k-means Clustering

Amirou Sanoussy

March 2024

1 Introduction

When approaching this problem, I was confused about how K-means clustering; in particular, I lacked intuition on how Lloyd's algorithm worked in conjunction with k-means clustering. In response, I began by watching many YouTube videos and articles on GeeksforGeeks to figure out how my program should function in the grand scheme of things.

2 $find - cluster$

When approaching this function, I had a hard time because this function was a helper function to the $calc - kmeans - next()$. So, I first looked up how step 2 of Lloyld's algorithm was supposed to work online. After some time, I gained a better intuition of how the function was supposed to work and began writing out an outline of how to approach the problem in a notebook. I began by initializing the cluster to 0, which is the variable responsible for eventually holding the index of the nearest cluster to the given point. I also initialized $min - dist - sq$ to DBL-MAX to help compare distance later as we iterate through all cluster centers.

The function then enters a loop, iterating over each cluster center. It calculates the squared distance between the point and the cluster center for each center. This is done by the $vec - dist - sq$ function that calculates the distance between two points in the dim dimension. Following that is the loop where if the calculated squared distance for the current cluster is less than the current $min - dist - sq$, the function updates the later variable with the new distance and sets $cluster$ to the current index, thus indicating that in this iteration, is currently nearest to the point.

After iterating through all cluster centers, the function returns the index of the nearest cluster center.

3 $calc - kmeans - next()$

I first initialize two arrays, $sums[]$ and $counts[]$, to store the sum of the points coordinate for each cluster and the count points in each cluster. Where I then

zero them out at the beginning.

The function then iterates through each point in the dataset. For each moment, it finds the nearest cluster center using the helper function $find-cluster$, then increments the count for that cluster and adds the points coordinated to the corresponding clusters' corresponding sums. Essentially, it groups the points by their nearest cluster and prepares for calculating the new centers.

After all the points are processed, I made the function iterate through each cluster center. If a cluster has no points assigned to it, an error is printed, and the program exits. However, if there are points, the function calculates the new position of each cluster center by dividing the sum of the point coordinates in that cluster by the number of points assigned to it. This mean calculation is done for each dimension of the cluster center, and the results are stored in $kmeans - nexr[]$.

4 Testing

```
(base) afroamir@matrix:~/cmda3634/CSA05$ gcc -o kmeans_main kmeans_main.c kmeans.c vec.c
(base) afroamir@matrix:~/cmda3634/CSA05$ time cat mnist_test.dat | ./kmeans_main 25 55 > mnist_test_22_25.dat

real    0m37.766s
user    0m37.725s
sys 0m0.057s
```

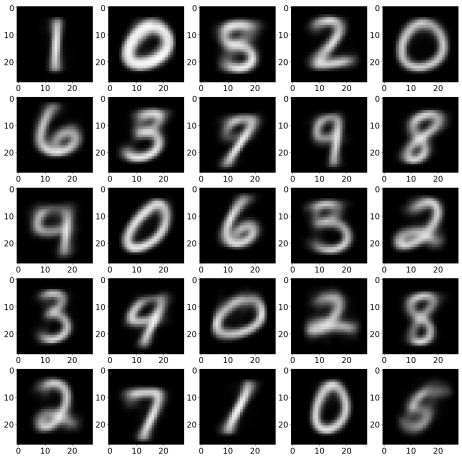

Figure 1: Visualizing kmeans on the MNIST Test Set

The integration of both the python script and the *kmeans.c* program to solve the color space clustering problem for images uses the strength of both program types to create an efficient solution. With Python, using libraries like PIL and NumPy is great at loading images and converting them into a format that is right for manipulation. Where this process involves reading the image in and flattening the color space from a 2D image into a 1D array.

Figure 2: Visualizing kmeans on the Bike Image

Now with using the *kmeans.c* c script in C, is very efficient for running computationally intensive images of varying sizes. This is because of C ability to run low level operations such as directly manipulating memory.

The reason why you would not want to use only C or Python for this problem is because for C in particular, it lack high level functionalities available in Python, especially for tasks like image loading/saving and rapid prototyping. In addition, the development in C might be slower since you would need to do everything manually, and the debugging would likely take up most of the time due to C lower-level language characteristics.

Now with Python, even using NumPy might not achieve the same level of performance as C for the k-means computation, especially on large datasets.