CSE427S

HOMEWORK 7

M. Neumann

Due: THU 25 OCT 2018 4PM

Getting Started

Update your SVN repository.

When needed, you will find additional materials for *homework x* in the folder `hwx`. So, for the current assignment the folder is `hw7`.

Hint: You can **check your submission** to the SVN repository by viewing `https://svn.seas.wustl.edu/repositories/<yourwustlkey>/cse427s_fl18` in a web browser.

SUBMISSION INSTRUCTIONS

WRITTEN:

- all written work needs to be submitted electronically in *pdf format*[1] via GRADESCOPE
- provide the following information on *every* page of your `pdf` file:
  - name
  - student ID
  - wustlkey (your `wustlkey` indicates the location (SVN repository) for your code submissions; without this information we will not be able to grade your code!)
- start every problem on a *new page*
- **FOR GROUPS**: make a **group submission** on GRADESCOPE and provide the following information for **all group members** on every page of your `pdf` file: names, student IDs, and location of code submissions (**one** student's `wustlkey`)[2]

CODE:

- code needs to be submitted via SVN repository commit (detailed submission instructions are provided whenever code submissions are required)
- make sure to always use the required *file name(s)* and *submission format(s)*
- **comment your code** to receive maximum credit

[1] Please, **type your solutions** or use **clear hand-writing**. If we cannot read your answer, we cannot give you credit nor will we be able to meet any regrade requests concerning your writing.

[2] It is sufficient to commit code to one SVN repository. If you do not specify the repository for your group's code submission clearly, we will only check the first repository, sorting wustlkeys alphabetically.

Problem 1: Computing Word Co-Occurrence (50%)

In Lab 6 we discussed co-occurrence based recommendations. A very similar Big data application is to compute the co-occurrence of words in text documents. These word co-occurrences, also called bi-grams and their frequencies can be used as features in many text mining and natural language processing tasks such as document categorization or plagiarism detection.

You will implement a MAPREDUCE program that counts the words-pairs in `shakespeare` that occur right next to each other. For this application the order of the words actually matters, so you will have to adapt the pseudo code of job 2 in the RS2 slides accordingly. For instance, in the toy example below (now,no) is a different bi-gram than (no,now).
Use the stubs provided in:

```
~/workspace/word_co-occurence/src/stubs/
```

(a) Implement the word co-occurrence MAPREDUCE program using the *pairs* approach. Do **not** perform any preprocessing such as stemming or stop-word filtering. Comment your code to receive maximum credit.

Use the following **test input** to test your implementation:

```
No now is no now no time
```

The output for this test input would be:

```
is,no     1
no,now    2
no,time   1
now,is    1
now,no    1
```

Once, everything works run your job on the `shakespeare` data.

(b) Check the output and provide the following statistics:

- How many different word pairs did your program produce?
- How often does the following word pairs occur:
  - the,lovers
  - loved,you
  - you,loved
  - verona,julia

HINT: the UNIX command **grep** will be useful to find those pairs in the output of your MAPREDUCE program.

(c) What is the actual *communication cost* of the job you just ran on the `shakespeare` input data? Include the input to all tasks of your MAPREDUCE job in your communication cost calculation.

HINT: You can find the required statistics to compute the actual communication cost of your job from the HADOOP provided **job counters** in the YARN Resource Manager Web UI. If you forgot how to access job counters, check hw6 problem 2(b).

(d) What is the drawback of merely counting the co-occurrence of words *directly* next to each other? Create two example sentences to illustrate the problem and describe the problem using your example.

(e) How does a *stripes* approach implementing word co-occurrence compare to your implementation? Consider both communication cost with respect to the number of different words k (size of the vocabulary) of your text document(s) and memory requirements of the reduce() functions in your answer.

Submit the implementation files of your Driver and Mapper.
To add the .java and results files to your SVN repo before committing run:

```
$ svn add WordCo.java
$ svn add WordCoMapper.java
$ svn commit -m 'hw7 problem 1 submission' .
```

Problem 2: Collaborative Filtering - Similarity Measures (20%)

(a) Show formally that the **normalized cosine similarity** measure corresponds to the **Pearson correlation**.

(b) **Quality vs. implementation effort and efficiency**

- From a quality perspective, what is the benefit of using the normalization (i.e., Pearson correlation instead of cosine similarity)?
- From an implementation and data storage perspective, what is the disadvantage of using the normalization (i.e., Pearson correlation instead of cosine similarity)?

(c) **Jaccard similarity**

- What is the main advantage of using the Jaccard similarity for collaborative filtering?
- What is the main disadvantage of the Jaccard similarity measure?
- Can you think of a way to pre-process the rating data to overcome this problem?

Problem 3: Collaborative Filtering in MAPREDUCE (30%)

The *dual approach* to collaborative filtering is to compute item-item similarites instead of user-user similarites and use those to fill in the missing values in the utility matrix.

(a) Explain two benefits of this approach.

(b) For the input data in the format `<user-id> <movie-id> <rating>` given below, compute the Mapper output, Reducer input and Reducer output for all MAPREDUCE jobs in a **collaborative filtering** approach to compute the **cosine similarity** between **all pairs of movies**. Round to two decimal places.

```
user1 movie1 1
user1 movie3 2
user1 movie2 3
user2 movie2 2
user2 movie3 3
user2 movie5 5
user3 movie1 1
user3 movie2 2
```

Bonus Problem (5% up to a max. of 100%) - no group work!

Write a review for this homework and store it in the file `hw7_review.txt` provided in your SVN repository (and commit your changes). This file should only include the review, **no other information** such as name, wustlkey, etc. Remember that you are not graded for the content of your review, solely it's completion.

Provide the star rating for `hw7` via this link: `https://wustl.az1.qualtrics.com/jfe/form/SV_aaAc0ILmBleyKzj`.
You can only earn bonus points if you write a meaningful review of **at least 50 words** and provide the corresponding star rating. Bonus points are given to the **owner of the repository only** (no group work!).

Submit your review in the file **`hw7_review.txt`** provided in the `hw7` folder in your SVN repository. **To commit the file run:**

```
$ svn commit -m 'hw7 review submission' .
```

Take 1 minute to provide a star rating for this homework – your star rating should match