Homework 11: PySpark

Due May 4, 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/Spring2022/STAT679/hw_instructions.html`. *Failure to follow these instructions will result in lost points.* Please direct any questions to the instructor.

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 Preliminaries: Set up a Storage Bucket (2 points)

Before we get started, create a storage bucket for this project, just like you did for Homework 10, this time called `NetID-stat679-hw11`, where `NetID` is your Wisconsin NetID in all lower-case letters. All settings should be the same as the bucket you created for Homework 10.

2 Warmup: Interactive PySpark on GCP (3 points)

Before we can do anything in PySpark, we have to get a server up and running.

1. Sign in to Google Cloud Platform, and make sure that you are in your project that you created for Homework 10. Recall that this project should be named `NetID-stat679s22`, where `NetID` is your Wisconsin NetID in all lower-case letters. Open Google Cloud Console and type

   ```
   gcloud dataproc clusters create CLUSTERNAME --region=REGION
   ```

   where `CLUSTERNAME` is the name you wish to give your cluster (e.g., `stat679hw11` or something like that; you are free to name this however you like) and `REGION` is a valid region.[1] You will need to wait a few minutes while Google Cloud sets up your cluster (i.e., gets some computers to serve as your nodes, installs necessary software on those computers, etc.). Once this process finishes, you will see a message to the effect of `Created [CLUSTERNAME] Cluster placed in zone [REGION]`. Once you have created this cluster, you should see it listed when you call

[1] See `https://cloud.google.com/compute/docs/regions-zones` or type `gcloud app regions list` in the console.

```
gcloud dataproc clusters list --region=REGION
```

in the console, where `REGION` is the same as the argument supplied when you created the cluster.

Important warning: any time you finish a working session (e.g., to take a break and come back again later), consider deleting your cluster with

```
gcloud dataproc clusters delete CLUSTERNAME --region=REGION
```

to ensure that you are not paying to leave a cluster sitting unused. Of course, when you come back to continue working, you will have to spin up the cluster again by following the instructions above. Bear in mind that any files that you create on the cluster are lost when you delete it, so be sure to move any files you want to keep into a storage bucket (we discuss this point at more length below).

2. Okay, now that we have a cluster up and running, let's try running an interactive PySpark session. To do that, we need to log onto our cluster. We will `ssh` to the master node on your Dataproc cluster. Double-check that your Dataproc cluster is up and running by calling

```
gcloud dataproc clusters list --region=REGION
```

again (`REGION` should be set to whatever region you requested when you created the cluster). If a cluster shows up in the list, go to the VM Instances dashboard,[2] where you should see a few entries listed. These correspond to the nodes in your cluster. The names of these instances should all be prefixed with your cluster name. One of them should end with `-m`. This is the master node in your cluster. To `ssh` to it, type the command

```
gcloud compute ssh MASTERNODE --project=PROJECT --zone=ZONE
```

in the console, where `MASTERNODE` is the name of your cluster with the added suffix `-m` (something like `CLUSTERNAME-m`), `PROJECT` is the name of your project (something like `NetID-stat679s22`), and `ZONE` is the specific zone that your cluster is in. This will have a form like `REGION` or `REGION-X`, where `REGION` is your specific region specified when you launched the cluster, and `X` is a letter or number. If you're not sure, you can find the zone of your cluster in the "Zone" column of the VM instances dashboard.

If all goes well, it won't look like much has changed, except you'll see that your prompt in the console has changed to something like `NetID@CLUSTERNAME-m`. Alternatively, you can type the command `hostname` in the console, which should produce an output of the form `CLUSTERNAME-m`.

Now you can start an interactive PySpark session by typing `pyspark` in the console. When you do this, you'll see some text appear, giving some setup information and information about the version of Spark, and then you'll see the interactive prompt (`>>>`). The `numbers.txt` file from lecture is available at

[2] `Compute Engine` → `VM instances` in the sidebar; see `https://cloud.google.com/compute/docs/instances` for more information

`gs://uw-stat679s22-hw11/numbers.txt`

Read it into an RDD in your PySpark interactive session and use a sequence of RDD transformations and RDD actions to compute how many of the numbers in the file are prime. You may make use of the function `is_prime`, which is defined in the Python file

`gs://uw-stat679s22-hw11/prime.py`

Save the answer in a variable called `number_of_primes` in your Jupyter notebook file for submission. **Note:** you can quit an interactive PySpark session either by typing `quit()` at the prompt or by typing `ctrl-D`.[3]

Please also copy-paste into your Jupyter notebook file the sequence of PySpark commands that you ran to obtain this answer. **Important:** paste these into a `Raw NBConvert` or `Markdown` cell, **not** a `Code` cell. If you paste these commands into a Jupyter `Code` cell, the grader script will try to run your PySpark commands in plain old Python, which will cause errors.

Reminder: if you aren't going to continue working on the next problem immediately, save credits by deleting your cluster.

3 Submitting a Job to Spark (6 points)

Now let's try writing a PySpark script and submitting it to your Dataproc server.

1. First things first: make sure that you have a Dataproc cluster up and running by typing

   `gcloud dataproc clusters list --region=REGION`

   where `REGION` is the region you specified upon cluster creation. Alternatively, you can pull up the VM instances dashboard to see a list of your currently-running VM instances (this list will include any running Dataproc clusters). If you don't have a Dataproc cluster up and running, follow the instructions from the previous problem to create one.

2. Now let's try running our example from lecture. The `ps_wordcount.py` script from the lecture slides is available at

   `gs://uw-stat679s22-hw11/ps_wordcount.py`

   (alternatively, you can download the demo code from this week's lecture and upload a copy to your own storage bucket).

   `gs://uw-stat679s22-hw11/war_and_peace.txt`

[3] `ctrl-D` inputs the "end-of-file" (EOF) symbol, which is a special ASCII character that basically means "end of transmission". Thus, it is a standard way to end an `ssh` or similar session in a terminal. See `https://en.wikipedia.org/wiki/End-of-file` for more.

contains a slightly modified version of the Project Gutenberg UTF-8 copy of Leo Tolstoy's *War and Peace*.[4] Submit a PySpark job to your Dataproc server that runs `ps_wordcount.py` on `war_and_peace.txt` and outputs the results to a directory

```
gs://NetID-stat679-hw11/WP_wordcount
```

where once again `NetID` is your NetID in all lower-case. Please also copy-paste the command that you called to launch this job into a `Raw NBConvert` cell or a `Markdown` cell in your Jupyter notebook file.

3. Concatenate the output of your script and store it in a file in your storage bucket at `gs://NetID-stat679-hw11/wp_output.txt`. Please also include a copy of this file in your submission.

4 Climate Data Revisited (9 points)

I used NOAA's Climate Data Online service[5] to collect daily historical temperature data for Madison, WI, which has been gathered at Dane County Airport since 1939. I have made this data available on GCP at [6]

```
gs://uw-stat679s22-hw11/NOAA_MSN_temps.csv
```

Each line of this file has the form

```
DATE,TMAX,TMIN
```

where `TMAX` and `TMIN` are integers describing the maximum and minimum temperatures (Fahrenheit) on a given day, and `DATE` encodes a date in the form `YYYY-MM-DD`.

You can see a few lines of the file by writing something like

```
gsutil cat -r 0-101 gs://uw-stat679s22-hw11/NOAA_MSN_temps.csv
```

to print out the first 102 bytes (six lines, at 17 bytes per line) of the file. This `-r` flag to the `gsutil cat` command is the closest thing (to the best of my knowledge, anyway) that `gsutil` has to the UNIX `head` command. **Important:** be careful when performing read operations like this with very large files. Reading multiple GBs or, worse, TBs of text into `less` or a similar command-line program can be very slow!

1. Write a PySpark script that reads two arguments from the command line, corresponding to an input file and an output directory, in that order (the same as the arguments for `ps_wordcount.py`) and computes the average maximum and minimum temperature for every year in the data set. The output should be of the form

```
YYYY, avgmax, avgmin
```

[4] `https://www.gutenberg.org/ebooks/2600`

[5] `https://www.ncdc.noaa.gov/cdo-web/`

[6] **Note:** Once again, this file is not, in reality, large enough to warrant using MapReduce or Spark, but it is good practice.

where `YYYY` is an integer encoding a year, and `avgmax` and `avgmin` are floats encoding the average maximum and minimum temperatures, respectively, for that year. **Note:** the precise formatting here does not matter—just make sure that your output has a line for each year in the data set and the maximum and minimum temperature are ordered correctly. So, for example, an output like

```
YYYY, (avgmax, avgmin)
```

is also fine. Save your script in a file called `ps_year_avgs.py` and include copies in both your storage bucket and your submission. If you wrote any additional Python code (e.g., function definitions in a separate Python file), please also include this in your submission. **Hint:** you may find the `reduceByKey` and `mapValues` transformations to be especially useful.

2. Run `ps_year_avgs.py` on the file `NOAA_MSN_temps.csv` in PySpark on a GCP Dataproc server. Concatenate the output of your job into a single file called `avgs.txt` and save this file in your storage bucket for this homework, and please also include a copy in your submission.

3. Write a PySpark script whose command line arguments are the same as those of `ps_wordcount.py` and `ps_year_avgs.py` and that computes, for each year in the data set, the day on which the maximum temperature was achieved and the day on which the minimum temperature was achieved (you may break ties as you see fit). That is, each row of the output should be of a form like

```
YYYY, MM-DD, mm-dd
```

where `MM-DD` encodes the month and day on which the maximum occurred and `mm-dd` encodes the month and day on which the minimum occurred. **Note:** the precise formatting here does not matter—just make sure that your output has a line for each year in the data set and the maximum and minimum temperature days are ordered correctly and in the correct `MM-DD` format, that is fine. So, for example, an output like

```
YYYY, (MM-DD, mm-dd)
```

or

```
YYYY, 'MM-DD' 'mm-dd'
```

is also fine. Save your script in a file called `ps_year_extremes.py`. Please include a copy of this script in your storage bucket and include a copy in your submission. If you wrote any additional Python code (e.g., function definitions in a separate Python file), please also include this in your submission. **Hint:** you may find it easiest to find the maximum and minimum separately, and then combine the two RDDs using the RDD transformation `join`.

4. Run your `ps_year_extremes.py` script on the file `NOAA_MSN_temps.csv` in PySpark on a GCP Dataproc server. Concatenate the output of your job into a single file called `extremes.txt` and save this file in your storage bucket for this homework, and please also include a copy in your submission.

What to turn in

Here is a list of what to turn in for each problem.

- **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.** You do not need to hand in anything for this problem. Simply make sure that you have successfully created the storage bucket as specified in the problem.
- **Problem 2.** Your Jupyter notebook should include a variable called `number_of_primes`, as well as a copy-paste of the sequence of PySpark commands that you ran in your interactive PySpark session to count the primes. **Important:** make sure that these are in a `Raw NBConvert` or `Markdown` cell, **not** a `Code` cell. If you paste these commands into a Jupyter `Code` cell, the grader script will try to run your PySpark commands in plain old Python, which will cause errors.
- **Problem 3.** Your Jupyter notebook file should include a copy-paste of the command that you called to launch the word-counting script. Remember to paste this into a `Raw NBConvert` cell or a `Markdown` cell, so avoid problems with the grader script. Your storage bucket should include a file

  ```
  gs://NetID-stat679-hw11/wp_output.txt
  ```

  that stores the full output of your PySpark script. A copy of this file should also be included in your submission.
- **Problem 4.** Your submission should include copies of the scripts `ps_year_avgs.py` and `ps_year_extremes.py`, and copies of both of these scripts should also be saved on your HW12 storage bucket. The outputs of these scripts, run on the Madison NOAA data, should be saved in files called `avgs.txt` and `extremes.txt`, respectively. Copies of both of these output files should be included in your submission and saved in your storage bucket.

Reminder: make sure you don't leave any clusters running on GCP! Running clusters cost money! Check the VM instance dashboard or call

```
gcloud dataproc clusters list --region=REGION
```

to make sure you don't have any running instances.