Niha Imam

04/15/2020

**CS 484 HW 3 Part 1 – Iris Clustering**

**CS 484 HW 3 Part 2 – Image Clustering**

**Username:** iateyourcookie

**HW 3 Part 1**

Rank: 97

V-Score: 0.72

**HW 3 Part 2**

Rank: 172

V-Score: 0.48

**Libraries Utilized:**

Random, NumPy, SciPy. Spatial, Pandas, MatPlotlib.pyplot, Sklearn.Decomposition, Time

**How to run:**

Make sure the train files are both in the same folder and the kmeans and the mnist files are also in the same folder. Make sure to change the input file names in both python files

The graph that will show the different k vs performance vectors will be plotted in the program at the end of mnist

The run time is ~30 minutes ☹

**Approach & Methodology for both parts**

Clustering is an unsupervised learning method which allows us to set of objects based on similar characteristics. One of the more common and famous methods of clustering is the K-means algorithm. The goal of this project was to implement our own kmeans algorithm and then implement it on the MNIST handwritten number dataset.

For this project I spent most of my trying to perfect my kmeans algorithm trying to perfect it because the second part of the project depended on it.

For part one I read in the data as a list of n dimensional data points. For the iris data set it became a list of 4-dimensional data points. I then set the value of k to my desired clusters and then started implementing the kmeans. For kmeans I primarily used the pseudo code in Professor Sean Luke’s’ class notes because the way he had explained it was much clearer to me. For my kmeans I start by setting up a max iteration, because k means works by finding the median of cluster and them moving points so I would prefer to find the optimal solution in at least 500 iterations. I then randomly set up k clusters from the data points provided. Initially I was setting one cluster at the first data point, the second at the middle data point and the third at the last data point but then I realized that would add bias to the kmeans so I resorted to randomly selecting k data points as initial clusters or centroids. Then I looped through all the data and for every data point I would compute the distance between the datapoint and the clusters and assign the data point to the closest clusters. For distance I used the Euclidean distance just because I used the same during the knn implementation too. After assigning all the data labels or new labels I would find the new median of the data based on every cluster and make them the new centroids. The last two steps would be repeated until the previous cluster assignment is the same as the current cluster assignment or if we have maxed out the number of iterations allowed. After that to check whether my clusters were being predicted properly, I used the sum or squared errors. but after reading up on I found that it may not be the best predictor, so I had to find another way to check my solutions. To check I imported the labeled iris data set and used the v score using my predicted labels and the correct labels which gave me a v-score of ~0.84.

After I was satisfied with my kmeans I had to move on the image clustering problem. The MNIST data set is a different beast than the iris dataset because it is dealing with images and the data isn’t as clearly separable. So, started off by reading in the data as a csv file. After looking through the data I realized that most of the dimensions was full of zeros so they wouldn’t be useful in computing the kmeans. I had to get rid of some of the dimensions. So, I Used PCA with the given number of dimensions to see which dimensions would be useful for the kmeans calculate. The plot showed that only about 40 dimensions should be saved. The I used PCA again but this time with 40 components so that I could get rid of some dimensions. I then fitted and transformed the data and ran my kmeans on the output. Then I had a slight problem. since clustering is an unsupervised learning method it’s hard to establish a good performance metric. But I needed something, so I went back to my kmeans implementation and implemented the sum or squared errors. so now when you call kmeans the return[0] will be the labels and the return[1] will be the calculated sse for the particular k.