Martian Classification Project

The Author

October 31, 2016

1 Introduction

Every object that you can see, and even most of those that you can't, have specific wavelengths of light that they emit and others that they absorb. Everything can be classified in this way. If an object looks red, it is because the object does not absorb red very well, and ends up reflecting red. Fortunately, we can use the spectrum of a rock to make educated guesses about what type of rock it is. This enables us to do a lot of important science on Mars, without actually having to be there.

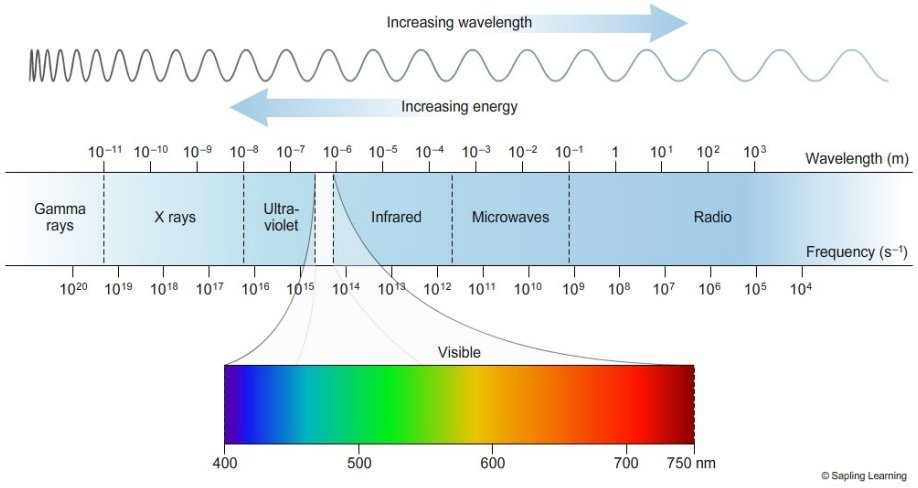

Figure 1: This is an image of the full electromagnetic spectrum. Every object we can perceive with our eyes emits visible light.

2 The Project, in a Nutshell

Partner and group work is encouraged. This is a difficult project with many parts. Copying somebody else's code is NOT ALLOWED, however seeking help and talking through it is important. If you are truly stuck, send me an email or ask me the question the following week.

Each image that is sent down is taken through one of its 13 monochromatic filters. Each image is a matrix of values between zeroes and ones describing the intensity of one wavelength of light. Using several pictures, we can create a spectrum curve and use it to group rocks together that have similar spectrums.

In this project, we will choose three pixels to use as our classes. These will be the spectrums of three different kinds of rocks you may pick out with your eyes. Using these three different spectrums, we will compare them to every other spectrum from each pixel using a mathematical examination called a dot product. Once they have all been compared, we will choose the class that each pixel best belongs too. Figure 2 describes the process through pictures. From a stack of the same pictures, we will extract an ordered list of values (ours will be between 0 and 1.) We will then compare these to our classes, rather than the forest classes they chose, and end with each pixel classified into one of these classes.

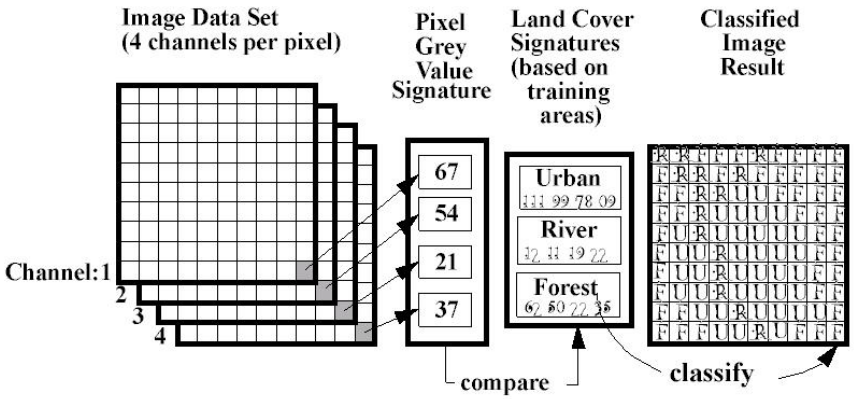

Figure 2: Beginning with a stack of bands, we can extract a list of data from a pixel called a vector. We use this vector to classify all the other pixels.

3 Part 1: Importing the .JPGs

We will now begin the project. I have provided a function call in R that will work if you fill in the functionality. Everywhere that it says '# YOUR CODE HERE' is where you will replace that line and write your own code.

The inputs to the 'classify' function are three coordinates (these are the pixels you will be classifying against) and then a band list. The band list is a list of paths (strings) to the images that are provided with the project. For this section, you will need to fill the pictureList with matrices that are produced by applying the function readJPEG() to each path string.

I have provided test cases to let you know if you are doing it correctly. Do not change the test cases, because if you do the code won't let you proceed to the next sections.

4 Part 2: Create the Picture Stack

At this point, you will have generated a list called pictureList. Each entry of the list is a matrix that is 256 by 256 entries. Each entry is a pixel with a value between 0 and 1. In this step, you will create a single array that has dimensions 256 x 256 x 6. So each 'slice' of the array is one of the images. Remember, you can always look online for help!

5 Part 3: Create output image templates

At this point you will have created an array that is a three dimensional box with dimensions 256 x 256 x 6. Each slice of this array is an image that is 256 x 256, so there are six slices. Now we need to create placeholders where we will place all of the calculations and data that we will generate. you will be creating 9 different variables in this section. These are as follows:

- dims: list of dimensions of one slice of the pictureStack.
- class1: matrix with dimensions of 'dims' (the variable above). Each entry is a zero. This will be where we keep our data for the first class
- class2: matrix with dimensions of dims. Each entry is a zero. This will be where we keep our data for the second class.
- class3: matrix with dimensions of dims. Each entry is a zero. This will be where we keep our data for the third class.
- red: matrix for red color values. Each entry is zero. This will be stacked so that we have a three dimensional array to output color instead of gray. This has the dimensions of dims.
- blue: matrix for blue color values. Each entry is zero. This will be stacked with the red and green matrices. This has dimensions of dims.
- green: matrix for green color values. Each entry is zero. This will be stacked with the red and blue matrices. This has dimensions of dims.

- rgbList: this is a list of the red, blue and green matrices.
- classifiedImage: This is a stack that has dimensions of 256 x 256 x 3 similar to part 1. There is 3 because each one will be a slice for red, blue or green. To make a full color image, we need a value for red, green and blue bands.

6 Part 4: Classification

Now we have all the things we need. We will now run our classification algorithm on each pixel. To do so, we need to know what a dot product is. The dot product is defined by this equation.

$$\vec{a} \bullet \vec{b} = \sum_{i=1}^{n} a_i * b_i = a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + ... + a_n * b_n \tag{1}$$

Every entry of a vector $\vec{a}$ is multiplied by the same entry in vector $\vec{b}$ and then added together. Geometrically this proves to be useful because it gives us a measure of the angle between vectors.

$$\vec{a} \bullet \vec{b} = ||a|| * ||b|| * Cos(\theta) \tag{2}$$

Where the $||a||$ is called the magnitude of the vector a:

$$||a|| = \sqrt{a_1^2 + a_2^2 + a_3^2 + ... + a_n^2} \tag{3}$$

Our classification parameter will be the angle between the vectors being examined. So each entry of each class matrix will be the dot product between respective class vector and the vector for that pixel. The code you write will be the following:

$$Cos(\theta) = \frac{\vec{a} \bullet \vec{b}}{||a|| * ||b||} \tag{4}$$

Iterate over every pixel in one of the class matrices and compare the vectors, then enter that value into the respective entry in that class matrix. You will end up writing three loops, one for each class.

7 Part 5: Create the final image

Now we have three matrices. I have given you the coordinates that define red, blue and green. In this section, you will iterate over each pixel of the classifiedImage. For every pixel, you will select the maximum value for the respective pixel from class1, class2 and class3.

If class1 has the maximum value, fill that pixel in with the red vector. If class 2 has the maximum value, fill it in with the green vector. If class 3 has the maximum value, fill it in with the blue vector. After doing this correctly, you will have created your final classified image, and all you have to do is save the image to view it.

To get the final product, run the following code:

```
classifiedImage <- classify(coord1, coord2, coord3, bandList);
writeJPEG(classifiedImage, 'classifiedImage.jpg');
```

Your file will be named 'classifiedImage.jpg.

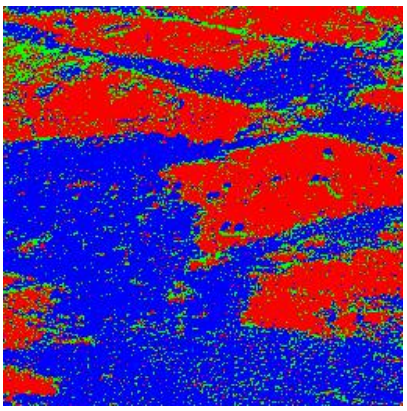

Figure 3: This will be your end result

If you have any questions feel free to contact me at SamuelEstrella226@gmail.com.