Big Data Hackathon 2017 Program

The goal is to take you through all the steps of a small big data project in a single day and give you some hands-on experience on Amazon's cutting-edge big data platform Amazon Web Services (AWS). The steps below are guidelines that you can follow – you can depart from them at any point and originality in what you do may bring rewards! **Above all, try to explore and have fun**!

Step 1: Understand the business case

We can use satellite data (remote sensing) to make decisions that confront climate change. Can satellite data help people decide what regions will continue to be safe to live in and which should be avoided? Where it's best to grow crops? To answer questions like these, we can use terrabytes of landsat satellite data sets that are publicly available on aws.

Step 2: Get your data

Technologies used: Amazon s3 (Simple Storage Server), Amazon Linux AMI (Amazon Machine Image), AWS CLI (command line interface)

Amazon has already peformed this step for us. They have made publicly available all Landsat 8 images on their s3 cloud storage system. The satellites follow a certain trajectory (https://landsat.gsfc.nasa.gov/wp-content/uploads/2013/01/wrs2.gif) characterized by paths and rows. For each path/row combination, a picture is taken every 16 days. If we are interested in the data for Antwerp (path 044 and row 034), we can see all available files through the command line interface:

aws s3 ls s3://landsat-pds/L8/044/034/

To copy a file to our local directory we would use

aws s3 cp s3://nasanex/Landsat/gls/2010/198/024/LT51980242010249KIS01.tar.gz /home/hadoop/data/Antwerp.tar.gz

although this is not necessary: we will be able to directly access the data on s3 from our computing framework (as will be explained in step 6).

More info on the meaning of the names of the files you encounter can be found here: https://aws.amazon.com/public-datasets/landsat/ . The metadata of the satellite pictures can be found in the csv file at the bottom of the page. This contains information like the path, row, acquisition date, and cloud cover at the moment of acquisition.

Step 3: Explore and clean your data

Technologies used: Amazon EMR (Elastic MapReduce), Spark (cluster-computing framework), PySpark (Spark Python API), Zeppelin notebook (open web-based notebook for interactive data analytics), hdfs (Hadoop Distributed File System)

EMR is a web service that uses Hadoop to process vast amounts of data. We have spun up an EMR cluster with Spark running on it that you can access (see Appendix: Set-Up Guide). We can use one of Spark's APIs (example below make use of pyspark) to explore the scenes csv file and gain insight into the cloud cover of the > 1 million pics available. A good overview of pyspark commands that will come in handy can be found here: https://spark.apache.org/docs/latest/sql-programming-guide.html. Here are some suggested steps for exploration of the available data:

- Download scenes file in your team's folder:
  ➔ *wget https://landsat-pds.s3.amazonaws.com/scene_list.gz*
- unzip it and rename to csv
- make a folder for your team on hdfs:
  ➔ *hdfs dfs -mkdir /user/hadoop/<team name>*
- make the file available on the cluster :
  ➔ *hdfs dfs -put scene_list.csv /user/hadoop/<team name>/scene_list.csv*)
- Open a Zeppelin note (notebook -> create new note with spark as default interpreter)
- load the csv file into a spark dataframe (all commands in Zeppelin should be proceeded by *%spark.pyspark*):
  ➔ make use of df = *spark.read.format("csv")* and indicate in the option that there is a header that contains the column names
- Cloudcover is a percentage tha expresses the probability that a pixel is covered by clouds. What values do you expect? Are there any values outside of this range? Does the data contain duplicates? Clean your data if necessary.
  ➔ apply *drop_duplicates()* to the dataframe and possibly a *filter* function to only look at the positive values
- Would it not be easier if we could just run a query on our data to get to know it better? Try to turn it into a query-friendly format!
  ➔ *createOrReplaceTempView* will do this for you (see link provided in the introduction to accomplish all steps below)
- It is hard to find good satellite pictures of Belgium...always so cloudy! Can you find the least cloudy day in Belgium in the available dataset?
  ➔ *Spark.sql("SELECT … FROM …)"* can now be used to write an sql-like query on the data
  The download_url column will give you an address that you can paste in the browser to see the picture. Now compare this to the most cloudy day. What a difference!
- Time for a vacation! Select the average cloudcover in Belgium and compare it to that of the Costa Blanca. You can use for instance Benidorm's coordinates. Possibly you will have to import the mean function
  ➔ *from pyspark.sql.functions import mean*
- July in Belgium is the sunniest month, averaging 7 hours of sunshine per day... Can you figure out the average cloudcover in the month of July using all of the available data?
- Can you figure out the cloudcover averages for all months...in just 1 sql command line? To test your results, try to answer the following question: in Southern California, two months (outside of the wintermonths December through February) are known to have a weather pattern that results in cloudy, overcast skies. This gives rise to appropriate nicknames for those months. Can you show the monthly average cloud covers, and find out what those two months are (nickname included!) using the dataset available for the coastal region around the city of San Diego?

Step 4: Store your findings

Technologies used: Amazon DynamoDB (NoSQL database), boto3 (AWS Software development kit for Python)

Users are charged depending on how heavily they make use of EMR 's computing power. It is possible that the EMR cluster will be terminated from time to time to avoid unnecessary charges when the cluster is idle. It would be very useful to permanently store our newly found results. We assume that you stored the results from step 3 in a dataframe that contains 12 months and the corresponding average cloud cover. Now we will save this in a nosql db using DynamoDB.

Follow the guidelines in http://boto3.readthedocs.io/en/latest/guide/dynamodb.html and see if you can write the whole dataframe containing the monthly averages to the DB using the batch_writer() function. In case you need inspiration, follow these steps:

- Import boto3 python library:
  ➔ *import boto3*
- Tell boto that you want to make use of dynamodb. Here you will need to specify the region in which our EMR is running
  ➔ *dynamodb = boto3.resource('dynamodb',region_name='eu-west-1'*)
- Create a table to store cloudcovers
  ➔ Make use of *table = dynamodb.create_table;* as explained in the tutorial. The CreationDate can be used as hashkey. A string for the team name could be used as range key. The combination hashkey/range key needs to be unique for each record.
- The schema names from the dataframe *month_avg* containing the query result can be changed into more appropriate names like 'months' and 'cloudavgs'. This will make the next step easier.
  ➔ *oldColumns = month_avg.schema.names*
  newColumns = ["month", "cloudavg"]
  month_avg = reduce(lambda month_avg, idx:
  month_avg.withColumnRenamed(oldColumns[idx], newColumns[idx]),
  xrange(len(oldColumns)), month_avg)
  month_avg.printSchema()
  month_avg.show()
- Save the results of your dataframe obtained by the query with the cloud averages in a list by performing a *collect* on the data.
  ➔ *months = month_avg.select('month').collect()*
  cloudavgs = month_avg.select('cloudavg').collect()
- Now import the averages for each month :
  ➔ Make use of the *batch_writer()* function in a for loop with maximum value 12
- *You can check if the table was filled with your data:*
  response = table.scan()
  items = response['Items']
  print(items)

Step 5: Build Visualizations

Technologies used: Amazon EMR, Zeppelin, Spark, Python, Amazon S3, HDFS

Now you can admire any place that interests you on the planet. A conversion between the path/row and latitude/longitude can be done through https://landsat.usgs.gov/wrs-2-pathrow-latitudelongitude-converter . The landsat 8 satellite produces data from 11 different spectral bands[1]. Depending on the purpose of the study, you can make use of a certain combination of bands. One of the most simpl operations is to generate RGB map. Here an image consists of bands 4-3-2.

Transforming the image into a data array:

Select an image of the place of interest on a day with clear skies. Import this image into a spark RDD (example code in step 1 below). Once the binary file is saved in an RDD, transform this data into an array using Numpy array. First we will need to convert our RDD to a byte array before we can import this into our Numpy array (step 2 below). This however cannot be visualized yet because it merely contains bytes in utf8. We need to decode it and return a Numpy Array (step3). After that we need to format this Numpy Array to uint8 (as this is the required format for matplotlib which we will use later on) instead of uint16. Therefore (step 4) divide the data by 255. This way we end up with data that can be visualized with matplotlib. Note that the recipe above only describes the process for one specific band(#3, i.e. green in the example below). You will need to duplicate the piece of code below two more times for the other bands by changing the s3-url and variable name accordingly.

In summary the performed steps are:

1. L3 = sc.binaryFiles('s3://landsat-pds/L8/139/045/LC81390452015058LGN00/LC81390452015058LGN00_B3.TIF.ovr').collect()

   It is very important to use the ".collect()" because this will import all the data given by this binary file into the RDD called L3.

2. file_bytes3 = np.asarray(bytearray(L3[0][1]), dtype=np.uint8)
3. R3 = cv2.imdecode(file_bytes3,cv2.IMREAD_UNCHANGED)
4. R3 = R3 / 255

Stacking the correct bands

Now the data is in the correct format for the 3 desired bands, we can stack the bands to produce a clear image. It is up to you to find a way to do this. For inspiration, take a look at following links:

1. https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.dstack.html
2. https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.imshow
3. https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ndarray.astype.html
4. Use show() at the end to show your image in Zeppelin.

[1] See https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-bands/ for detailed information on each band.

Step 6: Get predictive (bonus)

Technologies used: Amazon EMR, Zeppelin, Spark, Python, Amazon S3

Water is key in growing crops. Can you find a way to determine the water content of an image? You could start with one image and try to calculate how many pixels are water. Can you find out in what year they had the least amount of water for a specific picture? What other kinds of questions might this data help us answer?

The NWDI Normalized Water Different Index (NWDI)[2] seems a useful tool for our purpose. This makes use of the the NIR (Near Infrared) and SWIR (Shortwave Infrared) bands .

These will be the steps to adjust your previous code:

1. Create the needed data set by using the bands 7 and 5. Do the calculation you found for the NWDI and name this data set (e.g. imagetest).
2. Divide the data set again by 255 to get a number between 0 and 5.
3. Run a nested for loop to count the number of water pixels:
   # we need to initialize a counter

   ```
   counter = 0.0  #( the decimal ensures python will not see it as an integer )

   for rgb_value in imagetest:

           for value in rgb_value:

                   # by manual inspection of the rgb values we found out that 5 represents
                   water for  #this specific image

                   if value == 5:

                           counter = counter + 1.0
   ```

4. The initially obtained percentage will seem low. This is because we still need to clean our data. When inspecting an image we see it is rotated and we have some black borders that should not be counted when calculating the percentage. This can be done by adding
   ```
   if value == 0:
           int = int - 1.0
   ```
   into our already existing nested for loop. (don't forget to make a variable called "int" that you instantiate with the length of one band (not the whole array). If you find problems doing this take a look at RDD.shape to find out how you can find your size of your array.
5. Divide the "counter" by the "int" and multiply this by 100 to obtain the percentage of water for this specific map.

Step 7: Present your results

Showcase your findings of the previous day – try to be original. Even if your code is not working 100% properly, try to come up with a good plan of attack and show off your presentation skills!

[2] https://en.wikipedia.org/wiki/Normalized_difference_water_index

Want to know more?

Check out the following links if you want more information:

https://aws.amazon.com/ Secure cloud computing environment, suitable for big data projects.

https://aws.amazon.com/emr/ Framework to process vast amounts of data that can scale.

https://spark.apache.org/ Cluster-computing framework for big data processing. Several online courses are available on EdX and Coursera.

https://spark.apache.org/docs/0.9.0/python-programming-guide.html API for programming on Spark using Python.

https://aws.amazon.com/dynamodb/ Amazon's fast NoSQL database.

www.bigindustries.be/ The coolest big data consultancy company on the block.

Appendix: Set-up guide

1. All necessary files (such as the ppk key for logging in and this document) are available on https://drive.google.com/open?id=19wQDtkO2Vb4PoqtMNbxZ65Et1t7Pi3cW .
2. Required software installation when working under windows: putty to connect to the EMR master node.
3. We will need to add your IP address to allow inbound traffic to the EMR master node. Please check and provide us with your IP address e.g. through http://checkip.amazonaws.com/ .
4. Log on to the EMR's master node using putty. Under connection ->ssh ->Auth , browse on your pc to select the private key for authentication, using the ppk key that is provided (SSHKey.ppk). Your host name will be hadoop@ec2-54-77-222-53.eu-west-1.compute.amazonaws.com. Port is 22 and connection type ssh.
5. Once logged in, you will find a directory /home/Hadoop/<teamname>. Store all files your files in this directory.
6. Log in to Zeppelin using http://ec2-54-77-222-53.eu-west-1.compute.amazonaws.com:8890/ . You will be assigned a team number *x* and can use the credentials user = user*x*, password=password*x*. Now start a new note (notebook +) . Make sure you that you **let every command proceed by '%spark.pyspark'** to provide a spark environment.