CSE 474/574: Introduction to Machine Learning (Fall 2019) Cluster Analysis using Unsupervised Learning like KMeans and Gaussian Mixture Models on Fashion-MNIST

Sonali Naidu
Department of Computer Science
University at Buffalo
Buffalo, NY – 14226
sonaliye@buffalo.edu

Abstract

In this project we have worked on implementing cluster analysis using unsupervised learning on Fashion MNIST dataset. Clustering is one of the unsupervised machine learning technique where it deals with finding structure in collection of unlabeled data. The task here is to cluster images and identify it as one of many clusters. We have used three different clustering techniques i.e., KMeans algorithm, Autoencoder based KMeans and Autoencoder based Gaussian Mixture Models. The dataset used to train the model is Fashion- MNIST. We have 10 classes into which each of the image from dataset gets clustered.

Introduction

Unsupervised Learning:

In unsupervised learning, labels are not given to the learning algorithm thus leaving it on its own to find the structure. The goal here is to identify groups of similar examples within the data, which is called clustering, or to determine how the data is distributed in the space, which is known as the density estimation.

Clustering:

Clustering is one of the most important unsupervised learning problem which is defined as the process of organizing objects into groups whose members are similar in some way. A cluster is a collection of objects which are similar and dissimilar to objects belonging to other clusters. In this project we look into two clustering algorithms namely KMeans and Gaussian Mixture Model.

K-Means:

K-Means algorithm identifies the k number of centroids and assigns each datapoint to the nearest cluster, while keeping the cluster centroids as small as possible. The means in Kmeans refers to data averaging, i.e., finding the centroids. The centroid represents the center of the cluster.

The objective function od KMean is given by

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

Issues with KMeans-

KMeans doesn't work when clusters are not round shaped because it uses distance function to measure the distance from cluster center.

Gaussian Mixture Model:

Gaussian Mixture Model is a probabilistic approach to addressing the clustering problems. In this we describe each cluster by its centroid(mean0, covariance, and the size of the cluster(weight). In this, rather than identifying clusters by nearest centroids, we fit a set of k gaussians to the data. We estimate the gaussian distribution parameters such as mean, variance and weight of each cluster. After parameters learning for each data point we calculate the probabilities of it belonging to each of the clusters.

Mathematically, Gaussian Mixture Model is define as mixture of K gaussian distributions that means its weighted average of k gaussian distributions. We can write the data distribution as

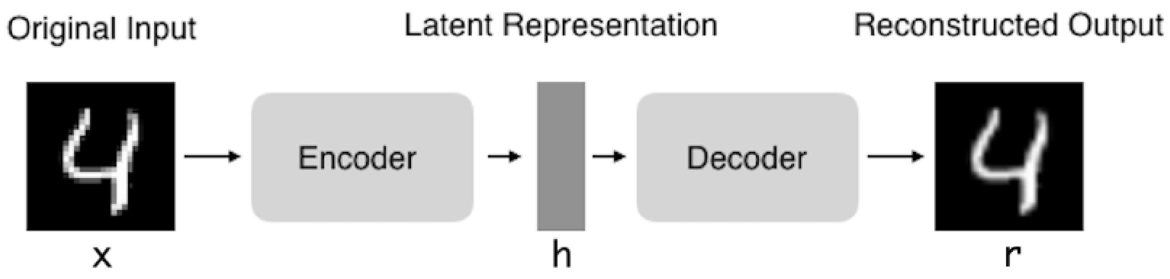
$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where N(x|mu_k,sigma_k) represents cluster in data with mean mu_k and co variance epsilon_k and weight pi_k. After learning these parameters we have learnt the k gaussian distribution from the data distribution that k gaussian distributions are clusters.

Auto Encoders:

Autoencoders are neural networks that aims to copy their inputs to their outputs by compressing the input into a **latent-space representation**, and then reconstructing the output from this representation. This kind of network is composed of two parts :

1. **Encoder:** This part of the network that compresses the input into a latent-space representation. It can be represented by an encoding function *h=f(x)*.
2. **Decoder:** This part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function *r=g(h)*.



Architecture of an Autoencoder

The autoencoder as a whole can thus be described by the function g(f(x)) = r where you want r as close as the original input x.

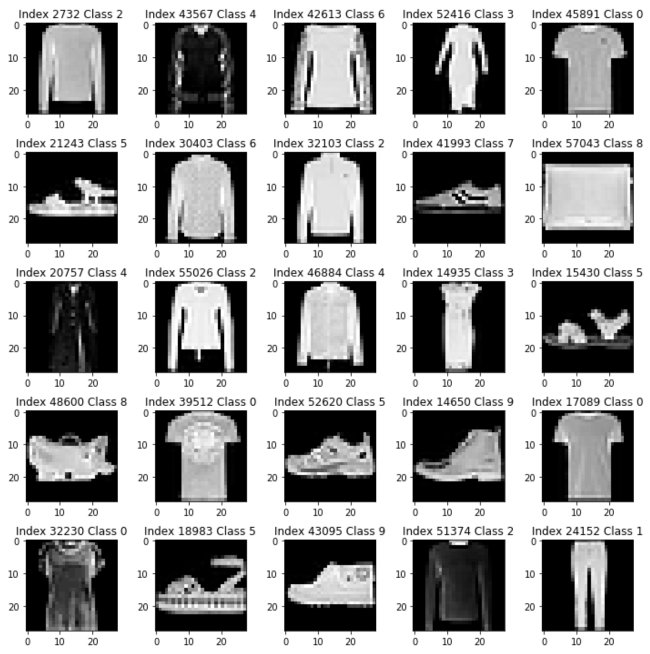
In this project we perform following three tasks

- Using KMeans algorithm cluster Fashion-MNIST dataset using Sklearns library.
- Build an Auto-Encoder based K-Means clustering model using Keras and Sklearns library.

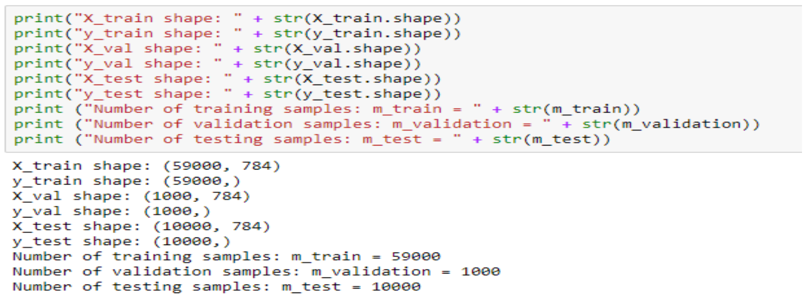
- Build an Auto-Encoder based Gaussian Mixture Model clustering model using Keras and Sklearns library. 4. Report the clustering accuracy for each of the task.

Dataset

The dataset used for training the cluster is the Fashion-MNIST dataset. It consists of 60,000 examples and testing set of 10,000 examples. Each image is a 28x28 grayscale image consisting of 28 pixels in height and 28 pixels in width thus resulting in a total of 784 pixels, associated with a label from 10 classes. The training and the test dataset have 785 columns and first columns consisting of class labels. The rest of the columns consists of the pixel-values of the associated image.



Below is the size of the training, test and validation sets.



```
print("X_train shape: " + str(X_train.shape))
print("y_train shape: " + str(y_train.shape))
print("X_val shape: " + str(X_val.shape))
print("y_val shape: " + str(y_val.shape))
print("X_test shape: " + str(X_test.shape))
print("y_test shape: " + str(y_test.shape))
print ("Number of training samples: m_train = " + str(m_train))
print ("Number of validation samples: m_validation = " + str(m_validation))
print ("Number of testing samples: m_test = " + str(m_test))
```

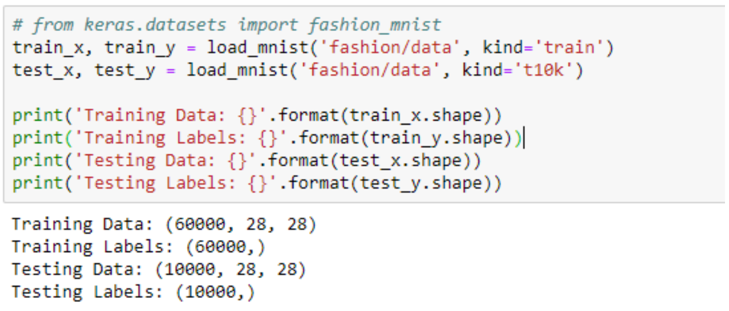
```
X_train shape: (59000, 784)
y_train shape: (59000,)
X_val shape: (1000, 784)
y_val shape: (1000,)
X_test shape: (10000, 784)
y_test shape: (10000,)
Number of training samples: m_train = 59000
Number of validation samples: m_validation = 1000
Number of testing samples: m_test = 10000
```

Preprocessing

Task 1: KMeans Clustering Algorithm

Preprocess:

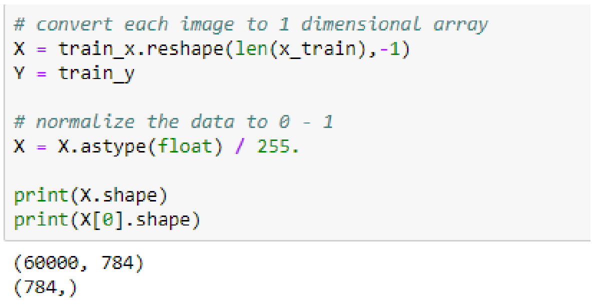
We start with loading the fashion MNIST dataset.



```
# from keras.datasets import fashion_mnist
train_x, train_y = load_mnist('fashion/data', kind='train')
test_x, test_y = load_mnist('fashion/data', kind='t10k')

print('Training Data: {}'.format(train_x.shape))
print('Training Labels: {}'.format(train_y.shape))
print('Testing Data: {}'.format(test_x.shape))
print('Testing Labels: {}'.format(test_y.shape))
```

```
Training Data: (60000, 28, 28)
Training Labels: (60000,)
Testing Data: (10000, 28, 28)
Testing Labels: (10000,)
```

Fashion MNIST consists of images that are 28 by 28 pixels, hence they will have a length of 784 once we reshape them into a 1-dimensional array. Find below the code snippets for the same.

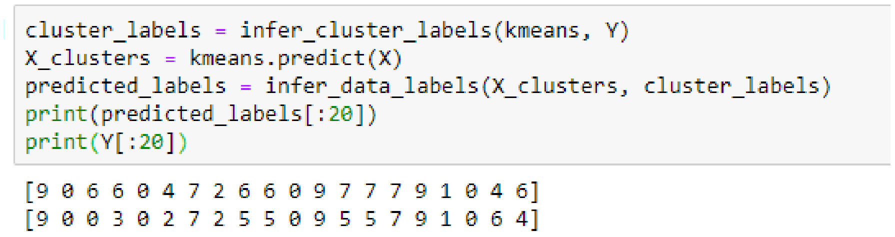


```
# convert each image to 1 dimensional array
X = train_x.reshape(len(x_train),-1)
Y = train_y

# normalize the data to 0 - 1
X = X.astype(float) / 255.

print(X.shape)
print(X[0].shape)
```

```
(60000, 784)
(784,)
```

KMeans Clustering Algorithm:

We have used the Mini-batch implementation of kmeans clustering provided by the scikit-learn. We have used this to dynamically reduce the time it takes to fit the algorithm to the data. The Fashion MNIST dataset consists of 10 different classes hence we take the number of clusters as 10 in our KMeans algorithm. The labels are assigned by KMeans algorithm refer to the cluster each array was assigned to and not the actual target. In order to fix this, we define functions that will predict which class corresponds to each cluster.

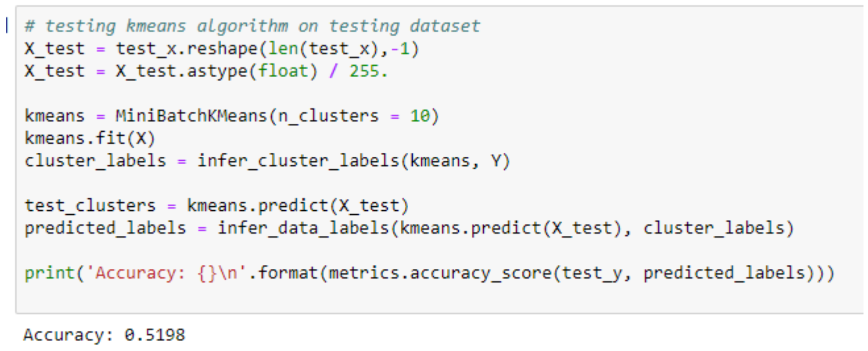


```
cluster_labels = infer_cluster_labels(kmeans, Y)
X_clusters = kmeans.predict(X)
predicted_labels = infer_data_labels(X_clusters, cluster_labels)
print(predicted_labels[:20])
print(Y[:20])
```

```
[9 0 6 6 0 4 7 2 6 6 0 9 7 7 7 9 1 0 4 6]
[9 0 0 3 0 2 7 2 5 5 0 9 5 5 7 9 1 0 6 4]
```

Evaluate Cluster Performance

We now determine accuracy of our algorithms. We have received an accuracy of 51.98 for number of clusters = 10

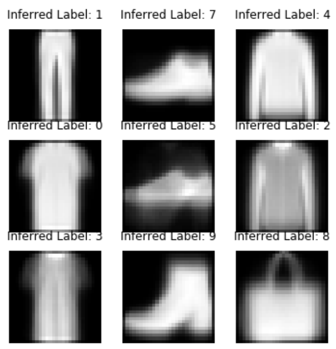


```
# testing kmeans algorithm on testing dataset
X_test = test_x.reshape(len(test_x),-1)
X_test = X_test.astype(float) / 255.

kmeans = MiniBatchKMeans(n_clusters = 10)
kmeans.fit(X)
cluster_labels = infer_cluster_labels(kmeans, Y)

test_clusters = kmeans.predict(X_test)
predicted_labels = infer_data_labels(kmeans.predict(X_test), cluster_labels)

print('Accuracy: {}\n'.format(metrics.accuracy_score(test_y, predicted_labels)))
```

```
Accuracy: 0.5198
```

Visualize the data

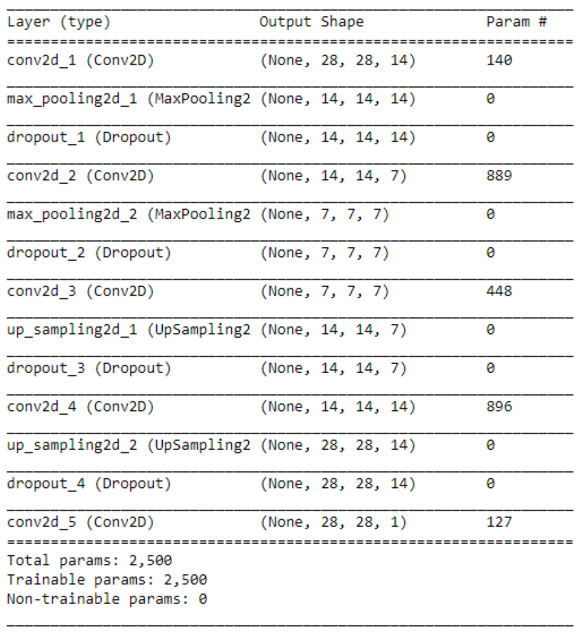
Below is the grid of images with their inferred labels.



Task 2: Autoencoder Based KMeans Clustering

It is difficult to cluster data in high dimensions as the distance between most pairs of points is similar. Using an autoencoder helps us represent high dimensional points in low dimensional space.

Find below the Autoencoder Summary used for the KMeans Clustering



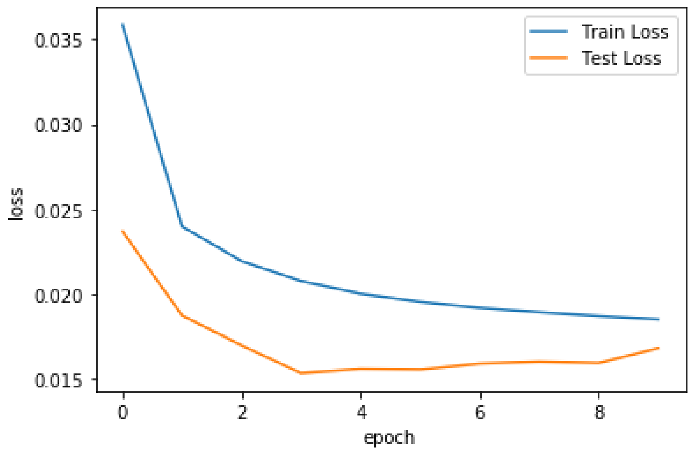
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 14)        140       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 14)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 14)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 7)         889       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 7)           0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 7)           0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 7)           448       
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 14, 14, 7)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 7)         0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 14)        896       
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 14)        0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 28, 28, 14)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 1)         127       
=================================================================
Total params: 2,500
Trainable params: 2,500
Non-trainable params: 0
_________________________________________________________________
```

After we train the autoencoder model, we input the encoded images into the KMeans algorithm for clustering.

```
# Cluster the training set
kmeans = KMeans(n_clusters=10)
clustered_training_set = kmeans.fit_predict(encoded_images)
```

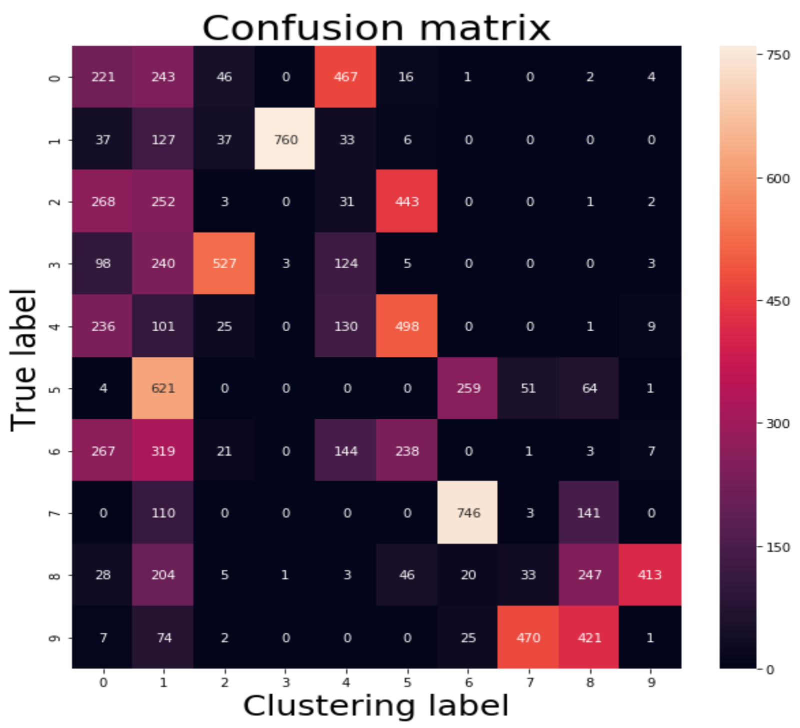
Find below the Graph for Training and Test Loss vs Number of Epochs.

Training and Test Loss vs Number of Epochs



Below is the confusion matrix for predictions made by the Clustering and the True Label.

Note: Label 1 in the confusion matrix might represent a class 5 as the classes are labelled by the KMeans algorithm.



	0	1	2	3	4	5	6	7	8	9
0	221	243	46	0	467	16	1	0	2	4
1	37	127	37	760	33	6	0	0	0	0
2	268	252	3	0	31	443	0	0	1	2
3	98	240	527	3	124	5	0	0	0	3
4	236	101	25	0	130	498	0	0	1	9
5	4	621	0	0	0	0	259	51	64	1
6	267	319	21	0	144	238	0	1	3	7
7	0	110	0	0	0	0	746	3	141	0
8	28	204	5	1	3	46	20	33	247	413
9	7	74	2	0	0	0	25	470	421	1

Task 3: Autoencoder Based GMM Clustering

Find below the Autoencoder Summary used for the KMeans Clustering

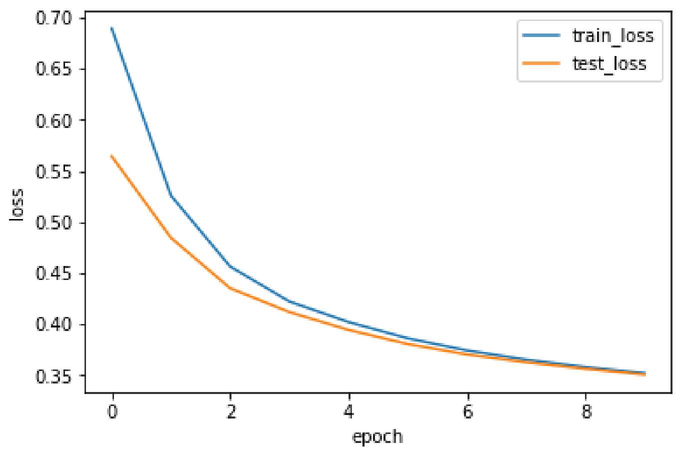
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         (None, 784)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 256)               200960    
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_9 (Dense)              (None, 6)                 774       
_________________________________________________________________
dense_10 (Dense)             (None, 128)               896       
_________________________________________________________________
dense_11 (Dense)             (None, 256)               33024     
_________________________________________________________________
dense_12 (Dense)             (None, 784)               201488    
=================================================================
Total params: 470,038
Trainable params: 470,038
Non-trainable params: 0
_________________________________________________________________
```

After we train the autoencoder model, we input the encoded images into the GMM algorithm for clustering.

```python
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=2).fit(encoded_imgs)
labels = gmm.predict(encoded_imgs)
labels
# decoded_imgs = decoder.predict(labels)
```

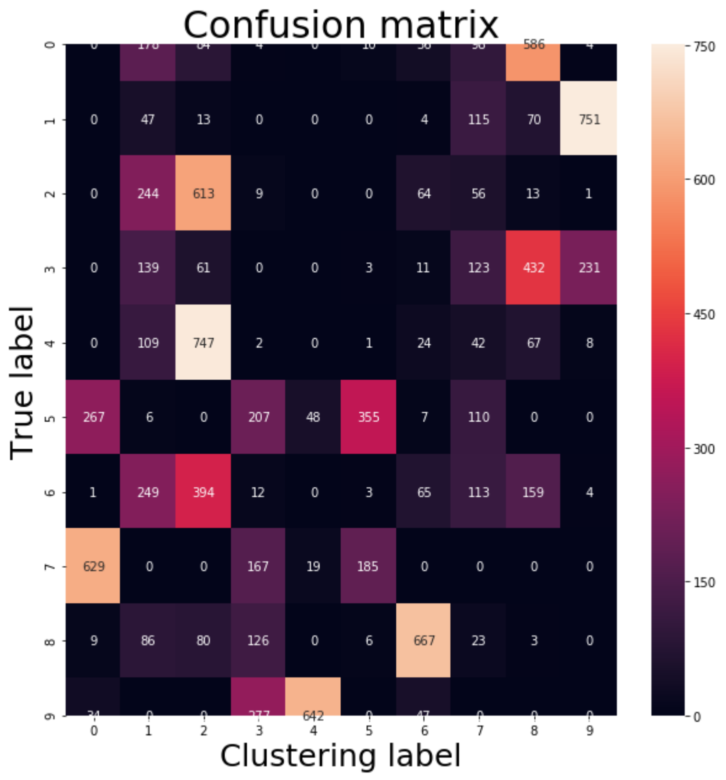
Find below the Graph for Training and Test Loss vs Number of Epochs.

Training and Test Loss vs Number of Epochs



Below is the confusion matrix for predictions made by the Clustering and the True Label.

Note: Label 1 in the confusion matrix might represent a class 5 as the classes are labelled by the KMeans algorithm.



Conclusion

The purpose of the clustering algorithm is to make sense of and extract values from large sets of structured and unstructured data. When we are working with huge volumes of unstructured data, it will be easier to analyze the data if it is partitioned into some sort of logical groupings. In simpler words, clusters are set of data points that have similar attributes and clustering algorithms group data points into different clusters based on similarities. Here we have performed cluster analysis using the Kmeans and the Gaussian Mixture Model Algorithms. We see that we get a better accuracy of clustering using autoencoders on KMeans and GMM.

References:

- http://marubon-ds.blogspot.com/2017/09/fashion-mnist-exploring.html
- https://medium.com/datadriveninvestor/k-means-clustering-for-imagery-analysis-56c9976f16b6
- https://www.kaggle.com/s00100624/digit-image-clustering-via-autoencoder-kmeans
- https://github.com/
- https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
- https://scikit-learn.org/stable/modules/mixture.html