CS-410 Text Information Systems – Spring 2024

HOMEWORK 2

Handed out: February 28, 2024 Due: March 20, 2024 at 23:59 CT

Submission via Canvas: upload a zip/notebook file with the name HW2-yourNETID-code.[zip—ipynb] *before the deadline (above)* Penalty for wrong formatting & file naming: **5% grade deduction**

PYTHON PROGRAMMING WITH HADOOP

For this homework, you will use AG's News Topic Classification Dataset and will also use a website of your choice for the web crawler. The dataset is available here: https://github.com/mhjabreel/CharCnn_Keras/tree/master/data/ag_news_csv (the original dataset is available at http://groups.di.unipi.it/gulli/AG_corpus_of_news_articles.html). As in Homework 1, you will work with the "description" field and assume a document_id field equal to the row number of the dataset. For this exercise, you can use any library for processing and handling data (Pandas is recommended). For your consideration, the code implementing word count with Hadoop is provided (in Canvas/Assignments). **You will use Google Colab to run the Notebook. Ensure that the files with your code are saved elsewhere. Colab runtime files are deleted when the runtime is terminated. I suggest to code your *.py solutions somewhere else or any *.py files could be lost otherwise. You won't be given extra time to submit your homework if you lose your files.** This homework could be solved with a few lines of code. Thus the homework **is not too programming-intensive but it may be time-consuming; so you must start early because issues may arise**.

1 Hadoop Setup (0 points)

Use the code provided for your reference - must use Google Colab. Make sure it runs error-free. The code does the following. 1. Installs Java and creates the Java home variable (needed to run Hadoop). 2. Downloads, uncompresses, and installs Hadoop. 3. Copies the resulting Hadoop folder to the location of your applications (usually /user/local).

2 Text Data Preprocessing and Word Frequency pre-computation: Parsing, Vocabulary Selection, Frequencies (25%)

Use the template scripts provided (mapper and reducer) and modify them to create mapper and reducer functions to: 1. Read the text of the corpus. 2. Remove stop-words, transform words to lower-case, remove punctuation and numbers, and excess whitespace, and do **stemming** of the words (the sample code already does some of it; make sure you implement the other steps). 3. Code a mapper.py script to count the words in the documents. 4. Code the reducer.py script to combine the information provided by the mapper, i.e., the word counts per document. Remember, MapReduce (Hadoop in this case) uses a pair: (*key*, *value*) to share information from the mapper

to the reducer. The *key* is the word/term in the example provided. You will need to modify the code to ensure the *value* not only contains the word frequencies but also the list of documents where the word appears (see the class slides/discussion for clarification and pseudocode) because you will need to build the scores for each pair query-document.

Once the mapper and reducer are created, you can run them with the single line command (here printed in various lines for readability -also available in the Notebook):

```
/usr/local/hadoop-3.3.0/bin/hadoop jar
/usr/local/hadoop-3.3.0/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar
-input /content/20news-18828/alt.atheism/49960 -output  /testout
-file /content/mapper.py -file /content/reducer.py -mapper 'python mapper.py' -reducer
'python reducer.py'
```

Before running this command, make sure the output directory /testout exists (and delete it every time you need to run the Hadoop command to run the mapper and reducer). Also, make sure both the mapper and reducer files have execution permission (`chmod u+rwx /content/mapper.py` or `chmod u+rwx /content/reducer.py`) after you upload them to the ~~`sample_data`~~ `/content/` directory of the G-Colab runtime files.

Once you run your solution (to compute the word frequencies and the list of documents where they appear) you can complete building your inverted index: Dictionary + Mappings. Ensure the reducer's output (located in `/root/testout/part-00000`) includes all the information you need to build the inverted index. You can construct the inverted index either directly in the reducer or in another script, which you will use to solve the next programming problems of this homework.

3 Document Relevance with Vector Space TF-IDF Model and Hadoop's Reducer (20%)

0. You will use the reducer's output file `/root/testout/part-00000` to create a vocabulary (a list of words) using the top 200 most frequent words in the text. Alternatively, you can modify the reducer to build and output the vocabulary directly. Sort the words in the vocabulary in decreasing order of frequency.

1. Create a script that automates the process of computing word relevances using a vector space representation using the TF-IDF using BM25 and document length normalization.

2. Run your implementation (if the TF-IDF BM-25 function) for the following queries:

- $q =$ "olympic gold athens"
- $q =$ "reuters stocks friday"
- $q =$ "investment market prices"

3. Run your implementation for a few words (3-4) from the vocabulary you constructed.

4 Probabilistic Text Retrieval (30%)

1. Using the same vocabulary you created in the previous section (no need to rerun Hadoop once you have the word frequencies), implement a script that automates the process of creating a probabilistic text retrieval system. It is similar to TF-IDF except that the frequencies are normalized and smoothing is required to avoid 0 probabilities. Use Jelinek-Mercer smoothing - make sure the value of λ you choose do not lead to inconsistent results.

2. Run your implementation (of the probabilistic retrieval with Jelinek-Mercer) for the following queries:

- $q =$ "olympic gold athens"
- $q =$ "reuters stocks friday"
- $q =$ "investment market prices"

3. Run your implementation for a few words (3-4) from the vocabulary you constructed.

5 Simple Web Crawler (25%)

Identify a webpage (e.g. Wikipedia) or set of webpages and create a simple web crawler. You can use python or a simple bash script for this. The web crawler should download around 10 pages from the target webpage(s) - for instance using the command wget. This will be your new corpus. Rerun your solution for problem 2 (including the word frequency count with Hadoop) but use this new corpus to create your vocabulary (instead of the AG's News Topic Classification Dataset). That means the mapper will need to access a set of webpages/files (instead of the rows of the CSV file) to build the inverted index. The sequence of execution of your solution is: Web crawler, Hadoop (mapper and reducer) that builds the new vocabulary based on the new corpus, and query-scorer. For the query scorer (as in Problem 3) you will need to run a set of 3 queries of your own design (from the new vocabulary) instead of the queries provided before.

PART II: EXTRA CREDIT

6 Document Relevance with Vector Space (15%)

In addition to Part I, implement a solution to Problem 3, but this time run 5-fold cross-validation to search for and print the optimal value of the BM-25 parameter k for each fold. Use either a sequential or grid search for k if needed. Cross-validation is a simple partitioning of the data into (in this case) 5 folds where you will use 4 folds as the training set and 1 as test set. Here is an example that can guide you: Scikit-Lern k-fold cross validation

WHAT TO TURN IN

Within the zip file with the answers, submit the following:
1. The updated CS410.HW2.BaseCode.ipynb with any new command or code needed to run your solutions. This may be the case, specially to run the solution with your web crawler.

2. The Python scripts for the mapper and reducer
3. The Python script(s) with the solution to each question
4. The output of the reducer (the file part-00000 located in `/root/testout`) that you used to build the TF-IDF and Probabilistic Retrieval function (inverted index).
5. A README.txt file that indicates how to run your code. If your code does not run you will get a grade over 50% of the maximum. Document the code indicating the answer to each problem/sub-problem. Also, make sure the function that answers each sub-problem prints or plots the adequate output.