INF553 Foundations and Applications of Data Mining

Summer 2020

Assignment 3

Deadline: June. 23th 11:59 PM PST

1. Overview of the Assignment

In Assignment 3, you will complete three tasks. The goal is to let you be familiar with Min-Hash, Locality Sensitive Hashing (LSH), and various types of recommendation systems.

2. Requirements

2.1 Programming Requirements

a. You must use **Python** to implement all the tasks. You can only use 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 2.3.2

We will use Vocareum to automatically run and grade your submission. You must test your scripts on **the 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 will be able to find Python implementations of some of the required functions on the web. Please do not look for or at any such code!

TAs 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 detected plagiarism to the university.

3. Yelp Data

In this assignment, we generated the review data from the original Yelp review dataset with some filters, such as the condition: "state" == "CA". We randomly took 80% of the data for training, 10% of the data for testing, and 10% of the data as the blind dataset.

You can access and download the following JSON files either under the directory on the Vocareum: resource/asnlib/publicdata/ or in the Google Drive (**USC email ONLY**):

https://drive.google.com/open?id=1MXm2nAiKNB6ypBtfc33UweQqo3LmLEtJ

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 (optional)

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

f. We do not share the blind dataset.

4. Tasks

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

a. [REQUIRED] Python scripts: **task1.py**, **task2train.py**, **task2predict.py**, **task3train.py**, **task3predict.py**

b. [REQUIRED] Model files: **task2.model**, **task3item.model**, **task3user.model**

c. [REQUIRED] Result files: **task1.res**, **task2.predict**

d. [REQUIRED FOR BONUS] Scala scripts: **task1.scala**, **task2train.scala**, **task2predict.scala**, **task3train.scala**, **task3predict.scala**; one Jar package: **hw3.jar**

e. [REQUIRED FOR BONUS] Model files: **task2.scala.model**, **task3item.scala.model**, **task3user.scala.model**

f. [REQUIRED FOR BONUS] Result files: **task1.scala.res**, **task2.scala.predict**

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

The memory limit for all tasks is **4GB**.

4.1 Task1: Min-Hash + LSH (3pts)

4.1.1 Task description

In this task, you will implement the Min-Hash and Locality Sensitive Hashing algorithms with Jaccard similarity[1] to find similar business pairs in the train_review.json file. We focus on **the 0 or 1 ratings** rather than the actual ratings/stars in the reviews. 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. Table 1 shows an example. **Your task is to identify business pairs whose Jaccard similarity is >= 0.05.**

[1] If you want to learn more about how to use Min-Hash and LSH with the Cosine similarity, take a look at here: https://github.com/soundcloud/cosine-lsh-join-spark

	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 0 or 1 ratings.

Hint: 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:

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

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

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

You could also consider generating hash functions dynamically for each permutation.

$$f(x) = \big((i * h1(x) + i * h2(x) + i * i) \% p\big) \% m$$

where i is the permutation index, $h1$ and $h2$ are fixed hash functions.

After you have defined all the hash functions, you will build the signature matrix using Min-Hash. Then you will divide the matrix into $\boldsymbol{b}$ bands with $\boldsymbol{r}$ rows each, where $\boldsymbol{b} \times \boldsymbol{r} = \boldsymbol{n}$ ($\boldsymbol{n}$ is the number of hash functions). You need to set $\boldsymbol{b}$ and $\boldsymbol{r}$ 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 as the example in Figure 1**. Each line represents a business pair ("b1", "b2") with one Json object. Order of b1 and b2 does not matter. There is no need to have an output for ("b2", "b1").

```
{"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

You could generate the ground truth that contains all the business pairs (from the train review set) whose Jaccard similarity is >=0.05. Then you could compare your task 1 outputs against the ground truth using the following metrics. Alternatively, you could use the total number of ground truth pairs (59435) for your calculation if your result doesn't have false positive. **Your accuracy should be >= 0.8.** The execution time on Vocareum should be less than **200** seconds.

Accuracy = number of true positives / number of ground truth pairs

4.2 Task2: Content-based Recommendation System (3.5pts)

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 set. Then you will use the system/model to predict if a user prefers to review a given business, i.e., computing the cosine similarity between the user and item profile vectors.

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

a. Concatenating all the review texts for the business as the document and parsing the document, such as removing the punctuations, numbers, and stopwords. There is no strict rule for parsing sentences. But you need a reasonable parser so that your model can have a good prediction.

b. You can also consider removing extremely rare words in order to reduce the vocabulary size, i.e., the count is **less than 0.0001%** of the total words for all records.

c. Measuring word importance using TF-IDF, i.e., term frequency * inverse doc frequency

d. For each business, use **top 200 words** with highest TF-IDF scores to describe the document

e. Creating the business profiles. How to define the profile is up to you. The vectors can be Boolean vectors with all the significant words. You could also use other methods like sets of the significant words for the businesses, or other representation that you think saves memory. The key point is to reduce the size of the profiles to avoid potential out of memory problem

f. Creating corresponding user profiles by aggregating the profiles of the businesses that each user has reviewed

During the predicting 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 will be considered as a valid pair if their **cosine similarity is >= 0.01**. You should **only** output these **valid** pairs. If a business or user in the test set was not in the training set, your model would not be able to predict or output them. It is fine to ignore them.

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 for the content-based model.

Prediction format:

You must write the results in the JSON format using **exactly the same tags** as the example in Figure 2. Each line represents for a predicted pair of ("user_id", "business_id") with one Json object.

```
{"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 should be able to generate the content-based model as well as the prediction results **(1pt)**. We will compare your prediction results against the ground truth (i.e., the test reviews). **Your accuracy should be >= 0.7 for the test datasets (1pt)**, i.e., the number of identified pairs should be >= 70% of the total number of given user and business pairs. 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 (6pts)

4.3.1 Task description

In this task, you will build collaborative filtering recommendation systems with train reviews and use the models to predict the ratings for a pair of user and business. You are required to implement 2 cases:

- Case 1: Item-based CF recommendation system (3pts)

In Case 1, during the training process, you will build a model by computing the Pearson correlation for the business pairs that have **at least 3 co-rated users**. Your model only needs to contain the valid pairs that have positive Pearson similarity. During the predicting process, you will use the model to predict the rating for a given pair of user and business. You must use **at most N business neighbors** that are 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 (3pts)

In Case 2, during the training process, if the number of potential user pairs is too large to compute in memory, you could combine the Min-Hash and LSH algorithms in your user-based CF recommendation system. You need to (1) identify user pairs who are similar 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 that have **Jaccard similarity >= 0.01 and at least 3 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:

You should **ONLY** output the predictions that can be generated from the model.

Model format:

You must write the model in the JSON format using **exactly the same tags as the example in Figure 3**. Each line represents for a business pair (“b1”, “b2”) for item-based model (Figure 3a) or a user pair (“u1”, “u2”) for user-based model (Figure 3b). There is no need to have (“b2”, “b1”) or (“u2”, “u1”).

```
{"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 item-based model and (b) is an example user-based model

Prediction format:

You must write a target pair and its prediction in the JSON format using **exactly the same tags as the example in Figure 4**. Each line represents for a predicted pair of ("user_id", "business_id").


```
{"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

4.1.4 Grading

You should be able to generate the item-based and user-based CF models. We will compare your model to our ground truth. The number of similar pairs in your item-based model should **match at least 90%** of the pairs in the ground truth (0.5pt). The number of similar pairs in your user-based model should **match at least 50%** of the pairs in the ground truth (0.5pt).

The total number of ground truth pairs in task 3 case 1 is 55,7683 and case 2 is 78,3235.

Besides, we will compare your prediction results against the ground truth. You should **ONLY** output the predictions that can be generated from the model. Here is what **TA** does when calculating the RMSE. Understanding it is optional. For those pairs that your model cannot predict (e.g., due to cold start problem or too few co-rated users), we will first predict them with the business average stars for the item-based model and with 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. There is a tag **"UNK"** that is the overall average stars of the whole review, which can be used for predicting those new businesses and users. Then we use RMSE (Root Mean Squared Error) to evaluate the performance as the following formula (1pt/prediction):

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

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. For your reference, the table below shows the RMSE requirements for all the prediction tasks (1pt).

	Case 1 (Test)	Case 2 (Test)	Case 1 (Blind)	Case 2 (Blind)
RMSE	0.9	1.0	0.9	1.0

5. About Vocareum

a. You can use the provided datasets under the directory resource: /asnlib/publicdata/

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.

d. Make sure you use Python3.6 in Vocareum terminal: $ *export PYSPARK_PYTHON=python3.6*

e. During submission period, the Vocareum will **directly** evaluate the following result files that you upload: task1.res, task2.predict, task3item.model, and task3user.model. This means that you need to run your code in Vocareum terminal yourself to make sure that it could properly generate above files. The Vocareum will also run task3predict scripts and evaluate the prediction results for both test and blind sets.

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

g. Here are the commands (for Python scripts):

```
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
```

h. You will receive a submission report after Vocareum finishes executing your scripts. The submission report should show **the accuracy** for each task. We do **NOT** test the **Scala** implementation during the submission period.

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

j. The total execution time of submission period should be less than 600 seconds. The total execution time of grading period need to be less than 3000 seconds.

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

6. Grading Criteria

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

a. You can use your free 5-day extension separately or together. You must submit a late-day request via https://forms.gle/4WZa1wYPv8FWmtQP7 . This form is recording the number of late days you use for each assignment. By default, we will not count the late days if no request submitted.

b. There will be 10% bonus for **each** task if your Scala implementations are correct. Only when your Python results are correct, the bonus of Scala will be calculated. There is no partial point for Scala.

c. There will be no point if your submission cannot be executed on Vocareum.

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

e. There will be 20% penalty for the late submission within one week and no point after that.