

INF553 Foundations and Applications of Data Mining

SPRING 2020

Assignment 3

Deadline: March. 17th 3:00 PM PST

1. Overview of the Assignment

In Assignment 3, you will complete two tasks. The goal is to let you be familiar with 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 used in the previous assignments.

2. Assignment Requirements

2.1 Programming Language and Library Requirements

- a. There will be no scala bonus for this homework. 0.5 bonus point will be rewarded for the top 20% students whose RMSE of task2.3 is lowest.
- 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

Python 3.6.4, JDK 1.8 and Spark 2.4.4

We will use these library versions to compile and 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.

3. Yelp Data

In this assignment, the datasets you are going to use are from:

<https://drive.google.com/drive/folders/1-ZJ0vFGuvKqOxO-vp7lq-zkMj4tkyNnG?usp=sharing>

We generated the following two datasets from the original Yelp review dataset with some filters. 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.
- d. other datasets: providing additional information (like the average star or location of a business)

4. Tasks

4.1 Task1: Jaccard based LSH (2 point)

In this task, you will implement the Locality Sensitive Hashing algorithm with Jaccard similarity using **yelp_train.csv**.

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:

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

where p is any prime number and m is the number of bins. **Please Carefully design your hash functions.**

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 $b \times r = n$ (n is the number of hash functions). **You should carefully select a good combination of b and r in your implementation ($b > 1$ and $r > 1$).** 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)

`./spark-submit task1.py <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_af0TW1STxD-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**.

$$\text{Precision} = \text{true positives} / (\text{true positives} + \text{false positives})$$

$$\text{Recall} = \text{true positives} / (\text{true positives} + \text{false negatives})$$

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 ≥ 0.99 and recall ≥ 0.97** . If not, then you will get only partial credit based on the formula:

$$(\text{Precision} / 0.99) * 0.2 + (\text{Recall} / 0.97) * 0.2$$

Your runtime should be **less than 100 seconds**. If your runtime is more than or equal to 100 seconds, you will not receive any point for this task.

4.2 Task2: Recommendation System (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, but please don't include it as your training data.

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:

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

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: Item-based CF recommendation system with Pearson similarity (2 points)

Case 2: Model-based recommendation system (1 point)

Case 3: Hybrid recommendation system (2 point)

4.2.1. Item-based CF recommendation system

Please strictly follow the slides to implement an item-based recommendation system with Pearson similarity.

4.2.2. Model-based recommendation system

You need to use XGBRegressor(a regressor based on the decision tree) to train a model. You need to use this API https://xgboost.readthedocs.io/en/latest/python/python_api.html, the XGBRegressor inside package xgboost.

Please choose your own features from the provided extra datasets and you can think about it with customer thinking. For example, the average star of a user and the number of reviews most likely influence the prediction result. You need to figure out other features and train a model based on that. Use the validation dataset to validate your result and remember don't include it into your training data.

Here is a link to introduce the detail of xgbRegressor: <https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7>
<https://www.kaggle.com/gayathrydasika/xgb-regressor-basic>

4.2.3. Hybrid recommendation system.

Now you have the results from previous models and you need to think hardly and choose a way from the slides to combine them together and design a better hybrid recommendation system.

Here are two examples of hybrid:

Example1:

You can combine them together as a weighted average, which means:

$$final\ score = \alpha \times score_{item_based} + (1 - \alpha) \times score_{model_based}$$

The key idea is: the CF focuses on the neighbors of the item and the model-based RS focuses on the user and item themselves. Specifically, if the item has a smaller number of neighbors, then the weight of the CF should be smaller. Meanwhile, if two restaurants both are 4 stars and while the first one has 10 reviews, the second one has 1000 reviews, the average star of the second one is more trustable, so the model-

based RS score should weight more. You may need to find other features to generate your own weight function to combine them together.

Example2:

You can combine them together as a classification problem:

Again, the key idea is: the CF focuses on the neighbors of the item and the model-based RS focuses on the user and item themselves. As a result, in our dataset, some item-user pairs are more suitable for the CF while the others are not. You need to choose some features to classify which model should you choose for each item-user pair.

If you train a classifier, you are allowed to upload the pre-trained classifier model named "model.md" to save running time on the Vocareum. You can use pickle library, joblib library or others if you want. Here is an example: https://scikit-learn.org/stable/modules/model_persistence.html.

You also need to upload the training script named "train.py" to let us verify your model.

Some possible features (other features may also work):

Average star of user, average star of business, the variance of history review of a user or a business.

Number of reviews of a user or a business.

Yelp account starting date, number of fans.

The number of people think a users' review is useful/funny/cool. Number of compliments (Careful about these features. For example, sometimes when I visit a horrible restaurant, I will give full stars because I don't hope I am the only one who wasted money and time here. Sometimes people are satirical. :-))

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

Case1:

```
./spark-submit task2_1.py <train_file_name> <test_file_name><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: the name of the testing file (e.g., yelp_val.csv), including the file path

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

Case2:

```
./spark-submit task2_2.py <folder_path> <test_file_name><output_file_name>
```

Param: folder_path: the path of dataset folder, which contains exactly the same file as the google drive.

Param: test_file_name: the name of the testing file (e.g., yelp_val.csv), including the file path

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

Case3:

```
./spark-submit task2_3.py <folder_path> <test_file_name><output_file_name>
```

Param: folder_path: the path of dataset folder, which contains exactly the same file as the google drive.

Param: test_file_name: the name of the testing file (e.g., yelp_val.csv), including the file path

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
oxd0FmY0YWW4gFq5jJr-hg,ZSCEkqlzZKRrZUz98CXtNw,2.804287677476818
GGTF7hnQi6D5W77_qiKlqg,5PyqkF8zZbfgFDyAcLUehQ,4.688318401935079
```

Figure 3: 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 running time for predicting the validation data. The time limit of case3 is set to 30 minutes because we hope you think hard about that and try to improve that as much as possible, which will help you a lot in the competition.

	Case 1	Case 2	Case 3
RMSE	1.09	1.00	0.99
Running Time	130s	400s	1800s

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.

5. Submission

You need to submit following files on **Vocareum** with exactly the same name:

a. Four Python scripts:

- task1.py
- task2_1.py
- task2_2.py
- task2_3.py

6. 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. 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. If a plagiarism is detected, there will be no point for the entire assignment and we will report all detected plagiarism.
4. All submissions will be graded on the Vocareum. Please strictly follow the format provided, otherwise you can't get the point even though the answer is correct.
5. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
6. if you use Spark DataFrame, DataSet, sparksql, there will be 20% penalty.
7. We can regrade on your assignments within seven days once the scores are released. No argue after one week. There will be 20% penalty if our grading is correct.
8. There will be 20% penalty for late submission within a week and no point after a week.

7. Common problems causing fail submission on Vocareum/FAQ

(If your program runs seems successfully on your local machine but fail on Vocareum, please check these)

1. Try your program on Vocareum terminal. Remember to set python version as python3.6,

```
export PYSPARK_PYTHON=python3.6
```

And use the latest Spark

```
/home/local/spark/latest/bin/spark-submit
```

2. Check the input commend line format.
3. Check the output format, for example, the header, tag, typo.
4. Check the requirements of sorting the results.
5. Your program scripts should be named as task1.py task2_1.py etc.
6. Check whether your local environment fit the assignment description, i.e. version, configuration.
7. If you implement the core part in python instead of spark, or implement it in a high time complexity way (e.g. search an element in a list instead of a set), your program may be killed on the Vocareum because it runs too slow.
8. You are required to only use Spark RDD in order to understand Spark operations more deeply. You will not get any point if you use Spark DataFrame or DataSet. Don't import sparksql.