EE379K
Spring 2016
Dr. Sriram Vishwanath

Lab 2 – K-means

Assigned: 2/18/16 **Due: 3/2/16 by 11:59pm**

For this assignment you will be implementing two versions of the standard K-means algorithm and using them to performing color compression and movie database analysis.

Part 0: Setup

Go into your Git directory and do a "git pull" to get the new materials for this assignment. You should see a "lab2" directory if the pull was successful. If you do not see it, contact me.

In the lab2 directory there will be a file called "setup.sh". This script will install all the dependencies you need to complete the lab. First, open a terminal in the lab2 directory. Then, give the script execute permission by issuing the following command:

```
chmod +x setup.sh
```

Then run the script by typing:

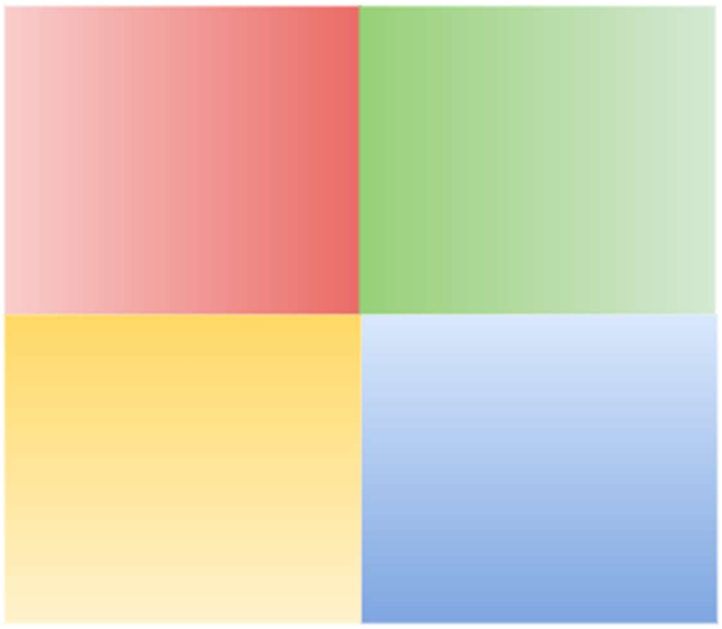
```
./setup.sh
```

You will be prompted for the sudo password, which is "sp2016" without quotes. You will also be asked to approve the installation of the dependencies. Approve them by typing **Y** and then pressing enter.

Contact me if the script fails.

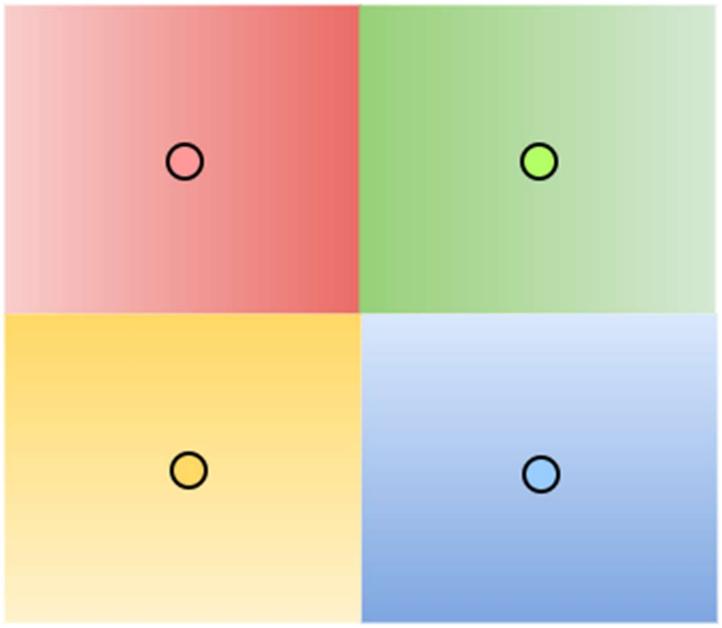
Part 1: K-means Palette Compression (30 pts)

Modern image formats use the RGB spectrum for encoding color information, where each pixel in the image has a Red, Green, and Blue value between 0 and 255. For example, a red pixel is encoded as **(255, 0, 0)**, a white pixel is **(255, 255, 255)**, and a purple pixel is **(255, 0, 255)**. The "palette" of a coloring scheme is defined as the set of all colors you can represent with the scheme. For RGB, the palette consists of 255 * 255 * 255 = **16581375** colors. This is a very large palette! One can imagine scenarios where the images don't need nearly as much color precision. For instance, let's say we are trying to encode the following image:



Clearly, we don’t need 16 million colors to encode this image. One solution to reducing palette size is simply to reduce the number of bits in the encoding. Instead of having our RGB values be in the range [0, 255], we could restrict them to something like [0, 16] which reduces our palette size to just 4096 colors.

It turns out that clustering can also be used for palette compression! If we run k-means (with k = 4) on the image above, we get the following centroids:



Note how the centroids gravitate towards the centers of the different regions. We can then use the colors under the centroids as our compressed palette. The compressed image looks like this:

By using k-means to do palette compression, we can choose the palette size by simply modifying the **k** parameter to the algorithm.

Your job: write a Python program that takes an input image and a **k** value, and then compresses the image using k-means. Use the compress.py file as your starter. The file must accept two inputs from the user: an input image in either jpg or png format, and a **k** value for the k-means algorithm. For example, if I wanted to compress an image called "test.png" using 32 colors, the command I would run is:

```
./compress.py test.png 32
```

Your program must save the compressed image it generates as "result.png".

Here is the general program flow:

1. Read in the input image and the **k** parameter
2. Pick **k** random pixels in the image as your initial centroids
3. Run k-means on the image until either the centroids converge or until you finish 100 iterations.
4. Once the k-means algorithm is finished, note down the colors of the pixels where the centroids happen to be. This is your compressed palette.
5. Go back through the original image. For each pixel, replace its color with the color in the compressed palette that it's closest to. Closeness is defined as the Euclidean distance between the two colors. For example, if your two colors are: (R_1, G_1, B_1) and (R_2, G_2, B_2), the distance between them is $\sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$
6. Save the compressed image as "result.png"

Restrictions:

1. This part does NOT use Hadoop or Amazon EMR. The program you write must work on its own.
2. **You may NOT use an external implementation of k-means. You will have to write your own.**
3. You must use the Pillow library for image manipulation. The library is quite easy to use, and the documentation is here: https://pillow.readthedocs.org/en/3.1.x/

Testing:

1. I have given you two test images to check your code. Note that k-means is inherently non-deterministic so you may get different images each time you run the code. This is perfectly fine.
2. Feel free to test with bigger images. However, running time increases as $O(n^2)$ so be wary of very large images (I wouldn't try anything bigger than 256x256).

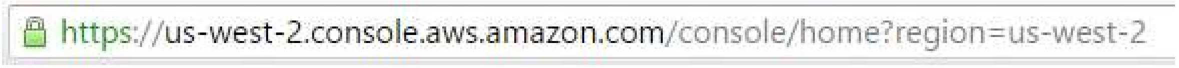
Part 2: MovieLens Cluster Analysis (70 pts)

For this part you will be doing cluster analysis on the MovieLens dataset, which is a dataset of movie ratings.

Setup:

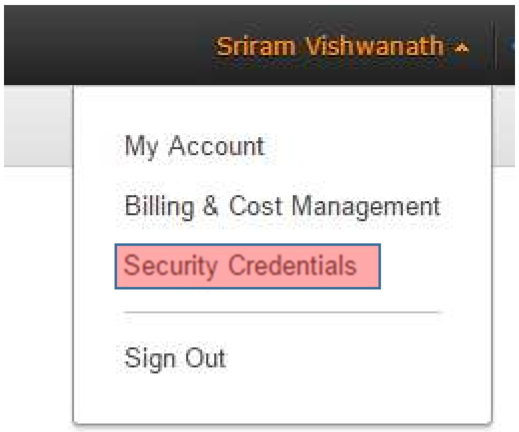
For the previous assignment, we used a raw implementation of Hadoop streaming to run our jobs locally. We also used the Amazon Web Services GUI to run our jobs on Amazon ElasticMapReduce. Both of these were suboptimal. This time around, we will be streamlining both of these tasks by using the **mrjob** Python library. This library takes care of the ugly intricacies of Hadoop streaming and will let you run your code without hassle. Mrjob will also let you run your code on EMR without using the GUI. However, we first need to setup the library:

1. Log onto your AWS console at aws.amazon.com
2. Once you are logged on, check the URL of the site. It will look something like this:

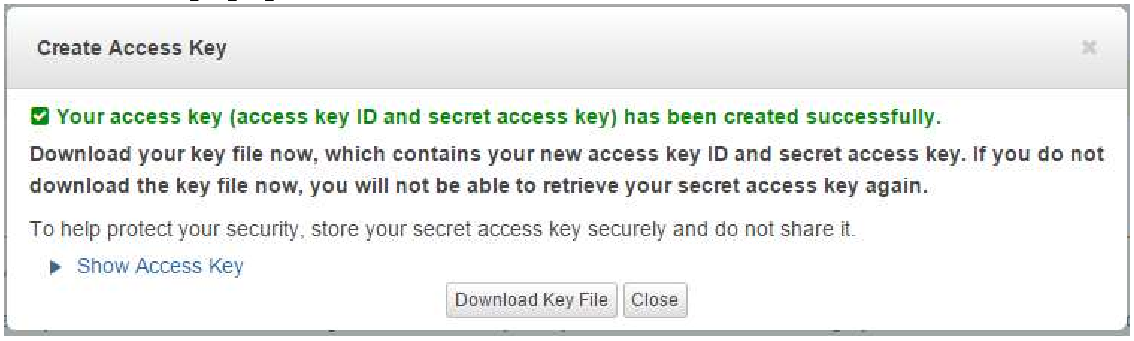


https://us-west-2.console.aws.amazon.com/console/home?region=us-west-2

3. Note down the "region". It may be **us-east-1**, **us-west-1**, or **ur-west-2**.
4. Now, click on your name and go to the "Security Credentials" page:



5. Click on the "Access Keys" menu and then select "Create New Access Key". You will see this window popup:

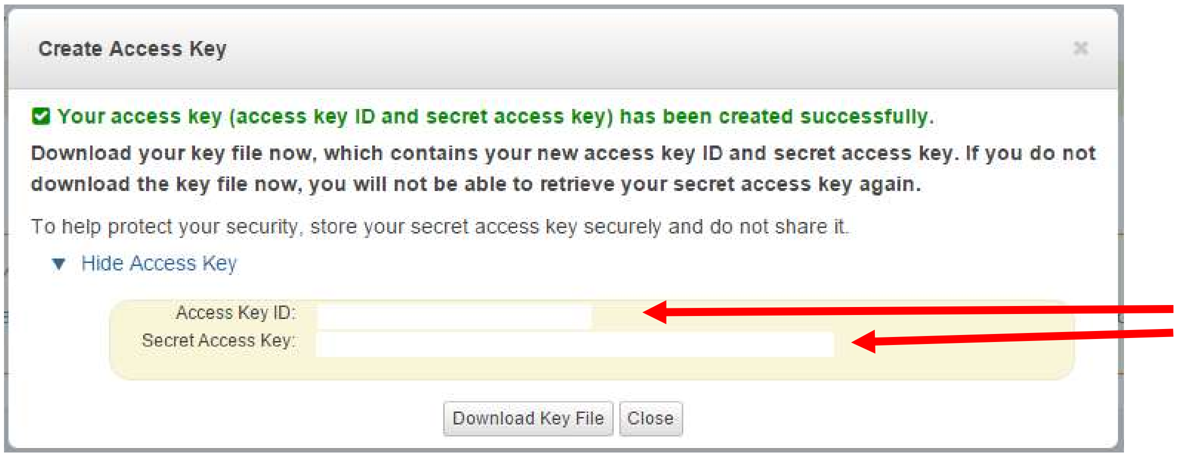


Your access key (access key ID and secret access key) has been created successfully.

Download your key file now, which contains your new access key ID and secret access key. If you do not download the key file now, you will not be able to retrieve your secret access key again.

To help protect your security, store your secret access key securely and do not share it.

6. Click on the "Show Access Key" tab and note down the **Access Key ID** and the **Secret Access Key**. Note that you will NOT be able to see your secret access key again if you close this window!

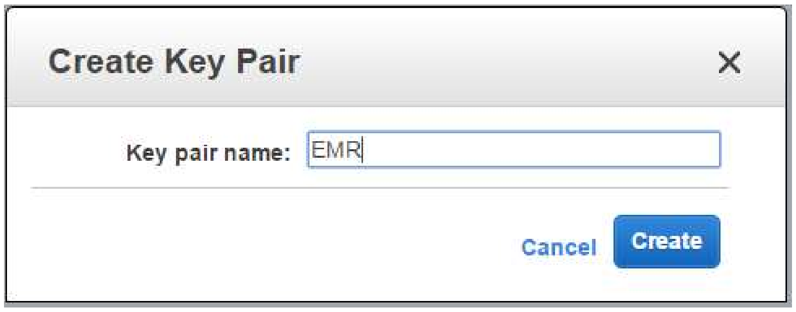


Your access key (access key ID and secret access key) has been created successfully.

Download your key file now, which contains your new access key ID and secret access key. If you do not download the key file now, you will not be able to retrieve your secret access key again.

To help protect your security, store your secret access key securely and do not share it.

7. Now, go to https://console.aws.amazon.com/ec2/home
8. On the lower left, click on "Key Pairs"
9. Click on "Create Key Pair". Name your key "EMR" and press Create. This will download a file called "EMR.pem" onto your machine. Do NOT delete it or move it.



EMR

10. In the lab2/part2 directory, there will be a ".mrjob.conf" file. If you open this file in a text editor, you will see this:

```
runners:
  emr:
    aws_access_key_id: <Enter your access key id>
    aws_secret_access_key: <Enter your secret access key>
    aws_region: <Enter your region>
    ec2_key_pair: EMR
    ec2_key_pair_file: ~/.ssh/EMR.pem
    ssh_tunnel_to_job_tracker: true
    ec2_instance_type: m3.xlarge
    num_ec2_instances: 3
```

11. Fill out your access key id, secret access key, and region that you noted down in the previous steps. Save and close the file.
12. Open a terminal in the lab2 directory. Type in the following commands to move the configuration file and SSH key file to their correct locations:

```
mv ~/Downloads/EMR.pem ~/.ssh/
chmod og-rwx ~/.ssh/EMR.pem
mv .mrjob.conf ~/
```

MrJob:

Familiarize yourself with MrJob: https://pythonhosted.org/mrjob/guides/quickstart.html Don't worry about installing it, that's already done for you. In general, you will be using MrJob to either test your code on your local machine or on EMR. To test your code on your local machine, use the following command (in the lab2 directory):

```
python your_mapreduce_program.py input1.txt input2.txt > output.txt
```

The above command will run the MapReduce program on the "input1.txt" and "input2.txt" files and dump the output to "output.txt". If you just want the output to be dumped onto the console, use this format:

```
python your_mapreduce_program.py input1.txt input2.txt
```

To run your code on EMR, all you have to do is use the following syntax:

```
python your_mapreduce_program.py -r emr input1.txt input2.txt > output.txt
```

And that will take care of the whole process!

I would highly suggest running one of their sample programs (which can be copy-pasted from the documentation) to make sure your local and EMR setup works.

<u>Your Job</u>:

The MovieLens dataset contains user ratings for movies. There are many versions of the dataset, which all differ in size. The smallest set contains 100,000 ratings. The largest set contains 20 million ratings. **Your task is to take the MovieLens dataset and cluster users into similar groups.**

In the lab2/part2 folder there is a folder called **ml-100k**. This folder contains the MovieLens 100k dataset, which is a dataset of 100,000 ratings from 1000 users on 1700 movies. <u>Read the README in ml-100k folder to learn about how the dataset is organized.</u> You will not be able to finish this part unless you understand the README.

Steps you must complete:

1. Remember that k-means needs a way to measure the "distance" between points. In your case, the points are users. You must come up with a way to measure "distance" between users. There are many possible metrics you can use. For example, the distance between two users could be "how many movies do both users rate a 3". Or it could be "how many horror movies do both users like?". The metric you use must incorporate **3 of the following**:
    a. User movie ratings
    b. Movie genres
    c. User ages
    d. User genders
    e. User occupations
    f. User zip codes

    Distance is age group and rating of horror movies

    Note that your distance metric can be as complex as you want. An example is "If both users are in the same age group, the distance is how many movies they both rate a 5. If they are not in the same age group, the distance is how many horror movies user A likes minus how many horror movies user B likes."

    Get creative with your distance metric! Start with a question you want to answer and figure out which metric best approximates a solution to your question.

2. Now, implement the standard k-means algorithm in MapReduce. Note that most implementations of k-means in MapReduce require you to run the same job multiple times. You may also need different MapReduce jobs depending on the stage of the algorithm (for

instance, you may need a separate MapReduce job for the first iteration). MrJob can handle all of this for you. Look at the documentation to figure out how to do it.

3. Test your k-means implementation on the ml-100k dataset to validate your distance metric. Your output can be in any format you want it to be in.

4. Once you are comfortable with your implementation, get the ml-1m dataset, which contains 1 million user ratings. Download it here: http://files.grouplens.org/datasets/movielens/ml-1m.zip

5. Now, run your k-means on this dataset on EMR using MrJob's EMR runner functionality (which I've outlined in the section above).

6. After you're done, **ensure that your clusters are terminated!**

7. Analyze your results. You may do this however you want (by plotting the data, writing a script to see which kinds of users ended up in which clusters). You may have to write more MapReduce jobs. How you do this is up to you and depends on the question you are trying to answer.

8. Prepare a short report (1 page max) answering the following questions:

    a. What was your distance metric?
    b. What question were you trying to answer and why does your metric answer that question?
    c. How did you choose your K? If you tried a variety of K's, which one worked best and why do you think that it did?
    d. How did you analyze your results?
    e. What insights did you gain from the analysis?

9. Turn in your report, your clustering output, and any plots or analysis scripts you wrote by putting them in the part2/results folder.

Note:

You may be tempted to use file i/o in your MapReduce program (i.e. f = open("myfile.txt")).

Do **not** do this! The only way to input files to your MapReduce program is through the command line prompt. If you use Python's file i/o your code will not work on EMR!