Cluster Analysis using Unsupervised Learning

Sachin Venugopal
University at Buffalo
sachinve@buffalo.edu

Abstract

This project shows the implementation of cluster analysis on fashion MNIST dataset, using unsupervised machine learning, where we train our model to group unlabeled data into 10 clusters and test the accuracy of our clustering models against the provided labels. We employ three models to do this task and compare accuracy and loss of the clusters generated by each of them.

1 Introduction

.

Unsupervised learning is a branch of machine learning that learns from data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data. It is one of the main three categories of machine learning along with supervised learning and reinforcement learning.

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 modeled using a measure of similarity which is defined upon metrics such as Euclidean or probabilistic distance.

We use two common clustering algorithms in this project:

- **K-Means Clustering**: Partitions the data into K distinct clusters based on the distance to the centroid of a cluster
- **Gaussian Mixture Models:** Models clusters as a mixture of multivariate normal density components.

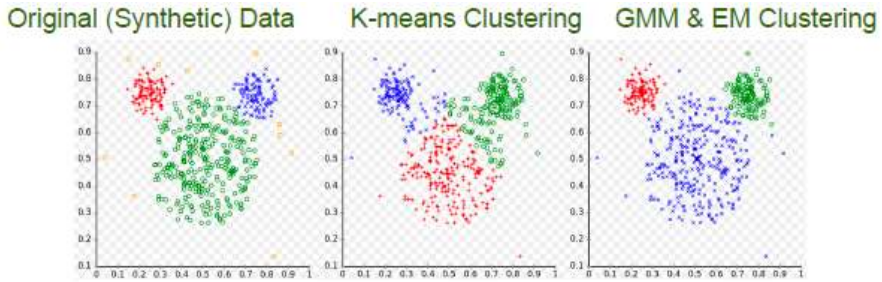

Fig1. K-Means Clustering VS GMM and EM Clustering

2 Dataset

We use the Fashion-MNIST dataset for the purpose of training and testing our classifier. 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 – 28 pixels in height and 28 pixels in width and each pixel has a single pixel value associated with where a higher value indicates a darker image.

The training and the test datasets have 785 columns each with the first column holding the class label specifying the category to which the article of clothing belongs.

Each image is associated with a label from 10 classes shown in the table below:

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

Fig 2a. Labels for Fashion-MNIST dataset

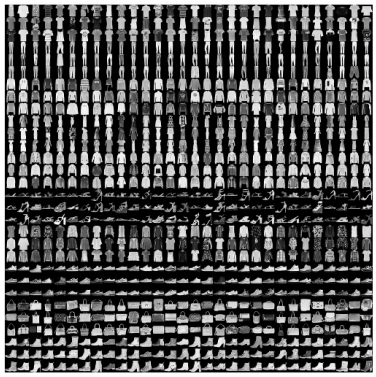

Fig.2b Data sample from Fashion-MNIST

3 Pre-Processing

The Fashion-MNIST dataset for the project is downloaded and partitioned into two NumPy arrays – one containing the 784 features and the other containing the class category for each of the entries in the dataset respectively. The data is also partitioned into 60000 examples for training and 10000 examples for testing. The data is already pre-processed and can be used in the Python program using the load_mnist function defined in util_mnist_reader.

The input test and training arrays containing the 784 features needs to be normalized by dividing by 255.0 so that the pixel values remain in the range of 0 to 1.

4 Architecture

4.1. K-Means Clustering

The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

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 (centroids are not, in general, points from the set of samples although the live in the same sample space). The algorithm needs the number of clusters to be specified.

The objective function for K-Means clustering is given by:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$$

which is the sum of square distances of each data point x_n to its assigned vector μ_k

Here r_{nk} is $\in \{0, 1\}$ is an indicator variable which describes which of the K clusters the data point x_n is assigned to. Its value is 1 if the data point belongs to the cluster k, else 0.

The goal is to find the values of r_{nk} and μ_k so as to minimize J.

The algorithm involves the following steps:

1) Specify number of clusters K.
2) Initialize the centroids by first shuffling the dataset and then randomly selecting K data points without replacement.
3) Assign data points to the nearest centroid determined by finding the shortest distance between the data points and each centroid. Keep iterating until there is no change in assignment of data points to clusters.
4) Compute the sum of square distance between the data points and all centroids.
5) Assign each data point to the cluster with the closet centroid.
6) Compute the centroid for the cluster by taking the average of all the data points that belong to each cluster.
7) This is done repeatedly until there is no change in assignments.

This approach K-Means follows is called **Expectation-Maximization.** The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.

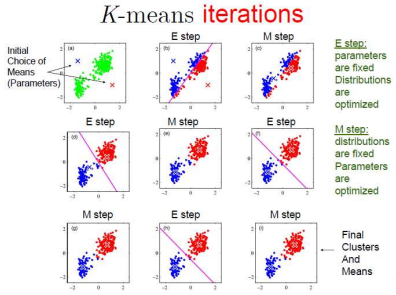

Fig.3 K-Means iterations for Old Faithful Dataset

4.2. Auto-Encoder

"Auto-Encoding" is a data compression algorithm where the compression and decompression functions are data-specific, lossy and unsupervised (learned automatically from examples rather than engineered by a human).

We implement the Auto-Encoder using a feed-forward neural network where the input is the same as the output. The Neural Network compresses the input into a lower-dimensional code and then reconstructs the output from this representation. The code is a compact "summary" or "compression" of the input, also called the *latent-space representation.*

An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code. There exists a distance function that represents 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 functions are chosen such that they are differentiable with respect to the distance function.

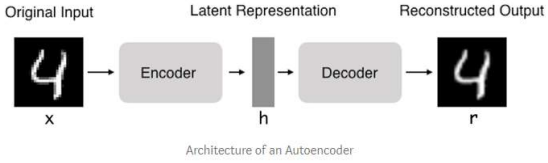

Fig.4 Autoencoder architecture

4.3 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. A Gaussian mixture model attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset.
In the simplest case, GMMs can be used for finding clusters in the same manner as *k*-means: by incorporating information about the covariance structure of the data as well as the centers of the latent Gaussians.

Under the hood, a Gaussian mixture model is very similar to k-means: it uses an Expectation–Maximization approach which qualitatively does the following:

1. Choose starting guesses for the location and shape.
2. Repeat until converged:
    - E-step: for each point, find weights encoding the probability of membership in each cluster
    - M-step: for each cluster, update its location, normalization, and shape based on all data points, making use of the weights

The result of this is that each cluster is associated not with a hard-edged sphere but with a smooth Gaussian model.

4.4 Model Implementation

4.4.1 K-Means Clustering

We first use the K-Means clustering function provided by the Scikit-Learn Library to cluster the training data from the MNIST dataset. The resulting clusters produced by the K-Means clustering function have cluster labels that have no relationship with the true value labels for our input training data.

We thus define a function that maps the predicted labels to the expected true value labels for the training data. The function groups the

We used the mapped labels returned by the map value function to predict the accuracy of our K-Means clustering implementation with the test data.

4.5.2 Auto-Encoder

We use Keras, a high-level Neural Network library, to implement an Auto-Encoder consisting of:

1) **Encoder Layer** consisting of:
    a. Input Layer which takes 784 inputs
    b. Dense layer of 512 Neurons with ReLU Activation
    c. Dense layer of 256 Neurons with ReLU Activation
    d. Dense layer of 128 Neurons with ReLU Activation
2) **Decoder Layer** consisting of:
    a. Input Layer which takes 128 inputs
    b. Dense layer of 256 Neurons with ReLU Activation
    c. Dense layer of 512 Neurons with ReLU Activation
    d. Dense layer of 784 Neurons with Sigmoid Activation

The Decoder uses the compressed latent features generated by the Encoder as input and tries to regenerate the original input image from it.

We use a Stochastic gradient descent as the optimizer and calculate the loss using Mean Squared Error.

4.5.3 Auto-Encoder with K-Means Clustering

We extract the Encoder from the Auto-Encoder by building a model with the Layers from the Encoding Stage of the Auto-Encoder and use it to perform dimensionality reduction on the input data. The K-Means function then uses the compressed data to form the clusters. We then use the test data to test the accuracy of K-Means function.

4.5.4 Auto-Encoder with GMM Clustering

Similar to the previous step, we use the Encoder built using the Encoding Layers of the Auto-Encoder to compress the data and pass this data to the Gaussian-Mixture Function provided by Scikit Learn, to generate clusters. We then test the accuracy of the generated clusters using the test data.

5 Results

5.1 K-Means Clustering:

Setting the hyperparameters for K-Means as:

- n_cluster:10
- max_iter: 300 (Default)

Training Accuracy	0.55355
Testing Accuracy	0.5601

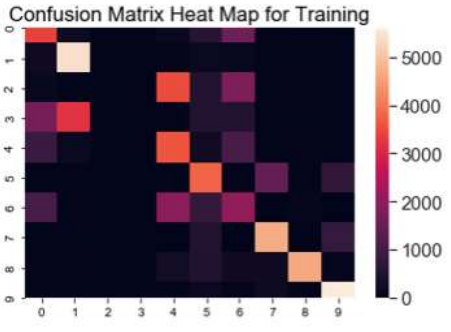

Fig5(a) Confusion Matrix – Training

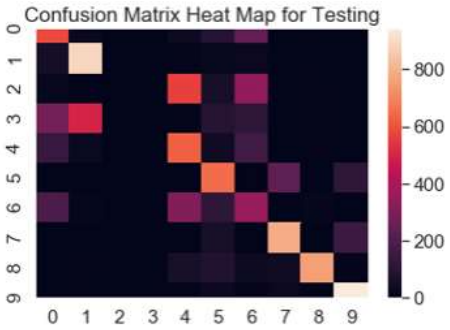

Fig5(a) Confusion Matrix – Testing

The accuracy obtained using the training data to fit the K-Means function and using the testing data is 0.56

5.2 Auto-Encoder

Setting the hyperparameters as:

- Number of Encoder Layers: 3
- Number of Neurons Per Layer:
  - Layer1: 512
  - Layer2: 256
  - Layer3: 128
- Number of Decoder Layers: 3
- Number of Neurons Per Layer:
  - Layer1: 256
  - Layer2: 512
  - Layer3: 784
- Number of Iterations: 50
- Batch size: 256

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_5 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_6 (Dense)              (None, 784)               402192
=================================================================
Total params: 1,132,944
Trainable params: 1,132,944
Non-trainable params: 0
```

Fig 6(a) Layers of Auto-Encoder

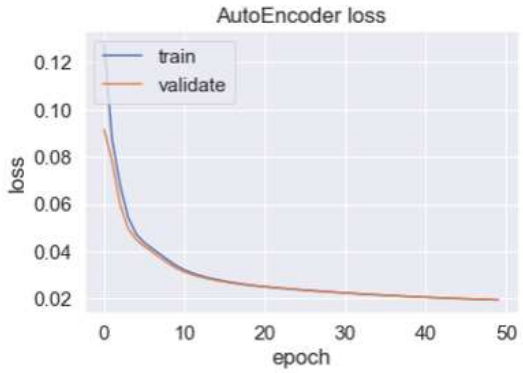

Fig 6(b). Loss VS Epoch

The loss significantly decreases with the number of epochs and layers. We have used the test data here for the purpose of validation and it fits well with the training data loss.

5.3 K-Means Clustering with Auto-Encoder

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
=================================================================
Total params: 566,144
Trainable params: 566,144
Non-trainable params: 0
_________________________________________________________________
```

Fig.7(a) Encoder Structure

Setting the hyperparameters as:

- n_cluster:10
- max_iter: 300 (Default)

Accuracy	0.5141

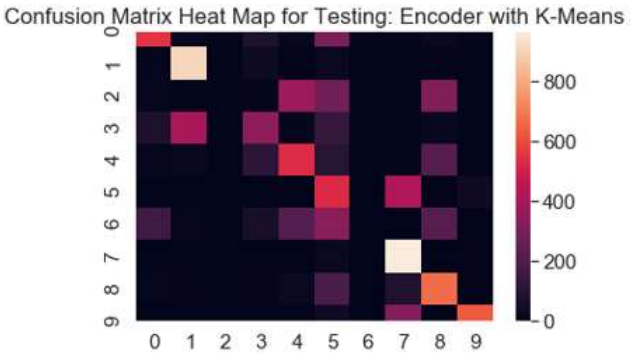

Fig.7(b) Confusion Matrix: K-Means with Encoder

The accuracy of the K-Means Clustering with the Encoder can be improved by increasing the size of the input or changing the compression ratio of the encoder.

5.4 Gaussian Mixture Model Clustering with Auto-Encoder

Setting the hyperparameters as:

- n_components:10
- max_iter: 100 (Default)

Accuracy	0.5575

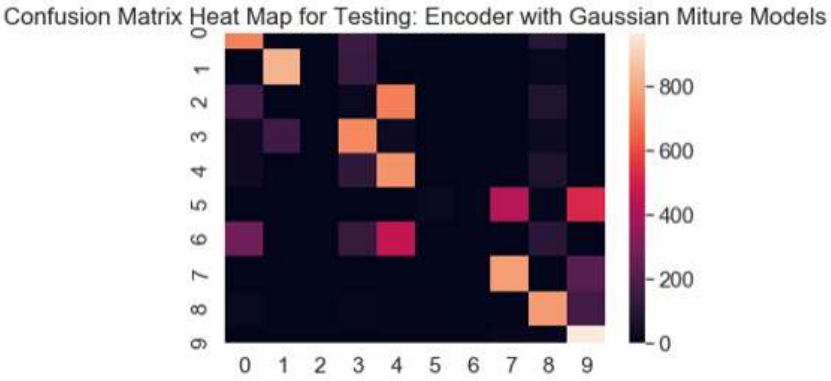

Fig.7(b) Confusion Matrix: GMM with Encoder

As seen from above, the Gaussian Mixture Model performs better than the K-Means clustering algorithm with the Encoder.

6 Conclusion

We use techniques of unsupervised learning in order to find clusters of data items in the feature space of unlabeled data and attempt to estimate the accuracy of the clustering model with respect to the ground truth. The K-Means clustering technique is first used as a baseline with the MNIST dataset of 784 features and we compare the predicted clusters to the true classes to which each data instance belongs. We then use an Encoder to reduce the feature space for the K-Means model and also use the same with a Gaussian Mixture Model and compare their accuracies.

There is a possibility of some datapoints that belong to a particular true-value class, falling into two different clusters or even going unaccounted for after clustering.
Using an Encoder reduces the feature space thereby enabling the Clustering model to improve the partitioning of the smaller feature space into discrete structures. While this is an enhancement for the clustering models, encoders that reduce the number of features of the input data could also be responsible for the loss of important feature information which could lead to inaccuracy in the resulting clusters.

References

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

[2] https://towardsdatascience.com/unsupervised-machine-learning-9329c97d6d9f

[3] https://www.geeksforgeeks.org/gaussian-mixture-model/

[4] https://en.wikipedia.org/wiki/K-means_clustering

[5] Logistic Regression and Machine Learning book, Christopher M. Bishop

[6] http://cs229.stanford.edu/notes/