Deadline + Late Penalty

Note : It will take you quite some time to complete this project, therefore, we earnestly recommend that you start working as early as possible.

- Submission deadline for the Project is **20:59:59 on 10th Jul, 2020** (Sydney Time).
- **LATE PENALTY: 10% on day-1 and 30% on each subsequent day.**

Instructions

1. This note book contains instructions for **COMP9313 Project 1**.
2. You are required to complete your implementation in the file `submission.py` provided along with this notebook.
3. You are not allowed to print out unnecessary stuff. We will not consider any output printed out on the screen. All results should be returned in appropriate data structures via corresponding functions.
4. You are required to submit the following files, via CSE `give` :
   - (i) `submission.py` (your code),
   - (ii) `report.pdf` (illustrating your implementation details)
   - **Note:** detailed submission instructions will be announced later.
5. We provide you with detailed instructions of the project in this notebook. In case of any problem, you can post your query @Piazza. Please do not post questions regarding the implementation details.
6. You are allowed to add other functions and/or import modules (you may have to for this project), but you are not allowed to define global variables. **All the functions should be implemented in** `submission.py` .
7. In this project, you may need to **CREATE YOUR OWN TEST CASES** in order to evaluate the correctness, while at the same time improving the efficiency of your implementation. **DO NOT COMPLETELY RELY ON THE TOY EXAMPLE IN THE SPEC!**
   - In order to create your own test cases, you are expected to use real datasets or randomly generated data, and generate hash functions by yourself.
8. The testing environment is the same as that of `Lab1` . **Note:** Importing other modules (not a part of the Lab1 test environment) may lead to errors, which will result in **ZERO score for the ENTIRE Project**.

- After completing the project, the students are **ENCOURAGED** to attempt for **BONUS** part. Detailed instructions for the **BONUS** part are given in the later part of this notebook.

Task1: C2LSH (90 points)

In this question, you will implement the C2LSH algorithm in Pyspark. Specifically, you are required to write a method `c2lsh()` in the file `submission.py` that takes the following four arguments as input:

1. **data_hashes**: is a rdd where each element (i.e., key,value pairs) in this rdd corresponds to (id, data_hash). `id` is an integer and `data_hash` is a python list that contains m integers (i.e., hash values of the data point).
2. **query_hashes** is a python list that contains m integers (i.e., hash values of the query).
3. **alpha_m** is an integer which indicates the minimum number of collide hash values between data and query (i.e., αm).
4. **beta_n** is an integer which indicates the minimum number of candidates to be returned (i.e., βn).

Note:

1. You don't need to implement hash functions and generate hashed data, we will provide the data hashes for you.
2. Please follow **the description of the algorithm provided in the lecture notes**, which is slightly different to the original C2LSH paper.
3. While one of the main purposes of this project is to use spark to solve the problems. Therefore, it is meaningless to circumvent pyspark and do it in other ways (e.g., collect the data and implement the algorithm without transformations etc.). Any such attempt will be considered as a invalid implementation, hence will be assigned **ZERO** score. Specifically, you are not allowed to use the following PySpark functions:

   - `aggregate`, `treeAggregate`, `aggregateByKey`
   - `collect`, `collectAsMap`
   - `countByKey`, `countByValue`
   - `foreach`
   - `reduce`, `treeReduce`
   - `saveAs*` (e.g. `saveAsTextFile`)
   - `take*` (e.g. `take`, `takeOrdered`)
   - `top`
   - `fold`

Return Format

The `c2lsh()` method returns a `rdd` which contains a sequence of candidate id's.

Notice: The order of the elements in the list does not matter (e.g., we will collect the elements and evaluate them as a set).

Evaluation

Your implementation will be tested using 3 different test cases. We will be evaluating based on the following factors:

- the correctness of implemented `c2lsh()`. The output will be compared with the result from the correct implementation. Any difference will be considered as incorrect.
- the efficiency of your implmentation. We will calculate the running time of `c2lsh()` in each testcase (denoted as T).

For each testcase (worth 30 points), the following marking criteria will be used:

- **Case 1, 0 points**: the returned `rdd` is incorrect, or $T > T_1$
- **Case 2, 10 points**: the returned `rdd` is correct, and $T_1 \geq T > T_2$,
- **Case 3, 20 points**: the returned `rdd` is correct, and $T_2 \geq T > T_3$,
- **Case 4, 30 points**: the returned `rdd` is correct, and $T_3 \geq T$.

Where $T_1 > T_2 > T_3$ depend on the testing environment and the test cases.

Task 2: Report (10 points)

You are also required to submit your project report, named: `report.pdf` . Specifically, in the report, you are at least expected to answer the following questions:

1. Implementation details of your `c2lsh()` . Explain how your major transform function works.
2. Show the evaluation result of your implementation using **your own test cases**.
3. What did you do to improve the efficiency of your implementation?

Bonus

In order to encourage the students to come up with efficient implementations, we allow bonus part of the project with a maximum of 20 points. Rules for the **BONUS** part are as under:

- Prerequisites:
  1. You must have obtained **90 points** for the implementation part.
  2. The **total running time** of your implementation for the three test cases is among the top-50 smallest running times of the class.
- All the submissions, satisfying the above-mentioned conditions will be tested against a more challenging dataset. Top-20 most-efficient and correct implementations will be awarded the bonus scores.
- We will rank the top-20 implementations in an increasing order w.r.t the running time. We will award 20 points to the most efficient implementation (i.e., the one with smallest running time), 19 points to the 2nd most efficient one, and so on. The implementation ranked 20-th on the list will get 1 bonus point.

How to execute your implementation (EXAMPLE)

In [1]:

```
from pyspark import SparkContext, SparkConf
from time import time
import pickle
import submission

def createSC():
    conf = SparkConf()
    conf.setMaster("local[*]")
    conf.setAppName("C2LSH")
    sc = SparkContext(conf = conf)
    return sc

with open("toy/toy_hashed_data", "rb") as file:
    data = pickle.load(file)

with open("toy/toy_hashed_query", "rb") as file:
    query_hashes = pickle.load(file)

alpha_m  = 10
beta_n = 10

sc = createSC()
data_hashes = sc.parallelize([(index, x) for index, x in enumerate(data)])
start_time = time()
res = submission.c2lsh(data_hashes, query_hashes, alpha_m, beta_n).collect()
end_time = time()
sc.stop()

# print('running time:', end_time - start_time)
print('Number of candidate: ', len(res))
print('set of candidate: ', set(res))
```



```
Number of candidate:  10
set of candidate:  {0, 70, 40, 10, 80, 50, 20, 90, 60, 30}
```

Project Submission and Feedback

For the project submission, you are required to submit the following files:

1. Your implementation in the python file `submission.py`.
2. The report `report.pdf`.

Detailed instruction about using `give` to submit the project files will be announced later via Piazza.

In [ ]:

```

```