Project 5 – Twitter Analyzer

Due: 11/22/2021 11:59:59 pm

Goal. In this project you will use Hadoop to build a tool for processing sets of Twitter posts (i.e. *tweets*) and determining which people, tweets, hashtags, and pairs of hashtags are most popular. The processing of the data will be divided into two passes.

1. Count the number of times a user, tweet, hashtag or pair of hashtags is mentioned/used across all tweets. This is done in `TweetPopularityMR`.
2. Sort the results from most to least popular. This is done in `TweetSortMR`.

Getting started. You will use Hadoop version 0.20.2 in order to complete this project. To do this, you need to ensure that Unix directory commands (such as `chmod`, which is used to change access permissions to files and directories) are available in your environment. Hadoop uses these commands to move data around throughout execution. Ensure that executables for these commands are on installed on your platform. Directions for Mac, Linux, and Windows are given below.

Installing executables for Unix commands. If your operating environment is a Mac or Linux machine, you do not need to do anything; the relevant Unix commands should all be available natively. If your operating environment is Windows, the easiest way to ensure accessibility of the relevant Unix commands is to install the Cygwin package (available for free at https://www.cygwin.com/). The `bin` folder of the installation (if you install using the defaults this folder is either at `C:\cygwin64\bin` for 64-bit installations or `C:\cygwin\bin` for 32-bit ones) contains the Windows executables for the commands Hadoop needs. Adding this folder to your Windows Path will permit Hadoop to access this executable; steps for doing this in Windows are given below. You will need to restart Eclipse for Hadoop to see this update.

- Open Control Panel → System and Security → System.
- Click the "Change settings" link on the resulting window.
- On the resulting System Properties window, select the "Advanced" tab, then click on Environment Variables.
- In the "System Variables" pane, select "Path", then click "Edit".
- You will see a list of folders, separated by ";". Go to the end and add a ";", then the folder you want to add, then click OK.

NOTE: If your machine has WSL installed, you will have to temporarily disable WSL to allow Hadoop to work with Cygwin. Follow the commands below to disable (and later re-enable) WSL:

- Go to "Control Panel" -> "Programs" -> "Turn Windows features on or off"
- Uncheck "Windows Subsystem for Linux" and click "OK"
- Restart your device. Hadoop should now work with Cygwin64

Downloading the project starter files. You will need to download and install the starter files from ELMS. These files include the Hadoop set-up files you will need, as well as the build-path settings necessary for the project. Also included in is a folder containing sample Twitter data you may use in your testing. ***Disclaimer: The tweets we will be using are real Twitter data created by actual users. The opinions of these users do not reflect those of the university, department, professor, or staff. Some content may also be offensive or upsetting. We have pre-processed the data to eliminate the need for you to read the content of the tweets, so read them at your own risk.***

Setting Up Hadoop. You will need to set up the build path for this project correctly; this entails adding the following JAR files in your project.

- `hadoop-0.20.2-core.jar` from the base Hadoop installation in the p5 project
- From the lib folder in the Hadoop installation: all jars with the prefix `commons-*`, and `log4j-1.2.15.jar`

To add these to the build path in Eclipse use the "Add JARs" option of the "Configure Build Path" option in the "Build Path" menu entry obtained when you right-click on the project name.

Inputs. Your program will take four arguments on the command line.

- Trending Parameter: `user`, `tweet`, `hashtag` or `hashtag_pair`
  - The key to be used for the map-reduce.
  - This decides which entity we will be ranking.
- Cutoff:
  - The minimum score required for an entry to be included in the final output of the program.
- In:
  - This is a path to the input CSV (Comma-Separated Value) file containing the tweets on which to perform the map-reduce.
  - Each line in the file represents a tweet that we conveniently organized for you.
- Out:
  - This is a path to a directory where files will be written at the end of map-reduce.
  - The directory must not exist when the program is run, or else Hadoop will throw an error.

As an example, running Main.java with the arguments

```
user 500 p5_in/testdata_1_mod.xls out
```

should result in ranking the users with at least a score of 500. Scoring is discussed below.

Classes. There are three classes provided in the skeleton. The sections of these classes which require implementing code are marked with `//TODO: IMPLEMENT CODE HERE`. See the Javadocs in each class for more detail.

- `Main.java`:
  This is the entry point for the program. The `TEMP_DIR` variable contains the location of the temporary directory, meant to store the output from `score()` and the input for `sort()`.
  - `main(args)`
    You can change this method as necessary to test your code. We provide a base implementation for you which creates a `Configuration`, reads the parameter options, and then calls `score()` and `sort()` with new `Jobs`.
- `Tweet.java`
  Objects in this class represents a tweet. **Do not modify this file.**
  - `getId()`
    Returns the unique ID of the tweet as a `Long`.
  - `getTimestamp()`
    Returns the time of the tweet in the form of `java.util.Date`.
  - `getUserScreenName()`
    Returns the screen name of the person who tweeted the tweet as a `String`.
  - `getHashtags()`
    Returns a `List<String>` of the hashtags used in the tweet
  - `getMentionedUsers()`
    Returns a `List<String>` of all of the users mentioned in the tweet.
  - `wasRetweetOfUser()`
    Returns `true` if the tweet is a retweet of a ***user***, `false` otherwise.
  - `getRetweetedUser()`
    Returns the screen name of the retweeted user if the tweet is a retweet of a user, `null` otherwise.
  - `wasRetweetOfTweet()`
    Returns `true` if the tweet is a retweet of ***another tweet,*** `false` otherwise.
  - `getRetweetedTweet()`
    Returns the id of the tweet as a `Long` of the retweeted tweet if the tweet is a retweet of another tweet, `null` otherwise.
  - `getTextContent()`
    Returns the content of the tweet as a `String`.
  - `createTweet(csvLine)`
    Takes in a line from a tweet CSV file and returns a `Tweet` object.
- `TrendingParameter.java`
  This interface is used to determine on which parameter to perform the map reduce. The choices are `USER`, `TWEET`, `HASHTAG`, or `HASHTAG_PAIR`. **Do not modify this file.**

- `TweetPopularityMR.java`:
  This class is responsible for taking each line from the input CSV and creating `<key, score>` pairs where the keys are either user screen names, tweet IDs, or hashtags, and the scores are the scores of the keys. Ensure that the output file for this class contains **positive values** for the score.
  - `TweetMapper`
    This subclass extends `Mapper` and is responsible for parsing CSVs and extracting the necessary fields from them. This work has been done for you with `Tweet.createTweet()`. The `TweetMapper` will map the user screen name, tweet id, or hashtags used (depending on `trendingOn`) to the relative weight for its appearance.
  - `PopularityReducer`
    This subclass extends `Reducer` and is responsible for taking the output of `TweetMapper` and calculating the score for each key.
  - `score(job, input, output, trendingOn)`
    This function is responsible for executing the map-reduce process using `TweetMapper` and `PopularityReducer`.
    - This function is partially implemented for you. You must fill in details of the job configuration using commands such as:
      - `setOutput<Key/Value>Class()`
      - `setMapOutput<Key/Value>Class()`
  - Scoring for users, tweets, hashtags, and hashtag pairs is different. The table summarizes how these scores are computed.
    - Note: For `TWEET`, do NOT grant +1 to retweets not in the dataset

TrendingParameter	**Score**
`USER`	(# tweets by user) + 3*(# times retweeted) + (# times mentioned)
`TWEET`	1 + 3*(# times retweeted)
`HASHTAG`	(# times hashtag is used)
`HASHTAG_PAIR`	(# tweets the given pair of hashtags occurs in)

- `TweetSortMR.java`:
  This class is responsible for taking a tab-delimited list of `String/Integer` key-value pairs (like the output from `TweetPopularityMR`) and performing a secondary sort. Entries are sorted by their values, the `Integers`, **from greatest to least**, using map-reduce (ordering of titles in case of ties doesn't matter). Any pair with an `Integer` less than `CUTOFF` should be **omitted**. The output should still be `String/Integer` pairs, with only the sorting changed.
  - `SwapMapper`
    This subclass of Mapper is responsible for reading the input and mapping it such that the data is sorted properly. It works in conjunction with `SwapReducer`.
  - `SwapReducer`

This subclass extends Reducer and is responsible for formatting the output correctly. It works in conjunction with `SwapMapper`.

- `sort(job, input, output, cutoff)`
  This function is responsible for executing the map-reduce process using `SwapMapper` and `SwapReducer`.
  - This function is partially implemented for you. You must fill in details of the job configuration using commands such as:
    - `setOutput<Key/Value>Class()`
    - `setMapOutput<Key/Value>Class()`

Output formatting. The string representation of integers your program should produce is the standard one. User names and hashtags are just strings (note that hashtags should not include a "#" in front). Pairs of hashtags must be turned into strings as follows.

1. The symbol "`(`" should be the first character in the string.
2. The hashtag (without the "`#`") that is second in alphabetical order should be given next.
3. The symbol "`,`" should then appear.
4. The hashtag (again, without the "`#`") that is first in alphabetical order comes next.
5. The symbol "`)`" then concludes the pair.

There should be no embedded spaces anywhere in the string. For example, the string representation for the pair of hashtags `#CMSC433` and `#Awesome` would be: `(CMSC433,Awesome)`

Running the program. You will need to provide the following command-line arguments to the program: `<user/tweet/hashtag/hashtag_pair> <cut off> <input file> <output directory>`.

This is best done in Eclipse by: Selecting "Run" -> "Run Configurations…" -> "Arguments" tab then listing the command line arguments under "Program arguments"

Before running, ***you must ensure that the*** `output` **and** `temp` ***directories do not exist/are deleted***, as Hadoop will automatically generate these. This can be done through your local machine's file system or by right clicking your project under Eclipse's "Package Explorer" -> "Refresh" and then deleting the necessary directories.

Testing. We will test your implementations of `TweetPopularityMR` and `TweetSortMR` both independently and together by validating their output. There will **NOT** be public tests on the submit server in the sense that the correct outputs will not be shared. Sharing of tests for this project is **encouraged**.

Submission. Upload a .zip file containing the `*.java` files inside the `cmsc433/p5` directory to Gradescope. The `*.java` files should be at the base of the .zip file (i.e. the .zip should NOT contain the `cmsc433` or `p5` directory). You may submit an unlimited number of times.

Grading. Project 3 is 27% public tests and 73% secret tests. There are a total of 3 public test and 9 secret tests (some worth 8 points, others worth 9 points) adding up to 100 points total.