CSE 574 Fall 2019 Project 3

Using Unsupervised Learning for Cluster Analysis of Fashion-MNIST Images

Srisai Karthik Neelamraju
UBID - 50316785
University at Buffalo
Buffalo, NY 14260
neelamra@buffalo.edu

Abstract

Identifying patterns from raw data and making valuable inferences from them plays a pivotal role in the present-day machine learning applications. Unsupervised learning, and clustering in particular, is a branch of machine learning that enables us address these tasks efficiently. Cluster analysis is typically employed to segregate data points into different clusters such that all data points belonging to a cluster share some similarity. In this project, we try to cluster images of fashion apparel using two clustering algorithms K-Means and Gaussian Mixture Models. Besides, since clustering high dimensional data is often computationally intensive, a neural network-based dimensionality reduction technique called autoencoder is introduced to speed up the computations. Observing the clustering performances on the test set revealed that the accuracy achieved using autoencoder based clustering on the latent space was better than what the default K-Means has achieved on the original uncompressed data space.

1 Introduction

Unsupervised learning is a branch of machine learning that is widely used for drawing patterns and inferences from datasets with no ground truths. This contrasts it from supervised learning which, on the flip side, requires output labels associated with each input instance. One of the prominent concepts in unsupervised learning is cluster analysis, which is often simply referred to as clustering [1]. Clustering is all about identifying identical data points in the data space and grouping them together into meaningful categories. K-Means and Gaussian Mixture Models are two widely used algorithms to achieve clustering. The former starts with assuming randomly picked data points as cluster centroids and keeps updating them as it learns the new points that are closer to these centroids; whereas the latter assigns conditional probabilities of a data point belonging to each cluster. However, one caveat here is that fitting a clustering model on high dimensional data is often very time-consuming and computationally intensive. As a result, dimensionality reduction techniques like Principal Component Analysis (PCA) are often employed to boost them. Autoencoder is one of such techniques that employs deep learning and neural networks to compress the data. The aim of this project is to analyze the performance of K-means, autoencoder based K-means and autoencoder based GMM clustering algorithms on clustering fashion images.

2 Dataset

For this project, the Fashion-MNIST dataset containing images of fashion apparel was used.

As described in the dataset description [2], this dataset was designed in order to replace the famous MNIST digits dataset as it was easily feasible for modern day neural networks to easily achieve near perfect scores on the latter. The training set consists of 60,000 grayscale images, each of size 28 × 28. Consequently, there are 784 features that describe each training example. Associated with each such training example is an integer that describes its class. There are exactly ten classes of images in the training set: T-shirt/top (0), Trouser (1), Pullover (2), Dress (3), Coat (4), Sandal (5), Shirt (6), Sneaker (7), Bag (8) and Ankle Boot (9). Integers from 0 to 9 are used as labels to denote these classes in that order as shown. On the other hand, the test set consists of 10,000 images. In the test set, there are exactly 1,000 images belonging to each of the ten possible classes. This was also used as a validation set for tuning the best possible hyperparameters of the autoencoder. For this project, training the K-Means and GMMs does not require the use of output labels since the task of clustering is unsupervised and eliminates the need for them.

3 Preprocessing

As described previously, the training and test data of the Fashion-MNIST dataset consists of 28 × 28 grayscale images. Hence, each image has 784 pixels with each pixel being an integer between 0 and 255. To improve the training performance of the autoencoder, these pixel values were subjected to some preprocessing as they were normalized to values between 0 and 1 (both inclusive) by dividing each of them with 255. Hence, essentially, min-max scaling was performed on the data set.

4 Implementation

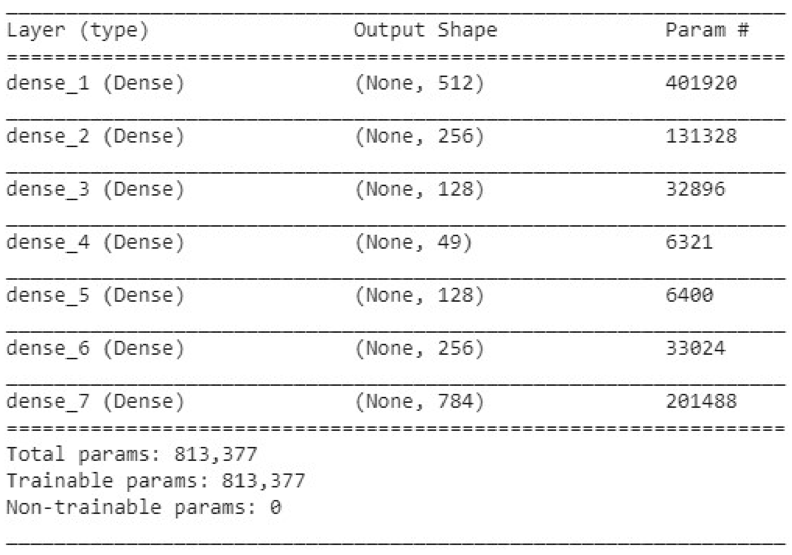
To cluster the images, three different approaches were considered. Firstly, the K-Means algorithm was employed on the 784-dimensional images directly. Later, using a multi-layer fully connected deep neural network based autoencoder, these images were compressed to have only 49 features. K-Means and Gaussian Mixture Models were then implemented on these compressed images in latent space. These techniques are described in this section.

4.1 K-Means

K-Means is a simple clustering algorithm that works by dividing the data points into disjoint subsets such that points in each subset are similar. For this algorithm to work, the number of clusters (say k) has to be known initially. It works by randomly choosing k data points as cluster centroids and then classifying each of the training samples into the cluster whose centroid is closest to the point in the data space. In sum, K-Means assigns each data point to exactly one cluster by minimizing the sum of squared Euclidean distances. This process of assigning each point to a single cluster is known as hard clustering. Later, each cluster centroid is updated based on all the points currently in its cluster. The algorithm stops when none of the centroids is being updated anymore. However, K-Means algorithm is sensitive to initialization and generates different clusters for different initial cluster centroids. Besides, the clusters formed are always spherical. For implementations in this project, Scikit-learn's cluster/KMeans module was used. K-Means was performed using 10 clusters on the 784-dimensional inputs. Implementing K-Means on such high dimensional input features often requires high computational time.

4.2 Autoencoder-based K-Means

For this part, autoencoder was introduced to increase the speed of K-Means algorithm. Essentially, instead of training K-Means on all the 784 features of the input, only a subset of these features in latent space were generated using a neural network and then used for clustering. A deep neural network based autoencoder was implemented with seven dense layers as shown in Figure 1. The first four layers of the network correspond to the encoder (encoding function of autoencoder) whereas the last three layers correspond to the decoder (decoding function of autoencoder). Thus, the output of layer 'dense_4' with 49 latent features was the encoded representation of 784-dimensional input.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 49)                6321
_________________________________________________________________
dense_5 (Dense)              (None, 128)               6400
_________________________________________________________________
dense_6 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_7 (Dense)              (None, 784)               201488
=================================================================
Total params: 813,377
Trainable params: 813,377
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Architecture of the deep neural network autoencoder implemented using Keras

All the layers in this architecture have ReLU activation function, apart from the output layers of the encoder and the whole network. These two layers use Sigmoid activation since we are trying to predict the values of normalized pixels which lie between 0 and 1. Mean squared error was the loss function and Adam Optimizer was used for optimizing the weights and the biases of each layer in the autoencoder. Different neural networks were tried out with different hyperparameters like the number of layers, learning rate and mini-batch size. However, using a mini-batch size of 256 and training the neural network for 100 epochs was found to be giving the best possible training and validation set losses.

After implementing the autoencoder to convert input dimension from 784 to 49, each of the training set instances is encoded using the encoder part of the autoencoder architecture as can be seen in Figure 2. Subsequently, K-Means algorithm is implemented on the encoded training images. This significantly reduced the running time of the algorithm.

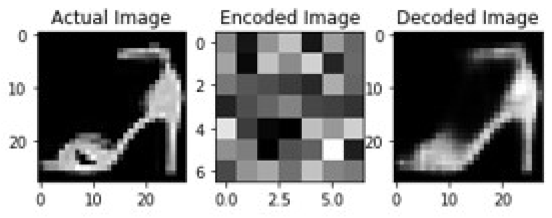
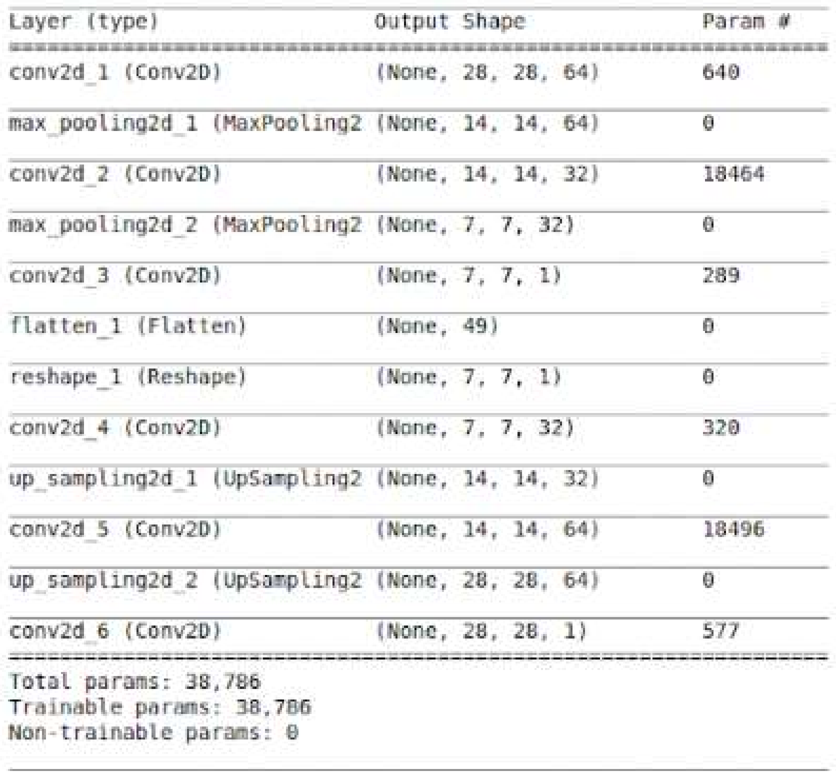


Figure 2: Functioning of the deep neural network autoencoder, depicting the actual training image, 7×7 compressed representation and the decoded image

This encoder representation doesn't really make any sense when looked at directly. However, when a convolutional autoencoder with architecture shown in Figure 3 was implemented, the encoded representation of the actual image resembled it as shown in Figure 4. Even though this implementation produced recognizable encoded images that would intuitively make more sense, its accuracy was almost similar to the deep neural network autoencoder and computational speed was slower than the latter. Since training a deep multi-layer neural network is much faster than training a convolutional neural network, the former was preferred for this project. Besides, based on trying different architectures, it was understood using a very deep convolutional autoencoder with 25 to 30 layers would definitely produce much better representations and higher clustering accuracies. Unfortunately, however, it was not feasible to train such deep networks with limited available computation resources.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        18464
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 1)           289
_________________________________________________________________
flatten_1 (Flatten)          (None, 49)                0
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 1)           0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 32)          320
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         577
=================================================================
Total params: 38,786
Trainable params: 38,786
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Architecture of the convolutional neural autoencoder implemented using Keras

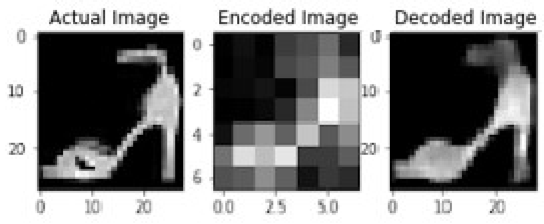


Figure 4: Functioning of the deep convolutional autoencoder, depicting the actual training image, 7 × 7 compressed representation and the decoded image

4.3 Autoencoder-based Gaussian Mixture Models

Gaussian Mixture Models (GMM) are an integral part of probabilistic clustering, which assumes that data is generated by sampling a continuous function. In contrast, K-Means just treats data as a bunch of points. Instead of assigning each data point to a single cluster, GMMs assign the point probabilistically to more than one cluster based on its distance from the cluster centroid; that is, lower the distance, higher the probability and vice versa. This process of assigning each point to more than one cluster is known as soft clustering. This is typically achieved by maximizing the log likelihood. In each iteration, the conditional probability of a point belonging to a cluster is evaluated. Subsequently, the gaussian distribution parameters (mean and variance) are updated for each cluster. The convergence of GMM is similar to K-Means, but GMMs are able to generate non-spherical clusters as they account for variance. For implementing this, GaussianMixture module of Scikit-learn was used. As described in section 4.1, implementing GMM on 784-dimensional input takes a lot of time. Hence, the autoencoder discussed in section 4.2 was again used to reduce the dimensionality of the input. Finally, GMM with 10 Gaussian components was trained on these 49-dimensional encoded training images and analyzed.

5 Results and Inferences

Firstly, basic K-Means algorithm had to be implemented using Scikit-learn. This required initial knowledge of the number of clusters (*k*). Even though it was known that the Fashion-MNIST dataset contains 10 classes, the number of clusters was supposed to be unknown as this was an unsupervised learning task. Consequently, the goal was to try out different values of *k* and observe which of these values fitted the dataset correctly. Hence, K-Means algorithm was implemented on the training set for 25 different values of *k* from 1 to 25 and with 10 random initializations for each of these 25 implementations. Figure 5 shows the sum of squared errors obtained for the best initialization in each case.

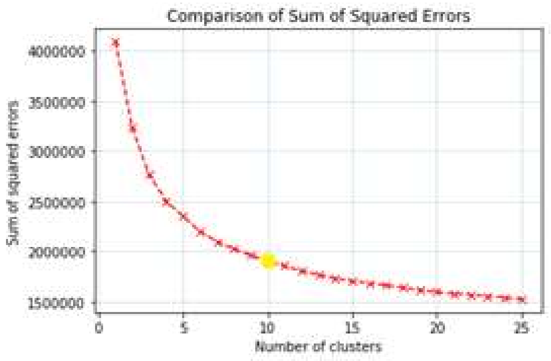
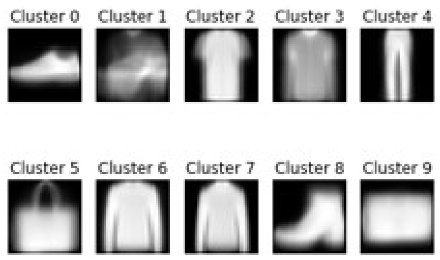
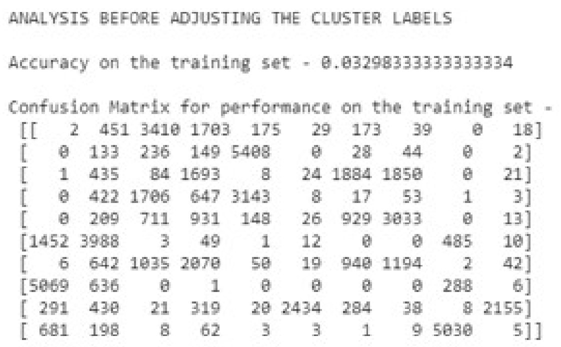


Figure 5: Comparison of SSE for different number of clusters in K-Means

As expected, using the K-Means elbow method suggested that the ideal number of clusters was 10 since the value of SSE decreased much slower for a greater number of clusters. So, the K-Means model with 10 classes was chosen as the best fit. Figure 6(a) shows the cluster centroids obtained after fitting the training data on this model and figure 6(b) shows the corresponding evaluation metrics of the model.



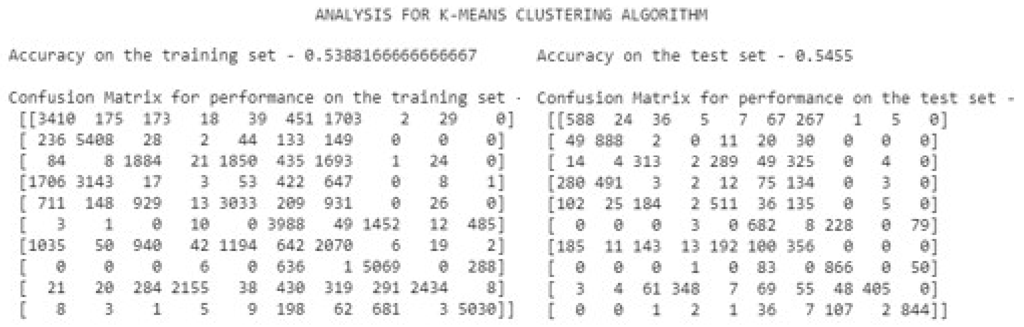


```
ANALYSIS BEFORE ADJUSTING THE CLUSTER LABELS

Accuracy on the training set - 0.03298333333333334

Confusion Matrix for performance on the training set -
 [[   2  451 3410 1703  175   29  173   39    0   18]
 [   0  133  236  149 5408    0   28   44    0    2]
 [   1  435   84 1693    8   24 1884 1850    0   21]
 [   0  422 1706  647 3143    8   17   53    1    3]
 [   0  209  711  931  148   26  929 3033    0   13]
 [1452 3988    3   49    1   12    0    0  485   10]
 [   6  642 1035 2070   50   19  940 1194    2   42]
 [5069  636    0    1    0    0    0    0  288    6]
 [ 291  430   21  319   20 2434  284   38    8 2155]
 [ 681  198    8   62    3    3    1    9 5030    5]]
```

(a) (b)

Figure 6: (a) Cluster centroids obtained, (b) Clustering performance on the training set for the best fit K-means model with 10 clusters

It was observed that the clustering accuracy was around 3%. However, this was because the ordering of clusters in the output of K-means was random and differed from the class numbers in the output labels. Hence, before calculating the clustering accuracy, the predicted labels were needed to be adjusted. This was achieved using SciPy's linear sum assignment method [3], which finds the adjusted labels using the original confusion matrix and returns them. Figure 7 shows the clustering accuracy and confusion matrix of the same model on the training and test sets after this step. A clustering accuracy of around 54% was obtained on both the training and the test sets.



```
ANALYSIS FOR K-MEANS CLUSTERING ALGORITHM

Accuracy on the training set - 0.5388166666666667

Confusion Matrix for performance on the training set -
 [[3410  175  173   18   39  451 1703    2   29    0]
 [ 236 5408   28    2   44  133  149    0    0    0]
 [  84    8 1884   21 1850  435 1693    1   24    0]
 [1706 3143   17    3   53  422  647    0    8    1]
 [ 711  148  929   13 3033  209  931    0   26    0]
 [   3    1    0   10    0 3988   49 1452   12  485]
 [1035   50  940   42 1194  642 2070    6   19    2]
 [   0    0    0    6    0  636    1 5069    0  288]
 [  21   20  284 2155   38  430  319  291 2434    8]
 [   8    3    1    5    9  198   62  681    3 5030]]
```

```
Accuracy on the test set - 0.5455

Confusion Matrix for performance on the test set -
 [[588  24  36   5   7  67 267   1   5   0]
 [ 49 888   2   0  11  20  30   0   0   0]
 [ 14   4 313   2 289  49 325   0   4   0]
 [280 491   3   2  12  75 134   0   3   0]
 [102  25 184   2 511  36 135   0   5   0]
 [  0   0   0   3   0 682   8 228   0  79]
 [185  11 143  13 192 100 356   0   0   0]
 [  0   0   0   1   0  83   0 866   0  50]
 [  3   4  61 348   7  69  55  48 405   0]
 [  0   0   1   2   1  36   7 107   2 844]]
```

Figure 7: Analyzing evaluation metrics on training and test sets for the basic K-Means model

Further, autoencoder with architecture as described in the previous section was implemented using Keras. After training the autoencoder for 100 epochs, both the training and the test set losses were found to be around of the order 10e-3. Figure 8 compares the training and validation losses for the autoencoder during the 100 epochs of training. The plot indicates that there has neither been an overfit nor an underfit of the training data.

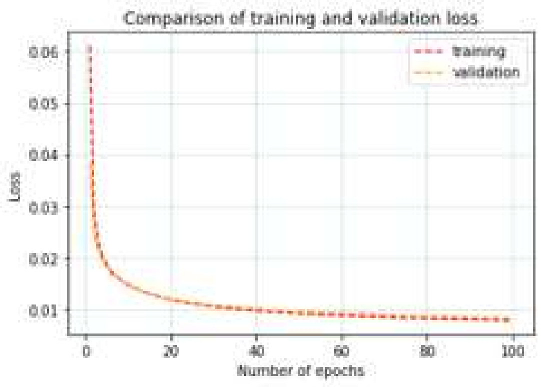
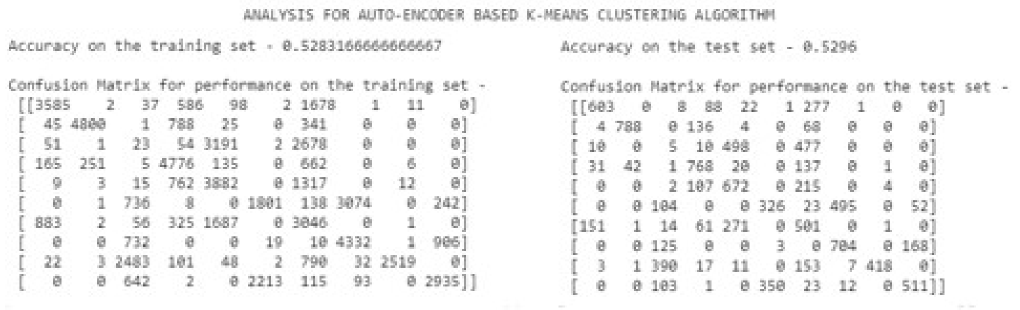


Figure 8: Comparing training and validation losses for the deep NN autoencoder

After training the autoencoder, the training images were compressed using the encoder part of the autoencoder and K-Means with 10 clusters was implemented on the encoded (compressed) training images. Figure 9 shows the clustering accuracy and confusion matrix of the autoencoder based K-Means model on the training and test sets after adjusting the cluster labels accordingly. A clustering accuracy of around 53% was obtained on both the training and the test sets.

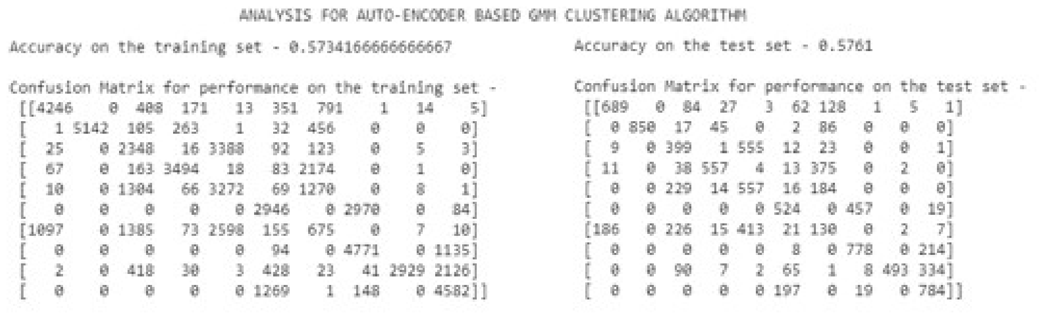


```
ANALYSIS FOR AUTO-ENCODER BASED K-MEANS CLUSTERING ALGORITHM

Accuracy on the training set - 0.5283166666666667

Confusion Matrix for performance on the training set -
 [[3585    2   37  586   98    2 1678    1   11    0]
 [  45 4800    1  788   25    0  341    0    0    0]
 [  51    1   23   54 3191    2 2678    0    0    0]
 [ 165  251    5 4776  135    0  662    0    6    0]
 [   9    3   15  762 3882    0 1317    0   12    0]
 [   0    1  736    8    0 1801  138 3074    0  242]
 [ 883    2   56  325 1687    0 3046    0    1    0]
 [   0    0  732    0    0   19   10 4332    1  906]
 [  22    3 2483  101   48    2  790   32 2519    0]
 [   0    0  642    2    0 2213  115   93    0 2935]]
```

```
Accuracy on the test set - 0.5296

Confusion Matrix for performance on the test set -
 [[603   0   8  88  22   1 277   1   0   0]
 [  4 788   0 136   4   0  68   0   0   0]
 [ 10   0   5  10 498   0 477   0   0   0]
 [ 31  42   1 768  20   0 137   0   1   0]
 [  0   0   2 107 672   0 215   0   4   0]
 [  0   0 104   0   0 326  23 495   0  52]
 [151   1  14  61 271   0 501   0   1   0]
 [  0   0 125   0   0   3   0 704   0 168]
 [  3   1 390  17  11   0 153   7 418   0]
 [  0   0 103   1   0 350  23  12   0 511]]
```

Figure 9: Analyzing evaluation metrics on the training and test sets for the autoencoder based K-Means clustering model

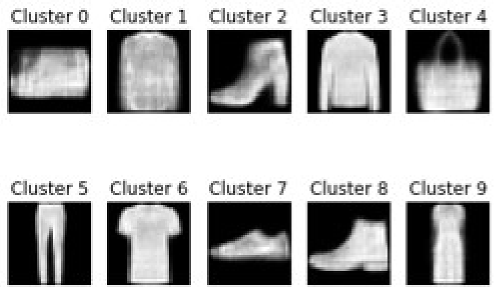
Finally, the same autoencoder was used to implement GMM clustering algorithm with 10 Gaussian components. Figure 10 shows the clustering accuracy and confusion matrix of the autoencoder based GMM on the training and test sets after adjusting the cluster labels accordingly. A clustering accuracy of around 57% was obtained on both the training and the test sets.



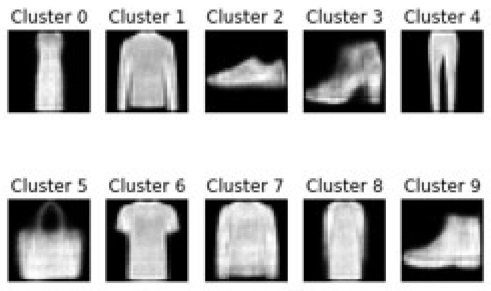
```
                         ANALYSIS FOR AUTO-ENCODER BASED GMM CLUSTERING ALGORITHM
Accuracy on the training set - 0.5734166666666667            Accuracy on the test set - 0.5761

Confusion Matrix for performance on the training set -       Confusion Matrix for performance on the test set -
 [[4246    0  408  171   13  351  791    1   14    5]         [[689   0  84  27   3  62 128   1   5   1]
  [   1 5142  105  263    1   32  456    0    0    0]          [  0 850  17  45   0   2  86   0   0   0]
  [  25    0 2348   16 3388   92  123    0    5    3]          [  9   0 399   1 555  12  23   0   0   1]
  [  67    0  163 3494   18   83 2174    0    1    0]          [ 11   0  38 557   4  13 375   0   2   0]
  [  10    0 1304   66 3272   69 1270    0    8    1]          [  0   0 229  14 557  16 184   0   0   0]
  [   0    0    0    0    0 2946    0 2970    0   84]          [  0   0   0   0   0 524   0 457   0  19]
  [1097    0 1385   73 2598  155  675    0    7   10]          [186   0 226  15 413  21 130   0   2   7]
  [   0    0    0    0    0   94    0 4771    0 1135]          [  0   0   0   0   0   8   0 778   0 214]
  [   2    0  418   30    3  428   23   41 2929 2126]          [  0   0  90   7   2  65   1   8 493 334]
  [   0    0    0    0    0 1269    1  148    0 4582]]         [  0   0   0   0   0 197   0  19   0 784]]
```

Figure 10: Analyzing evaluation metrics on the training and test sets for the autoencoder based Gaussian Mixture clustering model

By analyzing the confusion matrices for each of the three clustering techniques, it was observed that the images belonging to classes pullovers (2), coats (4) and shirts (6) were mostly clustered among one another. For autoencoder-based K-Means, only 5 pullovers out of 1000 in the test set were clustered correctly. Of the remaining pullovers, 498 were clustered into coats and 477 were clustered into shirts. This behavior can be understood by looking at the cluster centroids in Figure 11(a). There is no centroid corresponding to images of the class 'pullover', but two centroids corresponding to shirt (cluster 1) and coat (cluster 3). Since many images of these three classes look alike and their pixels have similar distributions, pullovers were wrongly clustered. On the flip side, the performance on classes like trouser was always high since images of no other class were similar to them.



(a)



(b)

Figure 11: Cluster centroids for autoencoder-based (a) K-Means and (b) GMM

6 Summary

In this project, three unsupervised learning models that used different clustering algorithms – basic K-Means, autoencoder-based K-Means and autoencoder-based Gaussian Mixture Model - were designed to cluster images from Fashion-MNIST dataset. It contained 60,000 training and 10,000 test images of size 28 × 28 (784 features). Initially, for the basic K-Means model, the ideal number of clusters was found out to be 10 using elbow method. Later, K-Means algorithm with 10 clusters was implemented and its performance was analyzed. Further, in an attempt to boost the computational speed of K-means, a deep NN based autoencoder was trained to reduce the dimensionality of the image to 49 features. Applying K-Means to the images in the latent space proved to be much faster. Finally, GMM was also applied on the encoded images. While the basic K-Means model gave a clustering accuracy of about 54%, the autoencoder-based K-Means achieved a slightly lesser accuracy of 53%. However, it was the autoencoder-based GMM that performed the best among the three models, giving a clustering accuracy of 57%. This demonstrated that using probabilistic (soft) clustering was efficient for this dataset as it contained many images that were similar but belonged to different classes. Besides, it was also observed that both the basic K-Means and the autoencoder-based K-Means badly suffered from the initialization problem as the clustering accuracies were very erratic for different centroid initializations.

References

[1] Bishop, C.M., 2006. *Pattern Recognition and Machine Learning*. Springer.

[2] https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/

[3] https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html

[4] CSE 574 Slides by Prof. Sargur Srihari

[5] Andrew Ng's 'Deep Learning by deeplearning.ai' on Coursera