Siddharth Patel
CS 484 – 001
Date: 11/21/2021

Part 1:

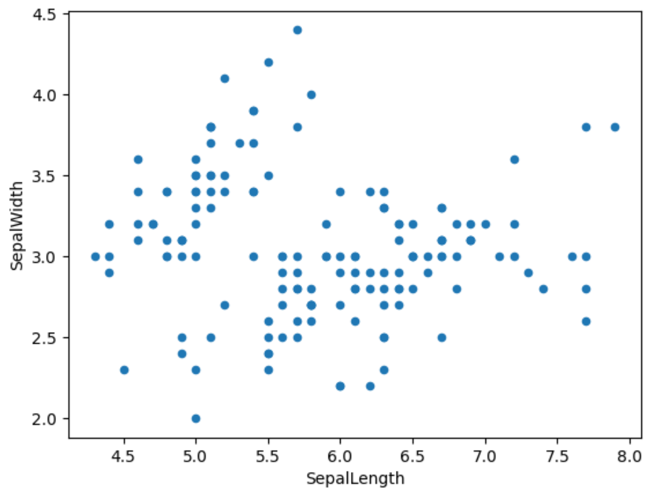
Overview

My registered name on miner is spatel53. My rank for homework 4, part 1 is 19 and the score is 95%.

K-means clustering is one of the simplest yet popular unsupervised machine learning algorithms. An unsupervised algorithm inferences from datasets using only input vectors without referring to known, or labelled, outcomes. The objective of K-means clustering is to group similar data points together by discovering the underlying patterns. To achieve this objective, K-means looks for a fixed number of clusters in a dataset also referred as the ‘k’. A cluster refers to a collection of data points aggregated together because of certain similarities. Your k-means algorithm starts off from a centroid. A centroid is random location representing the center of the cluster.

Preprocessing

I started my Iris dataset by storing the input data just like previous assignments. Using Pandas library’s read_csv() function, I loaded all the data into a data frame. After preprocessing the data, I generated a scatter plot with x = sepal length in cm and y = sepal width in cm. This would allow me to spot the possible clusters prior to k-means clustering algorithm implementation. As you can see, we can expect there to be at least 2 clusters, the one on the left and the one on the right. Depending upon the algorithm, the left clusters can be possibly broken up into pieces which you will see upon k-means algorithm implementation.

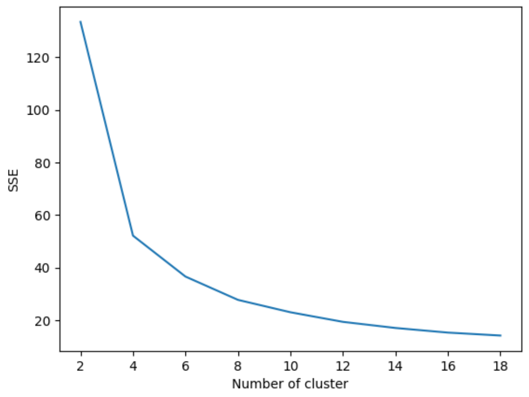


Implementation

To implement k-means algorithm from scratch, I started off by creating three random centroids based on the specification mentioning that you must have 3 sets of clusters. As a result, the number of centroids chosen were also 3. To make centroid selection fair, I used the randomizer, “np. random. choice()” which randomly picked three rows from the input data resulting in a 3x4 centroids. I even computed an initial distance from the input data to the three centroids using the cdist function with the ‘cosine’ metric from scipy.spatial.distance.

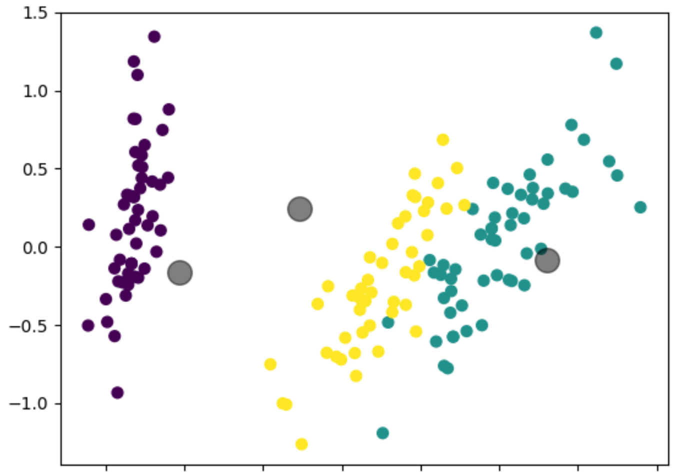
Once the centroids have been generated, I decided to calculate minimum distance from each point to the centroids. To do that, I ran a for loop iterating through all the rows in the dataset (150 rows) and used the mean between the input point and the centroids from preprocessing to basically shift the centroid towards their particular cluster.

These newly found centroids will also result in new distance computation between the centroids and the data points. To predict the output, I simply find the smallest value in each of the 150 rows and get its index.

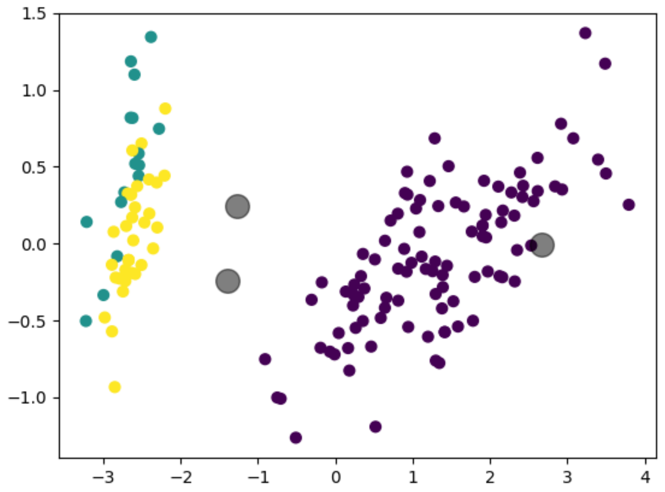


To see what the optimal number of clusters against the sum of squared error would be, I generated an exponential graph to get an idea. As shown in the graph, the optimal number of clusters should be around 6 with an SSE error of approximately 40.

Graph



Once the algorithm has been implemented, I decided to graph the clusters using a scatter plot. I used matplotlib. pyplot as plt library to observe the cluster. I also used sklearn. decomposition's PCA for dimensionality reduction. Since we have 4 columns meaning that originally our data is in 4-Dimension, it is necessary to reduce our data appropriately to 2D which is where the principal component analysis reduction comes in handy. The graph I came up with can be seen below. As you can see, I have shown the final clusters along with their centroids.



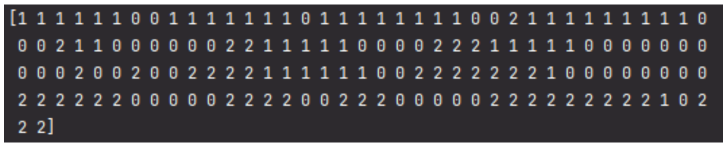
This helped me reassure that my k-means algorithm was accurate enough for starters. Of course, there are outliers among these data, but that is to be expected in k-means clustering as it is sensitive to noise.

I ran my code multiple times and came to conclusion that out of 12 runs, 11 times the algorithm results in accurate clustering, but 1 time it fails as shown. This could be because our centroids are chosen too close to each other, and the algorithm has trouble fitting each data point into a particular algorithm. I chose to run the k-means clustering 150 times since that is the length of the input data. This will ensure that our data points will be able to compute distance with the 3 clusters at least 1 time. I saw an improvement in accuracy with higher runs.

I also computed the distance metric using both cosine similarity and Euclidean along with many others, but the cosine similarity led to higher accuracy out of all the metrics.

Output

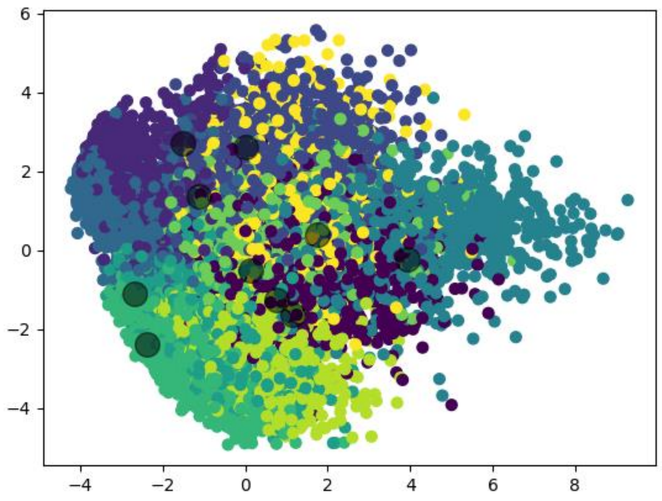
Finally, the computed output which range from 0 to 2 was stored in a list. Therefore, I created a new text file to store the output. I iterated through the list and added 1 to each value computed output and appended it to the newly generated text file since the expected output must be between 1 and 3 inclusive. Below, I have shown the output for Iris Dataset without the +1 incrementation.



```
[1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 2 1 1 1 1 1 1 1 1 1 0
 0 0 2 1 1 0 0 0 0 0 0 2 2 1 1 1 1 1 0 0 0 0 2 2 2 1 1 1 1 1 0 0 0 0 0 0 0
 0 0 0 2 0 0 2 0 0 2 2 2 2 1 1 1 1 1 1 0 0 2 2 2 2 2 2 2 1 0 0 0 0 0 0 0 0
 2 2 2 2 2 2 0 0 0 0 0 2 2 2 2 0 0 2 2 2 0 0 0 0 0 2 2 2 2 2 2 2 2 2 1 0 2
 2 2]
```

Part 2:

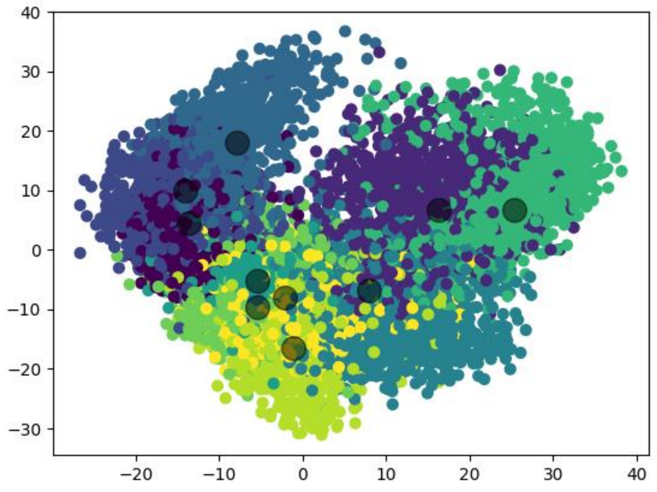
Overview:



My registered name on miner is spatel53. My rank for homework 4, part 2 is 52 at the time of writing this report and the score is 72%.

For part 2, we still have to use k-means algorithm from part 1, but our dataset this time is a 28 x 28-pixel handwritten number which has been scaled to a single vector of 784 x 1 pixel for each number ranging from 1 to 10. The problem is, we are given 10,000 handwritten numbers which means our data size is 10,000 x 784. As we learned in class, k-means clustering is sensitive to noise. A significant portion of the 784 pixels per digit is 0. We have two problems in our dataset: 1) lots of noise 2) An extremely large dataset.

Preprocessing



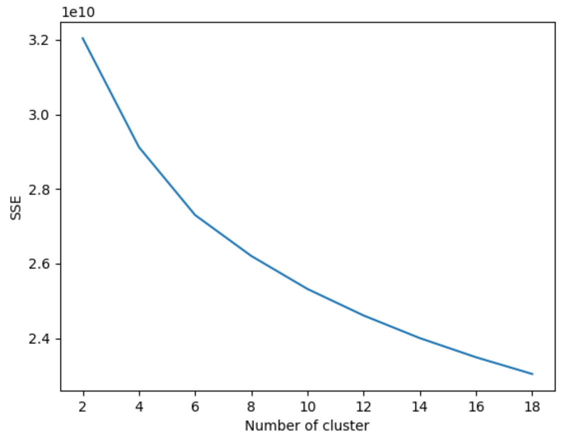
I would like to mention that prior to preprocessing, I ran my initial k-means algorithm like part 1 and ended up with 55% accuracy, but the algorithm took 30-40 minutes to run. Obviously, the accuracy was not as good.

My first approach to fixing the problem was scaling the data using the StandardScaler () and MinMaxScaler(). After computing and checking the accuracy on miner, I determined that there was not huge difference between the methods. Moreover, it led to poor accuracy than the original model meaning < 55%.

I then found out that what I must perform is dimensionality reduction, so I initially chose the PCA algorithm to reduce the number of features in the dataset. Again, this led to poor accuracy than the original algorithm. To see for myself, I generated a scatter plot shown.

I then did research on different types of dimensionality reduction and came across sklearn. manifold Isomap dimensionality reduction. Isomap is an algorithm which aims to recover full low-dimensional representation of a non-linear manifold considering that the manifold is smooth. With Isomap as the dimensionality reduction technique, I was able to achieve an accuracy of 70%.

To predict what would be the estimated/ appropriate number of clusters for image dataset, I computed the SSE error graph similar to part 1. Note that we have chosen the number of clusters to be 10. Personally, I think that we would have had improvement in accuracy if the number of clusters were either 6 or 8, since if you look closely, the point 6 leads to a sharp point on the graph. Ofcourse, that would mean that our output/ handwritten numbers posibility would be shortened from 0 to 6 leading to poor accuracy on miner.



Implementation

Once the preprocessing was completed, the k-means clustering algorithm was the same from part 1, so there are no changes there. Image dataset being a larger, more dense, populated with noise dataset, allowed me to test my k-means algorithm and made me realize that apart from a few tweaks, there was nothing wrong with my k-means algorithm particular. It was mainly the preprocessing that needed to be improved upon.

The number of times I decided to run my k-means clustering algorithm is equal to the length of the dataset which is 10,000 times. This was just a personal decision to ensure each row will at least get 1 chance to distance its data points in that row with the 10 clusters.

The graphs were also mainly associated with preprocessing which has been shown in the preprocessing section above.

Conclusion

Since my complicated dataset represented handwritten digits, this algorithm can be helpful in image detection given an improvement in accuracy. If we can detect and differentiate between similar handwritings, then such an algorithm can be used to prevent forgery in legal documents.