# Inf553 – Foundations and Applications of Data Mining

Summer 2018 Assignment 3
LSH & Recommendation System

Deadline: 06/25 2018 11:59 PM PST

## 1 Overview of the Assignment

This assignment contains two parts. First, you will implement an LSH algorithm, using Jaccard similarity measurement, to find similar movies. Second, you will implement a collaborative-filtering recommendation system. The datasets you are going to use are the MovieLens 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.

### 1.1 Environment Requirements

Python: 2.7 Scala: 2.11 Spark: 2.2.1

Student must use python to complete both Task1 and Task2.

There will be 10% bonus if you also use Scala for both Task1 and Task2 (i.e. 10 - 11; 9 - 9.9).

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.**

**In order for you to understand more deeply of the Spark, use RDD only, you won't get any point if you use Dataframe or Dataset.**

### 1.2 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!

TA will combine some python code on Github which can be searched by keyword "INF553" and every students' code, using some software tool for detecting Plagiarism.

**Do not share code with other students in the class!!**

### Submission Details

For this assignment you will need to turn in a Python or Scala program depending on your language of preference. We will test your code using the same datasets but with different support thresholds values.

This assignment will surely need some time to be implemented so please plan accordingly and start early!

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.

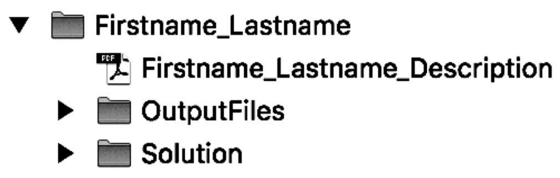


Figure 1: Submission Structure

## 2 Task 1: LSH (50 Points)

### 2.1 LSH Algorithm

In this task, you will need to develop the LSH technique using the ml-latest-small/ratings.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 movies 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 movie, then his contribution to the characteristic matrix is 1 while if he hasn't rated a movie his contribution is 0. Our goal is to identify similar movies whose Similarity is greater or equal to 0.5.

### 2.2 Jaccard based LSH (50 Points)

**Implementation Guidelines - Approach**

The original characteristic matrix must be of size [users] x [movies]. Each cell contains a 0 or 1 value depending on whether the user has rated the movie 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 x rows = n (n is the number of hash functions), in order to generate the candidate pairs. Remember that in order for two movies 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 movies 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 SimilarMovies.Goundtruth.05.csv.

**Example of Jaccard Similarity:**

| | user1 | user2 | user3 | user4 |
|---|---|---|---|---|
| movie1 | 0 | 1 | 1 | 1 |
| movie2 | 0 | 1 | 1 | 1 |

$$JaccardSimilarity(movie1, movie2) = \#movies\_intersection/\#movies\_union = 1/3$$

## Execution Requirements

The program that you will implement should take only one parameters. The parameter is the location of the ratings.csv file. The name of the output file must be Firstname_Lastname_SimilarMovies_Jaccard.txt. The content of the file must follow the same directions of question 1 in the Result Format 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%.

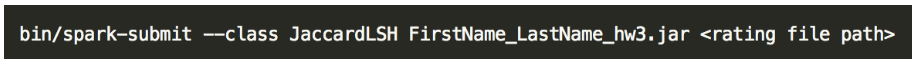
## Execution Example

The parameter passed to our program is the the location of the ratings.csv file. 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.

**Example of running application with spark-submit**

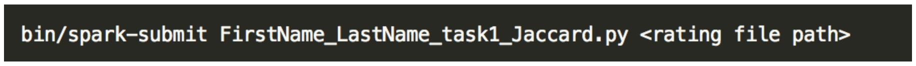
Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

Please use JaccardLSH as class name



```
bin/spark-submit --class JaccardLSH FirstName_LastName_hw3.jar <rating file path>
```

Figure 2: LSH: Command Line Format for Scala



```
bin/spark-submit FirstName_LastName_task1_Jaccard.py <rating file path>
```

Figure 3: LSH: Command Line Format for python

You don't need to specify the path of the output file in the commandline, you only need to save the file with the format Firstname_Lastname_SimilarMovies_Jaccard.txt. in the same path your program run.

## Result format

1. A file that contains the pairs of similar movies that your algorithm has computed using LSH followed by their Jaccard similarity. (40 Points)

Example Format:
$movie_1$, $movie_2$, $Similarity_{12}$
$movie_1$, $movie_3$, $Similarity_{13}$
...
$movie_n$, $movie_k$, $Similarity_{nk}$

The file must be sorted ascending first by the left-hand side movie and then sorted ascending by the right-hand side movie. For example, the first row on the above snippet should be shorted firstly by $movie_1$ and secondly by $movie_2$.

The file should be located under the OutputFiles directory of your submission with the name:
firstname_lastname_SimilarMovies_Jaccard.txt

2. For Similarity >= 0.5 compute the precision and recall for the similar movies that your algorithm has generated against the ground truth data file under the data folder in the .zip file. (10 Points)

**Expressions:**
Precision = true positives / (true positives + false positives)
Recall = true positives / (true positives + false negatives)

### 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:

$$(Precision/0.9) * 25 + (Recall/0.8) * 25$$

And your run time should be less than 240 seconds, or there'll be 20% penalty.

## 3 Task2: Model-based CF/ User-based CF / Item-based CF Algorithms (50 Points)

### 3.1 Datasets

The MovieLens datasets can be found in the following link: MovieLen

You will download two datasets: ml-20m.zip and ml-latest-small.zip. Once you extract the zip archives, you will find multiple data files. In this assignment, we will only use ratings.csv. However, you can combine other files to improve the performance of your recommendation system.

You will also download two testing files from Blackboard: testing_small.csv and testing_20m.csv. The testing datasets are a subset of the original dataset, each containing two columns: **userId** and **movieIds**. The file testing_small.csv (20256 records) is from ratings.csv in ml-latest-small, correspondingly the file testing_20m.csv (4054451 records) is a subset of ratings.csv in ml-20m. Your goal is to predict the ratings of every **userId** and **movieId** 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 ratings.csv file downloaded from MovieLenses using the testing data. Then by using the training data, you will need to **predict** rate for movies in the testing datasets. You can use the testing data as your ground truth to evaluate the accuracy of your recommendation system.

**Example:**

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

## 3.2 Model-based CF Algorithm (20 Points)

In task1, you are required to implement a Model-based CF recommendation system by using Spark MLlib. You can learn more about Spark MLlib by this link: MLlib

You are going to predict for both the small and large testing datasets mentioned above. In your code, you can set the parameters yourself to reach a better performance. You can make any improvement to your recommendation system: **speed**, **accuracy**.

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:

>=0 and <1: 12345 (there are 12345 predictions with a < 1 difference from the ground truth)
>=1 and <2: 123
>=2 and <3: 1234
>=3 and <4: 1234
>=4: 12

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

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

Where $Pred_i$ is the prediction for movie i, $Rate_i$ is the true rating for movie i, n is the total number of the movies. Read the Microsoft paper mentioned in class to know more about how to use RMSE for evaluating your recommendation system.

## 3.3 User-based CF Algorithm (30 Points)

In this part, you are required to implement a User-based CF recommendation system with Spark.

You are going to predict for only the small testing datasets mentioned above. You can make any improvement to your recommendation system: **speed**, **accuracy** (e.g., Hybird approaches). It's your time to design the recommendation system yourself, but first you need to beat the baseline.

After achieving the prediction for ratings, you need to compute the accuracy in the same way mentioned in Model-based CF.

### 3.4 Item-based CF integrating LSH result you got from Task 1 (10 Points Bonus)

In task 1, you have already found all the similar movies 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.**

## Result Format

1. Save the predication results in a text file. The result is ordered by **userId** and **movieId** in ascending order.

Example Format:
$user_1$, $movie_2$, $prediction_{12}$
$user_1$, $movie_3$, $prediction_{13}$
...
$user_n$, $movie_k$, $prediction_{nk}$

2. **Print the accuracy 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
```

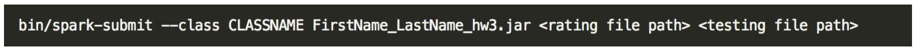
## Execution Example

The first argument passed to our program (in the below execution) is the rating csv file. The second input is 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.

**Example of running application with spark-submit**

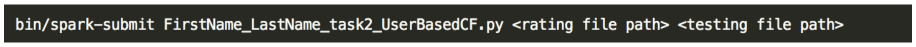
Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

Please use ModelBasedCF, UserBasedCF, ItemBasedCF as class name



```
bin/spark-submit —-class CLASSNAME FirstName_LastName_hw3.jar <rating file path> <testing file path>
```

Figure 4: CF: Command Line Format for Scala

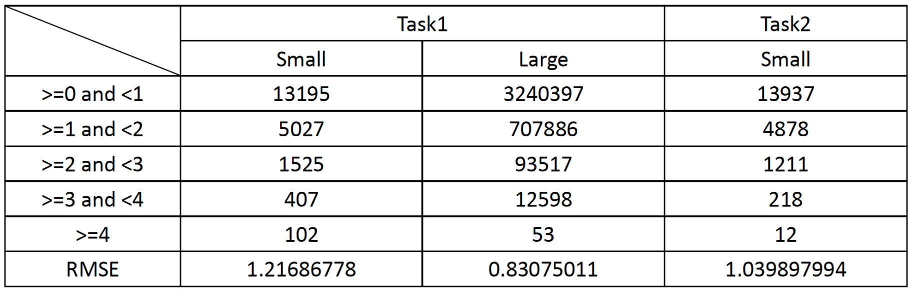


```
bin/spark-submit FirstName_LastName_task2_UserBasedCF.py <rating file path> <testing file path>
```

Figure 5: CF: Command Line Format for python

You don't need to specify the path of the output file in the commandline, you only need to save the file with the name format Firstname_Lastname_ModelBasedCF.txt. in the same path your program run.

## Base Line



| | Task1 | | Task2 |
|---|---|---|---|
| | Small | Large | Small |
| >=0 and <1 | 13195 | 3240397 | 13937 |
| >=1 and <2 | 5027 | 707886 | 4878 |
| >=2 and <3 | 1525 | 93517 | 1211 |
| >=3 and <4 | 407 | 12598 | 218 |
| >=4 | 102 | 53 | 12 |
| RMSE | 1.21686778 | 0.83075011 | 1.039897994 |

Figure 6: CF: Base Line

## 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 Jaccard LSH in task1.
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_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, 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:
Firstname_Lastname_task1_Jaccard.py
Firstname_Lastname_task2_ModelBasedCF.py
Firstname_Lastname_task2_UserBasedCF.py
*Firstname_Lastname_task2_ItemBasedCF.py
5. Required result files for task1 & 2:
Firstname_Lastname_SimilarMovie_Jaccard.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 be run with the commands you provide, your submission will be graded based on the result files you submit and 80% penalty for it.
2. If the files generated by your programs 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 you don't provide the source code and just the .jar file in case of a Java/Scala application there will be 100% penalty.**
5. If the running time is greater than the base line, there'll be 20% penalty for each part as mentioned above. 6. If your prediction result files miss any records, there will be 30% penalty
7. There will be 20% penalty for late submission within a week and 0 grade after a week.

8. You can use your free 5-day extension.
9. There will be 10% bonus if you use both Scala and python for the entire assignment.
**10. There will 0 grade if you use Dataframe or Dataset.**