COMS W4121: Compute Systems for Data Science

Assignment #3: Spark

Due March 30th, 2015 at 11:55PM

Overview

In this assignment, you will perform various data processing tasks using Apache Spark. The tasks include computing the **term frequency–inverse document frequency (tf-idf)** of a set of Wikipedia pages and implementing a simplified version of **PageRank** algorithm. The dataset you will be dealing with is the latest English Wikipedia database dump in XML format, and a dataset that describes the inter-page link within the English Wikipedia database.

Spark

Spark Programming Guide: Here

Spark Python API: Here

We also provide two sample Spark programs: **pagerank.py, and sort.py**. You may reference the sample code but use the function only after you have understood it thoroughly.

You are not allowed to use the Spark Machine Learning Library (mllib) or GraphX in this assignment.

Dataset

1. English Wikipedia Database Dump: Please refer to the instruction of assignment #2 for a brief overview of the English Wikipedia dataset. We will not use the chunk_xmls dataset in this assignment.

IMPORTANT: The format of the XML files used in this assignment is slightly different from the dataset used last time. Because there is no concept of "mapper" or "reducer" in Spark, we can no longer use trick of slicing large XML file into smaller ones we used last time. Instead, I have "flattened" the XML files, i.e., **each line in the file contains an entire page**. You may still use the Element tree for parsing the page file. The only difference is that the page text no longer contain the "newline(\n)" character, but it's not important in this assignment as we

are not counting n-grams but only words. You may want to take a look at the sample flattend XML file (sample.xml) in the assignment package.

2. English Wikipedia inter-page link dataset: This dataset describes the inter-page (link from one Wikipedia page to another Wikipedia page) link within the English Wikipedia database. The dataset contains two separate files. One specifies the source and destinations of the links. The pages are referenced by their page IDs. The other file contains the mapping from a page to ID its corresponding page title. **The entries in both files are tab-delimited.**

page_link.txt:

```
source_page_id    destination_page_id
source_page_id    destination_page_id
source_page_id    destination_page_id
...
```

page_id.txt:

```
page_id    page_title
page_id    page_title
page_id    page_title
...
```

1. Compute tf-idf (30 points)

What is Tf-idf:

Tf-idf stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model. Tf-idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification.

How to Compute:

The tf-idf weight is composed by two terms: the first computes the **normalized Term Frequency (TF)**, aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the **Inverse Document Frequency (IDF**), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear many times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

TF-IDF: **TF-IDF(t) = TF(t)*IDF(t)**

See below for a simple example.

Example:

Consider a document containing 100 words wherein the word cat appears 3 times. The term frequency (i.e., tf) for cat is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

Task:

Write a Spark program that computes TF-IDF of a set of Wikipedia pages.

Requirements:

Text Processing:

1. Count words that contains only [a-z] and [A-Z], i.e., English letters, and hyphens, e.g. `good-humored`.
2. All parsing should be case-insensitive. The output should be in lowercase.
3. Strip all punctuations and non-English letter characters that prefix or postfix a word.

Page Processing:

1. Check the number between the `<ns><\ns>` tag. This is the namespace of a Wikipedia page. Discard all pages that has namespace not equal to 0.
2. Discard pages with no text.
3. Count only the words that are enclosed by a `<text> ... <\text>` tag pair.
4. In this assignment, a page is defined as text that are enclosed by a `<page> ... <\page>` tag pair. Please notice that the XML files are formatted such that each line contains a single page.
5. Wikipedia contains fair amount of unicode characters. They are not English letters so apply rule #1.

Implementation details:

Input: Files under `hdfs:/datasets/hw3/p1_dataset/` . Please refer to the "Dataset" section for more information about the format of this dataset.

Output: The program should compute the TF-IDF of each word in a page (a "word" is defined below), and output the word that has the highest TF-IDF of each page and its tf-idf value.

What to submit:

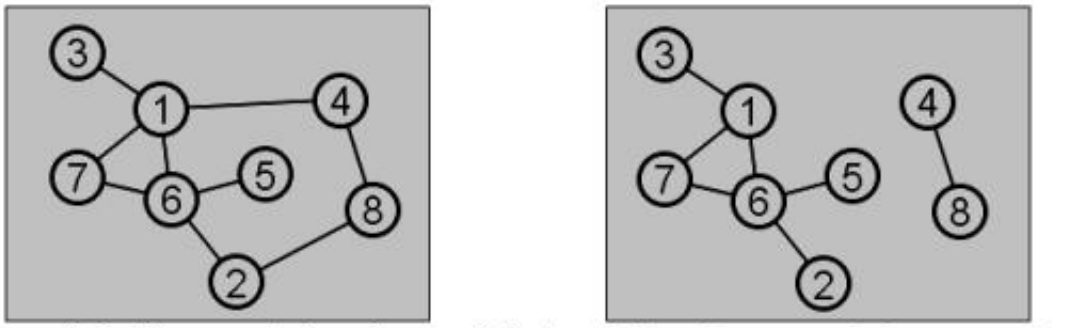
1. A Spark program (**p1_tf_idf.py**) that computes the TF-IDF of a input document/directory. The program should take the first command line argument as the path of the input file/directory.
2. A file (**p1_output.txt**) that contains the output. The output should be in the following format:

```
document_id document_title word tf_idf
document_id document_title word tf_idf
document_id document_title word tf_idf
...
```

---

2. Finding connected component using TF-IDF-based similarity measurement (50 points)

In graph theory, a connected component is a set of vertices in a graph that are linked to each other by edges. Following is an example of a graph that contains one connected component (left) and a graph that contains two connected component (right).



A graph that is connected, and a graph that consists of two connected components

TF-IDF similarity:

In Part 1, your program computes the TF of each word in a document and the IDF of the words across the dataset. We will name the TF-IDF of each word in a page the **TF-IDF feature** of this page. More formally, the TF-IDF feature vector $\mathbf{v}$ of a page p_i is defined as:

$$\mathbf{v}_{p_i} = (t^{(i)}_{w_1}, t^{(i)}_{w_2}, t^{(i)}_{w_3}, ..., t^{(i)}_{w_n})$$

, where t_{w_k} is defined as the tf-idf value of the k^{th} word in a list of words that appeared in the corpus.

For example, if a corpus contains two documents p_0 and p_1, where

p_0 = a, fox, jumps, over, a, dog
p_0 = a, cat, jumps, over, a, dog

, then the word list would be

{a, fox, cat, jumps, over, dog}

, and the TF-IDF feature vector of these two documents are:

$\mathbf{v}_{p_0} = (0, \log(2) * 1/6, 0, 0, 0, 0)$
$\mathbf{v}_{p_1} = (0, 0, \log(2) * 1/6, 0, 0, 0)$

When implementing the algorithm, you don't have to include the TF-IDF value of the words that does not appear in a document since they will be 0 anyways.

We can use this vector to measure the similarity between two documents by computing the Euclidean distance between the vectors of these two documents. The similarity between p_i and p_j is defined as:

$$s(i, j) = (\sum_{k=1}^{n} (t^{(i)}_{w_k} - t^{(j)}_{w_k})^2))^{1/2}$$

In other words, small s(i,j) value indicates similar documents

Task:

In this part of the assignment, you will write a Spark program that builds a graph among a set of Wikipedia pages using the page similarity measurement defined above. A path between two pages exists if the (dis)similarity measurement between these two pages is below certain threshold (s(i,j) < threshold). After you have built a graph, find all connected components of this graph, and then find the shortest document (contains least number of words within the text section) of each connected component.

Implementation details:

Input: Files under `hdfs:/datasets/hw3/p2_dataset/` . Please refer to the "Dataset" section for more information about the format of this dataset.

The similarity threshold you should use to produce the output is **0.3**.

Output: The program should:

1. Compute the TF-IDF of all pages in the dataset
2. Build a graph based on the (dis)similarity measurement defined above
3. Find all connected component in the graph
4. Output the total number of connected components
5. Output out page ID, page title of the shortest page (the page that contains least number of words in its text section) in each of the connected component as well as the number of pages in the connected component.

Requirements:

Same requirements as in part 1.

What to submit:

1. A Spark program (**p2_cc.py**). The program should take the first command line argument as the path of the input file/directory and the second command line argument as the similarity measurement threashold.
2. A file (**p2_output.txt**) that contains the output. The output should be in the following format:

```
total_number_of_connected_components
page_id page_title number_of_pages
page_id page_title number_of_pages
page_id page_title number_of_pages
...
```

3. PageRank on Shortest Document (20 points)

In this part of the assignment, you will implement a simplified PageRank algorithm described below. Then you will use your program from Part 2 to find the shortest page of each connected component in a dataset. Finally you will run the PageRank program on this set of short pages and find the highest ranked pages. You will be provided with the inter-page link dataset so that you don't have to extract the link information from the page dataset.

PageRank

PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page,one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

> PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

PageRank algorithm (simplified)

Suppose a small universe of four web pages: A, B, C and D. If all those pages link to A, then the PR (PageRank) of page A would be the sum of the PR of pages B, C and D.

$$\mathrm{PR(A)} = \mathrm{PR(B)} + \mathrm{PR(C)} + \mathrm{PR(D)}$$

But then suppose page B also has a link to page C, and page D has links to all three pages. One cannot vote twice, and for that reason it is considered that page B has given half a vote to each. In the same logic, only one third of D's vote is counted for A's PageRank.

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}$$

In other words, divide the PR by the total number of links that come from the page.

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}$$

Finally, all of this is reduced by a certain percentage by multiplying it by a factor q. For reasons explained below, no page can have a PageRank of 0. As such, Google performs a mathematical operation and gives everyone a minimum of 1 - q. It means that if you reduced 15% everyone you give them back 0.15.

$$PR(A) = \left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right) q + 1 - q$$

So one page's PageRank is calculated by the PageRank of other pages. Google is always recalculating the PageRanks. If you give all pages a PageRank of any number (except 0) and constantly recalculate everything, all PageRanks will change and tend to stabilize at some point. It is at this point where the PageRank is used by the search engine.

Task:

1. Write a Spark program that implements the simplified PageRank program. You may reference the implementation in **pagerank.py** in the assignment package, but use the function only after you have understood it thoroughly. **Use 0.15 as the "give back" parameter.**

2. Find the shortest page in each component, and run the PageRank on this set of short pages and find the highest ranked pages. The page link dataset is located under `hdfs:/datasets/hw3/p3_dataset/page_link.txt`.

Requirements:

Same requirements as in part 2.

Implementation details:

Input:

1. Page files `hdfs:/datasets/hw3/p3_dataset/page_files/`
2. Page link dataset `hdfs:/datasets/hw3/p3_dataset/page_link.txt`
3. The similarity threshold you should use to produce the output is **0.2**.
4. The number of iteration the PageRank algorithm should run is **10**.

Please refer to the "Dataset" section for more information about the format of these datasets.

Output: 10 Pages that has the highest rank among all short pages.

What to submit:

1. A Spark program (**p3_pagerank.py**). The program should take the first command line argument as the path of the input file/directory and the second command line argument as the number of iterations of PageRank.
2. A file (**p3_output.txt**) that contains the output. The output should be in the following format:

```
page_id page_title rank_value
page_id page_title rank_value
page_id page_title rank_value
...
```

4. Extra Credit

1. Same as in HW2 (25 points)
2. Implement the connected component in part 2 by yourself (5 points)

5. Hints and tips

Text processing with Python

Some useful functions and classes are:

```
strip()
isalpha()
lower()
translate()
string
string.punctuation
```

You may use The Element Tree for parsing XML files.