1 value depending on whether the user has rated the product or not. Once the matrix is built, you are free to use any collection of hash functions that you think would result in a more consistent permutation of the row entries of the characteristic matrix.

Some potential hash functions could be of type:

$$f(x) = (ax + b) \% m \quad \text{or} \quad f(x) = ((ax + b) \% p) \% m$$

where p is *any prime number* and m is the *number of bins*.

You can use any value for the *a, b, p or m* parameters of your implementation.

Once you have computed all the required hash values, you must build the Signature Matrix. Once

the Signature Matrix is built, you must divide the Matrix into *b* bands with *r* rows each, where *bands x rows = n (n is the number of hash functions),* in order to generate the candidate pairs. Remember that in order for two products to be a candidate pair their signature must agree (i.e., be identical) with at least one band.

Once you have computed the candidate pairs, your final result will be the candidate pairs whose **Jaccard Similarity is greater than or equal to 0.5**.

Example of Jaccard Similarity:

	user1	user2	user3	user4
product1	0	1	1	1
product2	0	1	0	0

Jaccard Similarity (product1, product2) = ***#products_intersection / #products_union = 1/3***

The program that you will implement should take two parameters as input and generate one file as an output. The first parameter must be the location of the *ratings.csv* file and the **second one must be the path to the output file followed by the name of the output file. The name of the output file must be *Firstname_Lastname_SimilarProducts_Jaccard.txt***. The content of the file must follow the same directions of question 1 in the *Questions & Grades Breakdown section* below. If your program does not generate this file or it does not follow the specifications as described in the following section, there would be a penalty of 50%.

Grading:

In order to get full credit for this question you should have *precision >= 0.9 and recall >= 0.8*. If not, then you will get partial credit based on the formula:

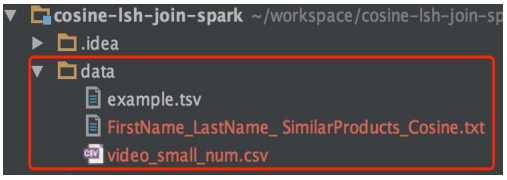
*(Precision / 0.9) * 15 + (Recall / 0.8) * 15*

And your run time should be less than **120 seconds**, or there'll be **20% penalty**.

2. Cosine based LSH (20%)

Please refer to https://github.com/soundcloud/cosine-lsh-join-spark and use the library to implement. You can clone the repository to your computer and learn from the existing code in *Main.scala* and finish the task.

The screenshot below demonstrates the structure of this project. **Please make sure to read input csv and write output result under */data* directory, which means your input path should be "data/yelp_train.csv" and your output file path should be "data/Firstname_Last Name_SimilarProducts_Cosine.txt"**



Grading:

In order to get full credit for this question you should have *precision >= 0.9 and recall >= 0.7*. If not, then you will get partial credit based on the formula:

*(Precision / 0.9) * 10 + (Recall / 0.7) * 10*

Input format:
Param: input_file_path: the name of the training file (e.g., yelp_train.csv), including the file path
Param: similarity method: the name of method to do the LSH (i.e., jaccard and cosine)
Param: output_file_path: the name of the prediction result file, including the file path

Result format:

1. **A file that contains the pairs of similar products that your algorithm has computed using LSH followed by their Jaccard similarity**. **(same for Cosine similarity)**

Example Format:
$product_1$, $product_2$, $Similarity_{12}$
$product_1$, $product_3$, $Similarity_{13}$
…
$product_n$, $product_k$, $Similarity_{nk}$

The file must be sorted ascending first by the left-hand side product and then sorted ascending by the right-hand side product. For example, the first row on the above snippet should be shorted firstly by $product_1$ and secondly by $product_2$.

Execution example:
Python:
spark-submit firstname_lastname_task1.py <input_file_path> < similarity method> <output_file_path>
Scala:
spark-submit --class firstname_lastname_task1 firstname_lastname_hw3.jar <input_file_path> < similarity method> <output_file_path>

Task2: Model-based CF/ User-based CF / Item-based CF Algorithms (50%)

In this task, 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 running time and accuracy. You can use the validation dataset (yelp_val.csv) to evaluate the accuracy of your recommendation systems.

You CANNOT use the ratings in the testing datasets to train your recommendation system. You can use the testing data as your ground truth to evaluate the accuracy of your recommendation system.

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

>=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 the following formula:

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

Where Predi is the prediction for business i and Ratei is the true rating for business i. n is the total number of the business you are predicting.

In task 2, you are required to implement:

1. **Model-based CF recommendation system by using Spark MLlib. (20%)**
   You can learn more about Spark MLlib by this link: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html
   We will grade on this task based on your accuracy. If your RMSE is greater than the baseline showing in the table below, there will be 20% penalty.

RMSE	1.30
Time(sec)	50s

2. **User-based CF recommendation system by yourself. (30%)**
   Below is the baseline for user-based CF. Both accuracy and time is measured. If your RMSE or run time is greater than the baseline, there'll be 20% penalty each.

RMSE	1.18
Time (sec)	180s

3. **Item-based CF integrating LSH result you got from task 1. (Bonus points: 10%)**
   In task 1, you've already found all the similar products pairs in the dataset. You need to use them to implement an item-based CF. Also measure the performance of it as previous one does.
   Comparing the result with CF without LSH and answering **how LSH could affect the recommendation system?** No baseline requirement for this one. **If you successfully implement it and have reasonable answer to the question, you can get the bonus points.**

Input format:

Param: train_file_path: the name of the training file (e.g., yelp_train.csv), including the file path
Param: test_file_path: the name of the testing file (e.g., yelp_val.csv), including the file path
Param: case_id: the case number (i.e., 1, 2, or 3)
Param: output_file_path: the name of the prediction result file, including the file path

Output format:

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.

Figure 2: Output example in CSV for task2

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

Execution Example:

Python:

spark-submit firstname_lastname_task2.py <train_file_path> <test_file_path> <case_id> <output_file_name>

Scale:

spark-submit --class firstname_lastname_task2 firstname_lastname_hw3.jar <train_file_path> <test_file_path> <case_id> <output_file_name>

Description File

Please include the following content in your description file:

1. Mention the Spark version and Python version
2. Same baseline table as mentioned in task 2 to record your accuracy and run time of programs in task 2
3. If you make any improvement in your recommendation system, please also describe it in your description file.

Submission Details

Your submission must be a .zip file with name: *<firstname>_<lastname>_hw3.zip*

Please include all the files in the right directory as following:

1. A description file: *<firstname>_<lastname>_desription.pdf*
2. Python scripts:
   <firstname>_<lastname>_task1.py
   <firstname>_<lastname>_task2.py
3. If you use Scala, all Scala scripts:
   <firstname>_<lastname>_task1.scala
   <firstname>_<lastname>_task2.scala
4. A jar package for all Scala file: *<firstname>_<lastname>_hw3.jar*
   If you use Scala for all tasks, please make all *.scala file into ONLY ONE *<firstname>_<lastname>_hw3.jar* file and strictly follow the class name mentioned above. And DO NOT include any data or unrelated libraries into your jar.
5. Required result files for task1 & 2. **NOTE: THIS LIST CONTAINS THE FILES WHICH SHOULD BE GENERATED BY YOU SUBMITTED CODE, DO NOT SUBMIT ANY RESULT FILE**:
   <firstname>_<lastname>_ SimilarProducts_Jaccard.txt
   <firstname>_<lastname>_ SimilarProducts_Cosine.txt
   <firstname>_<lastname>_ModelBasedCF.txt
   <firstname>_<lastname>_UserBasedCF.txt
   *<firstname>_<lastname>_ItemBasedCF.txt

* are required files for bonus points.

Grading Criteria:

1. If your programs cannot run with the commands you provide, your submission will be graded based on the result files you submit, and there will be an 80% penalty
2. If the files generated are not sorted based on the specifications, there will be 20% penalty.
3. If your program generates more than one file, there will be 20% penalty.
4. If the running time is greater than the base line, there'll be 20% penalty for each part as mentioned above.
5. **If your prediction result files miss any records, there will be 30% penalty**
6. **If you don't provide the source code, especially the Scala scripts, there will be 20% penalty.**
7. You can use your free 5-day extension.
8. There will be 10% bonus if you use Scala for the entire assignment.