Computational project 1

Goga Jincharadze

October 2024

1 Image compression with K-means clustering

If we have an image that is too big in terms of size (the amount of space it takes up in storage), we may want to make it smaller, compress it, while maintaining as much quality as possible.

This can be done in many ways and one of many is K-means clustering. We are aware that an image is just a matrix of RGB values. K-means clustering goes over those RGB values or equivalently, pixels and chooses a few of them as the so called centroids. These centroids are colors of a single pixel in the image. After choosing those centroids, we calculate the "distance" between every other pixel and the centroids. When we get the closest centroid to the given pixel, we set it's RGB value to that of centroid.

In simple terms, we just reduce the number of individual colors, if we choose 16 centroids, our resulting image will only have 16 different colors. When the image matrix has most of it's values the same, it's size is reduced.

1.1 Usage manual

0. (Important!) Make sure to have every required library imported.

1. Copy the uploaded code, "compress.py", into your IDE.

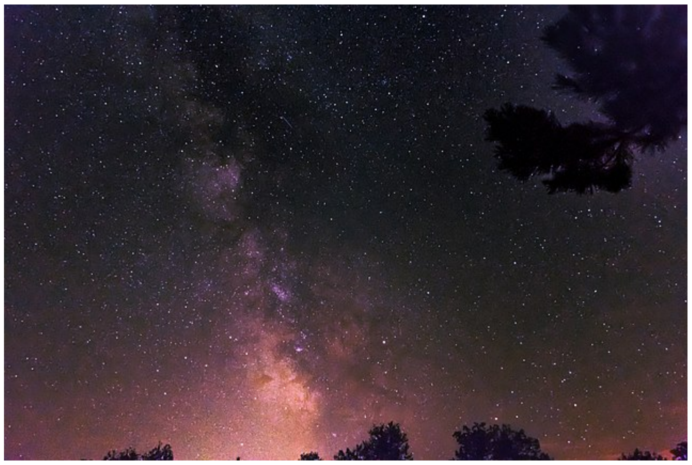
2. Create a folder to store the images that you want to compress and if you want, you can create a separate folder to store the compressed images or you can store them in the same folder.

3.Set the number of colors you want your image to have, by adjusting the variable "k". The lower it is, the smaller your image will be, but it's quality will also be worse.

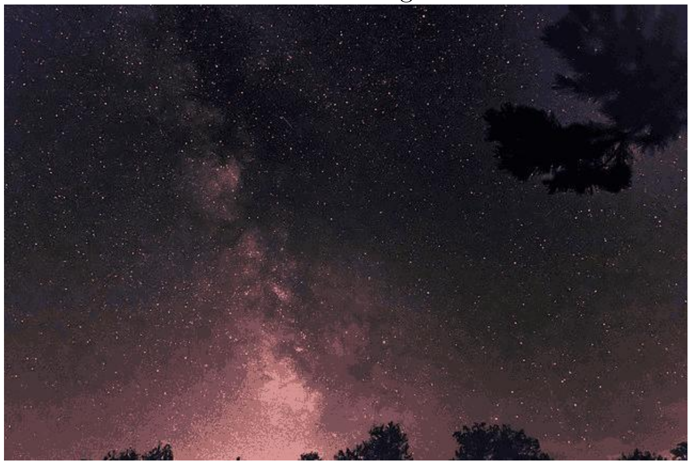
4. After you've set everything up correctly, run the code and wait (if the image is high resolution, it may take a few minutes because k-means is inefficient in it's nature for compressing images). After some time, you should see the

compressed image appear in the path you've specified and also, the sizes of the original and compressed images shall be printed on the terminal for comparison.

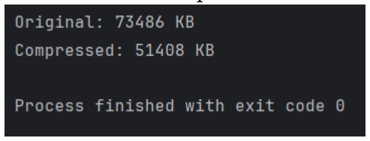
1.2 Example of a compressed image



This is the original



This is the compressed result



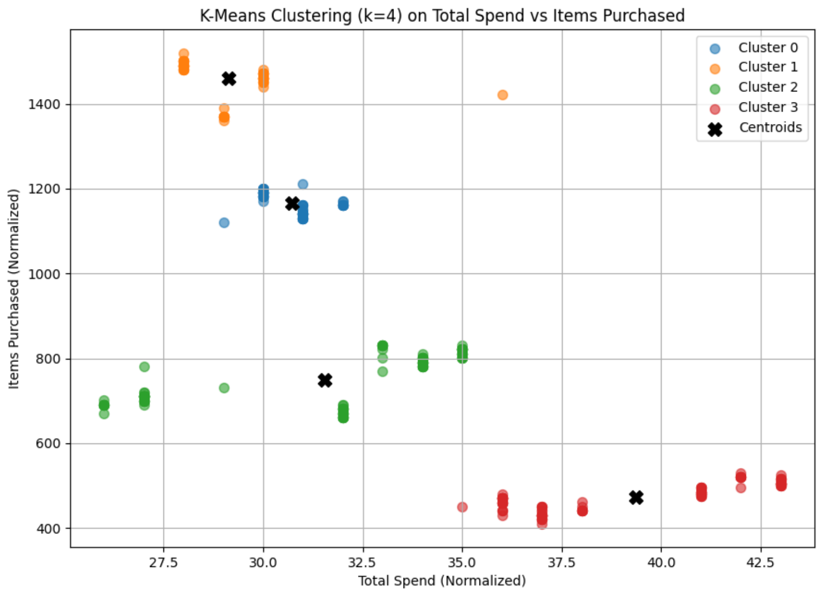
```
Original: 73486 KB
Compressed: 51408 KB

Process finished with exit code 0
```

And the sizes of the images

2 Customer segmentation in marketing with K-means clustering

When you're a big company with lot's of customers, you have to know them well, in order to offer them a product that they're most likely to buy. Each person is somewhat different (actually if everyone can be clustered then there is no person with an unique personality, can this algorithm be considered a mathematical proof of that? Anyways let's pretend it's not like that). Using K-means clustering, we can group them by different parameters, such as age, last time they purchased something, what they purchase, how much they spend, their personality, etc.

When we possess such information about patterns in our customers, we can extort money from them more efficiently by offering each individual something that they simply can't resist.



Here I have clustered these people into four different groups and mapped them by the relation between how much they spend and what they purchase. Like this, we will know which group spends the most and on what exactly.

2.1 Usage manual

0. (Important!) Make sure to have every required library imported.

1. Copy "Psychographic_segmentation.py" and "k_means_psychographic.py" into your IDE.

2. Create a folder in which you will have your data in .csv format. You can procure your own data on kaggle or any other similar website by searching customer behavior.

3. In "Psychographic_segmentation.py", specify the path to your .csv data file. The file must be normalized, meaning it shouldn't contain useless or non-numeric columns or else the algorithm will fail or give wrong results as k-means is not supposed to handle those. My code automatically drops any non-numeric data and useless ID columns, but you may have to modify it in case you have a customer behavior data that is not written by conventional standards.

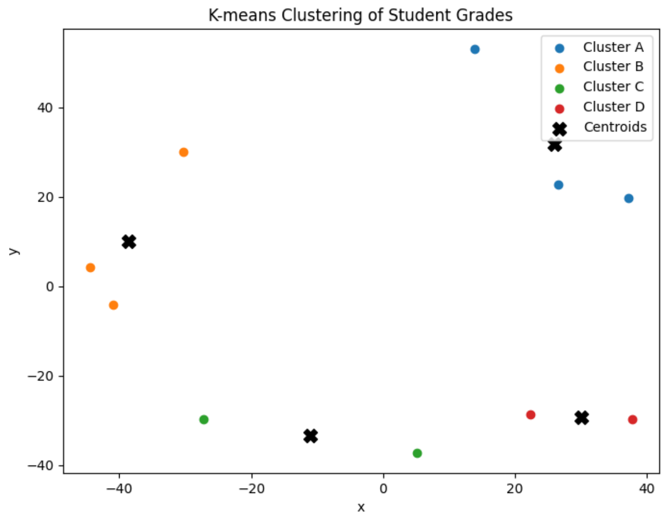
4. Run the "Psychographic_segmentation.py" and wait for the diagram to appear.

3 Ranking students for fellowship applications using k-means

This one has a problem, it does not cluster correctly, the points remain too scattered sometimes because k-means wasn't even meant to work on multidimensional data.

"The "curse of dimensionality" refers to various challenges that arise when working with high-dimensional data. As the number of dimensions increases, the volume of the space increases exponentially, making it difficult to sample data and leading to issues such as overfitting in models and increased computational complexity. This term was coined by mathematician Richard E. Bellman."

Despite this, here is the graphed clusters for 10 students in frobenious norm.



Instead of randomly choosing the four centroids, I used k-means++. Basically, I choose the first centroid randomly and then I choose the rest with probability proportional to the distance, making sure the centroids aren't close to each other to make this a bit better. Despite this, the clustering are still sub-optimal for high number of students.

The code generates an n number of 4x5 matrices, for which, each entry ranges from 51 to 100, this is done automatically at execution.

3.1 Usage manual

0. (Important!) Make sure to have every required library imported.

1. Copy the uploaded "ranking_students.py" code into your IDE.

2. Adjust how many students you want to have.

3. Adjust the matrix norm in which you want to calculate the distances, by default, it's set to frobenious norm.
4. Run the code and wait for the graph to show up.

4 Where I have used AI

I have written all k-means algorithms by myself, however, I have used ChatGPT to generate the visualization code all the times, as I am not adept with matplot library. I have also used it to create the function "generate_data" in the ranking students problem to randomly create n number of 4x5 matrices with their entries ranging from 51 to 100.