Lab 1 – Wikipedia Word Count (Local)

CC5212-1 – March 13, 2024

Today you are going to help me learn Spanish by finding out what are the most common Spanish words (in Wikipedia abstracts). Thanks in advance!

- First we are going to download some data:
  - `http://aidanhogan.com/teaching/data/wiki/es/es-wiki-abstracts.txt.gz`
  - `http://aidanhogan.com/teaching/data/wiki/es/es-wiki-abstracts-1k.txt`

  The first file contains the full dataset, covering all 598,385 abstracts[1] from Spanish Wikipedia (52.5 million words, 120MB GZipped, 330MB uncompressed – no need to decompress this file, we will work with it compressed). The second file contains a sample of the first one thousand abstracts that you can open to see. We will work with the large file: we will be counting the occurrences of each unique word to find out which are the most commonly used words in Spanish (in Wikipedia abstracts at least).
- There is some code to get you started. Some parts are done, some you will have to do. Download `mdp-lab01.zip` from U-Cursos, unzip it and import it as a Java project in Eclipse. To do this, after creating or selecting a Workspace, go to File > Import ... > General > Existing Projects into Workspace > Next > Browse ... > [Select the unzipped folder that's the direct parent of `src/`, etc.] > Select Folder > Finish.
- All files will be encoded in UTF-8. If you notice some encoding issues in your console, you might need to change the encoding of the console in Eclipse. See here:
  - `https://decoding.wordpress.com/2010/03/18/eclipse-how-to-change-the-console-output-encoding/`
- Please create a local blank directory to hold all your data for the lab. We will refer to this directory as PATH in future. At the end of the lab (if you are finished) you can delete this directory to save disk space.
- First we want to count the appearances of each unique word in the big file. The `RunWordCountInMemory` class will simply use an in-memory map to keep track of the number of times each word appears, sort the words by occurrence, and then output the top 100. We will run this over the big file. Will it work? How long will it take to count each word in all abstracts of Spanish Wikipedia? Try it and see; run the following:
  - `RunWordCountInMemory -i` PATH`/es-wiki-abstracts.txt.gz -igz -k 100`

  Specifically, to do this: Expand the project folder on the left > Expand the `src` folder > Right click on `RunWordCountInMemory` > Run As > Java Application. You will see an error in the console indicating a problem about missing arguments, and listing the options for arguments. To add arguments to the class, Right click on `RunWordCountInMemory` (again) > Run As > Run Configurations ... > Click the Arguments tab > In Program Arguments, add `-i` PATH`/es-wiki-abstracts.txt.gz -igz -k 100`. This indicates the input file[2] (the value after `-i`), that the input file is GZipped (`-igz` flag) and that we want the top 100 words (the value of `-k`). Now when you hit Run you should see progress indicated in terms of the number of lines read. It will also give an estimation of memory used at each step.

  Did it run? How was the runtime?

[1] An abstract is the first paragraph of text.

[2] Note that if you use a relative path for PATH, it is relative to the base directory of the Java project; if in doubt, use an absolute path starting with `/` in Unix/IoS, or `C:\` in Windows (if the path has spaces, make sure to wrap it with quotes: `""`).

- Not challenging enough it seems? Let's try something harder: now we will count n-grams. For example a 2-gram (aka. bigram) is a sequence of two consecutive words like "`por ejemplo`", a 3-gram is a sequence of three consecutive words like "`que raro es`". Again, there is code provided for this. You can try to run the following commands following the same steps for `RunWordCountInMemory`. Note that the class is different, and there's an additional argument, `-n`, whose value indicates the length of the n-gram.

  - `RunNGramCountInMemory -i` PATH`/es-wiki-abstracts.txt.gz -igz -k 100 -n 2`
  - `RunNGramCountInMemory -i` PATH`/es-wiki-abstracts.txt.gz -igz -k 100 -n 3`
  - `RunNGramCountInMemory -i` PATH`/es-wiki-abstracts.txt.gz -igz -k 100 -n 4`

  Each time you run it successfully, keep a note of the number of non-unique n-grams read, the unique n-grams read, the memory used, and the time taken; the first three data points are found just before the output in black (standard out stream), while the time should be in red at the very end (standard error stream, used for logging).

  Now please keep increasing n until your machine gives up ...

  If you are starting to run out of memory during execution, the program will slow and eventually stop (the garbage collector gets busier and busier trying to keep the program alive). Note you can add more memory by adding, e.g., `-Xmx4096M` to the VM arguments (this gives 4096 MB of heap memory to use in RAM; you can also try more if your machine supports it). To do this, go to where you add program arguments, but rather than adding arguments in the Program arguments window, add `-Xmx4096M` to the VM arguments (VM = Virtual Machine) window below it. Adding more memory should help you run wider n-grams until your system has no more memory to run. So add as much memory as you can and try to find the smallest n where you cannot do it in the memory available to the system.

  Now what to do? Maybe we can use the hard disk. But how?

- Let's call the smallest value of n for which the above failed "ν" (we will often use Greek letters in the lab instructions to indicate something you have to replace). First we can run the following to write the ν-grams out to a file, one ν-gram per line (all one command running the `ExtractNGrams` class):

  - `ExtractNGrams -i` PATH`/es-wiki-abstracts.txt.gz -igz -n` ν
    `-o` PATH`/es-wiki-`ν`grams.txt.gz -ogz`

  For example, for 3-grams, this will produce a file like:

```
...
que raro es
raro es este
es este ejemplo
este ejemplo de
...
```

  Now we just need to find how many times each unique ν-gram (in other words, each unique line) occurs in that file. How can we do this?

- We will sort the data, which will bunch the same lines together. Since it does not all fit in memory, we will have to sort it in memory in small *batches*, write each sorted batch to disk, and then merge all the batches. This is called an external (merge) sort. It is called external since it uses external memory (in this case, disk). It is called a merge sort since in the final stage, we will merge all the batch files. Some of the code is already provided for you, but you will have to code one part.

  - Open `ExternalMergeSort.java`
  - The `main` method is done for you. It reads input from the command line and passes them to ...
  - ... the `externalMergeSort` method, which is also done for you. It opens the input file, an output file and a temporary folder and then calls the methods needed to do the external merge sort. It relies on two methods: `writeSortedBatches` and `mergeSortedBatches`.

* `writeSortedBatches` is already done for you, but have a look: it reads the input file (that needs to be sorted), reads `batchSize` number of lines into memory, sorts them, and writes these sorted lines to a temporary batch file, empties the memory buffer and continues reading from the input.
* You need to code `mergeSortedBatches` method. This takes the list of temporary files created by the `writeSortedBatches` code and will merge-sort them into the output. (We will offer hints on this.)

Once you have completed `ExternalMergeSort.java`, to sort the file output previously, run:

– `ExternalMergeSort -i` PATH`/es-wiki-`ν`grams.txt.gz -igz -o` PATH`/es-wiki-`ν`grams-s.txt.gz -ogz -tmp` PATH`/tmp -b` β

β indicates the size of the batches: put it too high and you run out of memory; put it too low and you'll have so many batches your hard drive may explode (or just take a long time to merge the files). Try find a good value: bigger is better, but you should be sure to not run out of memory (remember to give more memory using `-Xmx`, and avoiding creating more than 1,000 batch files). This will produce a sorted file like:

```
...
que rara es
que rara la
que rara la
que rara la
que raro es
que raro es
...
```

- Now you need to count repeated lines/n-grams in the output of the previous stage. Since the file is sorted this is easy: the repeated lines will be consecutive, so we just need to scan the file and count repeated lines that appear together. Again the code is ready for you:

  – `CountDuplicates -i` PATH`/es-wiki-`ν`grams-s.txt.gz -igz -o` PATH`/es-wiki-`ν`grams-c.txt.gz -ogz`

  This will output a file like:

```
...
0000000001 que rara es
0000000003 que rara la
0000000002 que raro es
...
```

  The lines are ordered alphabetically based on the first sort. But we would like to find the most frequent lines (those with the highest count). Note the leading zeroes on the count: we add these so we can re-use the first sorting method to do a numeric sort. If we did not have these, then "200" could appear before "30" in the ordering but after "10". So the zeroes allow us to sort numbers as strings.

  So we can now sort the output of the last phase using the sort, but this time we also need to add a flag to sort in descending order: we'd like the highest counts to appear first in the file. We thus need to add an '`-r`' flag; the code will do the rest. We can use the same value for β as before.

  – `ExternalMergeSort -i` PATH`/es-wiki-`ν`grams-c.txt.gz -igz -o [path]/es-wiki-`ν`grams-c-s.txt.gz -ogz -tmp` PATH`/tmp/ -b` β `-r -k 100`

  Now we have the top ν-grams in the data! Note that the `-k 100` argument will print the first 100 sorted lines for you to the console after the sort finishes.

- Some questions to consider: Which of the methods above was faster and which was slower? Which of the methods above was more scalable? What were the benefits of using the hard disk vs. using main memory? Why did we use a map (aka. dictionary) in memory but sorting on disk? Which affected the runtimes more, the size of the data or the length of the n-grams? Would it be faster to count n-grams over $2a$ abstracts, or $2n$-grams in a abstracts?
- You might want to clean up your hard-drive when you're finished: delete the temporary folder, intermediate data, etc., in `PATH`.
- **Submit `ExternalMergeSort.java` to u-cursos (just that file) before the deadline indicated. Put the top 5 ν-grams (for the largest value of ν you ran using the disk) in comments at the very start of the source file. Please include the names of all members of your group in the comment on the homework in u-cursos.**