Computation Intelligence – First project documentation

Arya Araban – 9612762832

Part 1: K-Means

I used the Mini Batch K-Means clustering algorithm for the first part of the project. The advantage of this algorithm over the original K-Means is that it reduces the computational cost by not using the entire dataset in each iteration, but a subsample of a fixed size. This approach can significantly reduce the time required for the algorithm to find convergence with only a small cost in quality.

The main parameter which is modifiable for this algorithm is K (the number of clusters) In this section I'll evaluate the results of my cluster.py code for different K values(using the results of the "print_metrics" function that I wrote)

KMeans(10) → rand index: 35.33%, purity: 55.79%

KMeans(16) → rand index: 42.26%, purity: 61.69%

KMeans(64) → rand index: 65.86%, purity: 81.04%

KMeans(128) → rand index: 74.69%, purity: 86.95%

KMeans(256) → rand index: 80.79%, purity: 90.52%

As can be seen, if we make K have a value of 256, the algorithm performs with less than a 10% error (note that the metrics are found by evaluating on the test set).

A question to ponder is that why do we need 256 clusters when there are only 10 digits. This is because there can be multiple ways to write a particular number. The orientation and style of writing a number can be different and thus the algorithm views them as drastically different images. Hence we need more than 1 cluster to represent the images of a particular number.

Part 2: Standard Neural Network (MLP)

In the code that I implemented for the neural network, I wrote the class NN which has two parameters. The first one being the number of hidden layers inside the neural network(however, here I'll use only two hidden layers as a default), and the second one being the number of hidden units that the hidden layers have.

It's also worth noting that I used the Stochastic Gradient Descent (sgd) optimization algorithm for the network, and that each of the layers uses the Relu activation function (which is also the default activation function that Sci-Kit uses).

NN(2, 50) → score: 95.53%

NN(2, 100) → score: 96.5%

NN(2, 150) → score: 96.96%

NN(2, 200) → score: 97.05%

So, it can be seen that by using neural networks we've achieved over 97% accuracy (less than 3% error), which is even higher than what we were able to achieve by using K-Means clustering.

Part 3: MLP vs K-Means

For the task of classifying the value of a picture which contains a digit, If we use standard Neural Networks, we generally need to have the label for each of the images, whereas when we make use of clustering algorithms such as K-means, we don't have the labels of the training set.

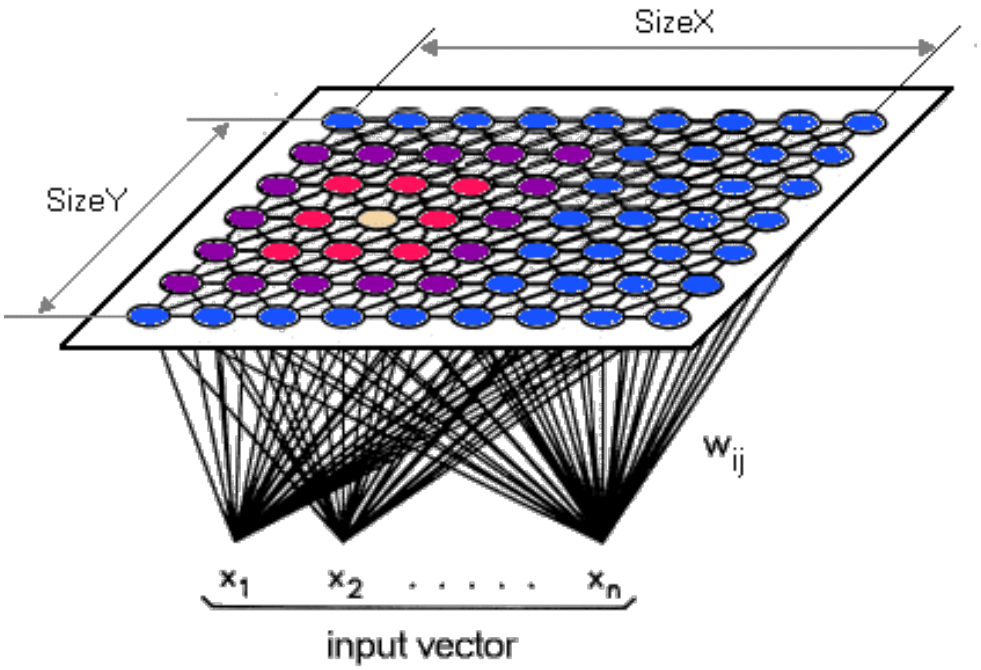
With that in mind, there are structures for neural networks which allow us to perform clustering tasks in an unsupervised manner, one of the most used MLP structures for unsupervised learning are known as **Self Organizing Maps** (AKA Kohonen Maps). SOMs reduces data dimensions and displays similarities among data, without having any need for labels.

Part 4: Using Self Organizing maps for Unsupervised Learning

SOMs allow reducing the dimensionality of multivariate data to low-dimensional spaces, which is usually 2 dimensions.

Observations are assembled in nodes of similar observations. Then nodes are spread on a 2-dimensional map with similar nodes clustered next to one another. Each node contains information on the number of observations it carries, and on representative values of the different input variables for these observations.

Self-Organizing Maps consist of two layers of units: A one dimensional input layer and a two dimensional competitive layer, organized as a 2D grid of units. This layer can neither be called hidden nor output layer.



In the SOM that I implemented, the sizeX is 16, and the sizeY is 10, Giving a total of 16*10 = 160 total outcomes. After the network is trained, when we make a prediction on an input image, the predicted output will be the closest to one unit out of these 160 neurons.

Part 5: Results of using self organized maps

As said in the previous section, I've used constant values for "sizeX" and "sizeY", however I've tuned the values of epochs and the learning rate over different values. The results for the accuracy and rand-index that were resulted are as follows:

SOM(epochs=20, learning rate =0.5) → accuracy: 70.78% rand index: 50.01%

SOM(epochs=30, learning rate =0.4) → accuracy: 72.53% rand index: 52.69%

SOM(epochs=35, learning rate =0.3) → accuracy: 74.12% rand index: 55.43%

SOM(epochs=60, learning rate =0.2) → accuracy: 74.52% rand index: 55.31%

Part 6: Conclusion

In this project we saw how we can use neural networks in order to perform clustering, without ever needing to have the labels at hand. Although the results we achieved by using SOMs seem a bit weaker than what we achieved using K-Means, By tuning the many hyper-parameters that we have, we may be able to achieve even better results than what we we were previously able to.

Although since tuning these hyper-parameters to near-optimal values may take a lot of time, it might be a better choice to use the standard clustering algorithms instead!