Homework 11: MapReduce in Python using `mrjob`

Due April 23, 11:59 pm

Worth 20 points

Instructions on writing and submitting your homework can be found on the course webpage at `http://pages.stat.wisc.edu/~kdlevin/teaching/Spring2021/STAT679/hw_instructions.html`. *Failure to follow these instructions will result in lost points.* Please direct any questions the instructor.

Note: this homework is longer than most of the previous ones in this course, in part because it tries to walk you through the process of getting started with a new tool, Google Cloud Platform. This aspect of the assignment is meant to challenge you to learn a new tool from scratch by reading documentation. This is tough, by design. **Start early** so that you have plenty of time to grapple with the material and ask for help if you need it.

Another note: You will spend a lot of this assignment running jobs remotely on a compute cluster rather than on your own laptop, so the set of things to turn in is slightly more complicated than previous assignments. For your convenience, the last page of this handout summarizes what you should turn in with your final submission.

1 Warmup: counting words with `mrjob` (3 points)

In this problem, you'll get a gentle introduction to `mrjob` on your machine locally. Then, we'll run the same job on Google Cloud Platform on a larger data file.

1. Write an `mrjob` program that takes text as input and counts how many times each word occurs in the text. Your script should strip punctuation like full stops, commas and semicolons, but you may treat hyphens, apostrophes, etc. as you wish. Simplest is to treat, e.g., "John's" as two words, "John" and "s", but feel free to do more complicated processing if you wish. Your script should ignore case, so that "Cat" and "cat" are considered the same word. Your output should be a collection of (word,count) pairs. Please save your script in a file called `mr_word_count.py` and include it in your submission.

2. To test your code, I have uploaded a simple text file to the course webpage:

   `http://pages.stat.wisc.edu/~kdlevin/teaching/Spring2021/STAT679/simple.txt` .

   Download this file and test your code on your local machine. The file is small enough that you should be able to check by hand whether your code is behaving correctly. Please include a copy of `simple.txt` in your submission. Save the output of running your script on this small file to a file called `simple_word_counts.txt` and include it in your submission.

2 Getting Set Up on Google Cloud Platform (3 points)

To run our MapReduce jobs on truly large data sets, we need to run in a distributed environment. That is, we need to run our job on multiple machines in parallel, instead of running only on our local machines. To do this, we will use Google Cloud Platform (GCP), Google's distributed computing service. Google has provided us with a grant, which will provide everyone in the course with free compute time on GCP.

1. The first thing you should do is claim your share of the grant money. Unfortunately, the automatic system that Google typically uses for distributing coupons went down the week that I was preparing this assignment, so we'll have to do this the old fashioned way. Watch your Canvas inbox for a message from me containing a coupon code. If you do not receive such a code by Thursday, April 8th, please contact me. You can claim your GCP credits at the following link:

   `https://console.cloud.google.com/education` .

   Please contact me **promptly** if you encounter any issues claiming your credits.

   These credits are valid on GCP until they expire on January 25, 2022. Any credits left over after completing this homework are yours to use as you wish. **Note:** Make sure that you claim your credits while signed in under your University of Wisconsin email, rather than a personal gmail account so that your project is correctly associated with your Wisconsin email. If you accidentally claim the credits under a different address, add your Wisconsin NetID email address as an owner (see below for further discussion of project ownership and roles on GCP).

2. Once you have claimed your credits, you should create a project, which will serve as a repository for your work on this problem. You should name your project `NetID-stat679s21`, where `NetID` is your UW NetID in all *lower-case letters*. Your project's billing should be automatically linked to your credits, but you can verify this fact in the billing dashboard in the GCP browser console. Please add the instructor (NetID `kdlevin`) as well as your grader Hongzhi Liu (NetID `hliu438`) as owners. You can do this in the IAM tab of the IAM & admin dashboard by clicking "Add" near the top of the page, listing our Wisconsin emails and specifying our Roles as Project → Owner.

3 Running a simple `mrjob` program on GCP (8 points)

To check that you have everything set up properly, let's try running your `mrjob` word counting program from Problem 1 on GCP.

1. The simplest way to transfer files to and from GCP (and store them on GCP) is using a *bucket*. This is the Google Cloud term for what is essentially a hard drive. Instructions on setting up a storage bucket can be found here:

   `https://cloud.google.com/storage/docs/creating-buckets`

   Create a bucket named `NetID-stat679-hw11`, where `NetID` is your Wisconsin NetID in all lower-case letters. Make sure you set "Access control" to *Uniform*. You may set the remaining options (location, location type, storage class) as you see fit.

The default values are all fine. Ignore the "Advanced settings", and click the blue "Create button". Once you have done all that, if you click the "Browser" button in the "Storage" menu on the left-hand sidebar, you should see your newly-created storage bucket listed.

2. Now it's time to put some files in your new bucket. In the storage browser, click on the bucket you created in the previous part of this problem and click "Upload files". Upload your `mrjob` program `mr_word_count.py` and the file `simple.txt` from the previous problem to your storage bucket. This is roughly analogous to moving a file from your local machine to a hard drive.

   Note: if you want a *completely optional* challenge, you can try using `wget` and the `gsutil` program in the Google Cloud shell to do this step, instead. In the blue bar at the top of your browser window in GCP, you should see an icon that looks like a terminal prompt.[1] If you click that icon, a terminal will open at the bottom of your browser window. This is the Google Cloud shell, which supports basic UNIX/Linux command line programs like `wget`. You can use `wget` to download `simple.txt` from my webpage, and then use the `gsutil` program to copy the file to your storage bucket. Type `gsutil help` in the Google Cloud shell to get started. If you're feeling even more adventurous, you can try installing the `gsutil` tool on your own local machine to copy files to your storage bucket,[2] though I would not recommend this.

3. Now let's try running `mr_word_count.py` on `simple.txt`, this time on GCP. Open the Cloud shell (click the icon that looks like a terminal prompt in the blue bar near the top of your browser window) and type `gsutil ls`. If your storage bucket is set up correctly, you should see its name displayed in the output as `gs://NetID-stat679-hw11`, where `NetID` is your NetID. Copy `mr_word_count.py` from your storage bucket to your local shell environment by typing

   ```
   gsutil cp gs://NetID-stat679-hw11/mr_word_count.py .
   ```

   at the command prompt. If all went well, you should see some output about copying your file and a message saying "Operation completed over 1 objects...". Use a similar command to copy `simple.txt` from your storage bucket to the shell session. Type `ls` and you should see both `mr_word_count.py` and `simple.txt` as being among your *local* files. Now, you should be able to run your `mrjob` program just like you did in Problem 1 by typing

   ```
   python2 mr_word_count.py simple.txt
   ```

   in the Cloud shell.[3] Use a redirect to store the output of this in a file called `shell.simple.out`, and use `gsutil cp` to copy this output file back to your storage bucket. Once you have done this, you should be able to see `shell.simple.out` as listed among the files in your bucket.

[1] Or see here: `https://cloud.google.com/shell`

[2] `https://cloud.google.com/storage/docs/gsutil`

[3] Note that we are using Python 2, not Python 3. In my experience, `mrjob` doesn't play especially nicely with Python3 on GCP, especially once we try running on a Hadoop cluster, so it's easiest for us to just avoid it. You shouldn't need to do anything special to get your `mrjob` program to run in Python 2, even though you probably wrote it to run in Python 3. The one possible thing that will need fixing is that `reduce` was moved into the `functools` module in Python 3. The first time you try to run `mrjob`, you may get an error that no such module exists. Install it in the `gcloud` console with `pip2 install mrjob`.

4. Okay, now things can start getting interesting. The whole point of using a service like GCP is that it gives us convenient access to cluster computing resources. So let's use GCP to run our `mrjob` program on a proper Hadoop cluster. On GCP, Dataproc[4] provides Apache Hadoop and related services. The `mrjob` documentation includes some instructions for running your `mrjob` program on Dataproc:

   `https://mrjob.readthedocs.io/en/latest/guides/dataproc-quickstart.html#running-a-dataproc-job` .

   You can try to run the example given there, adapted to our setting, by typing

   ```
   python2 mr_word_count.py -r dataproc README-cloudshell.txt.
   ```

   This may result in an error message (from Python, not from `gsutil`) that you must install `google-cloud-logging`, `google-cloud-storage` or both in order to connect to Dataproc. If you get a message along these lines, try running

   ```
   pip2 install mrjob google-cloud-dataproc google-cloud-logging google-cloud-storage
   ```

   to make sure that the Python installation running in your GCP shell has the requisite packages for interacting with the rest of GCP. Another common error is a Python error with the message

   ```
   TypeError:  __init__() got an unexpected keyword argument 'channel'.
   ```

   This is caused by the fact that the `detok-cloud-dataproc` library in Python has changed in its most recent version, and `mrjob` has not been updated accordingly.[5] You can solve this by calling

   ```
   pip2 install --force-reinstall --no-deps google-cloud-dataproc==1.1.1
   ```

   in the Google Cloud shell to revert the Python `google-cloud-dataproc` package to the earlier version that `mrjob` is expecting. If you are still getting errors after trying these two corrections, make a post in the discussion board, because it is almost certainly the case that your classmates are encountering a similar error. **Note:** Part of my goal in having you do all this is for you to see the kinds of annoying and/or weird errors that are typical when we starting trying to use multiple interacting tools, each of which is frequently being updated, often with API-breaking changes. If you don't like the idea of spending hours chasing down these kinds of weird errors, I suggest you find another line of work!

   When you successfully run an `mrjob` program, you'll see output along the lines of

   ```
   No configs found; falling back on auto-configuration
   No configs specified for dataproc runner
   using existing temp bucket mrjob-[REGION]-[alphanumerics]
   ```

   Followed by some logging information about creating temporary directories and "waiting for cluster to accept jobs". You will likely need to wait five or ten minutes for your job to finish running– a lot longer than running on your local machine. This is because GCP needs time to set up machines to serve as the nodes in your

[4] `https://cloud.google.com/dataproc`

[5] See, e.g., `https://stackoverflow.com/questions/64628864/mrjob-fails-to-create-cluster-on-dataproc-init-got-an-unexpected-keyword`

cluster before it can actually run your job. Essentially, GCP is recruiting a collection of machines for you to use temporarily as your compute cluster and installing the necessary software on those machines for them to be nodes in a Hadoop cluster, and only then is it running your job on that Hadoop cluster. This is an example of what are sometimes called *startup costs* in distributed computing. If we have a large job to run, having a lot of computers on which to run it in parallel should make things faster, but there is an up-front cost to pay in the time it takes to "spin up" all those computers,[6] as well as *teardown costs* once our job is finished running and GCP has to terminate all those worker nodes and free up the computing resources for other users.

Once your job finishes running, you should see the output of your program, followed by a bit more logging information. If everything looks correct, run your Hadoop word count script `mr_word_count.py` on `simple.txt` again, this time using the Dataproc cluster instead of running locally. Use a redirect to send the `stdout` output (i.e., the word counts) to a file called `hadoop.simple.out`, and send `stderr` (i.e., the logging information) to a file called `hadoop.simple.log`.[7] Use `gsutil` to copy these files to your storage bucket. You will notice that GCP created a few temporary storage buckets when it created your Hadoop cluster. These contain some logging information about creating and running your Hadoop cluster. You are welcome to delete these once your job is finished. Just be careful not to delete your `NetID-stat679-hw11` bucket.

5. I have made a larger file available publicly (okay, actually, it's still not really large enough to warrant using distributed computing, but large enough to be good practice). In the Cloud shell, try typing

   ```
   gsutil ls gs://uw-stat679s21-hw11/darwin.txt
   ```

   The output should include a file called `darwin.txt`. This is a slight modification of the Project Gutenberg plain text version of Charles Darwin's famous *On the Origin of Species*. To run your script on a file stored on a storage bucket, you can call

   ```
   python2 mr_summary_stats.py -r dataproc gs://uw-stat679s21-hw11/darwin.txt
   ```

   Run your script on this file, save its output in a file called `darwin_word_counts.txt`, and copy this file to your storage bucket. You should also download a copy of your output and include it in your submission.

6. Zipf's law[8] states that if one plots word frequency against frequency rank (i.e., most frequent word, second most frequent word, etc.), the resulting line is (approximately) linear on a log-log scale. Using the information in `darwin_word_counts.txt`, make a plot of word frequency as a function of word rank on a log-log scale for all words in the file `darwin.txt`. Give an appropriate title to your plot and include axis labels. Save the plot as a pdf file called `zipf.pdf`, and include it in your submission.

7. How "Zipfian" does the resulting plot look (It suffices for you to state whether or not your plot looks approximately like a line)?

[6] See also `https://en.wikipedia.org/wiki/Amdahl%27s_law`

[7] We can redirect both `stdout` and `stderr` by writing `command.sh > outfile.txt 2> errfile.txt`

[8] `https://en.wikipedia.org/wiki/Zipf's_law` Zipf's law is an example of what is called a *power law* (`https://en.wikipedia.org/wiki/Power_law`). For more about power laws, I recommend this survey paper by Mark Newman `https://arxiv.org/pdf/cond-mat/0412004.pdf`.

4 Computing Sample Statistics with `mrjob` (6 points)

In this problem, we'll compile some basic statistics summarizing a toy dataset. The file

`http://pages.stat.wisc.edu/~kdlevin/teaching/Spring2021/STAT679/populations_small.txt`

contains a collection of (class,value) pairs, one per line, with each line taking the form `class_label value`, where `class_label` is a nonnegative integer and `value` is a float. Each pair corresponds to an observation, with the class labels corresponding to different populations, and the values corresponding to some measured quantity.

1. Write an `mrjob` program called `mr_summary_stats.py` that takes as input a sequence of (label,value) pairs like in the file `populations_small.txt`, and outputs a collection of 4-tuples of the form `(label, number of samples, mean, variance)`, in which one 4-tuple appears for each class label in the data, and the mean and variance are the *sample* mean and variance, respectively, of all the values for that class label. Thus, if 25 unique class labels are present in the input then your program should output 25 lines, one for each class label. Please include a copy of `mr_summary_stats.py` in your submission. **Note:** I don't care whether you use n or $n-1$ in the denominator of your sample variance formula—just be clear which one you are using. **Note:** you don't need to do any special formatting of the output. That is, your output is fine if it consists of lines of the form `label [number,mean,variance]` or similar.

   Think carefully about what your key-value pairs should be here, as well as what your mappers, reducers, etc. should be. Should there be more than one step in your job? Sit down with pen and paper first! **Hint:** to compute the sample mean and sample variance of a collection of numbers, it suffices to know their sum, the sum of their squares, and the size of the collection.

2. Run your `mrjob` script on the `populations_small.txt` file and write the output to a file called `summary_small.txt`. Include this output file as well as a copy of `populations_small.txt` in your submission. Inspect your program's output and verify that it is behaving as expected, noting that the small file is small enough that you should be able to check by hand what the results should be.

3. I have uploaded a much larger data file to GCP, available at

   `gs://uw-stat679s21-hw11/populations_large.txt` .

   Once you are *sure* that your script is doing what you want on the small file, run it on this larger file. Don't forget to use the `-r dataproc` flag to tell `mrjob` to run on a Hadoop server rather than in the Cloud shell. Save the output in a file called `summary_large.txt`. Copy this file to your storage bucket, and download a copy to include in your submission.

4. Use `matplotlib` and the results in `summary_large.txt` to create a plot displaying 95% confidence intervals for the sample means of the populations given by the class labels in `populations_large.txt`. You will probably want to make a boxplot for this, but feel free to get creative if you think you have a better way to display the information. Make sure your plot has a sensible title and axis labels, save it as a pdf called `populations.pdf`, and include it in your submission.

What to turn in

Great job completing a very challenging assignment! There were a lot of moving parts in this homework, so here is a checklist to make sure you submit everything necessary.

- **Jupyter notebook file.** You should turn in a Jupyter notebook file, as usual, that includes your collaboration statements and summary of total time required for each problem.
- **Problem 1.** Your submission should include a copy of your `mr_word_count.py` as well as a copy of `simple_word_counts.txt`, which should contain the output of running your script on the file `simple.txt`. For convenience, please also include a copy of `simple.txt`.
- **Problem 2.** There is nothing to include in your submission for this problem. You should, however, take care that your GCP project is named per the instructions (`NetID-stat679s21`, where `NetID` is your UW NetID in all lower-case letters). Further, make sure that both the instructor (NetID `kdlevin`) and the grader Hongzhi Liu (NetID `hliu438`) are owners of the project.
- **Problem 3.** Make sure that you have created a storage bucket according to the instructions, with name `NetID-stat679-hw11`, `NetID` is your Wisconsin NetID in all lower-case letters. This storage bucket should contain the following files:
  - copies of your `mrjob` program `mr_word_count.py` and the file `simple.txt` from Problem 2.
  - a copy of `shell.simple.out`, which holds the result of running `mr_word_count.py` on `simple.txt`.
  - copies of `hadoop.simple.out` and `hadoop.simple.log`. These should contain the output and logging information, respectively, from running `mr_word_count.py` on the file `simple.txt`, this time on a Dataproc Hadoop server.
  - a file called `darwin_word_counts.txt`, containing the output from running `mr_word_count.py` on the file `darwin.txt`.

  Your submission should contain a copy of `darwin_word_counts.txt` and your plot `zipf.pdf`. Your Jupyter notebook should include code for generating `zipf.pdf` and a brief remark on the "Zipfianness" of the data.
- **Problem 4.** Your submission should include
  - a copy of the script `mr_summary_stats.py`
  - a copy of the test file `populations_small.txt`
  - `summary_small.txt`, containing the output of your script run on `populations_small.txt`
  - a copy of `summary_large.txt`

  Please make sure that a copy of `summary_large.txt` is also stored in your storage bucket created in Problem 3. Finally, your submission should include a copy of your plot, `populations.pdf`, and your Jupyter notebook should include code for generating this plot.