UE19CS322 Big Data Assignment 2

Implementation of Page Rank Algorithm with Embeddings for Wikipedia

This is the second assignment for the UE19CS322 Big Data Course at PES University. The assignment consists of 2 tasks and focuses on running MapReduce jobs to implement a modified version of page rank that leverages graph embeddings.

The files required for the assignment can be found here.

Assignment Objectives and Outcomes

1. The objective of this assignment is for the students to run iterative processing with Map Reduce and learn how the Map Reduce algorithm works.
2. At the end of this assignment, the student will be able to write and debug Page Rank code on Map Reduce.

Ethical practices

Please submit original code only. You can discuss your approach with your friends but you must write original code. All solutions must be submitted through the portal. **We will perform a plagiarism check on the code and you will be penalised if your code is found to be plagiarised**.

The Dataset

The dataset is a network of hyperlinks from a snapshot of English Wikipedia in 2013. An edge from `i` to `j` indicates a hyperlink on page `i` to page `j` . The dataset you will be working with is only a **subset of the entire network** which can be obtained from the Stanford Network Analysis Project.

Each line of the dataset consists of two values, the `source page` and the `destination page` separated by `\t` . The pages are denoted using a numerical ID. For example, the dataset may look like the following.

```
15   2
28   8
1    12
```

```
13  6
16  7
17  8
3   9
4   9
9   11
17  10
12  6
```

Software/Languages to be used:

1. Python `3.8.x`
2. Hadoop `v3.2.2` only

Marks

Task 1: 2 marks

Task 2: 2 marks

Report: 1 mark

Tasks Overview:

1. Load the data into HDFS.
2. Create `mapper.py` and `reducer.py` for Task 1 and Task 2
3. Run your code on the sample dataset until you get the right answer
4. Submit the files to the portal
5. Submit one page report based on the template and answer the questions on the report

Submission Link

Portal for Big Data Assignment Submissions

Submission Deadline

23rd October, 11:59 PM

Submission Guidelines

You will need to make the following changes to your `mapper.py` and `reducer.py` scripts to run them on the portal

1. Include the following shebang on the first line of your code

```
#!/usr/bin/env python3
```

2. Convert your files to an executable

```
chmod +x mapper.py reducer.py
```

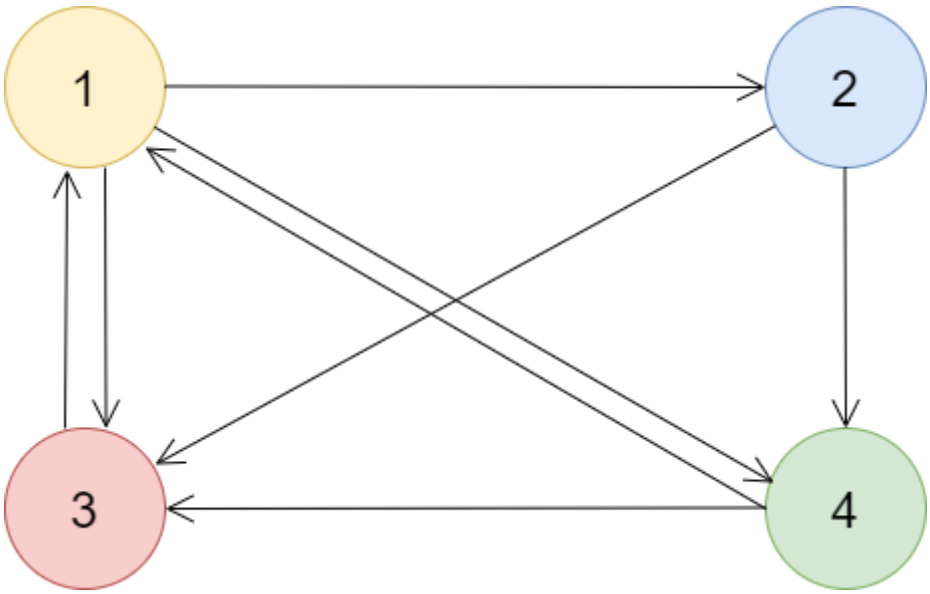
3. Convert line breaks in DOS format to Unix format (**this is necessary if you are coding on Windows** - your code will not run on our portal otherwise)

```
dos2unix mapper.py reducer.py
```

Check out a detailed list of submission guidelines here.

Task Specifications

The following graph will be used as an example to explain the sample input and outputs.



Task 1

Problem Statement

Convert an input file to an adjacency list using Map Reduce

Description

The input will be uploaded as a .txt file. Write Map and Reduce functions to read the input text file and generate two files representing the adjacency list and initial page ranks

Comments

1. The adjacency list should be written to HDFS, and the page rank vector should be written locally in a file called `v`
2. The path to the `v` file will be passed as a command line argument to the `reducer.py` file
3. **Important**: Never load the entire input text file or the adjacency list into your memory!
4. The input list may not be sorted, but it will be grouped by `from_node_id`

Input Format

The input to the mapper is the network of pages. The reducer receives the absolute path to the `v` file as a command line argument, where the initial page ranks are to be computed and stored.

Output Format

Display each node in the network along with its adjacent nodes. The output from the reducer *may* look like the following. The separator between the `from_node_id` and `list_of_adj_nodes` is up to your choice of implementation. The output should be sorted in **lexicographical** order of `from_node_id`.

```
from_node_id    list_of_adj_nodes
```

These initial page ranks should be written locally to a new file called `v` (**to be strictly followed**). The values are comma separated and newline delimited. The output should be sorted in **lexicographical** order of `node`.

```
node,pagerank
```

Example

1. Input network

```
2    3
2    4
1    2
1    3
1    4
3    1
4    3
4    1
```

2. `v` file containing initial page ranks, written locally

```
1,1
2,1
3,1
4,1
```

3. Output file containing adjacency list, written to HDFS

```
1    [2, 3, 4]
2    [3, 4]
3    [1]
4    [3, 1]
```

Task 2

Problem Statement

Iteratively calculate and update page ranks until convergence

Description

Write Map and Reduce functions to read the initial page ranks and the adjacency list file and using this calculate new page ranks. This process is repeated until convergence i.e, the new page ranks and the previous page ranks are similar.

The mapper will read the `v` file and the `adjacency list` and the reducer will compute the new page ranks based on the given equations.

(1) $Rank(p) = 0.15 + 0.85 \sum Contribution\ of\ nodes\ pointing\ to\ p$

where,

(2) $Contribution(p, q) = \frac{Rank'(p)\ .\ Similarity(p,q)}{Number\ of\ outgoing\ links\ from\ p}$

where $Rank'(p)$ is the *previous* rank of `p`, `p` is a node pointing to `q`, and

(3) $Similarity(p, q) = \frac{\overrightarrow{p}.\overrightarrow{q}}{|\overrightarrow{p}||\overrightarrow{q}|}$

where $\overrightarrow{p}$ and $\overrightarrow{q}$ are the vectors for page `p` and page `q` respectively which can be obtained from the `page embedding` file.

Comments

1. We will provide a bash script that will perform the following operations:
    - Mapper reads the `adjacency list`, `page embeddings` and `v` file and computes contributions
    - The `adjacency list` is read from HDFS
    - The `page embeddings` and `v` file are read locally, the paths to which are provided as command line arguments
    - Each page's embedding is a vector of size `5`, and this size will be fixed for all testcases
    - Reducer computes new page ranks writes output to `v1`
    - If values of `v` and `v1` are nearly similar (i.e, has reached convergence), exit
    - Else:
        - Delete `v` and rename `v1` to `v`
        - Redo from step 1
2. Reaching convergence means that the difference between the updated page ranks and the previous for every page should be < `CONVERGENCE_LIMIT`
3. The value of `CONVERGENCE_LIMIT` will be decided by the bash script - it will vary across different test cases
4. All ranks are to be **rounded off to 2 decimal places**. The **rounding off should only be done while printing to `STDOUT`** and not during computation.

Input Format

The mapper will receive two command line arguments, the absolute path to the `v` file and the absolute path to the `page embedding` file. The input to the mapper is the `adjacency list` generated from the previous task.

Output Format

For each page in the network, display the page's ID along with its updated page rank on a single line. The values are comma separated and newline delimited. The output should be sorted in **lexicographical** order of `node` .

```
node,pagerank
```

Example

Consider the following to be the input `page embeddings` for the provided sample network with 4 pages.

```
{
    "1": [
        0.032086015,
        0.108658746,
        0.34455177,
        0.54974973,
        -0.89134425
    ],
    "3": [
        -0.09123115,
        0.2637072,
        0.47960255,
        0.3426702,
        -0.9511681
    ],
    "4": [
        0.18350898,
        0.09125652,
        0.19741566,
        0.54505724,
```

```
        -0.91121626
    ],
    "2": [
        -6.809274e-05,
        0.19512779,
        0.31758854,
        0.24154831,
        -1.027901
    ]
}
```

Attached below are the **initial** `page ranks` for the provided network which have been generated from Task 1

```
1,1
2,1
3,1
4,1
```

Here is the `adjacency list` generated from the previous Task as well

```
1   [2, 3, 4]
2   [3, 4]
3   [1]
4   [3, 1]
```

The above `adjacency list` can be converted to the following matrix `M` where `M[i][j]` stores the initial contribution of page `i` to page `j` **before the similarity scores have been multiplied**.

#	page 1	page 2	page 3	page 4
page 1	0	0.33	0.33	0.33
page 2	0	0	0.5	0.5
page 3	1	0	0	0
page 4	0.5	0	0.5	0

As mentioned in equation `3`, the expected similarity matrix `S` will look like this, where `S[i][j]` is the similarity between pages `i` and `j` using the vectors obtained from the `page embedding` file.

#	page 1	page 2	page 3	page 4
page 1	1	0.95	0.96	0.98
page 2	0.95	1	0.98	0.93
page 3	0.96	0.98	1	0.91
page 4	0.98	0.93	0.91	1

Further, we can obtain the final contribution matrix `C` where `C[i][j]` contains the contribution `M[i][j]` multiplied by `S[i][j]`.

#	page 1	page 2	page 3	page 4
page 1	0	0.3135	0.3168	0.3234
page 2	0	0	0.49	0.465
page 3	0.96	0	0	0
page 4	0.49	0	0.455	0

Replacing the values of `p` and `q` as as `page 1` and `page 2` respectively in the equation `2`, we obtain the initial contribution of `page 1` to `page 2` as the following:

Initial page rank of `page 1` : 1
Number of outgoing links from `page 1` : 3
Initial contribution = 1/3 = 0.33

Multiplying the initial contribution of `page 1` to `page 2` with the similarity score between the two pages obtained from matrix `S`, we get the complete contribution of `page 1` to `page 2` as the following:

Initial contribution = 1/3 = 0.33
Similarity between `page 1` and `page 2` = 0.95
Complete contribution = 0.3135

The above process can be repeated to generate the values in all the cells of the matrices `M`, `S` and `N`.

Hence, the final page rank of `page 1` is given by equation `1` where,

New page rank of `page 1` after **one iteration** = 0.15 + 0.85 x (contribution of 3 + contribution of 4) = 0.15 + 0.85 x (0.96 + 0.49) = 1.3825

The updated page ranks are calculated for all pages to obtain the following result in the `v1` file.

```
1,1.38
2,0.42
3,1.23
4,0.82
```

The above steps are repeated until convergence