Unsupervised Learning

Soumita Das
University at Buffalo
UBIT Name: soumitad
UB Person Number: 50320170
soumitad@buffalo.edu

Abstract

Unsupervised Learning refers to the problem in Machine Learning when class labels of instances are not provided. In these types of problems the instances are clustered based on their feature similarities. In this work we have used K-Means and Gaussian Mixture Model (GMM) for clustering the MNIST Fashion Dataset. To address the problem of the large number of features we used Autoencoders for dimension reduction. Autoencoders are used to compress the images in the dataset, however it results in a lossy compression. However, the reduced dimesions of the images results in K-Means and GMM running more efficiently. Results have been compared for all the experiments using the above three concepts.

1 Introduction

Machine learning problems can be of two types, *supervised* and *unsupervised*. Classification that depends on the attributes and target variables of the training set, and then go on to predict class labels for unknown test samples is a category of *supervised* learning. On the other hand clustering techniques which are *unsupervised* learning groups input samples based on the homogeneity among their features.

In this project we address unsupervised learning. For clustering we firstly use K-Means algorithm. K-Means algorithm starts with a first group of randomly selected K centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids. The algorithm has been discussed in detail in the following sections.

However since K-Means is more inclined towards finding spherical clusters and perform hard clustering, we use the concept of Gaussian Mixture Model which is not dependant on shapes and perform soft clustering. Gaussian mixture models are a probabilistic model for representing normally distributed subpopulations within an overall population. Mixture models in general don't require knowing which subpopulation a data point belongs to, allowing the model to learn the subpopulations automatically. Since subpopulation assignment is not known, this constitutes a form of unsupervised learning.

Finally we use Autoencoder models to reduce the dimensionality of our Fashion MNIST dataset so that the K-Means and GMM model can run more efficiently. Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible.

2 Algorithms and Architectures

2.1 K-Means Clustering

The algorithm starts by initializing a random set of K points as centroids (μ) across the dataset. In each update step, all instances x are assigned to their nearest centroid μ (see equation 1). Each point must only be assigned to one centroid. If one instance is equally close to two or more centroids then one assignment will be chosen at random.

$$S_i^{(t)} = \left\{ x_p : \left\| x_p - \mu_i^{(t)} \right\|^2 \leq \left\| x_p - \mu_j^{(t)} \right\|^2 \forall j, 1 \leq j \leq K \right\} \tag{1}$$

Following this, a new centroid is assigned to each cluster by calculating the mean or average of all the data points belonging to that cluster (see equation 2).

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2}$$

The updating process continues till none of the centroids change their respective values.

2.2 Gaussian Mixture Model

When the data follows a mixture model i.e. the data looks multimodal, trying to fit a multimodal distribution with a unimodal (one "peak") model will generally give a poor fit. Since many simple distributions are unimodal, an obvious way to model a multimodal distribution would be to assume that it is generated by multiple unimodal distributions. For several theoretical reasons, the most commonly used distribution in modeling real-world unimodal data is the Gaussian distribution.

Thus, modeling multimodal data as a mixture of many unimodal Gaussian distributions makes intuitive sense. Furthermore, GMMs maintain many of the theoretical and computational benefits of Gaussian models, making them practical for efficiently modeling very large datasets. A multi-dimensional Gaussian model is shown below in Figure 1.

$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i)$$

$$\mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\left(-\frac{1}{2} (\vec{x} - \vec{\mu}_i)^{\mathrm{T}} {\Sigma_i}^{-1} (\vec{x} - \vec{\mu}_i) \right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

Figure 1: Gaussian Mixture Model Equations

where μ and $\sum$ are the mean and covariance matrix of the gaussian distribution respectively.

The EM Algorithm is as follows:
1. Randomly initialize Gaussian parameters.
2. E-Step : Compute the expected membership values for each data point for each cluster.
3. M-Step : Recompute Gaussian parameters using maximum likelihood estimation of parameters.

2.3 Auto-encoders

Autoencoders (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 :

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).

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 architecture of Autoencoders are shown in Figure 2.

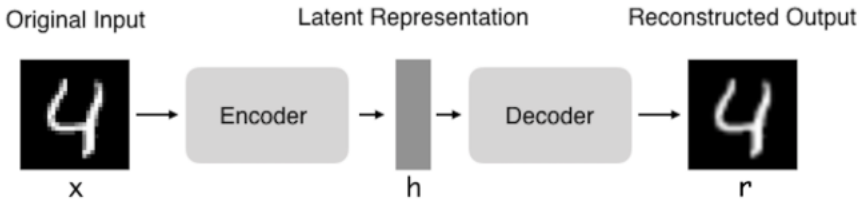

Figure 2: Architecture of Autoencoder

We mainly used the encoder portion of the autoencoder for dimension reduction. In this project three types of architecture for the autoencoder model has been used. Figures 3,4,5 shows the architecture of a simple autoencoder, deep autoencoder and convolution autoencoder.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
```

Figure 3: Simple Autoencoder

3 Experimental Setup

3.1 Data set

For training and testing of our clustering algorithms, we will use the Fashion-MNIST dataset. The 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.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_6 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_7 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_8 (Dense)              (None, 784)               101136
=================================================================
Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0
```

Figure 4: Deep Autoencoder

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
reshape_1 (Reshape)          (None, 4, 4, 8)           0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 1)         145
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
```

Figure 5: Convolution Autoencoder

3.2 Results and Discussions

The table below shows the comparison in results for K-Means and Gaussian Mixture Model after applying the three types of autoencoders discussed above. For experimental results we use accuracy

by mapping the clusters formed to our test labels given using linear assignment algorithm in sklearn. We also use the normalized mutual info score (NMI) and the adjusted mutual info score (AMI) from the sklearn.metrics. The following table shows the results. As we see in the table below, the Convolution autoencoder outperforms the Simple and the Deep autoencoders.

Algorithm	**Accuracy**	**NMI Score**	**AMI Score**
K-Means	0.4828	0.5128	0.5003
K-Means with Simple Autoencoder	0.4787	0.4784	0.4709
GMM with Simple Autoencoder	0.5376	0.5854	0.5669
K-Means with Deep Autoencoder	0.4705	0.4832	0.4732
GMM with Deep Autoencoder	0.6141	0.5995	0.5815
K-Means with Convolution Autoencoder	0.4811	0.5499	0.5387
GMM with Convolution Autoencoder	0.5609	0.6104	0.6011

Figure 6 shows the application of the convolution autoencoder on part of the Fashion MNIST dataset.

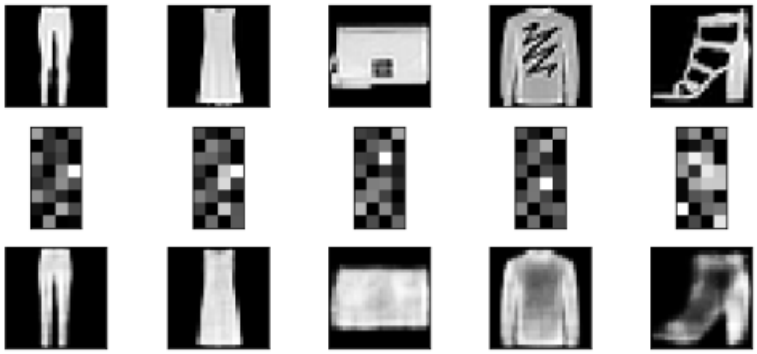

Figure 6: Convolution Autoencoder on MNIST data

For all the three architectures of autoencoders the training and validation loss has been shown for every epoch in Figures 7,8 and 9.

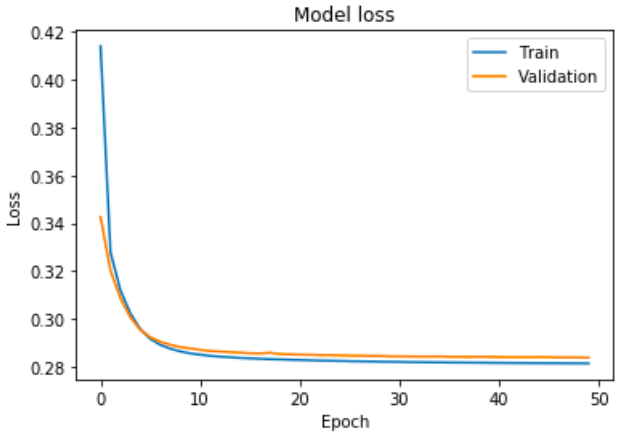

Figure 7: Training vs Validation Loss of Simple Autoencoder

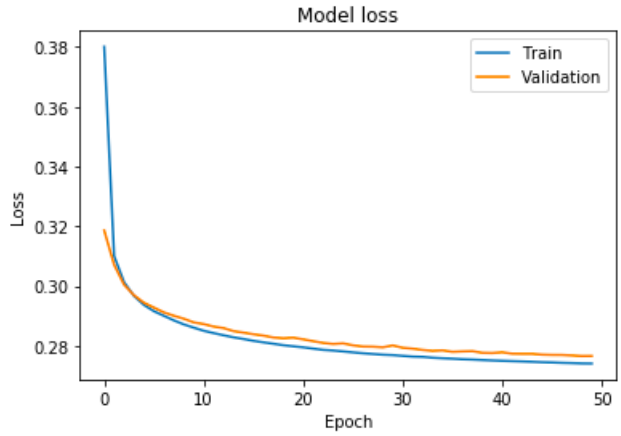

Figure 8: Training vs Validation Loss of Deep Autoencoder

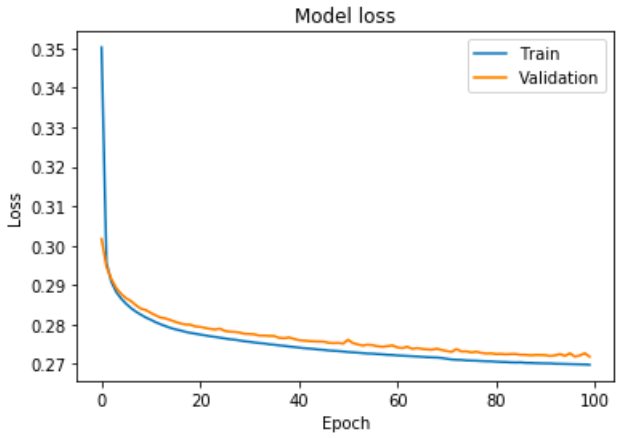

Figure 9: Training vs Validation Loss of Convolution Autoencoder

4 Conclusion

In this work the fashion MNIST dataset was clustered using the basic K-Means algorithm and the Gaussian Mixture model algorithm. To address the problem of high dimensionality, the concept of autoencoders have been used. The performance of three different types of autoencoders have been compared : a simple one, a deep autoencoder and finally an autoencoder with convolution layers. Results show how the accuracy, normalized_mutual_info_score(NMI) and adjusted_mutual_info_score (AMI) vary for each of the following.

5 References

[1] Pattern Recognition and Machine Learning by Christopher M. Bishop
[2] https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f
[3] https://brilliant.org/wiki/gaussian-mixture-model/
[4]https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1