CSE 545: Fall 2017 - Assignment 2

Big Data Analytics
Stony Brook University
CSE545 - Fall 2017

Assignment 2

Due: ~~11/22/2017 11:59pm~~ 11/27/2017 1:59am

Overall Task: Find similar regions of Long Island by comparing satellite imagery.

Objectives:

- Implement Locality Sensitive Hashing.
- Implement dimensionality reduction -- learn through doing.
- Gain further experience with Spark.
- Gain further experience with data preprocessing.
- Explore a different modality of data: satellite images.

Data:

The data are high resolution orthorectified satellite images. Orthorectified images satellite pictures which have been corrected for abnormalities due to tilt in photography or geography. Read more here: https://lta.cr.usgs.gov/high_res_ortho

There are two datasets:

1. **small_sample:** hdfs:/data/small_sample
   http://www3.cs.stonybrook.edu/~has/CSE545/assignments/a2_small_sample.zip -- 5 orthorectified images; small enough for ones own machine and for developing the code
2. **large_sample:** hdfs:/data/large_sample
   Approximately 50 images -- full data set -- the data we will test

Within each sample are additional zip files which contain, among other information, a tiff image file that defines each pixel of the image in terms of red, green, blue, and infrared (i.e. each pixel is represented by 4 integers). **Copy one of the zip files and take a look inside to find the tiff file and view it.**

Get access to class AWS Server (Due 11/11):

- Send Youngseo (yson@cs.stonybrook.edu) your *public* ssh key. Please send the email from one of your stonybrook.edu email addresses
- Once Youngseo acknowledges that you have been added to the server try to ssh in
  a. User name: (Youngseo will provide)
  b. Address: **ec2-107-23-109-58.compute-1.amazonaws.com**
     - port: 22
     (use your *private* (id_rsa or *.ppk) key on your end)
  c. Set spark environment variable to python3
     - add "export PYSPARK_PYTHON=python3" to .bashrc
     - run "source .bashrc"
  d. Test that spark shell works for you:

```
$ pyspark
Python 3.4.3 ….
...
>>> rdd = sc.binaryFiles('hdfs:/data/small_sample')
>>> rdd.map(lambda x: x[0]).collect()
['hdfs://ip-172-31-45-217.ec2.internal:8020/data/small_sample/3677453_2025190.zip', 'hdfs://ip-172-31-45-
217.ec2.internal:8020/data/small_sample/3677454_2025195.zip', 'hdfs://ip-172-31-45-
217.ec2.internal:8020/data/small_sample/3677500_2035190.zip', 'hdfs://ip-172-31-45-
217.ec2.internal:8020/data/small_sample/3677501_2035195.zip', 'hdfs://ip-172-31-45-
217.ec2.internal:8020/data/small_sample/3677502_2035200.zip']
(-5 for missing access deadline)
```

(NOTE: Youngseo may take up to 36 hours to enable access)

Step 1. Read image files into an RDD and divide into 500x500 images *(25 points)*

(a) Create an rdd of the orthographic zip files in the specified location. Pull the filenames out from the rest of the path (e.g. '/data/ortho/small_sample/3677453_2025190.zip' => '3677453_2025190.zip') and make sure to hang on to it.

(b) Find all the tif files and convert them into arrays (make sure to hang on to the image filename).The following method may be useful:

```
from tifffile import TiffFile

def getOrthoTif(zfBytes):
 #given a zipfile as bytes (i.e. from reading from a binary file),
# return a np array of rgbx values for each pixel
 bytesio = io.BytesIO(zfBytes)
 zfiles = zipfile.ZipFile(bytesio, "r")
 #find tif:
 for fn in zfiles.namelist():
  if fn[-4:] == '.tif':#found it, turn into array:
   tif = TiffFile(io.BytesIO(zfiles.open(fn).read()))
   return tif.asarray()
```

(c) Divide the image into 500x500 subimages. Each tif image should be 2500x2500 or 5000x5000 pixels, so you will end up with 25 500x500 images (image breaking the image into a 5x5 grid) or 100 500x500 images.

(d) Produce a final RDD where each record is of the form: (imagename, array).

Name for each 500x500 image (i.e. matrix) with the file name it was derived along with the cell number of the 500x500 pixel, labeling by rows and then columns (as depicted to the right). For example, if the file was "3677454_2025195.zip" the 500x500 portion corresponding to cell 18 would be labeled "3677454_2025195.zip-18"

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	34

(e) **Print the r, g, b, x values for the pixel (0,0) in the following images:: 3677454_2025195.zip-0, 3677454_2025195.zip-1, 3677454_2025195.zip-18, 3677454_2025195.zip-19

example output:

```
[('3677454_2025195.zip-0', array([114, 111, 109, 114], dtype=uint8)),
 ('3677454_2025195.zip-1', array([ 54,  53,  57, 117], dtype=uint8)),
 ('3677454_2025195.zip-18', array([ 79,  70,  66, 123], dtype=uint8)),
 ('3677454_2025195.zip-19', array([61, 57, 63, 84], dtype=uint8))]
```

Step 2. Turn each image into a feature vector *(25 points)*

For each 500x500 image (each image should be an RDD record):

(a) Convert the 4 values of each pixel into a single value,

`intensity = int(rgb_mean * (infrared/100))`

(i) rgb_mean: the mean value of the red, green, and blue values for the pixel

(ii) infrared: (the 4th value in the tuple for the pixel

E.g. if the pixel was (10, 20, 30, 65), rgb_mean would be 20 and infrared would be 65.

Thus, `intensity = int(20 * (65/100)) = 13`

(b) Reduce the resolution of each image by `factor = 10.` For example, from 500x500 to 50x50. To do this, take the mean intensity over factor x factor (10x10) sub-matrices. (Note one might build on the method for step 1(c); factor may change for later steps).

(c) Compute the **row** difference in intensities. Direct intensity is not that useful because of shadows. Instead we focus on change in intensity by computing how it changes from one pixel to the next. Specifically we say intensity[i] = intensity[i+1] - intensity[i] (the numpy function 'diff' will do this). Then we turn all values into 3 possible values: Convert all values < -1 to -1, those > 1 to 1, and the rest to 0.

e.g. if the row was: [10, 5, 4, 10, 1], then the diff would be [-5, -1, 6, -9] and the vector after conversion

would be [-1, 0, 1, -1] (notice there is one less column after doing this).
we call this **row_diff**
(d) Compute the **column** difference in intensities. Do this over the direct intensity scores -- not row_diff.
we call this **col_diff**
(e) Turn row_diff and col_diff into one long feature vector (row diff and col_diff do not have to be separate RDDs). Row_diff and col_diff are matrices. Flatten them into long vectors, then append them together. Call this **features**. Row_diff has 50rows x 49columns = 2450 values and col_diff has 49x50 = 2450 values. Thus, features will have 2450 + 2450 = 4900 values,
(f) **Print the feature vectors for: 3677454_2025195.zip-1, 3677454_2025195.zip-18 (just the numpy reduced string output: first 3 + last 3 values)

example output:

```
[('3677454_2025195.zip-1', array([ 1, -1, -1, ...,  1, -1, -1])),
 ('3677454_2025195.zip-18', array([-1,  1, -1, ...,  1, -1,  1]))]
```

Step 3. Use LSH, PCA to find similar images *(50 points)*

(a) Create a "signature" for each image: Pass the feature vector, in 128 similarly-sized chunks (e.g. a combination of 38 and 39 features per chunk), to an md5 hash function and take one *character* from the hash per chunk in order to end up with 128 *characters* as your signature for the image (note that the characters can be bits, ints, letters, or bytes -- your choice). This is analogous to the signature you get per document at the end of minhashing. Here, we are not using minhashing but the next steps for LSH are the same as if we had just produced a signature matrix using minhashing.
(b) Out of all images, run LSH to find approximately 20 candidates that are most similar to images: 3677454_2025195.zip-0, 3677454_2025195.zip-1, 3677454_2025195.zip-18, 3677454_2025195.zip-19. Treat each image as a column. Note that while you might have things stored as rows being signature vectors for images, the slides depict columns as the signature vectors.

(i) Tweak brands and rows per band in order to get approximately 20 candidates (i.e. anything between 10 to 30 candidates per image is ok). Note that there are 128 rows total, which you can divide evenly into 1, 2, 4, 8, 16, 32, or 64 bands, but you're also welcome to divide into a number that doesn't evenly fit, in which case just leave out the remainder (e.g. 3 bands of 5 rows, and ignore the last row).
(ii) Note that in LSH the columns are signatures. Here each RDD record is a signature, and so from the perspective of LSH, each record is a column.
(iii) While pre existing hash code is fine, you must implement LSH otherwise.
**print the ~20 candidates for 3677454_2025195.zip-1, 3677454_2025195.zip-18
(run on all 4 but only print for these 2)
(multiple correct answers)

(c) For each of the candidates from (b), use PCA and find the euclidean difference in low dimensional feature space of the images.

The following is provided as an aid to understand SVD:
CSE545: SVD Programming Handout

(i) Start by running PCA across the 4900 dimensional feature vectors of all images to reduce the feature vectors to only 10 dimensions. You must run this step across partitions within spark (use concepts of the course to figure out how to do this; there is no single correct approach and approximate solutions are ok). For example, one option is to run SVD separately on different samples (i.e. random batches) and then combine all the low dimensional batches into one. Solutions that are particularly efficient and accurate may receive extra credit. You may lookup how distributed statistics libraries distribute PCA/SVD but you must abide by policies of academic integrity and not copy code in part or whole.
(ii) Then, compute the euclidean distance between 3677454_2025195.zip-1, 3677454_2025195.zip-18, and each of their candidates within this 10 dimensional space. (can be done outside RDDs)
(iii) **print the distance scores along with their associated imagenames, sorted from least to greatest for each of the candidates you found in 3(b).
(multiple correct answers)

Extra Credit *(10 points)*

(d) Repeat (c), but instead change the resolution factor (step 2.(b)) to **5**, so you have 100x100 pixel images. Again, tweak bands and rows until you get approximately 20 candidates (note how changing the

resolution affected the number of bands/rows).
**print the distance scores along with their associated imagenames, sorted from least to greatest (this should run at the end of your code, after 3(c).iii)

Tips

- Use persist before step 2.(b) since you will go back and rerun on a different resolution
- The hash function used for creating signatures *before* locality sensitive hashing should be applied to 128 chunks in order to get similar patterns mapping to the same place.
- For the hash function within locality sensitive hashing, make sure you are using a sufficiently large number of buckets that it is rare 2 images hash to the same bucket unless they have the same pattern.appear within them.
- There are many places in step 3 where you need to make decisions in this assignment and there is no single correct answer. The idea is to be closer to a real world applied situation where there are endless options. Please be sure to comment your code clearly to explain the choices you have made.
- Note that the instructions for 3(a) have been updated. There is no longer a restriction to keep the signatures to 16 bytes. The signature can be made up of bits, ints, letters, bytes (or anything inbetween). This is to allow more flexibility to reach the 10 to 30 matching image requirement.
- small_sample and large_sample will naturally have different optimal band-sizes to get to 10 to 30 images. The small sample is just intended for development. The large sample is what we will test against and what your output should reflect.
- For steps 2 (c) and (d) row diff and col_diff do not have to be separate RDDs. It's explained as two sets of features to be clear about what is in each.

Guidelines.

All code should be placed into a single file: "a2_lastname.py" (replace ".py" with ".java" or ".scala" if appropriate).
We should be able to run the first file as is (i.e. "spark-submit a2_lastname.py"). Use a jar for Java or Scala if you have multiple file (make sure the jar includes the code).

Output of your code, from the large_sample, should also be submitted as "a2_lastname_output.txt"

Submission. Two **uncompressed** files (code and output) should be submitted via blackboard.

Python 3 libraries permitted: default file IO libraries, numpy/scipy (only for arrays and algebra, no statistical tests or regression besides those below), pprint, multiprocessing. No additional image processing libraries (beyond those below) may be used and only the default Spark 2.2.0 transformations and actions. If unsure, ask.
Valid (and likely useful) imports:

```
import io
import numpy as np
Import scipy.stats as ss
import zipfile
from tifffile import TiffFile
from PIL import Image #this is an image reading library
from numpy.linalg import svd
```

Questions / Clarifications: Please post questions on Piazza, so other classmates may see the answers. Questions posted within 48 hours of the deadline are not guaranteed a response before the deadline.

Academic Integrity: As with all assignments (sans the team project), although you may discuss concepts with others, you must work independently and insure your work and code is not visible to any classmates. You may also not copy any partial solutions from the Web or any other resources though you may reference algorithm descriptions and method parameter definitions.

Your home directories will be activated such that only you have access to your home directory. However, it is your responsibility to keep your files and access credentials to the class AWS server confidential. Sharing of files or access credentials with anyone in the class or outside the class will be considered cheating. Such information may only be shared with the professor or TAs.

Spark Cluster Considerations: Please be considerate that the spark cluster is being shared with a large number of your classmates.

- Please only submit one job at a time and use spark-submit (rather than spark shell) if your job is large.
- Do not try to adjust memory or any default settings when submitting your job.
- Do not try to run spark in local mode or to run your own local spark on the cluster.
- A complete version of the assignment can run in < 10 minutes on the cluster. Jobs that take longer than 1 hour are subject to being killed.
- Please do not store more than 1GB in your home directory. You can store an additional 2GB in hdfs:/user/USERNAME/. There is no need to store more than your code and the output in your home directory.
- Otherwise, enjoy having a medium-large spark cluster at your finger-tips!
(10 to 30 nodes, 300 to 800GB memory, 96 to 256 cores)