NYU CSGY 6513 Big Data

Assignment 1 - Hadoop HDSF & MapReduce – 120 points

Due Date: *Sunday, October 6th, 11:59PM Eastern*

Abstract:

Show proficiency using HDFS and writing a MapReduce program, including submitting to Hadoop and getting results out of HDFS.

Submission Rules

- Submit all code, pictures, and output files **as a ZIP**, using your **id and hw1:**
  for example, jcr365-hw1.zip.
- If we cannot run your program(s), you will not get full credit.
- Give attribution to any code you use that is not your original code. If this involves any AI generated work, include the prompts and LLMs used.

1. HDFS 20 points

Running any version of Hadoop (Dataproc, HPC, docker or otherwise), submit screen grabs (a picture in jpg, pdf or other suitable format) of the following:

a) create a directory **in HDFS** with this format: **netid-hw1** (e.g. mine will be 'jcr365-hw1').
Submit a screen grab of the output of a *Hadoop file listing* showing your home directory and your new directory in it.

b) Create a subdirectory in HDFS, `netid-hw1/data` and extract all input files into it.
Submit a picture of directory listings or otherwise show the input files in it.

2. Beginner's Language Models with MapReduce

N-Grams and Language Models

Background:

In the simplest form, a language model models the probability of a word or phrase appearing in a sentence of a text collection, or corpus. The building blocks of these language models are n-grams, where *n* refers to the number of words/tokens in the phrase.

A ***unigram*** (1-gram) model gives the probability of a single word/token appearing in the corpus; a bigram (2-gram) model models the probability of two words appearing in ***an exact sequence***.
An n-gram, then, models the probability of a ***sequence of n words appearing consecutively*** in a corpus.
https://en.wikipedia.org/wiki/N-gram

For example, given a corpus defined by the following sentences:

The cat in the hat is a funny book about a cat and a hat
The homework is strange and funny

The corpus, or **vocabulary**, would be the unique individual words:
[the, cat, in, the, hat, is, a, funny, book, about, and, homework, strange]

The **unigram** model (assuming a closed universe vocabulary) will assign a probability to each word appearing in a random sequence. Probability is computed using maximum likelihood estimation (MLE). For example: (using fractions for clarity)
P('the') = P(the) = 3/13
P(cat) = 2/13

For **bigrams**, we parse our corpus in pairs, as in:
`[the cat, cat in, in the, the hat, hat is, is a, a funny, funny book, ….]`
and the probabilities, with MLE, can be computed as:
P(the cat) = X/Y
X = number of times 'the cat' appears in the corpus and
Y = total unique entries (bigrams) in the corpus

Problem 2.1 10 Most likely words, 100 points:

Output the 10 most likely words in the input text.

For this problem, you do not need to compute the unigram (single word) probabilities. Recall from the earlier explanation that the denominator is constant across all words. So, you can just **count** words, then output the top 10. This is a 2 job problem. The output of job1 can feed into the job2.

You ***must*** **NOT** lowercase or normalize the input. You must tokenize punctuation as a single word/token. For example:
"The cat is purring, so do not wake her." should tokenize as:
"The", "cat", "is", "purring", "so", ",", "do", "not", "wake", "her", "."

- The input is lines of text: `TextInputFormat` is your input data class.
- In Hadoop MapReduce, you can pass it individual files, whole directories (which are recursed) or simple wildcard patterns to match multiple files.
- Remember:
  - Your mapper is guaranteed to exist only for a single *input split*.
  - Your reducer is guaranteed to exist only for a single *key.*
- You have full control to define what is a key and what is a value.
- Replace all multiple spaces to a single space before
- Ignore lines with less than 1 word.
- Mappers and Reducers cannot hold global state.
- You must use multiple jobs to solve this problem.
- Reducer parameters are one-way, one-shot iterators. You cannot double loop on the iterator.

Input for this problem: **hw1text.zip** (provided in class website)

Problem 2.2 Extra credit – Simple ID Tokenizer (100 points)

Use the word count given in class, or the output of Problem 2.1, and assign an increasing integer ID to each word (word***) in order of decreasing count***. ID's start at 1.

For example, if the word counts in the input are like this:

The, 123
house, 99
is, 88
cat, 76
…

Your output should be a table like this:

1, The
2, house
3, is
4, house
….

Input for this problem: **hw1text.zip** (provided in class website)