

## INF 553 – Spring 2018

### Assignment 3 LSH & Recommendation System

**Deadline: 03/25 2017 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 datasets you are going to use are the Amazon Review datasets. 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 the May 2 at 7 pm. You can continue to improve your recommendation accuracy and submit your scores to compete.

#### **Environment Requirements**

Python: 2.7 Scala: 2.11 Spark: 2.2.1

**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.

Your submission must be a .zip file with name: **<Firstname>\_<Lastname>\_hw3.zip**. The structure of your submission should be identical as shown below. The `Firstname_Lastname_Description.pdf` file contains helpful instructions on how to run your code along with other necessary information as described in the following sections. The *OutputFiles* directory contains the deliverable output files for each problem and the *Solution* directory contains your source code.

- ▼  Firstname\_Lastname
  -  Firstname\_Lastname\_Description
  - ▶  OutputFiles
  - ▶  Solution

## Datasets

We are continually using Amazon Review data. This time we use the Amazon Instant Video category. We have already transferred the string id of user and product to integers for your convenience. You should download two files from Blackboard:

1. [video\\_small\\_num.csv](#)
2. [video\\_small\\_testing\\_num.csv](#)

## Task1: LSH (50%)

### LSH Algorithm

In this task, you will need to develop the LSH technique using the [video\\_small\\_num.csv](#) file. The dataset is provided to you under the `/data` folder of the bundled.zip file of the assignment. 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 the sections 3.3 – 3.5 from Chapter 3 of the Mining of Massive Datasets book.

In this problem, we will focus on 0-1 ratings rather than the actual ratings of the users. To be more specific, if a user has rated a product, then his contribution to the characteristic matrix is 1 while if he hasn't rated a product his contribution is 0. **Our goal is to identify similar products whose Similarity is greater or equal to 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  $[users] \times [products]$ . 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:

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

or

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

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

$bands \times rows = n$  ( $n$  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. After computing the final similar items, you must compare your results against the provided ground truth dataset using the precision and recall metrics. The ground truth dataset contains all the products pairs that have Jaccard similarity above or equal to 0.5. The ground truth dataset is located under the `/data` folder of the assignment's bundled .zip file and named as `video_small_ground_truth_jaccard.csv`.

Example of Jaccard Similarity:

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

$Jaccard\ Similarity\ (product1, product2) = \#products\_intersection / \#products\_union = 1/3$

Execution Example

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%.

**Java/Scala Execution Example:**

Please use **JaccardLSH** as class name

```
→ spark-2.2.1-bin-hadoop2.7 bin/spark-submit --class JaccardLSH Firstname_LastName_hw3.jar <input path> <output path>
```

**Python Execution Example:**

```
→ spark-2.2.1-bin-hadoop2.7 bin/spark-submit Firstname_LastName_task1_Jaccard.py <input path> <output path>
```

Grading:

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

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

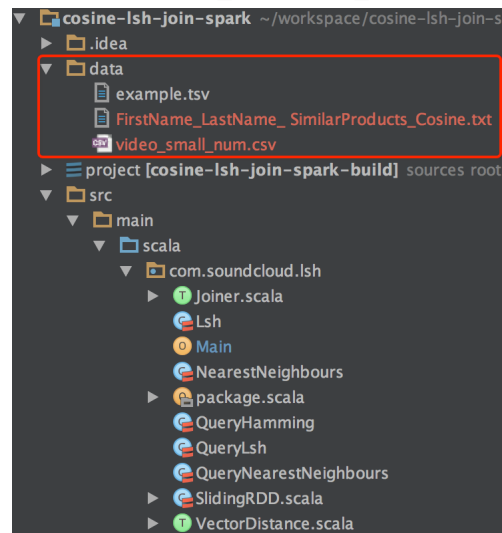
And your run time should be less than **120 seconds**, or there'll be **20% penalty**.

## 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. You **ONLY** need to submit your \*.scala file for this task. Please make sure to read input csv and write output result under */data* directory, which means your input path should be “data/video\_small\_num.csv” and your output file path should be “data/Firstname\_LastName\_SimilarProducts\_Cosine.txt”



Please also compare your results against the provided ground truth dataset using the precision and recall metrics. The ground truth dataset contains all the products pairs that have cosine similarity above or equal to 0.5. The ground truth dataset is located under the */data* folder of the assignment's bundled .zip file, and named as *video\_small\_ground\_truth\_cosine.csv*.

#### Grading:

In order to get full credit for this question you should have *precision*  $\geq 0.9$  and *recall*  $\geq 0.7$ .

If not, then you will get partial credit based on the formula:

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

#### 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<sub>1</sub>, product<sub>2</sub>, Similarity<sub>12</sub>

product<sub>1</sub>, product<sub>3</sub>, Similarity<sub>13</sub>

...

product<sub>n</sub>, product<sub>k</sub>, Similarity<sub>nk</sub>

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 sorted firstly by product<sub>1</sub> and secondly by product<sub>2</sub>.

2. For Similarity  $\geq 0.5$  compute the precision and recall for the similar products that your algorithm has generated against the ground truth data file under the data folder in the .zip file.

**Expressions:**

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

Recall = true positives / (true positives + false negatives)

## Task2: Model-based CF/ User-based CF / Item-based CF Algorithms

### (50%)

You are going to predict for *video\_small\_testing\_num.csv* datasets mentioned above. In your code, you can set the parameters yourself to reach better performance. You can make any improvement to your recommendation system: **speed, accuracy**.

The *video\_small\_testing\_num.csv* datasets are a subset of the *video\_small\_num.csv*, each having the same columns as its parent. Your goal is to predict the ratings of every <userId> and <productId> combination in the test files. You CANNOT use the ratings in the testing datasets to train your recommendation system. Specifically, you should first extract training data from the rating file downloaded from Amazon Review dataset using the testing data. Then by using the training data, you will need to **predict** rate for products in the testing datasets. You can use the testing data as your ground truth to evaluate the accuracy of your recommendation system.

**Example:**

Assuming *video\_small\_num.csv* contains 1 million records and the *video\_small\_testing\_num.csv* contains two records: (12345, 2, 3) and (12345, 13, 4). You will need to first remove the ratings of user ID 12345 on product IDs 2 and 13 from *video\_small\_num.csv*. You will then use the remaining records in the *video\_small\_num.csv* to train a recommendation system (1 million – 2 records). Finally, given the user ID 12345 and product IDs 2 and 13, your system should produce rating predictions as close as 3 and 4, respectively.

After achieving the prediction for ratings, you need to compare your result to the correspond ground truth and **compute the absolute differences**. You need to divide the absolute differences into 5 levels and count the number of your prediction for each level as following:

$\geq 0$  and  $< 1$ : 12345 //there are 12345 predictions with a  $< 1$  difference from the ground truth

$\geq 1$  and  $< 2$ : 123

$\geq 2$  and  $< 3$ : 1234

$\geq 3$  and  $< 4$ : 1234

$\geq 4$ : 12

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

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

Where  $Pred_i$  is the prediction for product  $i$ ,  $Rate_i$  is the true rating for product  $i$ ,  $n$  is the total number of the products. Read the Microsoft paper mentioned in class<sup>1</sup> to know more about how to use RMSE for evaluating your recommendation system.

In task 2, you are required to implement:

**1. Model-based CF recommendation system by using Spark MLlib. (20%)**

You can only use Scala to implement this task. 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.

$\geq 0$ and $< 1$	4679
$\geq 1$ and $< 2$	2031
$\geq 2$ and $< 3$	663
$\geq 3$ and $< 4$	292
$\geq 4$	35
RMSE	1.306126814254818

**2. User-based CF recommendation system by yourself. (30%)**

Below is the base line for user-based CF. Both accuracy and time is measured. If your RMSE or run time is greater than the base line, there'll be 20% penalty each.

$\geq 0$ and $< 1$	5178
$\geq 1$ and $< 2$	1857
$\geq 2$ and $< 3$	512
$\geq 3$ and $< 4$	130
$\geq 4$	23
RMSE	1.10331546222
Time (sec)	200

**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 base line requirement for this one. **If you successfully implement it and have reasonable answer to the question, you can get the bonus points.**

**Execution Example**

The first argument passed to our program (in the below execution) is the rating csv file. The second input is the path to the testing csv. Following we present examples of how you can run your program with spark-submit both when your application is a Java/Scala program or a Python script.

---

<sup>1</sup>Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In *Recommender systems handbook* (pp. 257-297). Springer US.

**A. Example of running a Java/Scala application with spark-submit:**

Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

You should use **ModelBasedCF**, **UserBasedCF**, and **ItemBasedCF** as your class name for 1, 2, and 3 in task 2 respectively.

```
→ spark-2.2.1-bin-hadoop2.7 bin/spark-submit --class <class name> Firstname_LastName_hw3.jar <rating file path> <testing file path> <output file path>
```

**B. Example of running a Python application with spark-submit:**

```
→ spark-2.2.1-bin-hadoop2.7 bin/spark-submit Firstname_LastName_task2_UserBasedCF.py <rating file path> <testing file path> <output file path>
```

**Result format:**

1. **Save the predication results in a txt file.** The result is ordered by <userId> and <productId> in ascending order.

*Example Format:*

user<sub>1</sub>, product<sub>2</sub>, prediction<sub>12</sub>

user<sub>1</sub>, product<sub>3</sub>, prediction<sub>13</sub>

...

user<sub>n</sub>, product<sub>k</sub>, prediction<sub>nk</sub>

2. **Print the accuracy and run time information** in terminal, and **copy this value** in your description file.

>=0 and <1: 12345

>=1 and <2: 123

>=2 and <3: 1234

>=3 and <4: 1234

>=4: 12

RMSE: 1.23456789

Time: 123 sec

## **Description File**

Please include the following content in your description file:

1. Mention the Spark version and Python version
2. Describe how to run your program for both tasks
3. The precision and recall for both Jaccard and Cosine based LSH in task 1.
4. Same baseline table as mentioned in task 2 to record your accuracy and run time of programs in task 2
5. 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. All Scala scripts:  
    <Firstname>\_<Lastname>\_task1\_Jaccard.scala  
    <Firstname>\_<Lastname>\_task1\_Cosine.scala  
    <Firstname>\_<Lastname>\_task2\_ModelBasedCF.scala  
    <Firstname>\_<Lastname>\_task2\_UserBasedCF.scala  
    \*<Firstname>\_<Lastname>\_task2\_ItemBasedCF.scala
3. 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.
4. If you use Python, then all python scripts: (Model-Based CF and Cosine-based LSH can only use Scala, please pack them into <Firstname>\_<Lastname>\_hw3.jar and follow the class naming convention as well)  
    <Firstname>\_<Lastname>\_task1\_Jaccard.py  
    <Firstname>\_<Lastname>\_task2\_UserBasedCF.py  
    \*<Firstname>\_<Lastname>\_task2\_ItemBasedCF.py
5. Required result files for task1 & 2:  
    <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.