DSCI553 Foundations and Applications of Data Mining

Summer 2021

Assignment 3

Deadline: June 15th 11:59 PM PST

1. Overview of the Assignment

In Assignment 3, you will complete three tasks. You will first implement Min-Hash and Locality Sensitive Hashing (LSH) to find similar businesses efficiently. Then you will implement various types of recommendation systems.

2. Requirements

2.1 Programming Requirements

a. You must use **Python & Spark** to implement all tasks. You can only use the standard Python libraries (i.e., external libraries like numpy or pandas are not allowed).

b. **You are required to only use Spark RDD**, i.e. no point if using Spark DataFrame or DataSet.

c. There will be 10% bonus for Scala implementation in each task. You can get the bonus only when both Python and Scala implementations are correct.

2.2 Programming Environment

Python 3.6, Scala 2.11, and Spark 2.3.0

We will use Vocareum to automatically run and grade your submission. You must test your scripts on **your local machine** and **the Vocareum terminal** before submission.

2.3 Write your own code

Do not share code with other students!!

For this assignment to be an effective learning experience, you must write your own code! We emphasize this point because you may find Python implementations of some of the required functions on the Web. Please do not look for or at any such code!

Plagiarism detection 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. We will report all detected plagiarism to the university.

3. Yelp Data

For this assignment, we have generated sample review data from the original Yelp review dataset using some filters, such as the condition: "state" == "CA". We randomly took 80% of sampled reviews for training, 10% for testing, and 10% as the blind dataset. (We do not share the blind dataset.) You can access and download the following JSON files either under the directory on the Vocareum: resource/asnlib/publicdata/ or on Google Drive (USC email only):

a. train_review.json

b. test_review.json – containing only the target user and business pairs for prediction tasks

c. test_review_ratings.json – containing the ground truth rating for the testing pairs

d. user_avg.json – containing the average stars for the users in the train dataset

e. business_avg.json – containing the average stars for the businesses in the train dataset

f. stopwords

g. We do not share the blind dataset.

4. Tasks

You need to submit the following files on Vocareum: (all in lowercase)

a. Python scripts: task1.py, task2train.py, task2predict.py, task3train.py, task3predict.py

b. Model files: task2.model, task3item.model, task3user.model

c. Result files: task1.res, task2.predict, task3item.predict, task3user.predict

d. Scala scripts: task1.scala, task2train.scala, task2predict.scala, task3train.scala, task3predict.scala; one jar package: hw3.jar

e. Model files: task2.scala.model, task3item.scala.model, task3user.scala.model

f. Result files: task1.scala.res, task2.scala.predict

g. [OPTIONAL] You can include other scripts to support your programs (e.g., callable functions).

4.1 Task1: Min-Hash + LSH (3.5pts)

4.1.1 Task description

In this task, you will implement the Min-Hash and Locality Sensitive Hashing algorithms with Jaccard similarity to find similar business pairs in the *train_review.json* file. We focus on **0/1 ratings** rather than the actual rating values in the reviews. In other words, if a user has rated a business, the user's contribution in the characteristic matrix is 1; otherwise, the contribution is 0 (Table 1). **Your task is to identify business pairs whose Jaccard similarity is >= 0.05.**

	user1	user2
business1	3	-
business2	-	3
business3	-	4
business4	5	4

	user1	user2
business1	1	0
business2	0	1
business3	0	1
business4	1	1

Table 1: The left table shows the original ratings; the right table shows the converted 0 and 1 ratings.

You can define any collection of hash functions to permutate the row entries of the characteristic matrix to generate Min-Hash signatures. Some potential hash functions are:

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

where p is any prime number; m is the number of bins. You can define any combination for the parameters (a, b, p, or m) in your implementation.

After you have defined all hash functions, you will build the signature matrix using Min-Hash. 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 need to set b and r properly to balance the number of candidates and the computational cost. Two businesses become a candidate pair if their signatures are identical in at least one band.

Lastly, you need to verify the candidate pairs using their original Jaccard similarity. Table 1 shows an example of calculating the Jaccard similarity between two businesses. Your final outputs will be the business pairs whose Jaccard similarity is >= 0.05.

	user1	user2	user3	user4
business1	0	1	1	1
business2	0	1	0	0

Table 2: Jaccard similarity (business1, business2) = #intersection / #union = 1/3

4.1.2 Execution commands

Python $ spark-submit task1.py <input_file> <output_file>

Scala $ spark-submit --class task1 hw3.jar <input_file> <output_file>

<input_file>: the train review set
<output_file>: the similar business pairs and their similarities

4.1.3 Output format

You must write a business pair and its similarity in the JSON format using **exactly the same tags like the example in Figure 1**. Each line represents a business pair, e.g., "b1" and "b2". For each business pair "b1" and "b2", you do not need to generate the output for "b2" and "b1" since the similarity value is the same as "b1" and "b2". You do not need to truncate decimals for the 'sim' values.

```
{"b1": "cYwJAgA6I12KNsd2rtXd5g", "b2": "Fid2ruy5s600SX4tvnrFgA", "sim": 0.032448377581120944}
{"b1": "7zecrDCEugcx8bgFn9LbLQ", "b2": "1VvxsddAoINg8TJXOZgEfg", "sim": 0.018867924528301886}
```

Figure 1: An example output for Task 1 in the JSON format

4.1.4 Grading

Your **task 1 outputs (2pt)** will be graded by precision and recall metrics defined below.

Precision = # true positives / # output pairs, Recall = # true positives / # ground truth pairs

Your precision should be **>= 0.95 (0.75pt**), and recall should be **>= 0.5 (0.75pt).** The execution time on Vocareum should be less than **200 seconds**. To evaluate the implementation, you can generate the ground truth that contains all the business pairs in the *train_review.json* file whose Jaccard similarity is >=0.05 and calculate precision and recall by yourself.

4.2 Task2: Content-based Recommendation System (4pts)

4.2.1 Task description

In this task, you will build a content-based recommendation system by generating profiles from review texts for users and businesses in the *train_review.json* file. Then you will use the model to predict if a user prefers to review a given business by computing the cosine similarity between the user and item profile vectors.

During the training process, you will construct the business and user profiles as follows:

a. Concatenating all reviews for a business as one document and parsing the document, such as removing the punctuations, numbers, and stopwords. Also, you can remove extremely rare words to reduce the vocabulary size. Rare words could be the ones whose frequency is **less than 0.0001%** of the total number of words.

b. Measuring word importance using TF-IDF, i.e., term frequency multiply inverse doc frequency

c. Using **top 200 words** with the highest TF-IDF scores to describe the document

d. Creating a Boolean vector with these significant words as the business profile

e. Creating a Boolean vector for representing the user profile by aggregating the profiles of the items that the user has reviewed

During the prediction process, you will estimate if a user would prefer to review a business by computing the cosine distance between the profile vectors. **The (user, business) pair is valid if their cosine similarity is >= 0.01. You should only output these valid pairs.**

4.2.2 Execution commands

Training commands:

Python $ spark-submit task2train.py <train_file> <model_file> <stopwords>

Scala $ spark-submit --class task2train hw3.jar < train_file> <model_file> <stopwords>

<train_file>: the train review set
<model_file>: the output model
<stopwords>: containing the stopwords that can be removed

Predicting commands:

Python $ spark-submit task2predict.py <test_file> <model_file> <output_file>

Scala $ spark-submit --class task2predict hw3.jar <test_file> <model_file> <output_file>

<test_file>: the test review set (only target pairs)
<model_file>: the model generated during the training process
<output_file>: the output results

4.2.3 Output format:

Model format: There is no strict format requirement for the content-based model.

Prediction format:

You must write the results in JSON format using **exactly the same tags** like the example in Figure 2. Each line represents a predicted pair of ("user_id", "business_id"). You do not need to truncate decimals for 'sim' values.

```
{"user_id": "1vXJWH7Lsdzsd8aU3SOsdA", "business_id": "ZzvfffV9kFY3ysdSgyRUBQ", "sim": 0.612348829899405}
{"user_id": "2svfwyX1hn2lsdjv5Sn36w", "business_id": "JAmQCczUclsDUfsdjNdjQA", "sim": 0.342154341436827}
```

Figure 2: An example prediction output for Task 2 in JSON format

4.2.4 Grading

You need to generate the content-based model and the prediction results **(2pt)**. We will grade your prediction results by calculating precision and recall using the ground truth (i.e., the blind reviews). The definitions of precision and recall are the same as the ones in task 1. Your precision should be >= **0.8 (1pt)** and recall should be **>= 0.7 (1pt)** for the blind datasets. The execution time of the training process on Vocareum should be less than **600 seconds**. The execution time of the predicting process on Vocareum should be less than **300 seconds**.

4.3 Task3: Collaborative Filtering Recommendation System (5pts)

4.3.1 Task description

In this task, you will build collaborative filtering (CF) recommendation systems using the *train_review.json* file. After building the systems, you will use the systems to predict the ratings for a user and business pair. You are required to implement 2 cases:

- Case 1: Item-based CF recommendation system **(2.5pts)**

During the training process, you will build a recommendation system by computing the Pearson correlation for the business pairs with **at least three co-rated users**. During the predicting process, you will use the system to predict the rating for a given pair of user and business. You must use **at most N business neighbors** who are the top N most similar to the target business for prediction (you can try various N, e.g., 3 or 5).

- Case 2: User-based CF recommendation system with Min-Hash LSH **(2.5pts)**

During the training process, you should combine the Min-Hash and LSH algorithms in your user-based CF recommendation system since the number of potential user pairs might be too large to compute. You need to (1) identify user pairs' similarity using their co-rated businesses without considering their rating scores (similar to Task 1). This process reduces the number of user pairs you need to compare for the final Pearson correlation score. (2) compute the Pearson correlation for the user pair candidates with **Jaccard similarity >= 0.01 and at least three co-rated businesses**. The predicting process is similar to Case 1.

4.3.2 Execution commands

Training commands:

Python $ spark-submit task3train.py <train_file> <model _file> <cf_type>

Scala $ spark-submit --class task3train hw3.jar < train_file> <model _file> <cf_type>

<train_file>: the train review set
<model_file>: the output model
<cf_type>: either “item_based” or “user_based”

Predicting commands:

Python $ spark-submit task3predict.py <train_file> <test_file> <model_file> <output_file> <cf_type>

Scala $ spark-submit --class task3predict hw3.jar <train_file> <test_file> <model_file> <output_file> <cf_type>

<train_file>: the train review set
<test_file>: the test review set (only target pairs)
<model_file>: the model generated during the training process
<output_file>: the output results
<cf_type>: either “item_based” or “user_based”

4.3.3 Output format:

Model format:

You must write the model in JSON format using **exactly the same tags like the example in Figure 3**. Each line represents a business pair (“b1”, “b2”) for the item-based model (Figure 3a) or a user pair (“u1”, “u2”) the for user-based model (Figure 3b). There is no need to have (“b2”, “b1”) or (“u2”, “u1”). You do not need to truncate decimals for ‘sim’ values.

```
{"b1": "eZcCFV-1vXJWH7Lsdzsd8a", "b2": "fB4fffV9kFY3ysdSgyRUBQ", "sim": 0.35478743759344955}
{"b1": "1vXJWH7Lsdzsd8aU3SOsdA", "b2": "HhVmDybpU7L50Kb5A0jXTg", "sim": 0.6204366813009298}
```

(a)

```
{"u1": "eZcCFV-1vXJWH7Lsdzsd8a", "u2": "fB4fffV9kFY3ysdSgyRUBQ", "sim": 0.35478743759344955}
{"u1": "1vXJWH7Lsdzsd8aU3SOsdA", "u2": "HhVmDybpU7L50Kb5A0jXTg", "sim": 0.6204366813009298}
```

(b)

Figure 3: (a) is an example of item-based model and (b) is an example of user-based model

Prediction format:

You must write a target pair and its prediction in the JSON format using **exactly the same tags like the example in Figure 4**. Each line represents a predicted pair of (“user_id”, “business_id”). You do not need to truncate decimals for ‘stars’ values.

```
{"user_id": "1vXJWH7Lsdzsd8aU3SOsdA", "business_id": "ZzvfffV9kFY3ysdSgyRUBQ", "stars": 3.607958829899405}
{"user_id": "2svfwyX1hn2lsdjv5Sn36w", "business_id": "JAmQCczUclsDUfsdjNdjQA", "stars": 1.442154461436827}
```

Figure 4: An example output for task3 in JSON format

4.3.4 Grading

You need to generate the item-based and user-based CF models. We will grade your model using precision and recall defined in task 1. For your item-based model, precision should be >= **0.9 (0.5pt)** and recall should be >=**0.9 (0.5pt)**. For your user-based model should, precision should be >= **0.4 (0.5pt)** and recall should be >=**0.5 (0.5pt)**.

Besides, we will compare your prediction results against the ground truth in both test and blind datasets. You should output the predictions **ONLY** generated from the model. Then we use RMSE (Root Mean Squared Error) defined in the equation below to evaluate the performance. For those pairs that your model cannot predict (e.g., due to cold start problem or too few co-rated users), we will predict them with the business average stars for the item-based model and the user average stars for the user-based model. We provide two files contain the average stars for users and businesses in the training dataset, respectively. The value of **"UNK"** tag, which can be used for predicting those new businesses and users, is the average stars for the whole reviews.

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

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 user and business.

The execution time of the training process on Vocareum should be less than **600 seconds**. The execution time of the predicting process on Vocareum should be less than **100 seconds**. RMSE for the item-based model in both test and blind datasets should be <=**0.91 (1.5pt)**, and for the user-based model in both datasets should be <=**1.01 (1.5pt)**. If the performance of only either one dataset reaches the threshold, you will obtain **1pt.**

5. About Vocareum

a. You can use the provided datasets under the directory: asnlib/publicdata/ (for Vocareum terminal, the directory is $ASNLIB/publicdata/). **So you do not need to upload the dataset to Vocareum.**

b. You should upload the required files under your workspace: work/

c. You must test your scripts on both the local machine and the Vocareum terminal before submission. **We do not accept the regrading request if the submission cannot be run or generate correct output formats on Vocareum.**

d. During the submission period, the Vocareum will directly evaluate the following result files: task1.res, task2.predict, task3item.model, and task3user.model. The Vocareum will also run task3predict scripts and evaluate the prediction results for both test and blind datasets.

e. During the grading period, the Vocareum will run both train and predict scripts. **If the training or predicting process fails to run, you can get 50% of the score only if the submission report shows that your submitted models or results are correct (regrading).**

f. Here are the commands that you can use to run Python scripts on Vocareum:

```
spark-submit task1.py $ASNLIB/publicdata/train_review.json task1.res
spark-submit task2train.py $ASNLIB/publicdata/train_review.json task2.model $ASNLIB/publicdata/stopwords
spark-submit task2predict.py $ASNLIB/publicdata/test_review.json task2.model task2.predict
spark-submit task3train.py $ASNLIB/publicdata/train_review.json task3item.model item_based
spark-submit task3predict.py $ASNLIB/publicdata/train_review.json $ASNLIB/publicdata/test_review.json task3item.model task3item.predict item_based
spark-submit task3train.py $ASNLIB/publicdata/train_review.json task3user.model user_based
spark-submit task3predict.py $ASNLIB/publicdata/train_review.json $ASNLIB/publicdata/test_review.json task3user.model task3user.predict user_based
```

g. You will receive a submission report after Vocareum finishes executing your scripts. The submission report should show **precision and recall** for each task. We do not test the Scala implementation during the submission period.

h. Vocareum will automatically run both Python and Scala implementations during the grading period.

i. The total execution time of the submission period should be less than **600 seconds**. The execution time of grading period needs to be less than **3000 seconds**.

j. Please start your assignment early! You can resubmit any script on Vocareum. We will only grade on your last submission.

k. You could add --driver-memory 4g --executor-memory 4g to your spark-submit command to limit its memory usage, in case your code could work properly in your local (with more resources) but would run into memory error. You can also use sc.setSystemProperty('spark.driver.memory', '4g') sc.setSystemProperty('spark.executor.memory', '4g') in your code.

l. Printing out too many outputs (printout) may crash the vocareum, so we recommend to use sc.setLogLevel("OFF") for your submission.

m. You can find a tutorial about Vocareum in the Week1 folder in D2L.

6. Grading Criteria

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

1. We do not have partial credits. For example, you will get **0** although your result covers 80% answer. **You will also get 0 if your outputs do not follow the format requirements (such as lexicographical order).** There will be no point if the total execution time exceeds the Section 4 evaluation metric.

2. You can use your free 5-day extension separately or together. You must submit a late-day request via https://forms.gle/syyUsyyTM684vf4K6. This form is recording the number of late days you use for each assignment. By default, we will not count the late days without your extension request. Please see the detail of late requests at Piazza. Also, you **are not able to use 5-day extension for the final HW (i.e., HW4)**.

3. There will be 20% penalty for the late submission within one week and no point after that. If you use your late days, there wouldn't be the 20% penalty.

4. There will be a 10% bonus for each task if both your python Scala implementations are correct. **The Scala bonus will not be calculated if your Python results are not correct.** There is no partial point for Scala.

5. There will be no point if your programs cannot be executed on Vocareum Please start your assignment early! You can resubmit on Vocareum. We will grade your last submission.

6. There is no regrading. Once the grade is posted on D2L, we will only regrade your assignments if there is a grading error. No exceptions.

7. There will be no point if your submission falls into the following situations:
    a. The submission cannot be executed on Vocareum. Each task will be run five times and graded on the best run.
    b. The execution failure on Vocareum is because of script naming issues, output file format issues.
    c. The code works on your local but it does not properly on Vocareum.