Analyzing Clustering Algorithms on Fashion MNIST dataset

Amrata Raykar
Department of Computer Science
University at Buffalo
Buffalo, NY 14214

Abstract

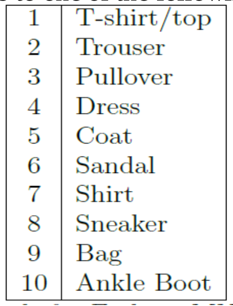
This project provides the clear understanding of features of unsupervised learning. We can see the performance difference between K-Mean clustering and Gaussian Mixture Matrix Clustering on Fashion MNIST dataset to classify datapoints into 10 clusters. An autoencoder lets us re-represent high dimensional points in a lower-dimensional space called as 'latent space'. We use this reduced dimensional representation for clustering. We can notice the accuracy improving over baseline K-means algorithm when we use Autoencoding and GMM clustering.

1 Introduction

In this project, we classify the datasets of Fashion MNIST into 10 clusters. Clustering is the process of grouping the datapoints into several groups called Clusters such that data points in same group are similar to each other. This determines intrinsic grouping among unlabeled dataset. We used Autoencoder to reduce the dimensionality of dataset to get better accuracy in clustering. This lower dimensional space is called as Latent Space. For clustering purpose, we use this dataset in latent space to improve on accuracy.
A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

2 Dataset

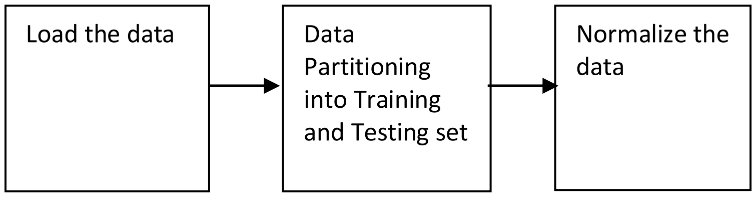
In this project we have used Fashion-MNIST dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Each image is 28 pixels 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.
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 belongs to one of the following cluster:



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

3 Pre - Processing

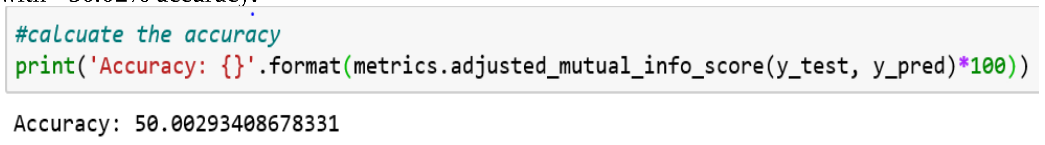
This process of scaling the data to a common range for the dataset is known as Normalization. We first partitioned the data and then normalized it and not vice-versa otherwise there is a chance of normalization of training data being affected by the values of validation and testing set. The dataset was split into two parts as training (60,000 samples) and testing (10,000 samples) set. Normalization of both the sets are done dividing each data point by 255, because 255 is the highest possible value of the pixel in a grayscale image. This process of scaling the data to a common range for the dataset is known as **Normalization**.



4 Architecture

The **K-Means** algorithm clusters data by trying to separate samples in n groups of equal variance. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields. The k-means algorithm divides a set of samples into disjoint clusters, each described by the mean of the samples in the cluster. The means are commonly called the cluster centroids; note that they are not, in general, points from set of samples, although they live in the same space.

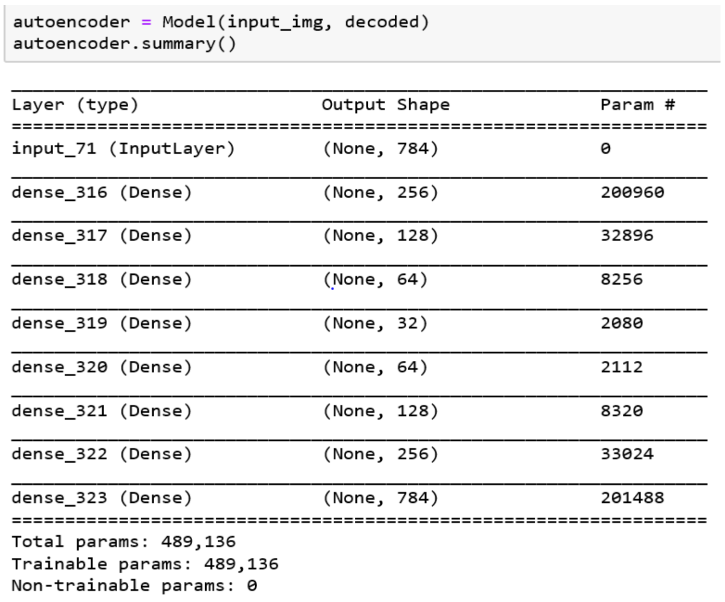
In Fashion MNIST dataset, we have dataset grouped into 10 clusters. This project clusters the data with ~50.02% accuracy.



```
#calcuate the accuracy
print('Accuracy: {}'.format(metrics.adjusted_mutual_info_score(y_test, y_pred)*100))
```

```
Accuracy: 50.00293408678331
```

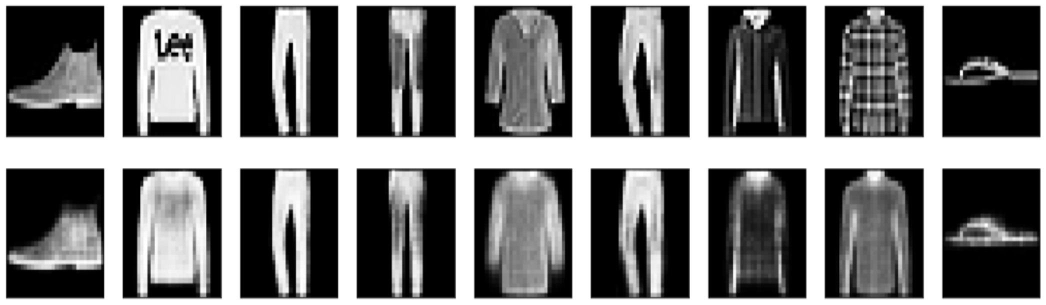
Autoencoding is a data compression algorithm where the compression and decompression functions are data-specific, lossy, and learned automatically from examples rather than engineered by any human. Clustering is difficult to do in high dimensions because the distance between most pairs of points is similar. An autoencoder lets us re-represent high dimensional points in a lower-dimensional space called as 'latent space'. We use this reduced dimensional representation for clustering.



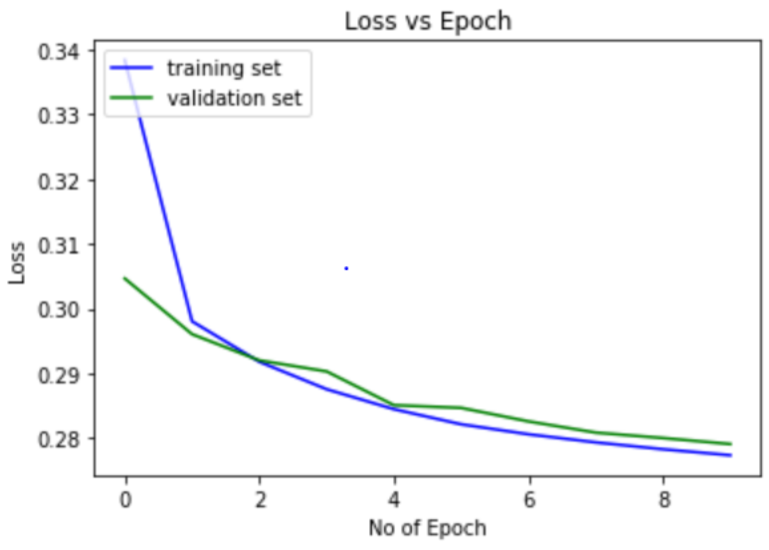
```
autoencoder = Model(input_img, decoded)
autoencoder.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_71 (InputLayer)        (None, 784)               0         
_________________________________________________________________
dense_316 (Dense)            (None, 256)               200960    
_________________________________________________________________
dense_317 (Dense)            (None, 128)               32896     
_________________________________________________________________
dense_318 (Dense)            (None, 64)                8256      
_________________________________________________________________
dense_319 (Dense)            (None, 32)                2080      
_________________________________________________________________
dense_320 (Dense)            (None, 64)                2112      
_________________________________________________________________
dense_321 (Dense)            (None, 128)               8320      
_________________________________________________________________
dense_322 (Dense)            (None, 256)               33024     
_________________________________________________________________
dense_323 (Dense)            (None, 784)               201488    
=================================================================
Total params: 489,136
Trainable params: 489,136
Non-trainable params: 0
```

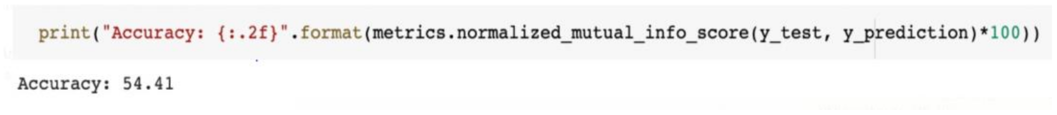
Autoencoders are lossy, which means that the decompressed outputs will be degraded compared to the original inputs. This can be seen in the image below. The first line of images are input datasets. Once passed through the encoder, the compressed images are degraded in dimensionality which is shown in the second line of images.



We can notice the loss recuding over the epoches for both training and validation set.



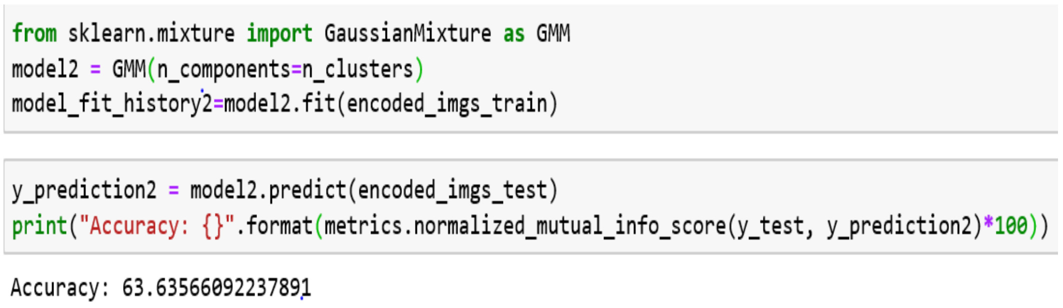
On using K-Means algorithm along with Autoencoder layer provides us the accuracy of ~54.41%.



```
print("Accuracy: {:.2f}".format(metrics.normalized_mutual_info_score(y_test, y_prediction)*100))
```

```
Accuracy: 54.41
```

A **Gaussian mixture model** is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.
The GaussianMixture object implements the expectation-maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models and compute the Bayesian Information Criterion to assess the number of clusters in the data. A GaussianMixture.Fit method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belongs to using the GaussianMixture.predict method.



```
from sklearn.mixture import GaussianMixture as GMM
model2 = GMM(n_components=n_clusters)
model_fit_history2=model2.fit(encoded_imgs_train)
```

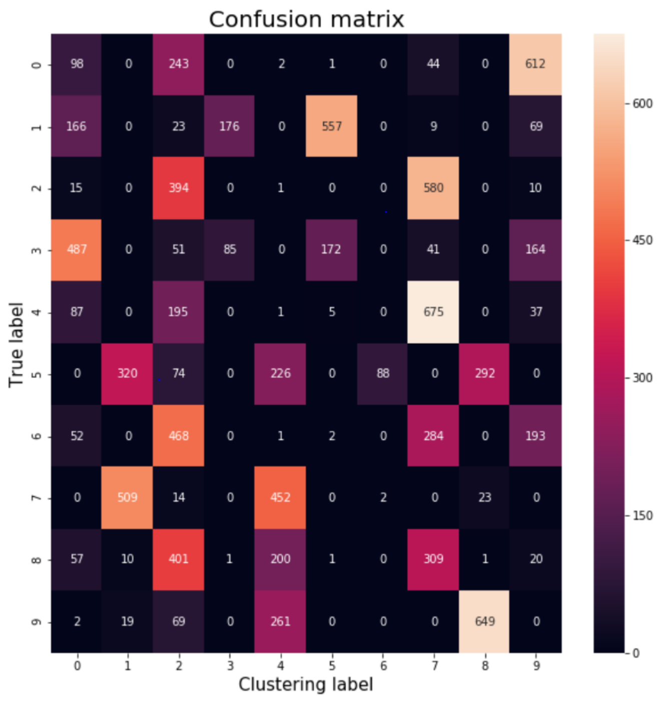
```
y_prediction2 = model2.predict(encoded_imgs_test)
print("Accuracy: {}".format(metrics.normalized_mutual_info_score(y_test, y_prediction2)*100))
```

```
Accuracy: 63.63566092237891
```

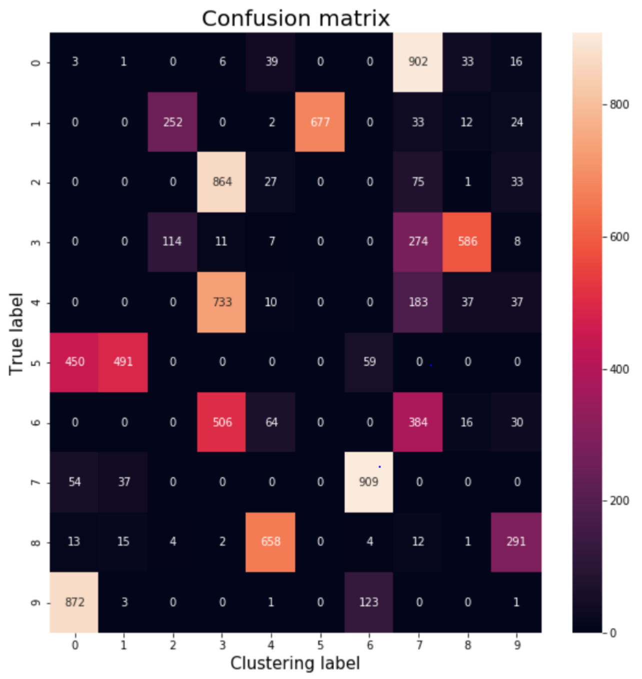
5 Results

After training the model on training data we evaluate the performance of the model on the test data. The accuracy of the model is evaluated using the **Confusion Matrix**. A confusion matrix is often used to describe the performance of the classification models for which the true values are already known.

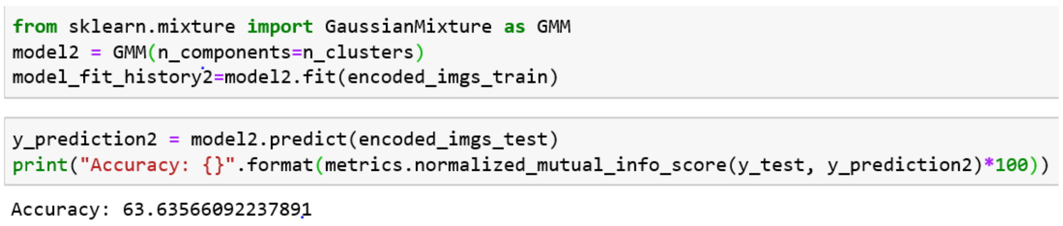
Confusion matrix for model of K-mean using Deep neural network and Autoencoder layer is given below:



Confusion matrix for model of using Deep neural network is given below:



We can see the accuracy improving using Gaussian Mixture Model with Autoencoder. I.e., ~63.63%



```
from sklearn.mixture import GaussianMixture as GMM
model2 = GMM(n_components=n_clusters)
model_fit_history2=model2.fit(encoded_imgs_train)
```

```
y_prediction2 = model2.predict(encoded_imgs_test)
print("Accuracy: {}".format(metrics.normalized_mutual_info_score(y_test, y_prediction2)*100))
```

```
Accuracy: 63.63566092237891
```

6 Conclusion

Through this project we learn the basics of unsupervised learning and clustering which uses Autoencoding to reduce the dimensionality of input dataset and hence increase the accuracy. Using Gaussian Mixture Matrix Algorithm for Clustering we accurately group unlabeled data with accuracy upto 63% which is improved over K-Mean clustering method in Neural Networks. We can fine tune our hyper parameters be it epochs and batch size such that our model converges faster without the problem of overfitting and at the same time generalize the solution for unseen test data.