Unsupervised Learning

Fashion MNIST Data Set

Snigdha Gupta
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
snigdhag@buffalo.edu

Abstract

This project aims at implementing unsupervised learning over Fashion MNIST data set. We use k-means clustering and Gaussian Mixture Model algorithms to implement unsupervised learning to draw inferences from the data set consisting of input data without labelled responses. A decent accuracy for the model is achieved by creating a compressed representation of the data using autoencoders.

1 Introduction

1.1 Unsupervised Learning

Unsupervised learning is a type of machine learning algorithm used to draw inferences from data sets consisting of input data without labelled responses.

The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. The clusters are modelled using a measure of similarity which is defined upon metrics such as Euclidean or probabilistic distance.

Common clustering algorithms include:

- **Hierarchical clustering:** builds a multilevel hierarchy of clusters by creating a cluster tree
- **k-Means clustering:** partitions data into k distinct clusters based on distance to the centroid of a cluster
- **Gaussian mixture models:** models clusters as a mixture of multivariate normal density components
- **Self-organizing maps**: uses neural networks that learn the topology and distribution of the data
- **Hidden Markov models:** uses observed data to recover the sequence of states

Application of Unsupervised Learning: Unsupervised learning methods are used in bioinformatics for sequence analysis and genetic clustering; in data mining for sequence and pattern mining; in medical imaging for image segmentation; and in computer vision for object recognition.

1.2 Clustering

The goal of clustering is to create groups of data points such that points in different clusters are dissimilar while points within a cluster are similar. Clustering allows you to automatically split the dataset into groups according to similarity.

Application of clustering: Clustering is widely used in various industries – customer segmentation, document clustering, image segmentation, recommendation engines etc.

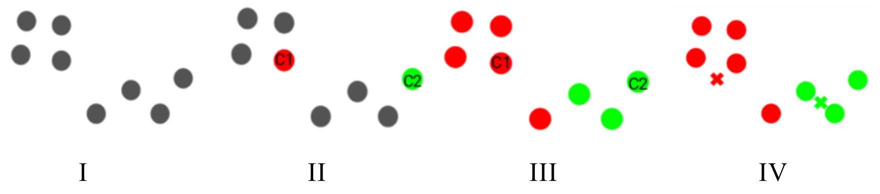
1.3 k-Means Clustering

k-means clustering, clusters the data points into *k* groups. A larger *k* creates smaller groups with more granularity, a lower *k* means larger groups and less granularity.

The output of the algorithm would be a set of "labels" assigning each data point to one of the *k* groups. In k-means clustering, the way these groups are defined is by creating a **centroid** for each group. The centroids are like the heart of the cluster, they "capture" the points closest to them and add them to the cluster. k-means clustering performs hard-clustering.

The steps to perform k-means is as follows:

1. Define k centroids, initialize at random
2. Find the closest centroid & update cluster assignments. Assign each data point to one of the k clusters. Each data point is assigned to the nearest centroid's cluster. Here, the measure of "nearness" is a hyperparameter — often Euclidean distance.
3. Move the centroids to the center of their clusters. The new position of each centroid is calculated as the average position of all the points in its cluster.
4. Keep repeating steps 2 and 3 until the centroid stop moving a lot at each iteration (i.e. until the algorithm converges).



[I] Cluster contains 8 data points.
[II] Randomly selecting two (c1, c2) points as centroids.
[III] Assign all points to the closest cluster centroids.
[IV] Recompute centroids of newly formed clusters. (Here the two X in red and green are new centroids)

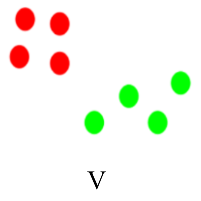


Figure 1 k-means clustering for k=2

[V] Repeat III and IV steps until the centroids of newly formed clusters do not change.

1.4 Gaussian Mixture Model

Gaussian mixture models can be used to cluster unlabeled data in much the same way as k-means. GMMs are used to implement soft-clustering.

A **Gaussian mixture** is a function that is comprised of several Gaussians, each identified by $k \in \{1,\ldots, K\}$, where K is the number of clusters of our dataset. Each Gaussian k in the mixture is comprised of the following parameters:

- A mean μ that defines its centre.

- A covariance Σ that defines its width. This would be equivalent to the dimensions of an ellipsoid in a multivariate scenario.
- A mixing probability π that defines how big or small the Gaussian function will be.

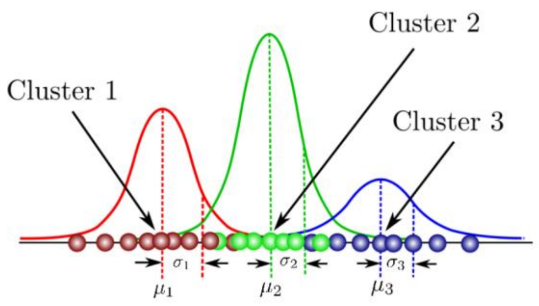


Figure 2 Gaussian functions for k=3

Each Gaussian explains the data contained in each of the three clusters available. The mixing coefficients are themselves probabilities and must meet $\sum_{k=1}^{K} \pi_k = 1$ condition. To determine optimal value for these parameters, ensure each Gaussian fits the data points belonging to each cluster. This is achieved by maximum log likelihood. The below equation is the Gaussian density function:

$$N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}\, |\Sigma|^{1/2}} \exp\left(\frac{-1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

Here x represents data points, D is the number of dimensions of each data point. μ *and* Σ are mean and covariance respectively.

The log likelihood is given by:

$$lnN(x|\mu, \Sigma) = \frac{-D}{2} ln2\pi - \frac{1}{2}\ln\Sigma - \frac{1}{2}(x-\mu)^T \Sigma^{-1}(\mathrm{x}-\mu)$$

To address the fact that we need the parameters of each Gaussian (i.e. variance, mean and weight) in order to cluster the data segregate which sample belongs to what Gaussian in order to estimate those very same parameters.

At a high level, the **expectation maximization** algorithm can be described as follows:

1. Start with random Gaussian parameters (θ)
2. Repeat the following until convergence:
   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.

1.5 Autoencoders

Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of **representation learning**. It is a data compression algorithm where the compression and decompression functions are data-specific, lossy, and learned automatically from examples rather than engineered by a human. Autoencoder uses compression and decompression functions which are implemented with neural networks.

To build an autoencoder, we need three things: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of the data and the decompressed representation (i.e. a "loss" function). The encoder and decoder will be chosen to be parametric functions (typically neural networks), and to be differentiable with respect to the distance function, so the parameters of the encoding/decoding functions can be optimized to minimize the reconstruction loss, using Stochastic Gradient Descent.

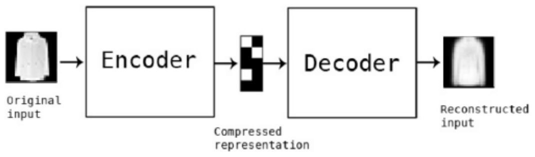


Figure 3 Autoencoder

Optimizer: Adam
We are using 'Adam' optimizer since it works well in practice and outperforms other adaptive techniques. Adam is Adaptive Moment Estimation. It uses relatively very low memory and works well with little tuning of parameters. The default learning rate is 0.01. Adam computes estimates of the mean and variance and further introduces bias correction. Adam is basically RMSprop + momentum. Adam provides both a smart learning rate annealing and momentum behaviors of the algorithms.
Loss: Binary Cross Entropy
We use binary cross entropy since we have multi-label classification.

2 Data Set

2.1 Zalando's Fashion MNIST

Fashion-MNIST dataset by e-commerce Zalando—consists 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.

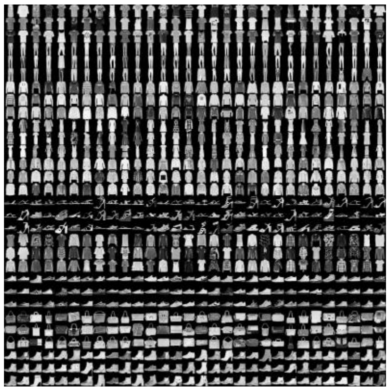


Figure 4 Fashion MNIST data set by Zalando

The Fashion MNIST dataset is identical to the MNIST dataset in terms of training set size, testing set size, number of class labels, and image dimensions:

- 60,000 training examples
- 10,000 testing examples
- 10 classes
- 28×28 grayscale images

The first column consists of the class labels and represents the article of clothing. The rest of the columns contain the pixel-values of the associated
image.

Each training and test example is assigned to one of the labels as shown below:
1 T-shirt/top
2 Trouser
3 Pullover

4 Dress
5 Coat
6 Sandal
7 Shirt
8 Sneaker
9 Bag
10 Ankle Boot

3 Preprocessing

The transformation applied to the data before feeding it to the algorithm is called pre-processing. This is required to convert the raw data into a clean data set, since the raw data is not feasible for the analysis.

The data set comprises of 784 columns. We scale these values to a range of 0 to 1 before feeding them to the neural network model. In order to do so, we divide the values by 255. It's important that the *training set* and the *testing set* be preprocessed in the same way.

Each image is normalized between a range of 0 to 1 by dividing each pixel value by 255.0 (RGB) to convert the image into gray scale. We do this so that the classes, when assigned weights can be compared.

It is important to verify that the data has been normalized correctly.

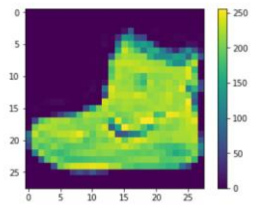


Figure 5 Non-normalized input data

Figure 6 Normalized input data

For each task data is also reshaped depending upon the scenario.

4 Architecture

Task 1

In the first task we simply apply k-means clustering algorithm to fashion MNIST data set. This method is fast and effective algorithm for many problems. However, k-means clustering's distance metrics is limited to original data space and becomes ineffective when dimensionality is high (example: Images).

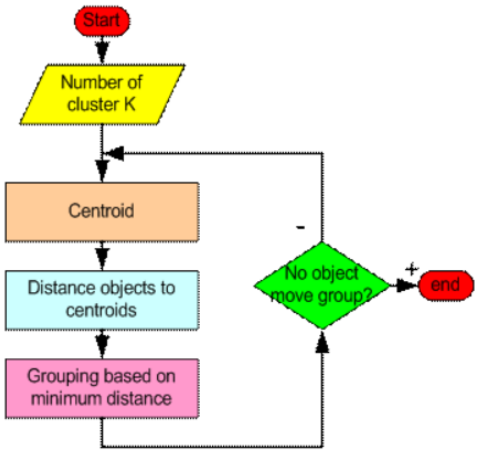

Start
Number of cluster K
Centroid
Distance objects to centroids
Grouping based on minimum distance
No object move group?
-
+
end

Figure 7 Flowchart of k-means algorithm

Task 2 and 3

We train the model using **Adam Optimizer** for **50 epochs** and **128 batch size**. The loss is calculated using **binary cross-entropy** function with a **0.01 learning rate**. Task 2 and 3 use convolutional autoencoders. This means that encoding and decoding models use convolutional neural networks instead of fully-connected networks. The encoded images are transformed from 28x28x1 to a 3D array of dimensions 4x4x8. However, to visualize the same, we flatten the image to a vector length of 128. We reshape the image back to 4x4x8 after flattening and perform un-sampling to get back a 28x28x1 image.

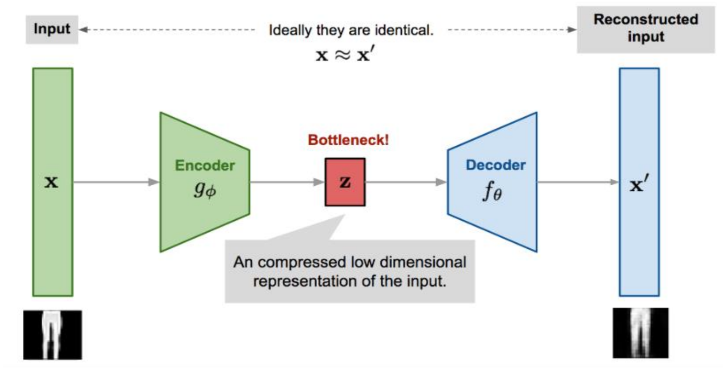

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

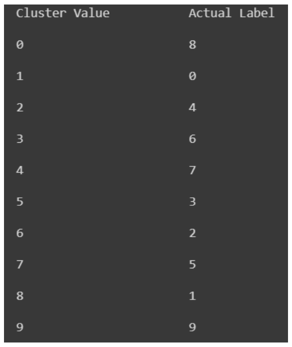
Figure 8 Autoencoder and decoder architecture

5 Results

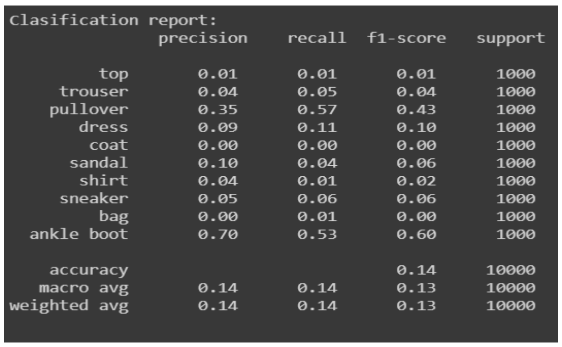
Task 1

Using k-means clustering an accuracy of **48.43%** was obtained. Since the actual labels are not aligned with cluster labels, we manually align them through code (figure 9). On successful alignment we map the clusters (figure 10[a]).

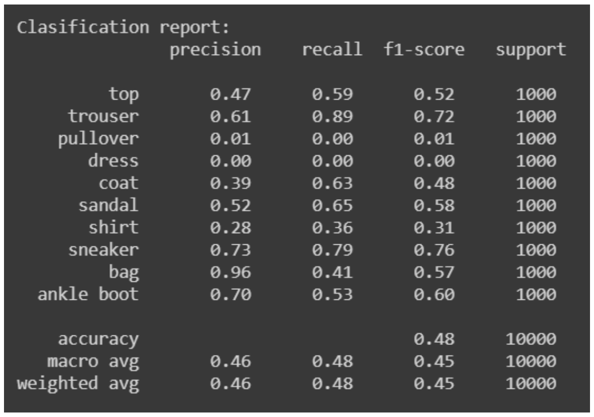
We also create the confusion matrix (figure 10[b]).



Cluster Value	Actual Label
0	8
1	0
2	4
3	6
4	7
5	3
6	2
7	5
8	1
9	9



```
Clasification report:
              precision    recall  f1-score   support

         top       0.01      0.01      0.01      1000
     trouser       0.04      0.05      0.04      1000
    pullover       0.35      0.57      0.43      1000
       dress       0.09      0.11      0.10      1000
        coat       0.00      0.00      0.00      1000
      sandal       0.10      0.04      0.06      1000
       shirt       0.04      0.01      0.02      1000
     sneaker       0.05      0.06      0.06      1000
         bag       0.00      0.01      0.00      1000
  ankle boot       0.70      0.53      0.60      1000

    accuracy                           0.14     10000
   macro avg       0.14      0.14      0.13     10000
weighted avg       0.14      0.14      0.13     10000
```



```
Clasification report:
              precision    recall  f1-score   support

         top       0.47      0.59      0.52      1000
     trouser       0.61      0.89      0.72      1000
    pullover       0.01      0.00      0.01      1000
       dress       0.00      0.00      0.00      1000
        coat       0.39      0.63      0.48      1000
      sandal       0.52      0.65      0.58      1000
       shirt       0.28      0.36      0.31      1000
     sneaker       0.73      0.79      0.76      1000
         bag       0.96      0.41      0.57      1000
  ankle boot       0.70      0.53      0.60      1000

    accuracy                           0.48     10000
   macro avg       0.46      0.48      0.45     10000
weighted avg       0.46      0.48      0.45     10000
```

Figure 9 Accuracy after mapping actual labels with cluster labels for k-means

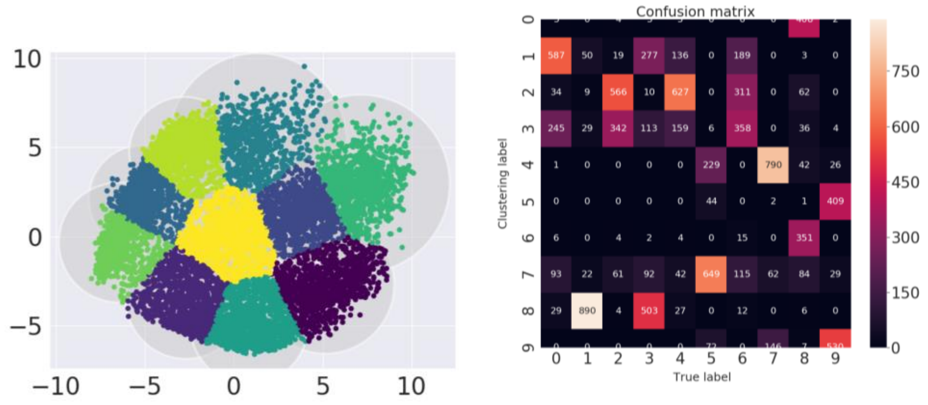
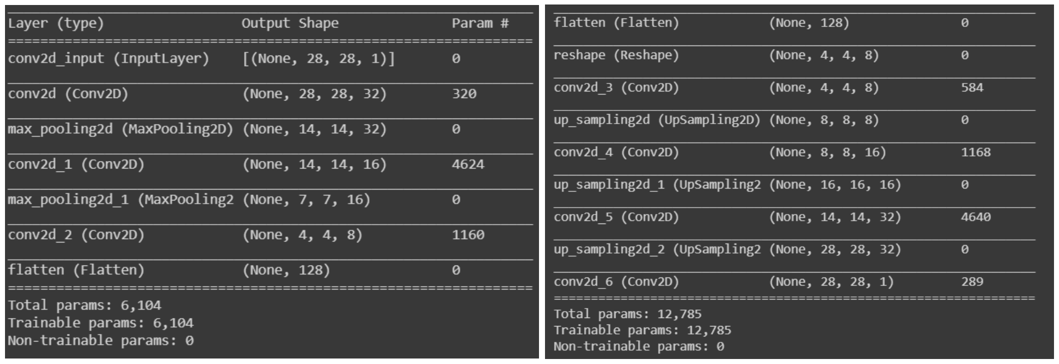


Figure 10 (a) k-means cluster representation (b) Confusion matrix of k-means algorithm

Task 2 and 3

We train the model using encoders and decoders. The autoencoder has the following architecture:



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_input (InputLayer)    [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 16)        4624
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 8)           1160
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
=================================================================
Total params: 6,104
Trainable params: 6,104
Non-trainable params: 0
```

```
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
reshape (Reshape)            (None, 4, 4, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 16)          1168
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 32)        4640
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 12,785
Trainable params: 12,785
Non-trainable params: 0
```

Figure 11 (a) Encoder model (b) Decoder model

Together the above two models form the autoencoder architecture.

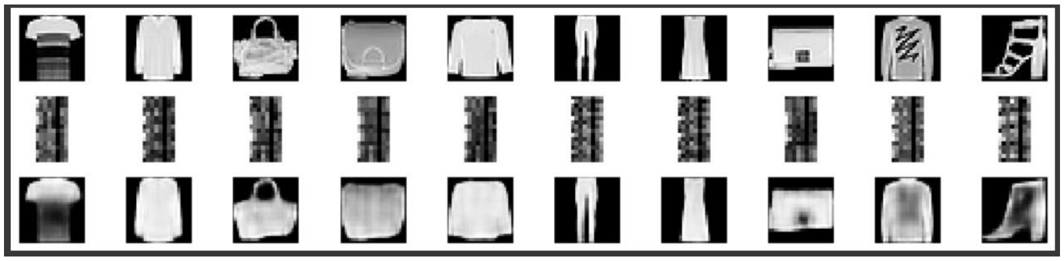


Figure 12 Visualization of how autoencoder compresses image and decodes the image. (a) Input Image, (b) Compressed Image, (c) Output Image

The training vs validation loss is represented below:

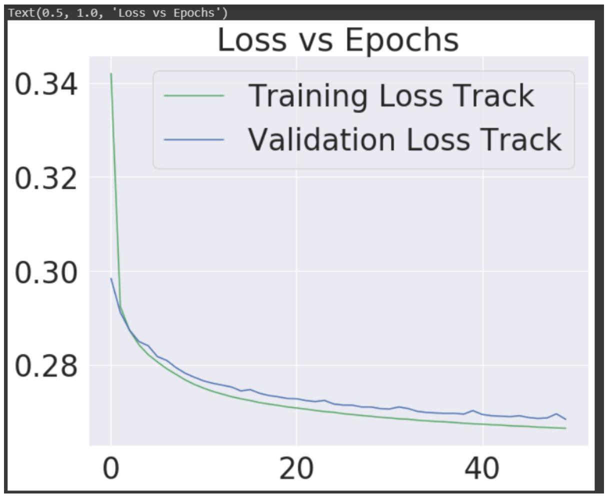
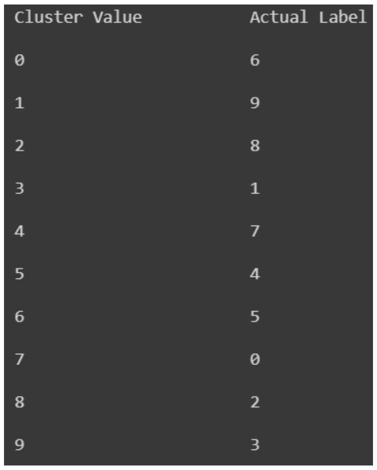
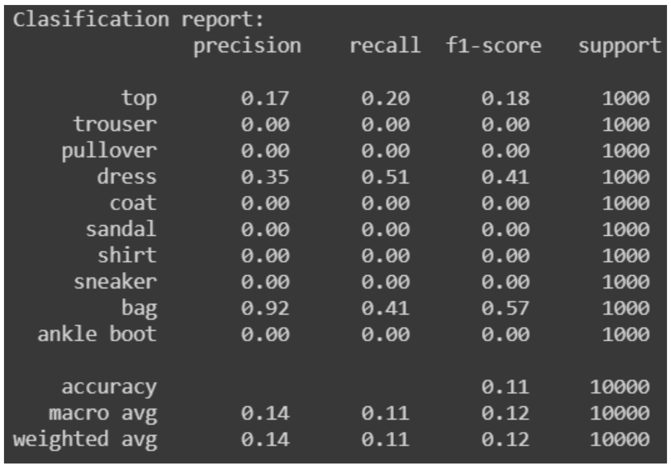


Figure 13 Loss vs Epochs

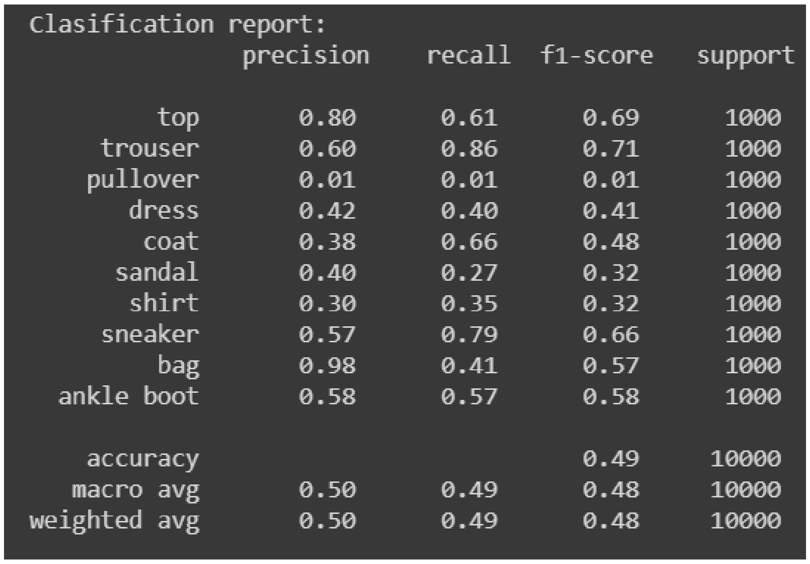
An accuracy of **50.97%** was obtained for **Task 2 (Autoencoder using k-means).**



Cluster Value	Actual Label
0	6
1	9
2	8
3	1
4	7
5	4
6	5
7	0
8	2
9	3



```
Clasification report:
              precision    recall  f1-score   support

         top       0.17      0.20      0.18      1000
     trouser       0.00      0.00      0.00      1000
    pullover       0.00      0.00      0.00      1000
       dress       0.35      0.51      0.41      1000
        coat       0.00      0.00      0.00      1000
      sandal       0.00      0.00      0.00      1000
       shirt       0.00      0.00      0.00      1000
     sneaker       0.00      0.00      0.00      1000
         bag       0.92      0.41      0.57      1000
  ankle boot       0.00      0.00      0.00      1000

    accuracy                           0.11     10000
   macro avg       0.14      0.11      0.12     10000
weighted avg       0.14      0.11      0.12     10000
```



```
Clasification report:
              precision    recall  f1-score   support

         top       0.80      0.61      0.69      1000
     trouser       0.60      0.86      0.71      1000
    pullover       0.01      0.01      0.01      1000
       dress       0.42      0.40      0.41      1000
        coat       0.38      0.66      0.48      1000
      sandal       0.40      0.27      0.32      1000
       shirt       0.30      0.35      0.32      1000
     sneaker       0.57      0.79      0.66      1000
         bag       0.98      0.41      0.57      1000
  ankle boot       0.58      0.57      0.58      1000

    accuracy                           0.49     10000
   macro avg       0.50      0.49      0.48     10000
weighted avg       0.50      0.49      0.48     10000
```

Figure 14 Accuracy after mapping actual labels with cluster labels for k-means using autoencoder

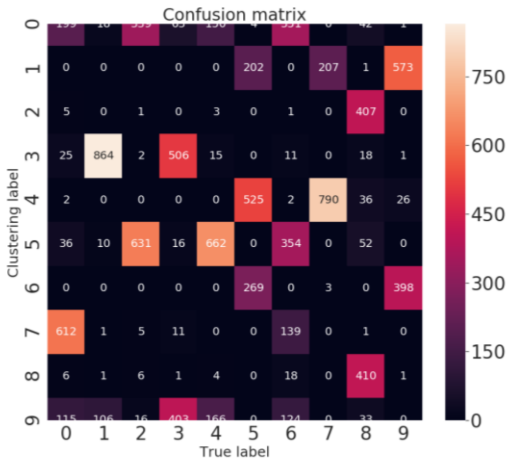
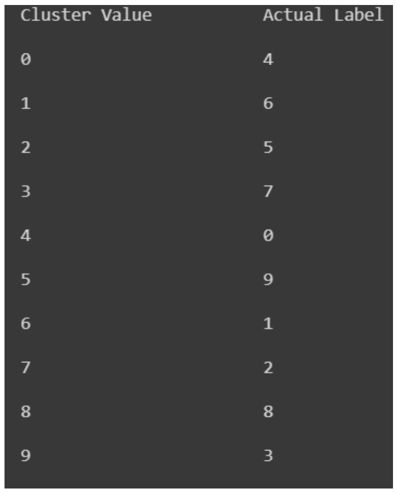
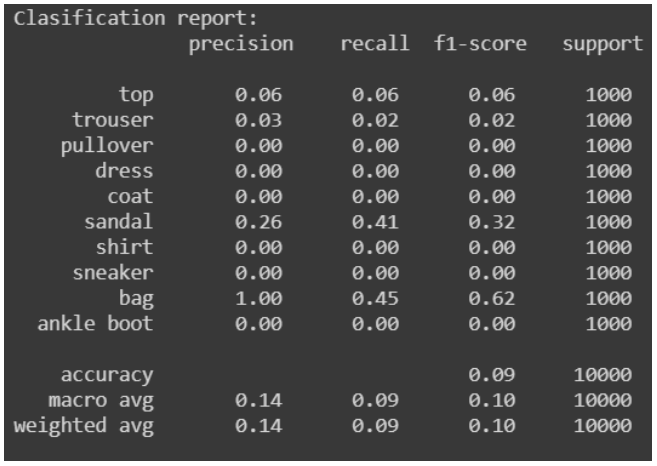


Figure 15 Confusion matrix of k-means using autoencoder

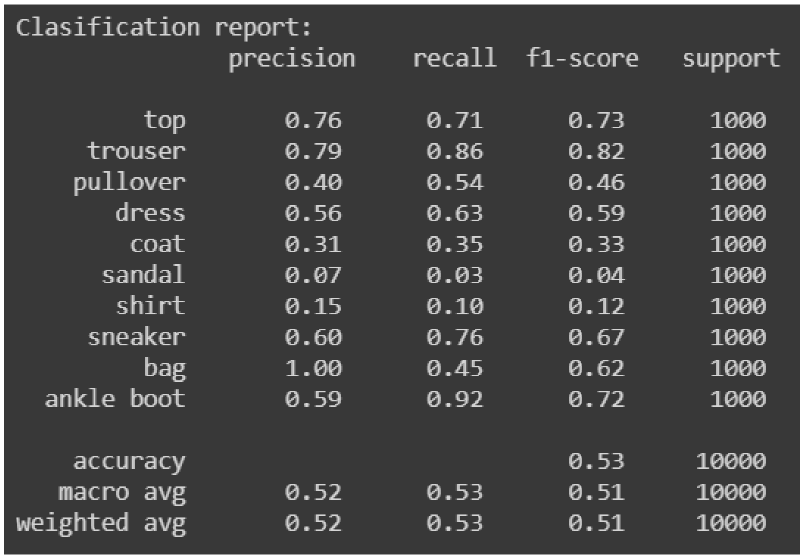
For **Task 3 (Autoencoder using GMM)** an accuracy of **53.9%** was achieved, which is the personal best until now.



```
Cluster Value                    Actual Label

0                                4

1                                6

2                                5

3                                7

4                                0

5                                9

6                                1

7                                2

8                                8

9                                3
```



```
Clasification report:
              precision    recall  f1-score   support

         top       0.06      0.06      0.06      1000
     trouser       0.03      0.02      0.02      1000
    pullover       0.00      0.00      0.00      1000
       dress       0.00      0.00      0.00      1000
        coat       0.00      0.00      0.00      1000
      sandal       0.26      0.41      0.32      1000
       shirt       0.00      0.00      0.00      1000
     sneaker       0.00      0.00      0.00      1000
         bag       1.00      0.45      0.62      1000
  ankle boot       0.00      0.00      0.00      1000

    accuracy                           0.09     10000
   macro avg       0.14      0.09      0.10     10000
weighted avg       0.14      0.09      0.10     10000
```



```
Clasification report:
              precision    recall  f1-score   support

         top       0.76      0.71      0.73      1000
     trouser       0.79      0.86      0.82      1000
    pullover       0.40      0.54      0.46      1000
       dress       0.56      0.63      0.59      1000
        coat       0.31      0.35      0.33      1000
      sandal       0.07      0.03      0.04      1000
       shirt       0.15      0.10      0.12      1000
     sneaker       0.60      0.76      0.67      1000
         bag       1.00      0.45      0.62      1000
  ankle boot       0.59      0.92      0.72      1000

    accuracy                           0.53     10000
   macro avg       0.52      0.53      0.51     10000
weighted avg       0.52      0.53      0.51     10000
```

Figure 16 Accuracy after mapping actual labels with cluster labels for GMM using autoencoder

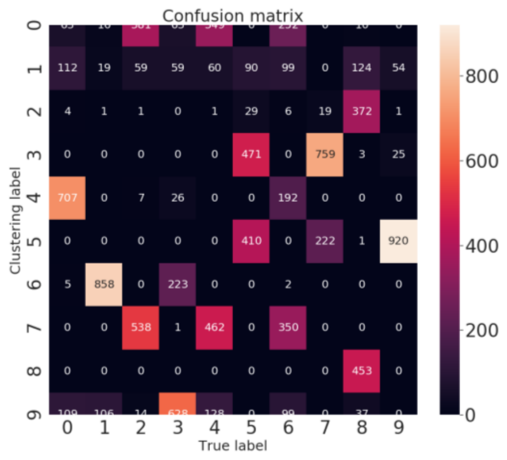


Figure 17 Confusion matrix of GMM using autoencoder

6 Conclusion

It can be concluded from our results that unsupervised learning can be used to cluster fashion MNIST data set efficiently. After training on approximately 60k samples with 784 features, we were able to create a model that segregated data into 10 clusters with an accuracy of **54%** (**autoencoders using GMM**). We were able to improve the accuracy of the model by implementing autoencoders. With a faster GPU and better machine configuration, the training of the model can be more vigorous, thereby increasing the accuracy of the model.

References

[1] https://www.mathworks.com/discovery/unsupervised-learning.html

[2] https://medium.com/machine-learning-for-humans/unsupervised-learning-f45587588294

[3] https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e

[4] https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

[5] https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

[6] https://www.kaggle.com/residentmario/keras-optimizers

[7] https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/

[8] https://arxiv.org/abs/1412.6980

[9] Andrew ng Machine Learning Course: https://www.coursera.org/learn/machine-learning/home/welcome