Massive Data Sets
Assignment No. 3

Note 1 **This assignment is to be done individually**

Note 2 Working with other people is prohibited.

Note 2 Sharing queries or files with other people is prohibited.

A note on Academic Integrity and Plagiarism

Please review the following documents:

- Standards for Professional Behaviour, Faculty of Engineering:
  `https://www.uvic.ca/engineering/assets/docs/professional-behaviour.pdf`
- Policies Academic Integrity, UVic:
  `https://www.uvic.ca/students/academics/academic-integrity/`
- Uvic's Calendar section on Plagirism:
  `https://www.uvic.ca/calendar/undergrad/index.php#/policy/Sk_0xsM_V`

Note specifically:

> Plagiarism
> Single or multiple instances of inadequate attribution of sources should result in a failing grade for the work. A largely or fully plagiarized piece of work should result in a grade of F for the course.

Submissions will be screened for plagiarism **at the end of the term**.
You are responsible for your own submission, but you could also be responsible if somebody plagiarizes your submission.

1 Objectives

After completing this assignment, you will have experience:

- Using Scala and Spark
- Jaccard measure of set difference
- Minhash and local sensitivity hashing algorithms for the comparisons of large collections of sets.

2 Set difference

In this assignment we will implement the Jaccard metric for set difference. But for large collections of sets Jaccard is very expensive. For this reason we will implement the Minhash set comparision algorithm as an approximation for Jaccard, and further improve it with a Local Hash Sensitive version of Minhash.

3 Your task, should you choose to accept it

3.1 Preliminaries

Download from Brightspace the assignment files. As with Assignment 2, there are data and source code files.

Spark will be installed by sbt. You do not need to install it manually. However, make sure that you are not running Java 17 (Java 11 is ok).

3.2 Data files

The data files contain one set per line. Each line is a list of tokens. Each token is separated by a fixed string (see Command Line options below). The first token of a line is its record id, and the rest the sequence of tokens that make that record.

We will shingle each record. The number of tokens per shingle is given in a command line option. For example, assume the separator is `;` and the number of tokens per shingle is 3. The record:

```
id20;And;this;is;the;sea
```

corresponds to the set of shingles {('And','this,'is'),('this,'is','the'),('is','the','sea')} with id *id20*. Note that all shingles should be of the same size.

3.3 Command line options

Your program is run with the following arguments:

```
<filename> <delimiter> <minimumSimilarity> <shingleSize> <hashCount>
    <bandSize> <doJaccard> <doAllMinHashes> <printHashCoefficients>
     <outputHashFunctions>
```

- filename. Each line is a set, where each element of the set is delimited file by a given character.
- delimiter. A one character string that indicates the separator between tokens in a set. Use quotes around it to avoid the shell or sbt to interpret the character.
- minimumSimilarity: minimum similarity to report (between 0 and 1).
- shingleSize: number of tokens per shingle.
- hashCount: number of hash functions to use (for minHash)
- bandSize: size of the band to use (for LSH)
- doJaccard: boolean value (true or false). Indicates whether to do Jaccard comparison
- doAllMinHashes: boolean value (true or false). Indicates whether to do compare all the minHashes.
- outputHashFunctions: boolean. If true, print the coefficients of the hash functions (for debugging)

For example, the command (in one single line):

```
sbt --error 'set showSuccess := false'
     'run data/articles_100.txt  " " 0.8 3 100 4 true true false
```

uses the file `data/articles_100.txt` with delimiter space, minimum similarity of 0.8, size of the shingle equal to 3, 100 hashes, band size equal 4, do Jaccard comparison, do all minhashes comparison, and do not output hash function coefficients.

Its output to standard output is (some progress messages will be printed to standard error):

```
Computing similarity with parameters:
    Filename: data/articles_100.txt
    Separator: [ ]
    Minimum similarity: 0.8
    Shingle size: 3
    Hash count: 100
    Band size: 4
    Compute Jaccard similarity: true
    Compute all minHashes similarity: true
    Print hash coefficients: false

Pair                    Jaccard             minHash                   lsh
------------------------------------------------------------------------
t1088,t5015             0.98054             0.99000             0.96000
t1297,t4638             0.98062             0.96000             0.84000
t1768,t5248             0.98031             1.00000             1.00000
t1952,t3495             0.97844             0.97000             0.88000
t2023,t980              0.97916             0.95000             0.80000
------------------------------------------------------------------------
```

4 Set comparison algorithms

Implement the following functions, all found in the file `minhash.scala`.

4.1 shingle_line

This function converts an input line to a record with its corresponding set of hashes of shingles.

```
def shingle_line(stList: List[String], shingleSize:Int):Shingled_Record = ???
```

This function takes two parameters:

1. `stList`: The tokens in the input file, in the order they appear.
2. `shingleSize`: number of tokens per shingle

It should return a single record, which has 3 components:

1. Its `id`
2. The number of tokens shingled (do not include the id in this count).

3. A option set of the shingles. Each shingle is a list of exactly shingleSize elements. If the record contains less than `shingleSize` elements, then its shingles should be None.

- Throw an exception (any exception) if the list of tokens is empty.
- Use the function `utils.hash_string_lst` to compute the hash of the list of tokens of the shingle (they should be in the same order as they appear in the record).

Hint: you will find the method `sliding` useful to create the shingles.

4.2 compute_jaccard_pair

Compute the Jaccard similarity between two sets of shingles.

```
def compute_jaccard_pair(a:Shingled_Record, b: Shingled_Record): Similarity = ???
```

The two parameters are the records, each containing a set of shingles.

4.3 minhash_record

Compute the minhash of a given record

```
def minhash_record(r: Shingled_Record,
                   hashFuncs: List[Int => Int]) : Min_Hash_Record = ???
```

Takes 2 parameters:

1. `r`: record to minhash
2. `hashesFunc`: a list of n functions to use to compute the n hashes of the shingles. The parameter to each function is the shingle.

The return value is a `Min_Hash_Record` which contains the id of the record, and a vector of the hashes. This vector is the same size as the `hashFuncs`

4.4 find_jaccard_matches

```
def find_jaccard_matches(records: RDD[Shingled_Record],
                         minSimilarity:Double):Matches = ???
```

Takes 2 parameters:

1. `records`: an RDD of shingled records (as created by invocations to `shingle_line`)
2. `minSimilarity`: matches should have a similarity bigger or equal to this value.

Compute the Jaccard similarity between all records, but return only those that have at least the given similarity. `Matches` is an `Array` of `Similarity`. `Similarity` contains the ids of the two matching sets and their similarity. The invariant of the `Similarity` objects is that `idA` should be less than `idB`.

Hint: Use the method `cartesion` of RDD

4.5 find_minhash_matches

Find the matching sets using the MinHash algorithm.

```
def find_minhash_matches(records:RDD[Min_Hash_Record],
                         minSimilarity:Double): Matches = ???
```

Takes 2 parameters:

1. `records`: an RDD of shingled records (as created by `shingle_line`)
2. `minSimilarity`: matches should have a similarity bigger or equal to this value.

Similar to `find_jaccard_matches` but using the minHash algorithm instead of Jaccard.

4.6 find_lsh_matches

Find the matching sets using the Local Sensitivity Hashing algorithm.

```
def find_lsh_matches(minHashes: RDD[Min_Hash_Record],
                     minSimilarity: Double, bandSize: Int): Matches = ???
```

Takes 2 parameters:

1. `records`: an RDD of shingled records (as created by `shingle_line`)
2. `minSimilarity`: matches should have a similarity bigger or equal to this value.

Similar to `find_jaccard_matches` and `find_minhash_matches` but using the LSH algorithm on the minHashes.

- Start by creating a RDD of `Singled_Record`, of the same size as the input, but only have $\lceil hashCount/bandSize \rceil$ hashes.
- Note that the last band might have less than bandSize elements (that is ok).
- Use the function `utils.hash_int_lst` to compute the hash of the band.

5 Tests

For each test I have provided:

1. The dataset
2. The command line
3. The expected output

Your program's output should be identical to the expected output for the same command line and input dataset.

See Makefile

6 Other information and restrictions

I recommend you start with test-01. Make sure the hash coefficients match. If they do not, email me.

Violating any of the following restrictions will result in a **grade of zero**:

1. You cannot use any var binding.
2. You cannot add any imports to the program.

You can share test cases but you are forbidden from sharing source code.

7 What to submit

Submit, via Brigthspace:

1. the file minhash.scala