Exercise 1: Text Processing Fundamentals using MapReduce (100p)

Data-intensive Computing 2024S; v1.1

Submission Deadline: **April 25, 2024 (23:59)**

Goal

In this assignment, we will apply basic MapReduce programming techniques to process large text corpora.

Preliminaries

You need to achieve a minimum of 35 points on every individual exercise.

Environment

You will be using your account on the 12-node Hadoop Cluster `ssh01.lbd.hpc.tuwien.ac.at`. Please use your SSH client of choice to connect to the cluster.

You can implement and compile your MapReduce programs either on the cluster or, more conveniently, use your Python IDE of choice to develop your MapReduce programs locally and transfer the files to the cluster.

You will use the python-based [mrjob] (https://mrjob.readthedocs.io/en/latest/#) implementation as introduced in class.

Act responsibly on the login node! Do not "probe" the data with trial and error using non-MapReduce jobs run locally. It will slow down the cluster for everyone. Keep an eye on the resources your jobs are using and kill them if necessary. **Do NOT wait for your experiments until the last moment** If everyone launches their jobs a day before deadline, the cluster might experience resource starvation and you might not able to complete the assignment.

You are working on shared infrastructure, expect that there will be cluster related issues (downtime, ssh node issues).

Dataset used

In this exercise, we will be using the *Amazon Review Dataset 2014*, which contains 142.8 million reviews from 24 product categories. The new releases can be found here (https://amazon-reviews-2023.github.io/). The data set ha already been prepared for the assignment and is available in a publicly readable **HDFS directory** on the cluster. For development, we provide a smaller set, to be found at:

hdfs:///user/dic24_shared/amazon-reviews/full/reviews_devset.json

A copy for local development can also be found on TUWEL. This file contains a 0.1% sample of a preprocessed version of the full Amazon review data set combined over all categories and extended with category information in JSON format.

Note that evaluation of submissions will be performed on the full dataset (56gb), to highlight the efficiency of implementations. The full data set can be accessed at

hdfs:///user/dic24_shared/amazon-reviews/full/reviewscombined.json

for testing purposes. However, act responsibly with the available resources.

File format

In the file, each line corresponds to a review in JSON format. The following entries are contained in each dictionary:

- **reviewerID** - `string` - the ID of the author of the review
- **asin** - `string` - unique product identifier
- **reviewerName** - `string` - name of the reviewer
- **helpful** - `array of two integers [a,b]` - helpfulness rating of the review: a out of b customers found the review helpful
- **reviewText** - `string` - the content of the review; this is the text to be processed
- **overall** - `float` - rating given to product **asin** by reviewer **reviewerID**
- **summary** - `string` - the title of the review
- **unixReviewTime** - `integer` - timestamp of when review was created in UNIX format
- **reviewTime** - `string` - date when review was created in human readable format
- **category** - `string` - the category that the product belongs to

For reference, the first entry in the file reads as follows:


```
{"reviewerID": "A2VNYWOPJ13AFP", "asin": "0981850006", "reviewerName": "Amazon Customer \"carringt0n\"", "helpful": [6, 7], "reviewText": "This was a gift for my other husband.  He's maki
```


Task

As a preparation step for text classification, we want to select terms that discriminate well between categories. Write MapReduce jobs that calculate chi-square values for the terms in the review dataset.

For preprocessing, make sure to perform the following steps:

- **Tokenization** to unigrams, using whitespaces, tabs, digits, and the characters ()[]{}.!?,;:+=-_"'`~#@&*%€$§\/ as delimiters
- **Case folding**
- **Stopword filtering**: use the stop word list [on TUWEL] (stopwords.txt) . In addition, filter out all tokens consisting of only one character.

Write MapReduce jobs that efficiently

- **Calculate chi-square values** for all unigram terms for each category
- **Order the terms** according to their value per category and preserve the top 75 terms per category
- **Merge the lists** over all categories

Output

- Produce a file `output.txt` from the development set that contains the following:
  - One line for each product category (categories in alphabetic order), that contains the top 75 most discriminative terms for the category according to the chi-square test in descending order, in the following format: `<category name> term_1st:chi^2_value term_2nd:chi^2_value ... term_75th:chi^2_value`
  - One line containing the merged dictionary (all terms space-separated and ordered alphabetically)
- Produce a `report.pdf`, that contains detailed report including at least four sections (1. Introduction, 2. Problem Overview, 3. Methodology and Approach, and 4. Conclusions). The Methodology and Approach section shoul have a figure **illustrating your strategy and pipeline in one figure** (1 page maximum) that shows the data flow clearly and indicate the chosen `<key,value>` design (all input, intermediary, and output pairs). The overall report should not exceed more than 8 pages (A4 size).

Important notes:

- Efficiency of the implementation is a crucial aspect of this assignment. Consider carefully how you design your implementation in order to avoid unnecessary overheads and calculations while achieving the expected final results. As stated above, evaluation will be carried out on the full data set. Solutions that do not produce a result within a given timeframe will be considered to be inefficient. For reference, the best times obtained in previou semesters are under 20 minutes for a Python implementation.
- For all parts, make sure to **document all code** including an explanation for the method signatures, including an explanation of the **`<k,v>` pairs** and all **classes provided (map, sort, combine, partition, reduce etc.)**.
- Make sure that all paths are **relative**. **Your solution will be graded based on the submission to TUWEL, NOT the files in your home directory!** Therefore, do not expect any files or dependencies to be present. **Parameterize the HDFS input path** such that the full dataset can be supplied easily.

Scoring

- Correctness and resource/runtime efficiency
  - Output correctness (selected terms): 25p
  - Code/program correctness: 20p
  - Runtime efficiency: 20p
- Code documentation: 10p
- Schematic overview (clarity, completeness,..): 10p
- Final report: 15p

Maximum total score: 100 points

Submission

Submission Files

Submit a single file named `<groupID>_DIC2024_Ex1.zip` that contains:

- `output.txt`: results obtained
- `report.pdf`: A written report
- `src/`: subdirectory containing all **documented** source code of your MapReduce implementation and a **script to run all jobs in the correct order with all necessary parameters**.

Submission procedure

Submit your solution via TUWEL before **April 25, 2024 (23:59)**