Assignment 2

Sping 2019

Due Date: Monday March 11, 2019

Instructions

- This assignment will involve writing code in the form of Scala classes. The code for the 2 parts should be in different classes in the same Scala project. The project should be compiled into a jar file that can run on AWS cluster. You should include build.sbt or equivalent pom.xml file.
- All instructions for compiling and running your code must be placed in the README file.
- You should use a cover sheet, which can be downloaded from here
- You are allowed to work in pairs i.e. a group of two students is allowed. Please write the names of the group members on the cover page.
- **You have a total of 4 free late days for the entire semester. You can use at most 2 days for any one assignment. After four days have been used up, there will be a penalty of 10% for each late day. The submission for this assignment will be closed 2 days after the due date.**
- Please ask all questions on Piazza, not via email.

1 Search Engine for Movie Plot Summaries

In this part, we will work with a dataset of movie plot summaries that is available from the Carnegie Movie Summary Corpus site. We are interested in building a search engine for the plot summaries that are available in the file "plot_summaries.txt" that is available under the Dataset link of the above page.

You will use the tf-idf technique studied in class to accomplish the above task. For more details on how to compute tf-idf using MapReduce, see the links below:

1. IR using MapReduce
2. Good introduction from Coursera Distributed Programming course
3. Chapter 4 of the reference book Data-Intensive Text Processing using MapReduce.

This assignment has to be done using Scala code that can run on a Spark cluster. You would need to create a project using IntelliJ Scala, and create a class called **MoviePlot** with the requirements below. Your project should be able to create an executable jar file that can be run on a AWS EMR. Details of getting started with EMR were discussed in class.

Below are the requirements of the project:

1. **Program Arguments**: Your class should have two parameters:
   1. Location of the directory that contains files from the unzipped **MovieSummaries** folder that you downloaded earlier from `http://www.cs.cmu.edu/~ark/personas/data/MovieSummaries.tar.gz`. This location could be on Amazon S3 storage.
   2. Search term(s) entered by the user
2. You will need to remove stopwords and then create tf-idf for each term and each document
3. You will create a tf-idf for every term and every document (represented by Wikipedia movie ID) using the MapReduce method.
4. There can be two cases:
   (a) **User enters a single term**: You will output the top 10 documents with the highest tf-idf values for that term.
   (b) **User enters a query consisting of multiple terms**: An example could be "Funny movie with action scenes". In this case, you will need to evaluate *cosine similarity* between the query and all the documents and return top 10 documents having the highest cosine similarity values.
   You can read more about cosine similarity at the following resources:
   - http://text2vec.org/similarity.html : *Read the cosine similarity section*
   - https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/
   - https://courses.cs.washington.edu/courses/cse573/12sp/lectures/17-ir.pdf

   For the search terms entered by the user, you will return the list of **movie names** sorted by their relevance values in descending order. Note again, that you have to return movie **names**, and not movie ID. You would need to use the movie.metadata.tsv file to lookup the movie names.
5. You can display output of your program on the screen

Remember that you have to write your code in the form of a Scala class called **MoviePlot** that should be inside a Scala project that should run on AWS.

2 PageRank for Airports

In class, we studied the PageRank algorithm and how it can be used to rank nodes in a graph in order of their importance. Details about this algorithm and its implementation using MapReduce can be found in Chapter 5 of the reference book Data Intensive Text Processing using MapReduce. You can also look at the slides available at http://lintool.github.io/UMD-courses/bigdata-2015-Spring/slides/session05.pdf.

The dataset for this project will be a graph that shows connections between various airports. This data is available at:Bureau of Transportation website. You would need to do the following to download the data:

1. Go to the url https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236 and get the data for the latest month available.
2. We are only interested in the following fields to create the graph
   - Reporting_Airline
   - Origin (Origin Airport Code)
   - OriginCityName
   - Dest (Destination Airport Code)
   - DestCityName
3. Download and unzip to get a csv file.

Below are the requirements of the project:

1. **Program Arguments**: Your class should have three parameters:
   1. Location of the csv file containing the fields mentioned above
   2. Maximum number of iterations to run
   3. Location of output file
2. You will compute the page rank of each node (airport) based on number of inlinks and outlinks. There may be multiple connections between two airports, and you should consider them independent of each other to compute the number of inlinks and outlinks. For example, if node A is connected to node B with an out-count of 10 and node C with an out-count of 10, then the total number of outlinks for node A would be 20.
3. You have to limit yourself to maximum number of iterations specified by the input parameter number 2.
4. You will use the following equation to compute PageRank:

$$\mathrm{PR}(x) = \alpha \times \frac{1}{N} + (1-\alpha) \times \sum_{1}^{n} \frac{PR(t_i)}{C(t_i)}$$

   where $\alpha = 0.15$ and x is a page with inlinks from $t_1, t_2, \ldots, t_n$, $C(t)$ is the out-degree of t, and N is the total number of links in the graph.

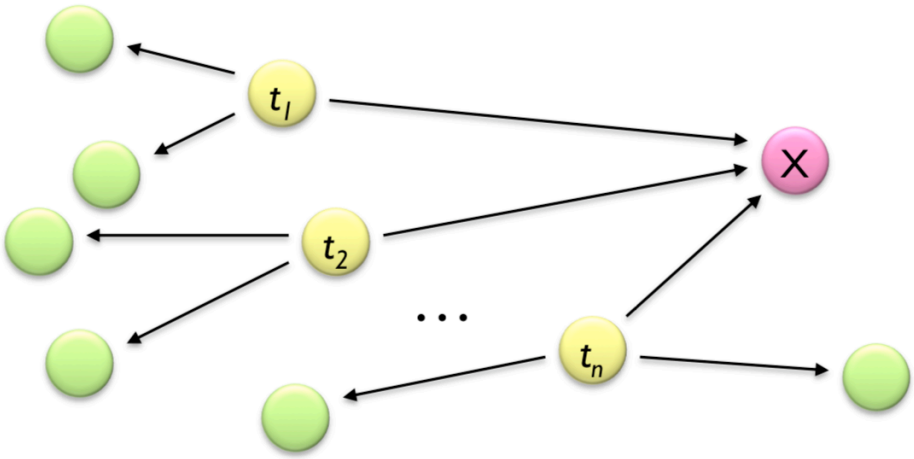

Figure 1: PageRank computation

You can initialize all the PageRank values to be 10.0.

5. You should create a class in the same project as part 1, and name your class as **PageRank**. Your output should be stored in the location specified by the third parameter. The output should contain the airport code and its PageRank, and data should be sorted by the PageRank in a descending order.

Remember to write your code in the form of a Scala class with the above specified number of parameters and include the class in the same project as part 1.