CS-410 Text Information Systems – Spring 2025

HOMEWORK 2

Handed out: March 10, 2025 Due: April 2, 2025 at 23:59 CT

Submission via Gradescope: the 8 scripts (.py files) which are indicated below. Submission must be done before the deadline (above). Penalty for wrong formatting & file naming: as indicated in the Syllabus. Late submission policy: as indicated in the Syllabus.

PYTHON PROGRAMMING WITH HADOOP

For this homework, you will use AG's News Topic Classification Dataset (MP 1-4,6) and a website of your choice for the web crawler (MP5). 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 to process and handle data (Pandas is recommended). For your consideration, the code implementing word count with Hadoop is provided (in Canvas/Assignments). **You can use Google Colab to run the reference Notebook `HW2.ipynb` and test your code, but you will have to upload your solution files through Gradescope.** The `HW2.ipynb` file is not part of the solution and should not be uploaded. **Ensure that the files with your code are saved elsewhere. Colab runtime files are deleted when the runtime is terminated. I suggest coding 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.** Some problems of 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. The reference code provided 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). Test this code and make sure it runs error-free. You will use this for answering the next problems. Also, note that although you will submit your code via Gradescope, you will use Google Colab to test your code locally in your web browser before submitting it. Gradescope will only give you feedback on two things for this homework: 1: whether the file submitted has the adequate name, and 2: whether it runs error-free. All the other grading items in these instructions (PDF) will be hidden for the student at submission time. The auto-grader will not provide any grades at this point.

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 (printed in various lines here, 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
-files /content/mapper.py -files /content/reducer.py
-input /content/train.csv
-output /testout
-mapper 'python mapper.py'
-reducer 'python reducer.py'
```

Before running this command, make sure the output directory /testout exists (was created) after running your script (and delete it before you run the Hadoop command to execute 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 `/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.

In summary, the files you have to upload to Gradescope for this problem are: `mapper.py, reducer.py` (both with your changes), and `inverted_index.py`

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 (of the TF-IDF BM-25 function) for the following queries and print the top and bottom 5 document IDs and scores:

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

3. Run your implementation for a query consisting of a few words (3-4) from the vocabulary you constructed and print the top and bottom 5 document IDs and scores.

The file you have to upload to Gradescope for this problem: `3_tfidf.py`

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. This is similar to TF-IDF except that the frequencies are normalized and you need to apply smoothing to avoid probabilities == 0. Use Jelinek-Mercer smoothing - make sure the value of λ you choose does not lead to inconsistent results or numerical issues.

2. Run your implementation (of the probabilistic retrieval with Jelinek-Mercer) for the following queries and print the top and bottom 5 document IDs and scores:

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

3. Run your implementation for a query consisting of a few words (3-4) from the vocabulary you constructed and print the top and bottom 5 document IDs and scores.

The file you have to upload to Gradescope for this problem: `4_jm_smoothing.py`

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. Make sure to crawl the data by parsing URLs within the webpage to scrap content from the linked pages. 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 —> `query-scorer` As in Problem 3, for the query scorer, you will need to run a set of 3 queries (with 2 or 3 words each) of your design (from the new vocabulary) instead of the queries provided before (see reference notebook `HW2.ipynb`).

The files you have to upload to Gradescope for this problem: `5_crawler.py` and `5_tfidf.py`

PART II: EXTRA CREDIT (OPTIONAL)

6 Document Relevance with Vector Space (15%)

This is an optional problem. 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 across folds. Use a sequential search for k if needed. Cross-validation is simply partitioning the data into (in this case) 5 folds where you will use 4 folds as the training set and 1 as a test set. Here is an example that can guide you: Scikit-Learn k-fold cross validation

The file you have to upload to Gradescope for this problem is: `6_cv_k.py`

WHAT TO TURN IN

Within the zip file with the answers, submit the following 8 files:
1. The Python scripts for the mapper, reducer, and inverted index (3 files)
2. The Python script(s) with the solution to each question (5 files, or 4 if you don't submit the optional extra credit)
If your code does not run (as tested by Gradescope) you will get a grade over 50% of the maximum. Document the code indicating the answer to each problem/sub-problem. Also, ensure that the function that answers each sub-problem prints or plots an adequate output.

AUTOGRADER NOTE

This section is also part of the instructions but is not the Machine Problem statements. This section is intended to provide you with additional context about the Autograder on Gradescope. However, the main instructions on what to turn in, print, plot, or compute are in the problem statements above.

Platform: The autograder uses Python 3 over Ubuntu 22.04 with the libraries indicated below.

Libraries: The libraries available are:

```
numpy
nltk
pandas
scikit-learn
bs4
```

If you need a library other than these send a private note to the instructor and TAs through Campuswire so that they can verify the eligibility of the library to add it to the auto-grader. Note that you are supposed to implement both the TF-IDF and Probabilistic Text Retrieval models so you can't use a library that implements them. However, you may need other libraries to access or store data.

Coding: Again, the auto-grader will only verify whether a file was submitted and whether it runs error-free. It is up to you to ensure all the requirements in this document have been coded. All `print` operations from your side (the ones you included for debugging – not the mandatory print operations for each machine problem above) will be available on the grader so you can use that for debugging. However, remove the debugging print statements from the final submission of your work (do not remove the print statements that are part of the MPs). The auto-grader will take about 9 minutes to run your code for HW2. If it takes more, check your code.

Grading/Regrading: Gradescope will indicate immediately if your code runs, that is about 50% of the grade. However, your grade will only be available after the submission period ends and all grading (including manual components) concludes. Regrading can be inquired through private notes on Campuswire.

Autograder software issues Please, report any **software issues** through a private note on Campuswire. If you don't see any output from your submission it is not that the grader broke, but that the code uploaded has some issue. For instance, a library not installed in the Gradescope Grader is imported (see note above), or the name of your script is not formatted as indicated in the instructions at the top of this document.