DSCI553 Foundations and Applications of Data Mining

Spring 2023

Assignment 3

Deadline: Mar 23 11:59 PM PST

1. Overview of the Assignment

In Assignment 3, you will complete two tasks. The goal is to familiarize you with Locality Sensitive Hashing (LSH), and different types of collaborative-filtering recommendation systems. The dataset you are going to use is a subset from the Yelp 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 points if you use Spark DataFrame or DataSet.

2.2 Programming Environment

Python 3.6, JDK 1.8, Scala 2.12, and Spark 3.1.2

We will use these library versions to compile and test your code. There will be no point if we cannot run your code on Vocareum. On Vocareum, you can call `spark-submit` located at /opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit`. (*Do not use the one at `/home/local/spark/latest/bin/spark-submit (2.4.4))

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/1SufecRrgj1yWMOVdERmBBUnqz0EX7ARQ?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 are not sharing the test dataset.

d. other datasets: providing additional information (like the average star or location of a business)

4. Tasks

Note: This Assignment has been divided into 2 parts on Vocareum. This has been done to provide more computational resources.

4.1 Task 1: Jaccard based LSH (2 points)

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 x 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)

Python: /opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit task1.py <input_file_name> <output_file_name>

Scala: spark-submit --class task1 hw3.jar <input_file_name> <output_file_name>

Param: input_file_name: the name of the input file (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 because of 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 sorted in lexicographical order.** This means if there are 2 businesses with business IDs "abd" and "abc" respectively and their similarity is 0.7, the output file should have

abc, abd, 0.7

And not

abd, abc, 0.7

The entire file also needs to be sorted in lexicographical 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 **precision and recall metrics**.

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

*(Precision / 0.99) * 0.4 + (Recall / 0.97) * 0.4*

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 Task 2: Recommendation System (5 points)

In task 2, you are going to build different types of recommendation systems using the **yelp_train.csv** to predict the ratings/stars for given user ids and business ids. You can make any improvement to your

recommendation system in terms of 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 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 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: 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.

Note: Since it is a CF-based recommendation system, there are some inherent limitations to this approach like cold-start. You need to come up with a default rating mechanism for such cases. This includes cases where the user or the business does not exist in the training dataset but is present in the test dataset. This is a part of the assignment and you are supposed to come up with ways to handle such issues on your own.

4.2.2. Model-based recommendation system

You need to use XGBregressor (a regressor based on 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 the package xgboost.

Please use version **0.72** of xbgoost package on your local system to avoid any discrepancies you might see between the results on your local system and Vocareum.

Please choose your own features from the provided extra datasets and you can think about it with customer thinking. For example, the average stars rated by a user and the number of reviews most likely influence the prediction result. You need to select 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.

4.2.3. Hybrid recommendation system.

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

Here are two examples of hybrid systems:

Example 1:

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 items 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 trustworthy, so the model-based RS score should weigh more. You may need to find other features to generate your own weight function to combine them together.

Example 2:

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 items 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 you should 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 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 stars of a user, average stars 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 who think a users' review is useful/funny/cool. Number of compliments (Be careful with these features. For example, sometimes when I visit a horrible restaurant, I will give full stars because I hope I am not the only one who wasted money and time here. Sometimes people are satirical. :-))

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

Case 1:

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

Case 2:

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

Case 3:

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
oxdOFmY0YWW4gFq5jJr-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 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 consider this factor and try to improve on it as much as possible (hint: this will help you a lot in the competition project at the end of the semester).

	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

b. [OPTIONAL] hw3.jar and Four Scala scripts:
   - task1.scala
   - task2_1.scala
   - task2_2.scala
   - task2_3.scala

6. Grading Criteria

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

1. You can use your free 5-day extension separately or together. (Google Forms Link for Extension: https://forms.gle/edH8jw1mJjrLFRcm8)
2. There will be a 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 plagiarism is detected, you will receive no points for the entire assignment and we will report all detected plagiarism.
4. All submissions will be graded on Vocareum. Please strictly follow the format provided, otherwise you won't receive points even though the answer is correct.
5. If the outputs of your program are unsorted or partially sorted, there will be a 50% penalty.
6. Do **NOT** use Spark DataFrame, DataSet, sparksql.
7. We can regrade your assignments within seven days once the scores are released. We will not accept any regrading requests after a week. There will be a 20% penalty if our grading is correct.
8. There will be a 20% penalty for late submissions within a week and no points after a week.
9. Only if your results from Python are correct will the bonus of using Scala be calculated. There are no partial points awarded for Scala. See the example below:

Example situations

Task	Score for Python	Score for Scala (10% of previous column if correct)	Total
Task1	Correct: 3 points	Correct: 3 * 10%	3.3
Task1	Wrong: 0 point	Correct: 0 * 10%	0.0
Task1	Partially correct: 1.5 points	Correct: 1.5 * 10%	1.65
Task1	Partially correct: 1.5 points	Wrong: 0	1.5

7. Common problems causing fail submission on Vocareum/FAQ

(If your program runs seem 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

/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit

2. Check the input command line format.

3. Check the output format, for example, the header, tag, typos.

4. Check the requirements of sorting the results.

5. Your program scripts should be named as task1.py task2.py etc.

6. Check whether your local environment fits 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 Vocareum because it runs too slowly.

8. You are required to only use Spark RDD in order to understand Spark operations more deeply. You will not get any points if you use Spark DataFrame or DataSet. Don't import sparksql.

9. Do not use Vocareum for debugging purposes, please debug on your local machine. Vocareum can be very slow if you use it for debugging.

10. Vocareum is reliable in helping you to check the input and output formats, but its function on checking the code correctness is limited. It can not guarantee the correctness of the code even with a full score in the submission report.

11. Some students encounter an error like: **the output rate …. has exceeded the allowed value ….bytes/s; attempting to kill the process.**

To resolve this, please remove **all print statements** and set the Spark logging level such that it limits the logs generated - that can be done using sc.setLogLevel . Preferably, set the log level to either **WARN** or **ERROR** when submitting your code.