Unsupervised learning on Fashion MNIST dataset

Abhishek S. Dalvi
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
adalvi@buffalo.edu

Abstract

The purpose of this project is to classify fashion MNIST dataset using K-means clustering, K-means and Guassian Mixture Model Clustering using embedding of the data obtained from an auto-encoder.

1 Introduction

Several ways have been proposed for unsupervised learning, but in this project will compare the results obtained from few of this algorithms i.e K-means clustering, K-means clustering using image embedding from an auto-encoder and Guassian Mixture Model Clustering using image embedding from an auto-encoder.

2 Dataset

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

NOTE :- The dataset provides labels for each category which will not be used as this will be an unsupervised learning project.

Table 1: Labels(not used)

0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

3 Pre-Processing

3.1 Data Partitioning

The whole dataset is partitioned into three parts, Testing, Validation and Testing randomly in

the ratio of 70%, 10% and 20% respectively. The partioning is done using sk-learns train-test-split function.

3.2 One hot Encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. it is done manually (in 1st network) and also done by using to_categorical function.

3.3 Normalization

A standard scalar normalization is applied on the training data i.e dividing each value by 256(the max value) to standardize the values between 0 to 1.

4 Architecture

4.1 Naive K-means clustering

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K.

The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

- The centroids of the K clusters, which can be used to label new data.

- Labels (Cluster no.) for the training data. (each data point is assigned to a single cluster)

The algorithm starts with an initial estimate for the K centroids (taken as 10 in the project; can be also found out by **elbow method**), which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

1. Data assignment step:

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance. More formally, if ci is the collection of centroids in set C, then each data point x is assigned to a cluster based on

$$\underset{c_i \in C}{\operatorname{argmin}}\ dist(c_i,\ x)^2$$

where dist(L·) is the standard (L2) Euclidean distance. Let the set of data point assignments for each ith cluster centroid be Si.

2. Centroid update step:

In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

$$c_i = \frac{1}{|S_i|}\sum_{x_i \in S_i} x_i$$

The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of

iterations is reached).

4.2 Auto-encoder Architecture

Auto-encoders (AE) are neural networks that aims to copy their inputs to their outputs. They work 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 :

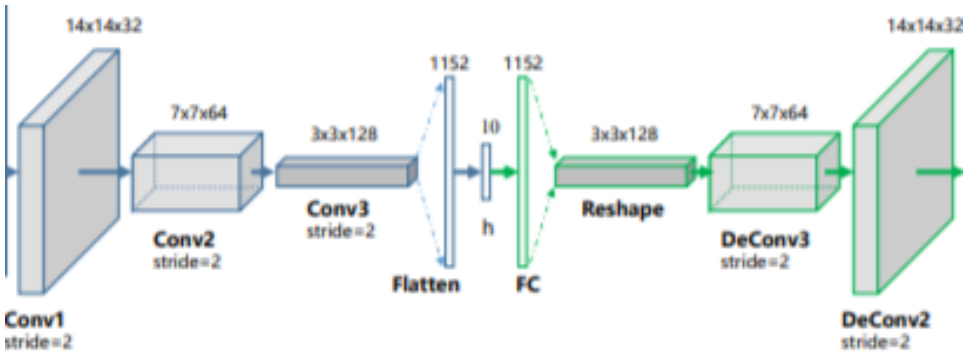
1) Encoder: This is the part of the network that compresses the input into a latent-space representation. It can be represented by an encoding function h=f(x).

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).

The auto-encoder 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.

The below diagram shows the architecture of the auto-encoder used in the project.

Figure 1: Auto-encoder Architecture



Loss function used is mean squarred error and the optimizer used is RMSprop. The model is trained for 100 epochs with batch size of 64.

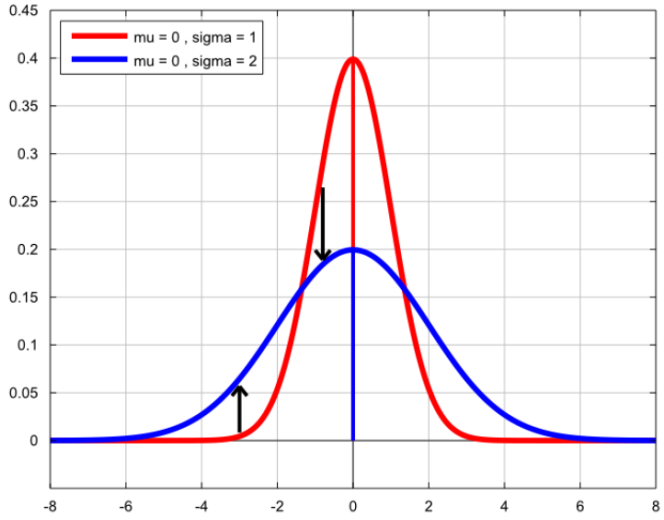
4.3 Gaussian Mixture Model

Gaussian Mixture Models (GMMs) is a clustering algorithm where it assume that there are a certain number of Gaussian distributions, and each of these distributions represent a cluster. Hence, a Gaussian Mixture Model tends to group the data points belonging to a single distribution together. The algorithm is as follows:-

- Start with random Gaussian parameters (θ)
- Repeat the following until we converge:
- a) Expectation Step: Compute p(zi = k | xi, θ). In other words, does sample i look like it came from cluster k?
- b) Maximization Step: Update the Gaussian parameters (θ) to fit points assigned to them.

Gaussian mixture models can be used to cluster unlabeled data in much the same way as k-means. There are, however, a couple of advantages to using Gaussian mixture models over k-means. First and foremost, k-means does not account for variance. By variance, we are referring to the width of the bell shape curve.

Figure 2: Gaussian Distribution



Thus, by seeing the above curve the guassian model can make clusters really specific, i.e unlike k-means model which places a circle (or, in higher dimensions, a hyper-sphere) at the center of each cluster, with a radius defined by the most distant point in the cluster.

The second difference between k-means and Gaussian mixture models is that the former performs hard classification whereas the latter performs soft classification. In other words, k-means tells us what data point belong to which cluster but won't provide us with the probabilities that a given data point belongs to each of the possible clusters.

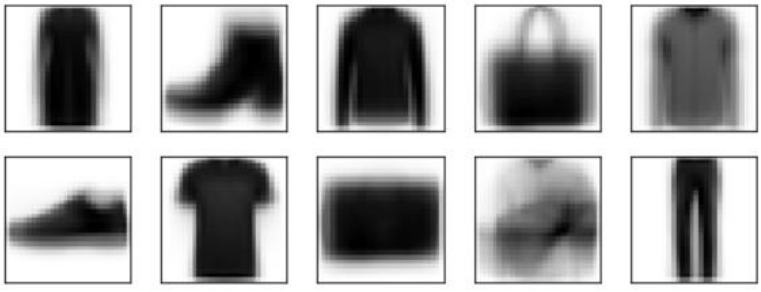
5 Results

Because k-means knows nothing about the identity of the cluster, the 0–9 labels may be permuted. We can fix this by matching each learned cluster label with the true labels found in them by using the mode in the data.

5.1 K-means (Without dimension reduction)

On applying K-means we see that the clusters centers obtained are roughly giving us the 10 categories.

Figure 3: Cluster Centers from K-means



After prediction we the confusion matrix as follows:

Figure 4: Confusion matrix K-means

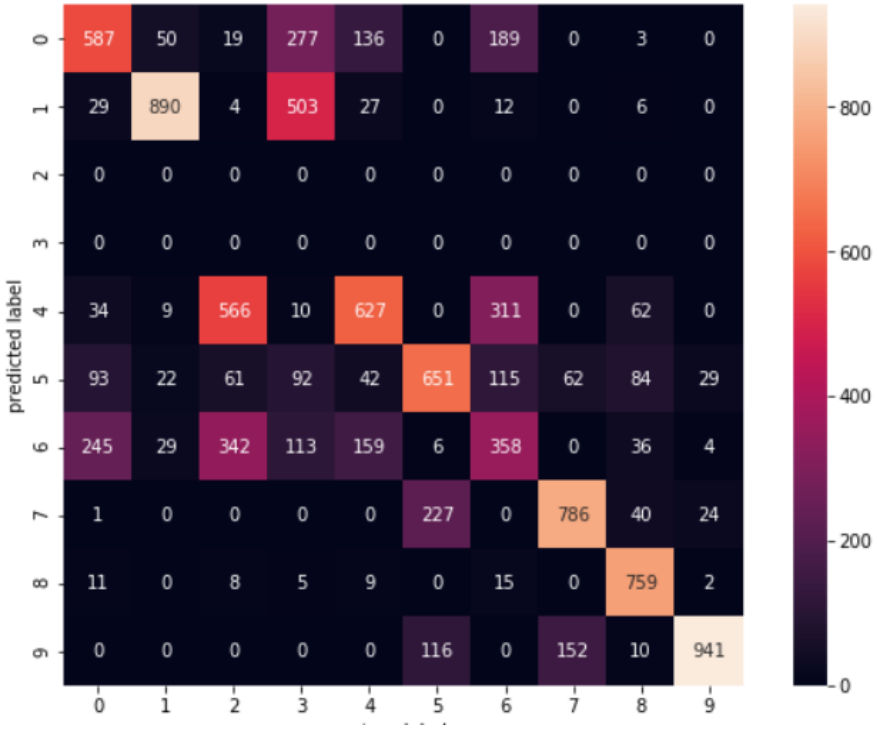


Figure 5: Accuracy of K-means

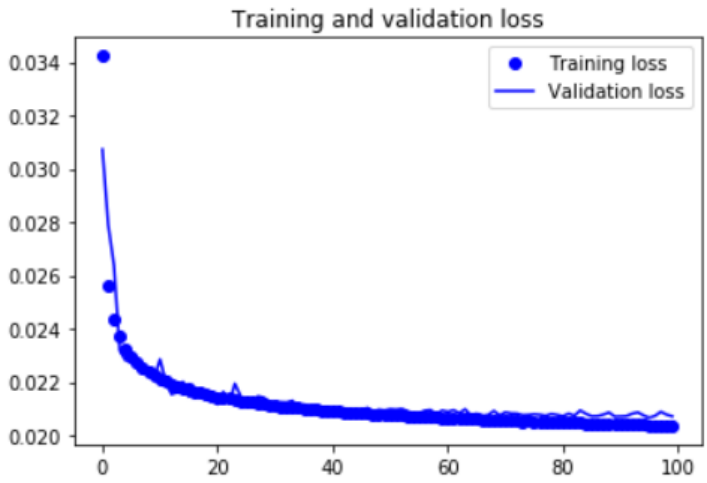
```
Out[25]: 0.5599
```

The Accuracy Obtained is roughly **56%**, and it is because K- means applied in a higher dimensions doesn't work out well, as the number of dimensions are large(784), calculations become difficult.

5.2 K-means using Auto-encoder embedding.

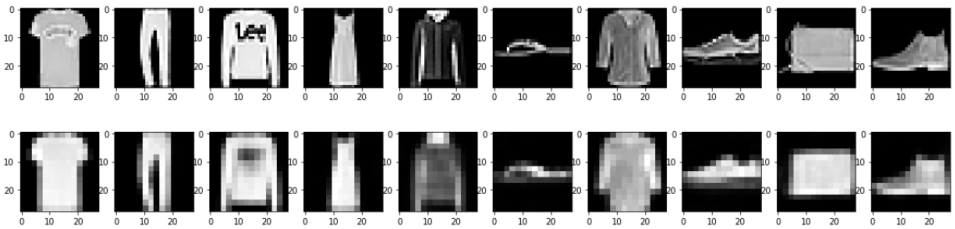
We train the autoencoder for 100 epochs and stop at there as we see the validation loss being almost constant after the 80th epoch.

Figure 6: Training vs Validation loss



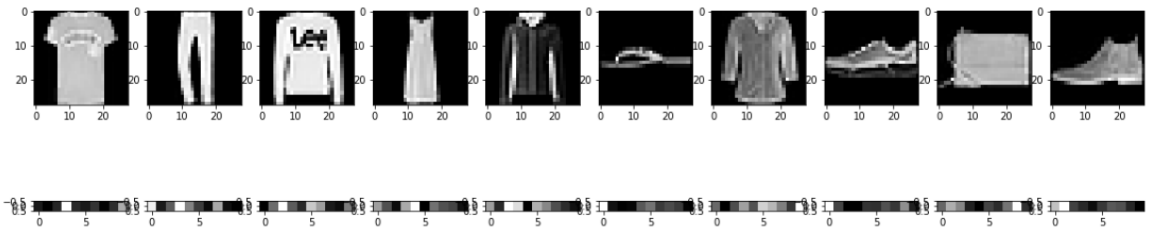
We pass some of the test data to see how the autoencoder is working.

Figure 7: Autoencoder input and output on Test Images



The images shown below how the test images look like when they are encoded in 10 dimensions.

Figure 8: Encoding of Test Images



After training the data we see observe the cluster centres which are in 10 dimensions.

Figure 9: Cluster Centers after Applying K-means on encoded training set (in values and visualized using plot)

```
[[ 92.91263   28.545515  38.844875 101.21614   96.989525  27.152925
   28.720516  56.889328  22.781723  23.73184 ]
 [ 47.19959  111.34349   33.59588   28.827866  50.319393  45.98233
   39.78293   55.347717  39.08201   32.07566 ]
 [ 40.700237  43.333164 139.81567   86.718544  53.971394  93.78134
   99.059006  59.30939   57.352787  79.93756 ]
 [ 31.079994  26.22086   46.270287  63.03607   31.704031  50.4328
   45.537914  30.780268  31.895634  42.44749 ]
 [ 82.16551  164.3995    60.93206   56.598267  50.538486  30.28525
   75.35089  108.87134   59.825844  60.307274]
 [ 70.53561   46.610542  70.51906  150.6259    96.633095  50.52431
   66.966225  53.432777  54.276882  53.184036]
 [ 64.08894   49.79169   58.832832  64.46622   26.729061  46.628532
  169.3431    44.271996  68.66142   38.209503]
 [ 71.162415  74.16441   94.15785   40.01586   24.95371   41.70951
   63.708263  42.149906 145.53613  100.31045 ]
 [ 60.599316  34.78147   43.13856  150.19452   61.01029   55.413467
   61.954136  46.18748   61.59098  138.11969 ]
 [140.16428   67.2382    31.26103   24.146477  28.12815   40.07991
   39.816544  42.04037   53.563244  32.48298 ]]
```

```
1 fig, ax = plt.subplots(2, 5, figsize=(8, 3))
2 centers = kmeans.cluster_centers_.reshape(10, 1, 10)
3 for axi, center in zip(ax.flat, centers):
4     axi.set(xticks=[], yticks=[])
5     axi.imshow(center, interpolation='nearest', cmap=plt.cm.bina
```

Figure 10: Confusion Matrix for K-means with autoencoder

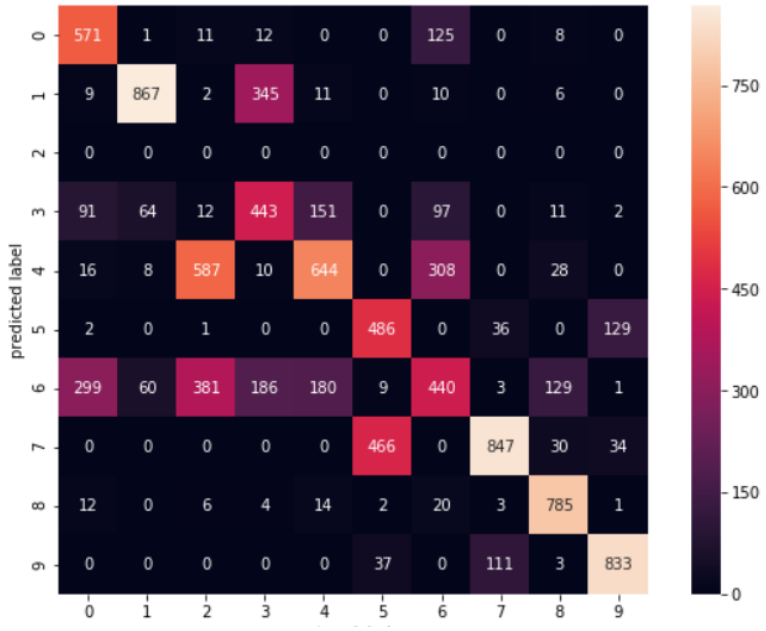


Figure 11: Accuracy for K-means with autoencoder

```
In [29]:  1 from sklearn.metrics import accuracy_score
          2 accuracy_score(y_test, labels)

Out[29]: 0.5916
```

By reducing the dimensions, we see that we get **59.16% accuracy**; which is a better accuracy than the K-means method without the encoder.

5.3 GMM using Auto-encoder embedding

The cluster centers, images obtained look approximately same as the previous method as encodings of images are used.

Figure 12: Confusion matrix for GMM by using image embedding

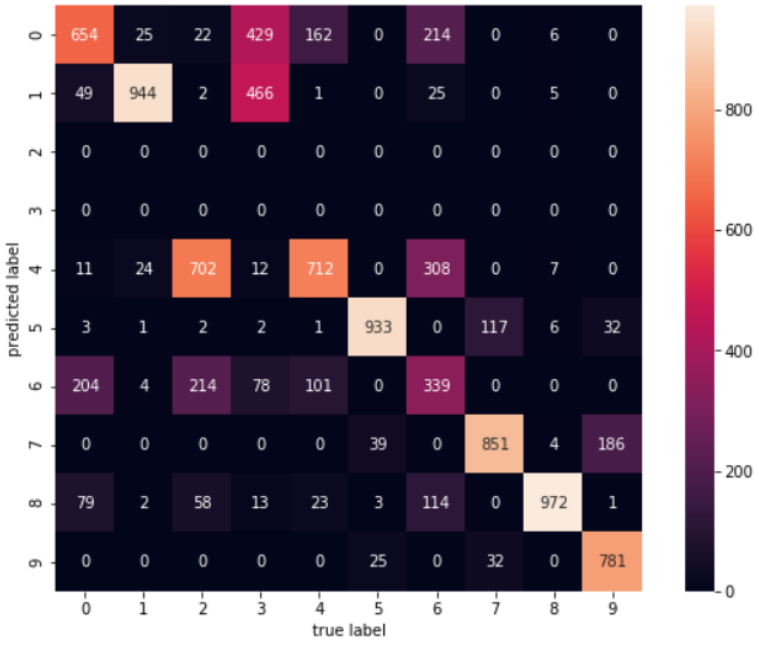


Figure 13: Accuracy for GMM by using image embedding

```
In [34]:  1 from sklearn.metrics import accuracy_score
          2 accuracy_score(y_test, labels)

Out[34]: 0.6186
```

We see that GMM gives the best accuracy as expected; of **61.86% accuracy** which is approximately 62%.

6 Conclusion

It can be concluded that by reducing dimensions, clustering algorithms work better and GMM is better than K-means as GMM takes into account variance, thus gives better clusters.

It can also be noted that significant improvement isn't obtained by reducing dimensions. This is because auto-encoders is a minimization problem where the **reconstruction cost is minimized**. We can also say that the **K-L divergence minimization** (making input and output similar) is considered on the input of the encoder and the output of the decoder and has considered no aspect of the image embedding. Therefore, it can be said that the **neighborhood distance is lost** which is essential for clustering.

A better way to cluster would be by using **T-SNE**(T-Distributed Stochastic Neighbor embedding) for embedding which focuses more on the local structure by preserving neighborhood distance.