Tutorial 3 - Answers

Hadoop with neonatology dataset

Step 0: Install HADOOP and HIVE

```
$ git clone https://github.com/rebremer/AzureDevTestLabHadoopSparkOnUbuntuScript.git
$ cd AzureDevTestLabHadoopSparkOnUbuntuScript/Artifacts/HadoopHive
$ chmod 777 HadoopHiveUbuntuArtifact.sh
$ ./HadoopHiveUbuntuArtifact.sh
```

Step 1: Fetch the neonatology data

The first command creates a directory called `neo` and moves to this directory.
The second command downloads the `neonatology.zip` file, the -O flag is used to set the name, just for clarification.

```
$ mkdir ~/neo && cd "$_" # shortcut for mkdir and cd
$ wget -O neonatology.zip \
https://github.com/Infomdss2018/infomdss/blob/master/tutorial_4/neonatology.zip?raw=true
```

Step 2: Extract the neonatology data

The `sudo` command is used to run the `apt-get` statement as a root-user, so the `install` command has access to installation directories. `Apt-get` is a package manager which will find the latest version of `unzip` and installs it.

```
$ sudo apt-get install unzip
```

Unzip in action: this will extract the content of `neonatology.zip`.

```
$ unzip neonatology.zip # you can find the password on teams
```

The `hdfs` commands should be familiar by now.

```
$ hdfs dfs -mkdir ~/input_neo
$ hdfs dfs -put Opname-and-lab-2.csv ~/input_neo/
$ hdfs dfs -put Opname-and-lab-3.csv ~/input_neo/
```

Assignments

Part I: Hadoop preprocessing

1. *Merge the two data files "`Opname-en-lab-2.csv`" and "`Opname-en-lab-3.csv`" within Hadoop into one integrated raw data file titled "`neo.csv`" (hereafter: "data file").*

```
$ hdfs dfs -getmerge Opname-and-lab-2.csv Opname-and-lab-3.csv neo.csv
```

Now your current folder (neo) should have the following content:

```
$ ll
total 1521508
drwxrwxr-x  2 labuser labuser      4096 Sep 28 21:03 ./
drwxr-xr-x 17 labuser labuser      4096 Sep 28 21:06 ../
-rw-r--r--  1 labuser labuser 766146643 Sep 28 20:57 neo.csv
-rw-r--r--  1 labuser labuser   5985532 Sep 28 20:57 .neo.csv.crc
-rwxrwxr-x  1 labuser labuser       733 Sep 28 18:45 neo_mapper.py*
-rw-rw-r--  1 labuser labuser  19704588 Sep 28 18:34 neonatology.zip
-rwxrwxr-x  1 labuser labuser       790 Sep 28 18:46 neo_reducer.py*
-rw-rw-r--  1 labuser labuser 404185017 Mar  9  2017
Opname-and-lab-2.csv
-rw-rw-r--  1 labuser labuser 361961626 Mar  9  2017
Opname-and-lab-3.csv
```

2. *Clean the data file by removing any empty lines.*
3. *Clean the data file by removing identical rows.*

Mapper

```
#!/usr/bin/env python

import sys
```

```
"""
Assignment: remove empty lines and identical rows.

One way of doing this:
- Mark empty and duplicate lines with the mapper
- Remove all marked lines with the reducer
"""

# marker:
# remove line => 1
# keep line => 0
remove_line = 0

# You can store the current line (which becomes the previous line in the next iteration)
# By comparing the previous line with the current line, you can determine if two lines are equal
prev_line = None

# iterate over every line from the data-set
for line in sys.stdin:
  line = line.strip()

  if line == "" or prev_line == line:
     remove_line = 1
  else:
     remove_line = 0

  print("{0}\t{1}".format(remove_line, line))

  prev_line = line
```

Reducer

```
#!/usr/bin/env python

import sys

"""
Assignment: remove empty lines and identical rows.

One way of doing this:
- Mark empty and duplicate lines with the mapper
- Remove all marked lines with the reducer
"""

# default values
line = None
remove_line = 0

for line_raw in sys.stdin:
  line_incl_marker = line_raw.strip()

  # note the try/except statement to handle empty lines
  try:
     remove_line, line = line_incl_marker.split("\t")
```

```
except ValueError:
    remove_line = 1

# Note: line now contains the remove_line marker for example:
# 1 \t <line>
# remove_line = the part before the \t
# line = the part after \t

# note the int-conversion
if int(remove_line) == 0:
    # if the line should not be removed, print it
    print('{0}'.format(line))
```

Command

Since you've just created (or downloaded) the mapper and reducer scripts, you have to make them executable, by using the `chmod` command, the `+x` part adds executable rights.

```
$ chmod +x neo_mapper.py # provide execution permission
$ chmod +x neo_reducer.py # provide execution permission
```

Now we can run the hadoop command with the mapper and reducer scripts on the data, we run this command with `sudo` (as root user) so it can find the hadoop-streaming file. The `-E` flag indicates that all environmental variables should be accessible by the root user (so the hadoop command can find the basic java installation folder and other parameters necessary for running the scripts).

```
$ sudo -E /usr/local/hadoop/bin/hadoop jar
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.1.1.jar
-input ~/input_neo \
-output ~/output_neo \
-mapper ./neo_mapper.py \
-reducer ./neo_reducer.py
```

Executing this command should result in an output that ends with:

```
2018-09-28 21:09:38,178 INFO mapred.LocalJobRunner: Finishing task:
attempt_local121151640_0001_r_000000_0
2018-09-28 21:09:38,178 INFO mapred.LocalJobRunner: reduce task
executor complete.
2018-09-28 21:09:38,740 INFO mapreduce.Job:  map 100% reduce 100%
2018-09-28 21:09:38,741 INFO mapreduce.Job: Job
job_local121151640_0001 completed successfully
2018-09-28 21:09:38,768 INFO mapreduce.Job: Counters: 30
```

```
        File System Counters
                FILE: Number of bytes read=11698337174
                FILE: Number of bytes written=11780325414
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
        Map-Reduce Framework
                Map input records=1722812
                Map output records=1722812
                Map output bytes=771315079
                Map output materialized bytes=778206465
                Input split bytes=2323
                Combine input records=0
                Combine output records=0
                Reduce input groups=1
                Reduce shuffle bytes=778206465
                Reduce input records=1722812
                Reduce output records=1722812
                Spilled Records=3445624
                Shuffled Maps =23
                Failed Shuffles=0
                Merged Map outputs=23
                GC time elapsed (ms)=1522
                Total committed heap usage (bytes)=3943038976
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=772390223
        File Output Format Counters
                Bytes Written=772132175
2018-09-28 21:09:38,770 INFO streaming.StreamJob: Output directory:
/home/labuser/output_neo
```

Part II: Map/Reduce processing

Create mappers and reducers to answer the following questions:

1. *How many measurements (that is rows) are available for each patient in the data file? Note: that the patient id is denoted as* `PseudoID_voorkeur`.

You can solve this question by only using a reducer, since all the necessary content is already present in the data. For this reason, is very simple mapper is included, which only print the current lines.

The reducer consists of three different parts (1) first the patient id is extracted from a preprocessed line line_list[0], lines that are formatted differently or contain usuited values for patient id are ignored within the try-except statement, which will try to parse a patient id as integer and continues to the next line in the data-file when this is not possible. The next part (2) will check if the patient id from the current line is the same as the patient id from the previous line, if this is the case, the counter is increased (representing the number of measures of the current patient). When the current patient id is not equal to the previous patient id, the number of measures and the patient id from the previous patient are printed. Lastly (3), the last line patient in the file is also printed.

Mapper

```
#!/usr/bin/env python

import sys

# iterate over every line from the data-set
for line in sys.stdin:
  print(line.strip())
```

Reducer

```
#!/usr/bin/env python

import sys

prev_patient_id = None
counter = 1

for line_raw in sys.stdin:
  # line preprocessing
  line_raw = line_raw.strip()
  line_list = line_raw.split(';')

  # try to parse the patient id as int, if this results in an error, continue with the next line in the data-file
  try:
    int(line_list[0])
  except ValueError:
    continue

  current_patient_id = line_list[0]
```

```
  # if the current patient id equals the previous patient id, increase the number of measures,
  # else print the number of measure from the previous patient id
  if current_patient_id != prev_patient_id:
      if prev_patient_id:
          print('{0}\t{1}'.format(prev_patient_id, counter))
      counter = 1
      prev_patient_id = current_patient_id
  else:
      counter += 1

# process the last patient from the data file
if prev_patient_id:
  print('{0}\t{1}'.format(prev_patient_id, counter))
```

2. *How many patients are in the data file?*

The strategy of the tasks is comparable with the previous task, again the patient id is extracted and an integer parse is made. The key difference here is that we do not need to count the number of measures, so we only increase the counter when the current patient id is different from the previous patient id.

Mapper

```
#!/usr/bin/env python

import sys

# iterate over every line from the data-set
for line in sys.stdin:
  print(line.strip())
```

Reducer

```
#!/usr/bin/env python

import sys

prev_patient_id = None
counter = 1

for line_raw in sys.stdin:
  # line preprocessing
  line_raw = line_raw.strip()
  line_list = line_raw.split(';')

  # try to parse the patient id as int, if this results in an error, continue with the next line in the data-file
  try:
      int(line_list[0])
  except ValueError:
```

```
        continue

    current_patient_id = line_list[0]

    # increase the counter when the current patient id is different from the previous patient id
    if current_patient_id != prev_patient_id:
        if prev_patient_id:
            counter += 1
        prev_patient_id = current_patient_id

# when all patient are counted, print the result
print('NUM OF PATIENTS: {0}'.format(counter))
```

3. *What is the hospitalisation duration for each patient, given that '`Opnamedatum_OPN`' records the moment hospitalisation and '`Ontslagdatum_OPN`' denotes the discharge date?*

Mapper

```
#!/usr/bin/env python

import sys

# iterate over every line from the data-set
for line in sys.stdin:
    print(line.strip())
```

Reducer

Here is where things get interesting, we need to compare two dates and times which are denoted by a unfamiliar notation. So the key problem of this reducer is to parse a date the looks like "`08JAN2014`" to a datetime python object (which is easy to work with when you want to compare dates and times).

Since the hospitalization datetimes are equal in every line of the same patient, we can just ignore duplicates.

Note: some hospitalisation times are negative, these should be filtered.

```
#!/usr/bin/env python

from datetime import datetime
import sys
import time

month = {"JAN": '1', "FEB": '2', "MAR": '3', "APR": '4', "MAY": '5',
```

```
      "JUN": '6', "JUL": '7', "AUG": '8', "SEP": '9', "OCT": '10',
      "NOV": '9', "DEC": '12'}

prev_patient_id = None

for line in sys.stdin:
  line = line.strip()
  line_l = line.split(";")

  # add a default value is patient id
  patient_id = -999

  # ignore non-integer patient ids
  try:
     patient_id = int(line_l[0].strip())
  except ValueError:
     continue

  # different measures of the same patient
  if prev_patient_id == patient_id:
     continue

  prev_patient_id = patient_id

  # extract the hospitalization date and release date from the data-line, all values should be integers
  # except the months fields, which are parsed to integers with the month-dictionary at the top of the script.
  try:
     hospitalization_id = line_l[1].strip()
     hospitalization_date_day = int(line_l[39].strip()[:2])
     hospitalization_date_month = int(month.get(line_l[39].strip()[2:5]))
     hospitalization_date_year = int(line_l[39].strip()[5:])
     hospitalization_time_hour = int(line_l[40].strip()[:2].strip(':'))
     hospitalization_time_minutes = int(line_l[40].strip()[3:])

     release_date_day = int(line_l[41].strip()[:2])
     release_date_month = int(month.get(line_l[41].strip()[2:5]))
     release_date_year = int(line_l[41].strip()[5:])
  except (IndexError, TypeError):
     continue

  # When the release-time is empty, we use the hospitalisation time.
  if line_l[42] == '':
     release_time_hour = hospitalization_time_hour
     release_time_minutes = hospitalization_time_minutes
  else:
     release_time_hour = int(line_l[42].strip()[:2].strip(':'))
     release_time_minutes = int(line_l[42].strip()[3:])

  # Use the extracted hospitalisation datetime to create a python datetime object
  hospitalization_date_time = datetime(hospitalization_date_year,
                                       hospitalization_date_month,
                                       hospitalization_date_day,
                                       hospitalization_time_hour,
                                       hospitalization_time_minutes, 0)

  # Use the extracted resleas datetime to create a python datetime object
  release_date_time = datetime(release_date_year,
                               release_date_month, release_date_day,
                               release_time_hour,
                               release_time_minutes, 0)
```

```
# convert a python datetime object to an integer (representing the datetime in seconds since 01-01-1970)
# see https://en.wikipedia.org/wiki/Unix_time
# the differences between hospitalization and release datetime in seconds, is the hospitalization duration
d1_ts = time.mktime(hospitalization_date_time.timetuple())
d2_ts = time.mktime(release_date_time.timetuple())
hospitalization_duration = int(d2_ts - d1_ts) / 60

print("{0}\t{1}\t{2}\t{3}\t{4}".format(patient_id, hospitalization_id,
                                       hospitalization_date_time,
                                       release_date_time,
                                       hospitalization_duration))  # time in minutes
```

4. *What is the average hospitalisation duration for a patient?*

The output from the previous question is used to calculate the average hospitalisation duration. The script itself is comparable with question 1 and 2, but now a variable total_hospitalisation_time is increased with the hospitalisation duration of each patient. In the last step, the average hospitalisation duration is calculated by dividing the total hospitalisation time over the number of patients in the dataset.

Mapper

```
#!/usr/bin/env python

import sys

# iterate over every line from the data-set
for line in sys.stdin:
  print(line.strip())
```

Reducer

```
#!/usr/bin/env python

import sys

number_of_patients = 0
total_hospitalisation_time = 0

for line_raw in sys.stdin:
  line_raw = line_raw.strip()
  line_list = line_raw.split('\t')

  hospitalisation_time = float(line_list[4])

  # only include positive hospitalisation durations greater than zero
  if hospitalisation_time > 0:
     total_hospitalisation_time += hospitalisation_time
```

```
        number_of_patients += 1

# calculate and print the average hospitalisation duration
print('NUMBER OF PATIENTS: {0}'.format(number_of_patients))
print('AVERAGE HOSPITALISATION TIME: {0}'.format(float(total_hospitalisation_time) / number_of_patients))
```

Command

Note we use the data generated in question 3

First we have to merge all output data from question 3, in this example the data is stored in `output_neo_q3`, but you can have chosen a different name for your output folder. The merged data is stored in the `hosp_duration.csv` file.

```
$ hdfs dfs -getmerge output_neo_q3/* hosp_duration.csv
```

Now can create a new input directory with the `hosp_duration.csv` file with the commands from part 1 question 1. For example:

```
$ hdfs dfs -mkdir ~/input_neo_q4
$ hdfs dfs -put hosp_duration.csv ~/input_neo_q4/
```

And then we can run the hadoop command with the new input directory, note that the `neo_mapper_q4.py` and `neo_reducer_q4.py` should contain this questions mapper and reducer.

```
$ sudo -E /usr/local/hadoop/bin/hadoop jar
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.1.1.jar
-input ~/input_neo_q4 \
-output ~/output_neo_q4 \
-mapper ./neo_mapper_q4.py \
-reducer ./neo_reducer_q4.py
```

5. *Which patients are hospitalized the longest? Make a top ten.*

For this assignment, a mapper and reducer are used over the dataset from assignment 3. The mapper prints the hospitalisation duration and the patient id, which is used by the reducer to create a op ten list.

Note we use the data generated in question 3

Mapper

```
#!/usr/bin/env python

import sys

for line_raw in sys.stdin:
  line_raw = line_raw.strip()
  line_list = line_raw.split('\t')

  # get the patient id and the hospitalisation duration
  patient_id = line_list[0]
  hospitalisation_time = float(line_list[4])

  # print the hospitalisation duration and patient id
  print('{0}\t{1}'.format(hospitalisation_time, patient_id))
```

Reducer

```
#!/usr/bin/env python

import sys


# create rank list, which contains the patients with the longest hospitalisation duration
patient_rank_list = []

for line_raw in sys.stdin:
  line_raw = line_raw.strip()
  line_list = line_raw.split('\t')

  # get the patient id and the hospitalisation duration
  patient_id = line_list[1]
  hospitalisation_time = float(line_list[0])

  # loop through the values in the rank list,
  # if the current hospitalisation duration is greater than one of the values in the list
  # insert the current patient id hospitalisation duration
  # since the length of the rank_list increases when a patient is added, we use break to avoid an infinite loop
  for n, v in enumerate(patient_rank_list):
     if hospitalisation_time > v[1]:
        patient_rank_list.insert(n, [patient_id, hospitalisation_time])
         break

  # if the rank list contains less than ten values
  # store the current patient and the current hospitalisation duration
  if len(patient_rank_list) < 10:
     patient_rank_list.append([patient_id, hospitalisation_time])


# print the top ten of longest hospitalized patients, including their rank.
for n, v in enumerate(patient_rank_list):
```

```
  if n <= 10:
     print('#{0}\t{1}\t{2}'.format(n, v[0], v[1]))
```