Cluster Analysis using Unsupervised Learning

Deepanshu Yadav
UB Person No:
50321285
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
dyadav@buffalo.edu

Abstract

In this study, cluster analysis was performed on Fashion MNIST dataset using unsupervised machine learning. It was observed that after using KMeans algorithm to cluster original data space in the first part of the study, the model was not able to match the Fashion MNIST data to correct classes with significant accuracy. In the latter parts, Auto-Encoder based K-Means clustering model and Auto-Encoder based Gaussian Mixture clustering model were developed using open-source libraries, Keras and SKLearns. Performance of the three models in terms of clustering accuracy was compared.

1 Introduction

Unsupervised learning is where we only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning, there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data. Unsupervised learning problems can be further grouped into clustering and association problems. A clustering problem is where we want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

In the first part of this study, clustering analysis was done using KMeans clustering algorithm. The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. 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.

In the second part, Auto-Encoder based K-Means clustering model was built to cluster the condensed representation of the unlabeled fashion MNIST dataset. “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 a human. Autoencoder uses compression and decompression functions which are implemented with neural networks.

In the third part, Auto-Encoder based Gaussian Mixture Model clustering model was built to cluster the condensed representation of the unlabeled fashion MNIST dataset. 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 Related Work

The work in this report includes reading the dataset and preprocessing it before using it for clustering analysis using the various models. The first model uses KMeans algorithm which 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 the second model, a convolutional autoencoder is trained and optimum solution is reached when loss is minimized and accuracy is significant. The results are plotted against epochs to obtain the loss curves to check improvements with every epoch. Then the data from autoencoder is used to get output using KMeans. In the third part, Gaussian Mixture Model is used to generate the predicted labels. The following sections of the report include dataset reading, pre-processing the data, developing model architecture and then the results for all the three models. These present detailed analyses of each step involved in developing the models and testing them.

3 Dataset

For training and testing of the classifiers, the Fashion-MNIST dataset was used. 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 (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

Fig 1: Example of how the data looks like

Each training and test example are assigned to one of the labels as shown in table 1.

Table 1: Labels for Fashion-MNIST dataset

S. No.	Description
1	T-shirt/top
2	Trouser
3	Pullover
4	Dress
5	Coat
6	Sandal
7	Shirt
8	Sneaker
9	Bag
10	Ankle Boot

4 Pre - Processing

The datasets were loaded in Python using *util_mnist_reader* provided by authors of the dataset. Preprocessing of the dataset was done using NumPy library of Python.
The dataset containing the 784 attributes was normalized to remove the bias which could have occurred due to large variations in values between the columns as they varied from 0 – 255 in the color range.

When using a convolutional layer as the first layer to our model in the second part of the study while developing Auto-Encoder, we need to reshape our data to (n_images, x_shape, y_shape, channels). The channels are set to 1 for grayscale images which we have in our dataset.

X_train = X_train.reshape(-1, 28,28, 1)
X_test = X_test.reshape(-1, 28,28, 1)

5 Model Architecture

Overview: K-Means is one of the simplest unsupervised learning algorithms that solve the clustering problems. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). The main idea is to define k centers, one for each cluster.

5.1 K-Means:

To process the learning data, the K-means algorithm starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids
It halts creating and optimizing clusters when either the centroids have stabilized, i.e., there is no change in their values because the clustering has been successful or the defined number of iterations has been achieved. In this algorithm, firstly, we have to randomly initialize points called the cluster centroids (K). K-means is an iterative algorithm and it does two steps:

5.1.1 Cluster assignment: The algorithm goes through each of the data points and depending on which cluster is closer, it assigns the data points to one of the three cluster centroids.

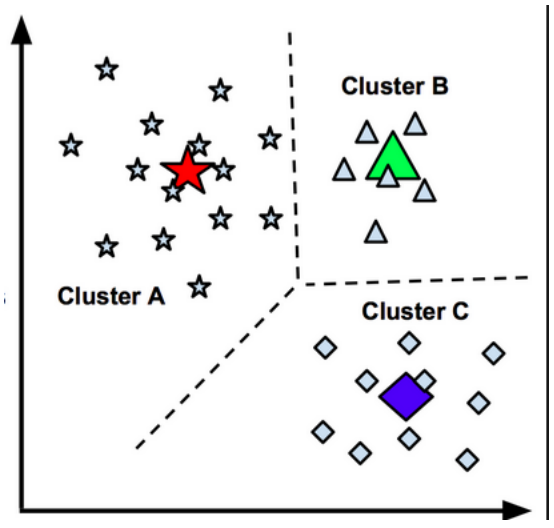


Fig 2: Data being clustered

5.1.2 Move centroid step: Here, K-means moves the centroids to the average of the points in a cluster. In other words, the algorithm calculates the average of all the points in a cluster and moves the centroid to that average location. This process is repeated until there is no change in the clusters (or possibly until some other stopping condition is met). K is chosen randomly or by giving specific initial starting points by the user. In this study, we assign K as 10.

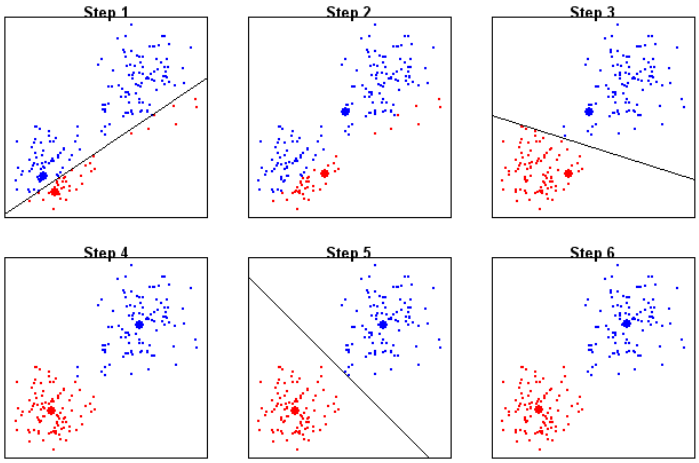


Fig 3: K-Means at work

5.2 Gaussian Mixture Model:

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 below mentioned equation defines a Gaussian Mixture.

$$p(X) = \sum_{k=1}^{K} \pi_k N(x \mid \mu_k, \Sigma_k)$$

where,

$p(X)$ = Probability density as a linear function of densities of all K distributions

π_k = Mixing coefficient for k-th distribution

μ_k = Mean for k-th distribution

Σ_k = Covariance for k-th distribution

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.

K-means model is that it 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. This works well when data is circular as shown in Fig 4. But it fails when data is of other shapes as shown in Fig 5.

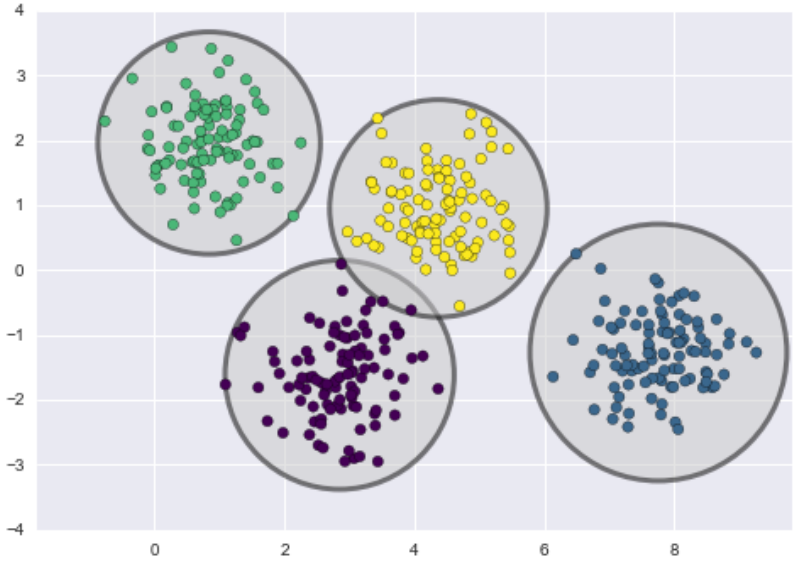


Fig 4: KMeans on Circular Data

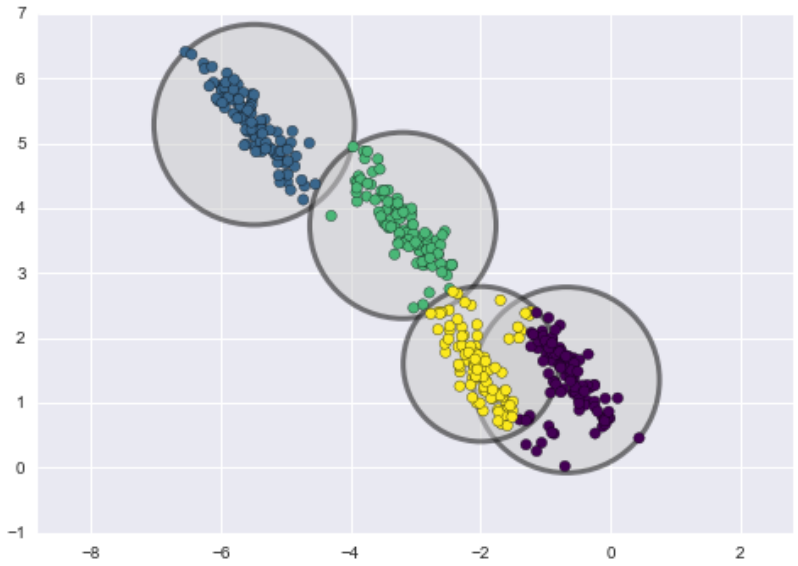


Fig 5: KMeans on Data of Other Shapes

In contrast, Gaussian mixture models can handle even very oblong clusters as shown belo in Fig 6.

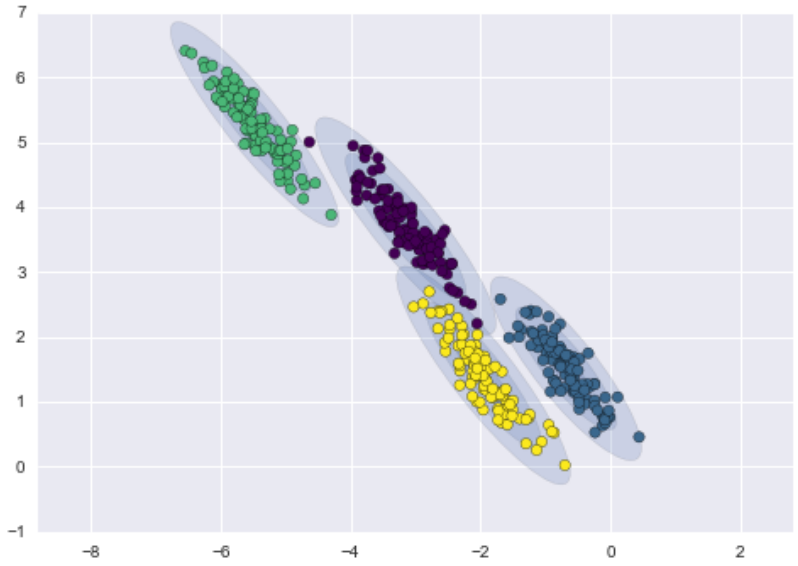


Fig 6: Gaussian Mixture Model in work

5.3 Auto-Encoder:

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 a human. Autoencoder uses compression and decompression functions which are implemented with neural networks as shown in Fig 7.

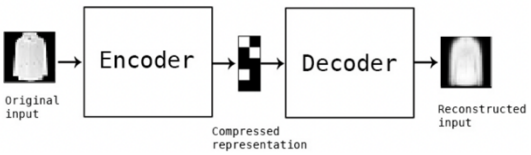


Fig 7: AutoEncoder

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 your 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.

In this study, we built an autoencoder using Conv2D, MaxPooling2D and UpSampling2D classes from Keras library for encoder and decoder. Three convolutional layers were added to the original data. The model summary after adding the layers is presented in the table below.

Table 2: Auto-Encoder Model Summary

Layer (type)	Output Shape	Param #
conv2d_29 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_9 (MaxPooling2	(None, 14, 14, 16)	0
conv2d_30 (Conv2D)	(None, 14, 14, 8)	1160
max_pooling2d_10 (MaxPooling	(None, 7, 7, 8)	0
conv2d_31 (Conv2D)	(None, 4, 4, 8)	584
flatten_5 (Flatten)	(None, 128)	0
reshape_5 (Reshape)	(None, 4, 4, 8)	0
conv2d_32 (Conv2D)	(None, 4, 4, 8)	584
up_sampling2d_13 (UpSampling	(None, 8, 8, 8)	0
conv2d_33 (Conv2D)	(None, 8, 8, 8)	584
up_sampling2d_14 (UpSampling	(None, 16, 16, 8)	0
conv2d_34 (Conv2D)	(None, 14, 14, 16)	1168
up_sampling2d_15 (UpSampling	(None, 28, 28, 16)	0
conv2d_35 (Conv2D)	(None, 28, 28, 1)	145

Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0

Now the autoencoder model is compiled and trained for various number of epochs and batch sizes. This process is repeated for a number of iterations known as epochs, which is also a hyperparameter. Number of epochs are chosen such that cost gets minimized. In this study, epochs are varied for developing the encoder.

5.4 Auto-Encoder with K-Means Clustering:
The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of- squares criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \epsilon C} (\left|\left|x_i - \mu_j\right|\right|^2)$$

Inertia can be recognized as a measure of how internally coherent clusters are. Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high-dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called curse of dimensionality). Running a dimensionality reduction algorithm such as Principal component analysis (PCA) or Auto-encoder prior to k-means clustering can alleviate this problem and speed up the computations.

By training the auto-encoder, we have the encoder learned to compress each image into latent floating-point values. We then used K-Means to generate the cluster centroids, which is the 10 cluster centers in the latent feature space. We then cluster compressed latent codes produced by AutoEncoder using K-Means.

5.5 Auto-Encoder with GMM Clustering:
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.

In this part of the study, the output of the autoencoding is sent as input to the GaussianMixture Model. As the output of the autoencoder in our study is at layer 4, we send this as a parameter to the GMM model to get the clusters and associated accuracy.

5.6 Evaluation Metrics
In this study, two metrics- accuracy and confusion matrix have been used to evaluate the performance of the clustering analysis.

For clustering accuracy, Normalized Mutual Information (NMI) is used. It is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation).

$$NMI(Y, C) = \frac{2 * I(Y; C)}{[H(Y) + H(C)]}$$

where,
Y = Class labels
C = Cluster labels
H(Y) = Entropy of class labels
H(C) = Entropy of cluster labels
I(Y;C) = Mutual Information between Y and C

$$I(Y; C) = H(Y) - H(Y|C)$$

where,
H(Y|C) = Conditional entropy of class labels within each cluster

In this function, mutual information is normalized by some generalized mean of H(labels_true) and H(labels_pred)), defined by the average_method of SKLearns library. This metric is independent of the absolute values of the labels: a permutation of the class or cluster label values won't change the score value in any way.
This metric is furthermore symmetric: switching label_true with label_pred will return the same score value. This can be useful to measure the agreement of two independent label assignments strategies on the same dataset when the real ground truth is not known.

For aligning confusion matrix, Scikit-learn provides a function linear_assignment to apply the Hungarian algorithm. We used it to create aligned confusion matrix for the models. Since linear_assigment method minimizes a cost which should be positive, we defined the function _make_cost_m to transform the confusion matrix into a cost matrix.

6 Results of Experimentation

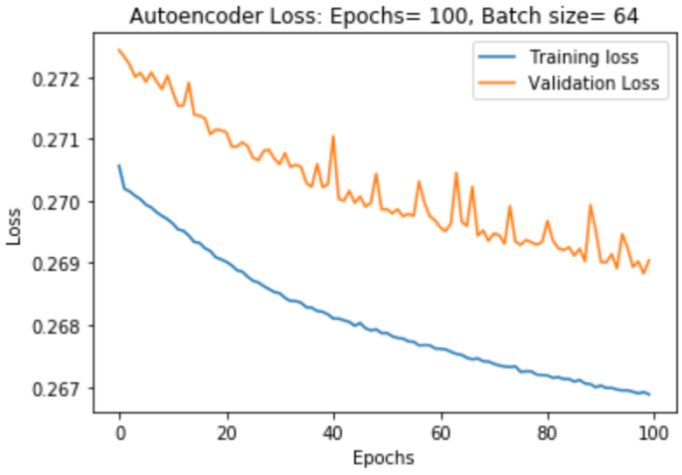
The task to be performed was predicting the class (1 – 10) of each sample with some confidence using KMeans, Auto-Encoder with KMeans, and Auto-Encoder with GaussianMixture model.
Evaluation metrics: I have used Accuracy and Confusion Matrix metrics for each dataset for evaluation of the efficiency of the models. Also, the cost values for training and validation datasets have been evaluated with respect to epochs. As the weights are initialized with random values before running the model every time, the model gives varied values of the evaluation metrics every time the model is run.

K-Means:
The hyperparameters for the first model, i.e. iterations, were set to get the optimum results by trial and error method. Accuracy was calculated using Normalized Mutual Information for various number of iterations.

For iterations = 100, NMI Score for Training Data = 51.07%
For iterations = 100, NMI Score for Test Data = 50.29%

For iterations = 1000, NMI Score for Training Data = 51.17%
For iterations = 1000, NMI Score for Test Data = 51.47%

Auto-Encoder Training:
The hyperparameters for the second model, i.e. epochs, layers and batch size were set to get the optimum results by trial and error method. Cost was calculated for various values and the one with minimum cost was chosen as the final hyperparameters for the model. Figures below represent loss plots for various values of hyperparameters.



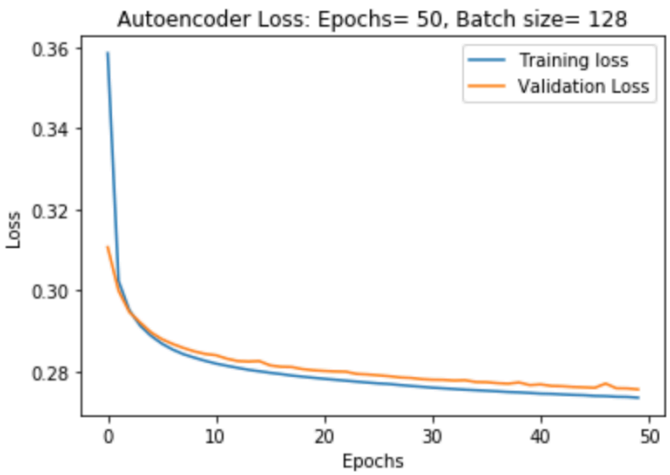


Fig 8: Train and Validation Loss for Epochs: 100, Hidden Layers: 3, Batch size: 64
Fig 9: Train and Validation Loss for Epochs: 50, Hidden Layers: 3, Batch size: 128

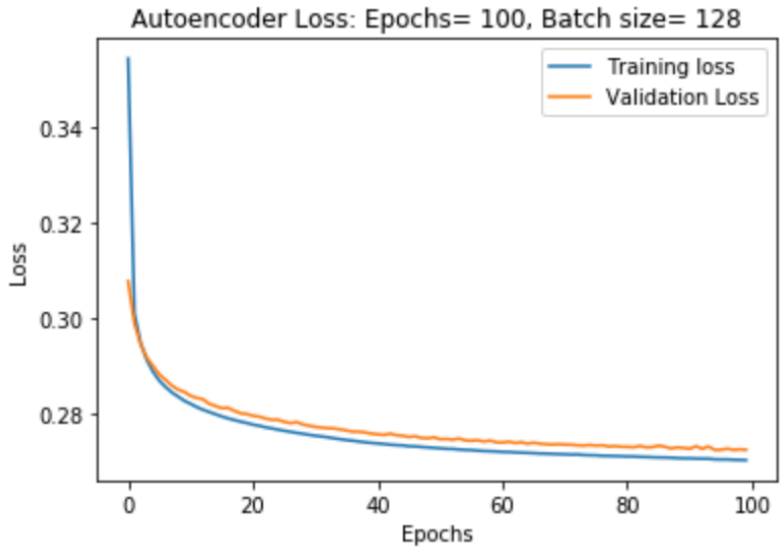


Fig 10: Train and Validation Loss for Epochs: 100, Hidden Layers: 3, Batch size: 128

Auto-Encoder with K-Means Clustering:
The results for final model trained for 100 epochs, with 3 hidden layers and batch size of 128 and then predicted with the help of K-Meansare as mentioned below:

Training Accuracy (NMI Score): 51.67%
Test Accuracy (NMI Score): 54.84%

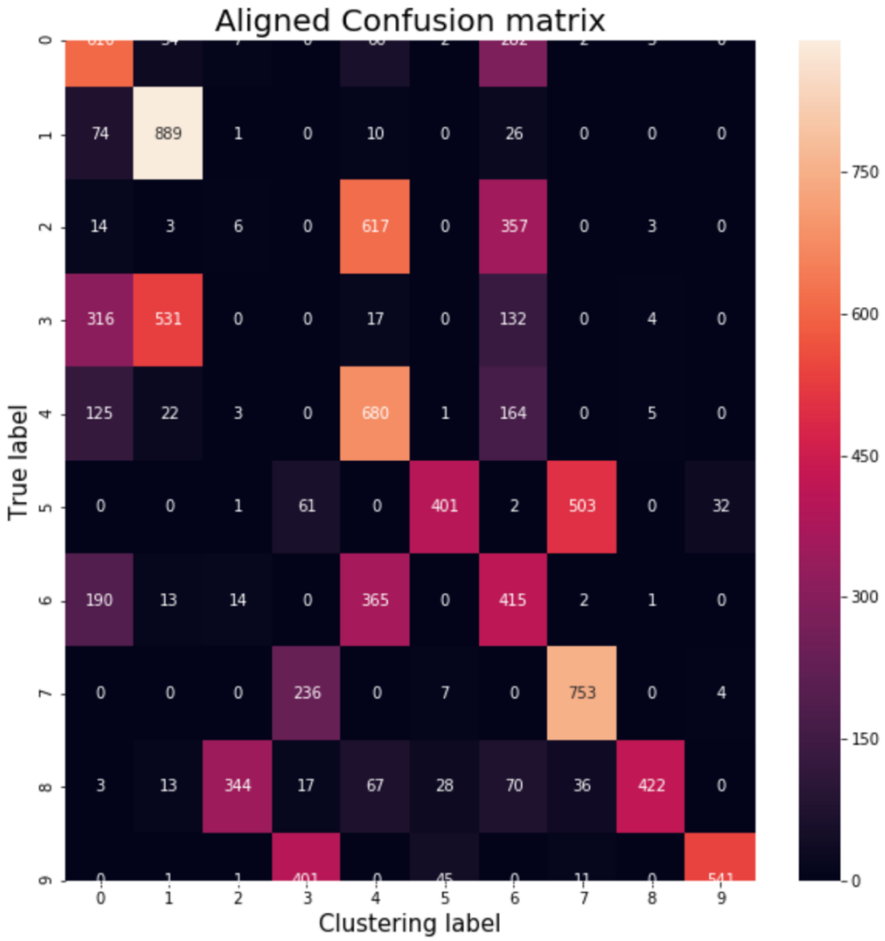


Fig 11: Aligned Confusion Matrix for Auto-Encoder with K-Means Clustering for Test Set

Auto-Encoder with Gaussian Mixture:
The results for final model trained for 100 epochs, with 3 hidden layers and batch size of 128 and then predicted with the help of Gaussian Mixture Model are as mentioned below:

Training Accuracy (NMI Score): 57.05%
Test Accuracy (NMI Score): 60.32%

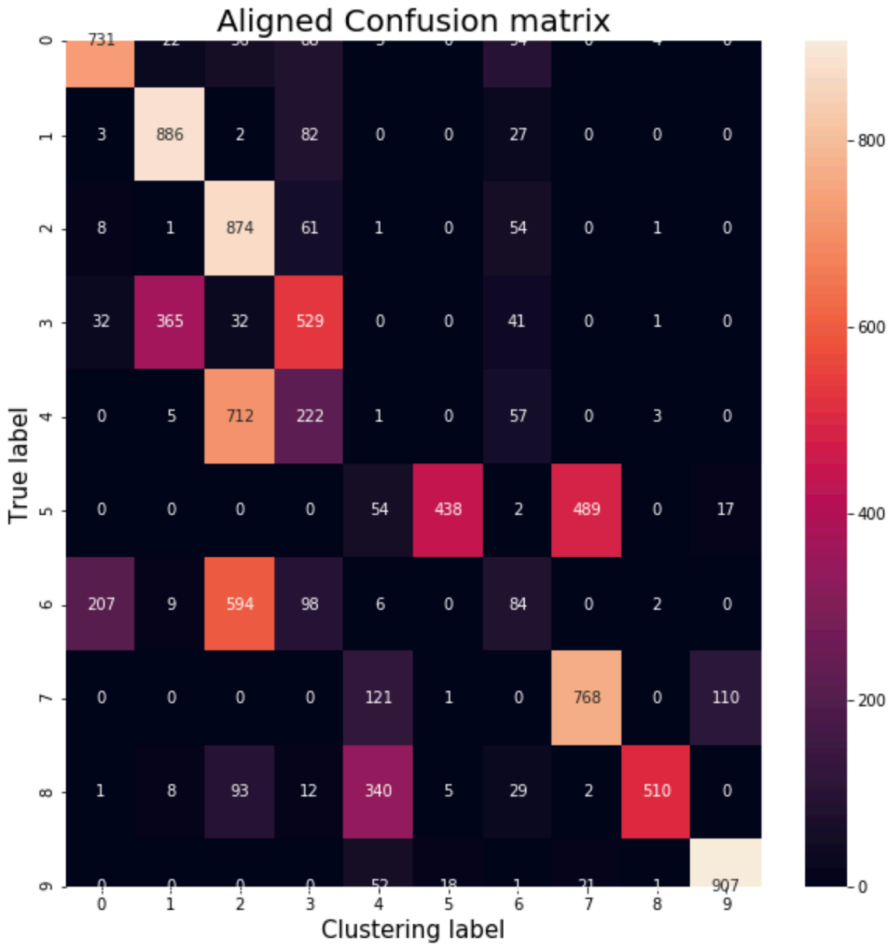


Fig 12: Aligned Confusion Matrix for Auto-Encoder with GMM for Test Set

From the results obtained from the three models, we can observe that the accuracy and confusion matrix are getting refined when we move from K-Means to Auto-Encoder with K-Means and then to Auto-Encoder with GaussianMixtureModel. The accuracy for test set is found to be 52% for K-Means Clustering, 55% for Auto-Encoder with K-Means and 61% for Auto-Encoder with GaussianMixtureModel which implies that the models developed are working with significant efficiency to classify image data.

7 Conclusion

In this study, clustering analysis was done using various algorithms of unsupervised machine learning to classify Fashion MNIST data into 10 classes. Models were trained for various values of hyperparameters. The evaluation metrics obtained showed significant efficiency of the models in classifying the data into respective classes for a particular set of hyperparameters for each model. The performance of the models when measured with accuracy and confusion matrix metrics was observed to improve when we moved from K-Means to Auto-Encoder with K-Means and then to Auto-Encoder with GaussianMixtureModel.

References

[1] https://www.kaggle.com/s00100624/digit-image-clustering-via-autoencoder-kmeans
[2] https://ramhiser.com/post/2018-05-14-autoencoders-with-keras/
[3] https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/
[4] https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f
[5] https://scikit-learn.org/stable/modules/mixture.html
[6] https://smorbieu.gitlab.io/accuracy-from-classification-to-clustering-evaluation/
[7] https://medium.com/@dilekamadushan/introduction-to-k-means-clustering-7c0ebc997e00
[8] https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
[9] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html
[10] https://course.ccs.neu.edu/cs6140sp15/7_locality_cluster/Assignment-6/NMI.pdf
[11] http://www.vlfeat.org/api/kmeans-fundamentals.html
[12] https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95