
CLUSTER ANALYSIS ON MNIST DATASET USING UNSUPERVISED LEARNING

1. USING K-MEANS CLUSTERING
2. AUTOENCODERS USING K-MEANS CLUSTERING
3. AUTOENCODER USING GAUSSIAN MIXTURE MODEL

Rugved Hattekar

#50320920

MASTER'S ROBOTICS AND AUTONOMOUS SYSTEMS

University at Buffalo

Buffalo, NY, 14226

ABSTRACT:

The project mainly focuses on the task to perform the clustering analysis on the fashion MNIST dataset using unsupervised learning. The unsupervised learning algorithms are designed in such a way that they infer patterns from the data without labels. Mainly the unsupervised learning uncover previously unknown patterns in the dataset, but these patterns are approximate. The K-means clustering or the GMM algorithms are generally make a cluster of by considering the Euclidean distance of the dataset from the chosen mean and try to separate the similar data. The given project applies the K-means clustering algorithm on a MNIST dataset and also forms an Autoencoder with K-means and GMM. The Auto-Encoders are data compression algorithm, in which the data is compressed and then again recompressed using a decoder.

INTRODUCTION:

In the project the K-Means and GMM algorithms are implemented using the Keras and the SK-Learns Library, both of them are highly popular for implementing MNIST Data sets of image classification. The given MNIST dataset is of fashion images and asks to train the data of 50000 instances and test it on remaining 10000 instances of 28X28 image each. Through prediction the algorithm should detect the given class of the problem. MNIST data set is very known for implementing machine learning algorithms with higher accuracy. This dataset contains 60,000 greyscale which are usually classified into 10 categories. Usually images are in a low quality at 28*28 pixel with a scale between 0:255.

DATASET:

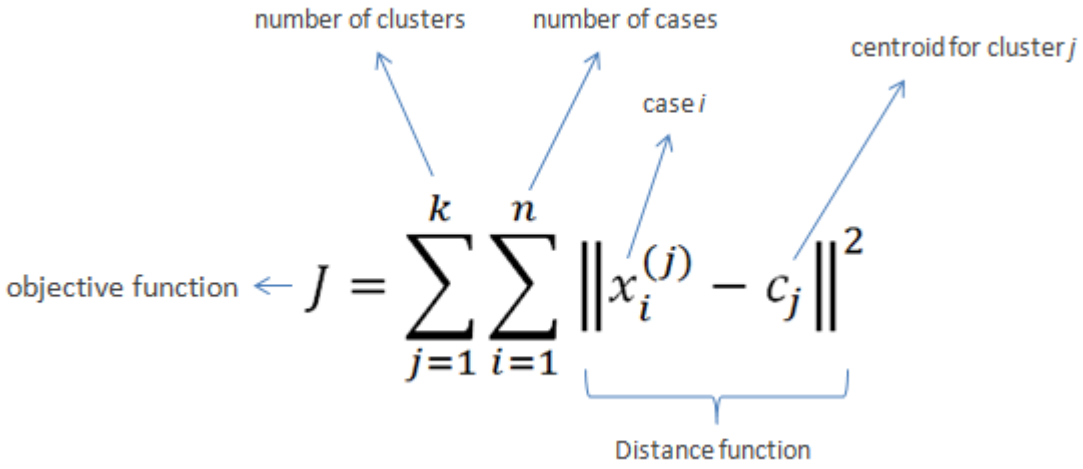
For training and testing of our classifiers, we will use the Fashion-MNIST dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. 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. 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. Each training and test example is assigned to one of the labels as shown in table.

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

ARCHITECTURE:

K-MEANS CLUSTERING:

The K-Means clustering algorithm is useful in the case of unsupervised learning when the data has to be separated. The K in the K-means clustering stands for the number of clusters to be made frm the given dataset of N instances and D-dimensions. The K-means clustering algorithm initially considers the random means in the dataset depending upon the value of K provided. Then the Euclidean distance of the each point near to the mean is calculated and the given point is then assigned to the one of clusters depending upon the Euclidean Distance. The K-means algorithm uses the distortion measurement function J, which is nothing but the sum of squared distances of each data points to its assigned vector. In many cases the J is minimized by taking the derivative of the J.



Equation 1: The above image describes the J the distortion measurement function from the K-means algorithm.

After differentiating this expression keeping the Cj constant, and if we solve further for Cj, we get the expression which sets Cj as mean of all the data points X(i). Hence it is called as K-Means Clustering.

K-means clustering Algorithm :

Consider the given no of K clusters, Also the data set of dimensions D with N instances, represented as $\{X_1, X_2, X_3, \ldots, X_N\}$ the X is an Euclidean variable, meaning the X is an euclidian distance between two points

Let's introduce M_K, M_K is the set of D-dimensh vectors with $K = 1 \ldots K$ (No of clusters).

The K-means algorithm splits the data in two parts by considering the distance between the two points (one is data sample and another is chosen cluster point).

$r_{n,k}$ represents the assignment of data point X_n to the corresponding cluster suppose :

$$r_{n,k} = 1$$

means, the X_n is assigned to cluster K for $K \neq j$

$$r_{n,j} = 0 .$$

The distortion measure for the K-means algorithm can be given by

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

which represents the sum of squares of the distances of each data point to its assigned vector M_K

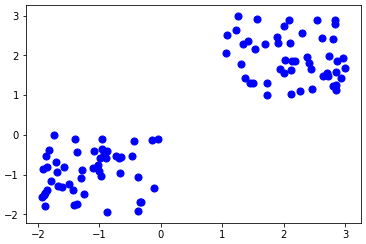


Image Courtesy: “Geeksforgeeks.com”

As you can see from the above picture the k-means clustering takes round shapes because of Euclidean Distance.

To find the optimum cluster for given instance of the data point, we minimize the distortion function J.

The simple way to do it is to take the derivative of the quadratic function;

$$2 \sum_{n=1}^{N} r_{nk} (x_n - \mu_k) = 0$$

And solving for μ_k

$$\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

The denominator in this expression is equal to the number of points assigned to the cluster k, so we set μ_k equal to the mean of all the data points x_n, assigned to cluster k.

GAUSSIAN MIXTURE MODEL

Gaussian Mixture model is another and more robust algorithm of clustering the unsupervised data. The Gaussian distribution is used in the Gaussian mixture model to cluster the data points. The Gaussian model takes care of data that is oddly shaped whereas the K-means algorithm only captures only circular shaped data. The Gaussian model able to capture any shape data because it considers the covariance of the data along with the mean of the data. The K-means algorithm only considers the Euclidean distance of the data which limits its properties to capture oddly shaped data. The Gaussian mixture models also have the data models which can be helpful to choose the value of K (No of clusters).

The Equations for the Gaussian mixture model are given by:

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma_k(x_n) x_n}{\sum_{n=1}^{N} \gamma_k(x_n)}$$

After differentiation we get,

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma_k(x_n)(x_n - \mu_k)(x_n - \mu_k)^T}{\sum n=1^N \gamma_k(x_n)}$$

Where the denominator denotes the total number of sample points in the Kth Cluster.

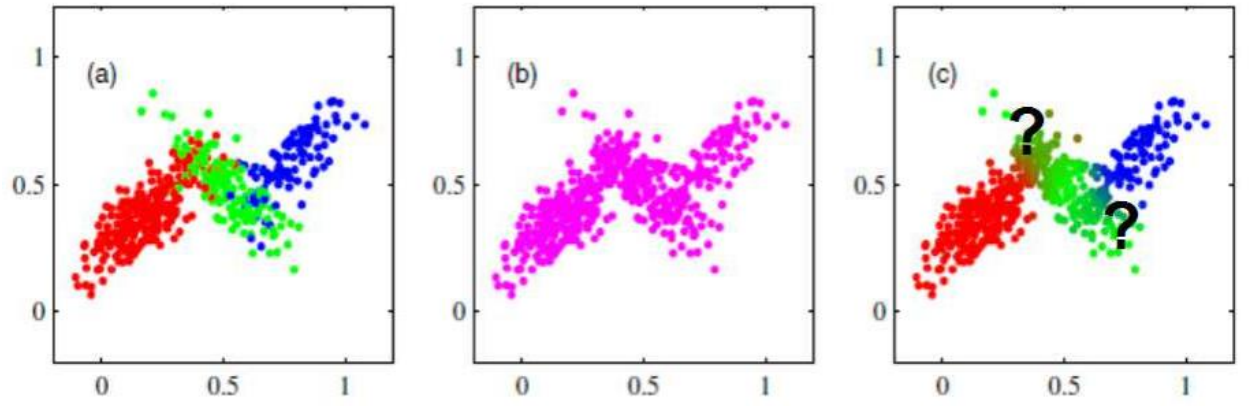


Image Courtesy : “”Geeksforgeeks.com”

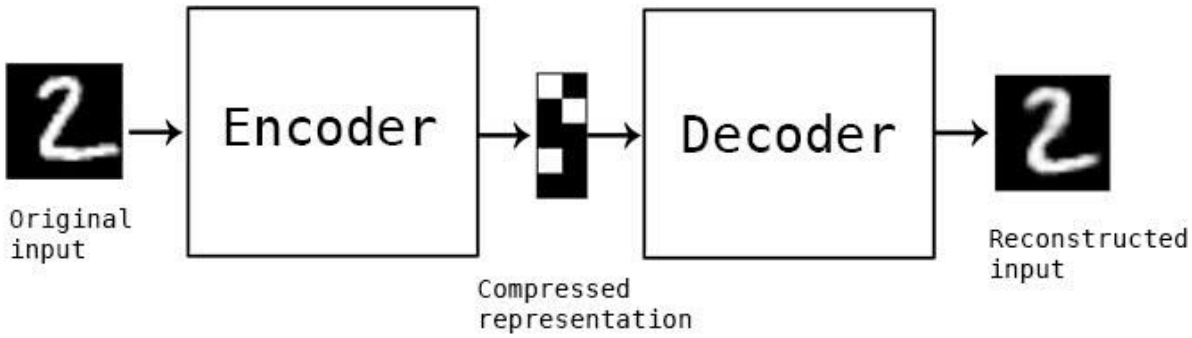
As it can been observed that GMM can have oddly shaped clusters.

AUTO-ENCODERS:

As the Handout mentions, ”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 Figure 2. To build an autoencoder, you 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 optimize to minimize the reconstruction loss, using Stochastic Gradient Descent.”



Clustering is difficult task to perform in Higher Dimensions, the distance between most of the points is very difficult to track. Therefore the Auto-Encoders are used to represent the data in the lower dimensions. This helps in the computations as well.

RESULTS:

Part 1: K-Means-Clustering in Unsupervised Learning:

```
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import util_mnist_reader
from sklearn.preprocessing import scale
from sklearn.metrics import confusion_matrix


X_train, y_train = util_mnist_reader.load_mnist('data/fashion', kind='train')
X_test, y_test = util_mnist_reader.load_mnist('data/fashion', kind='t10k')

x_train_sc=scale(X_train.astype(np.float64))
x_test_sc=scale(X_test.astype(np.float64))

kmeans = KMeans(n_clusters=10, init='random',max_iter=500,random_state=0).fit(X_train)
a=kmeans.labels_
y_pred=kmeans.predict(X_test)

c_m2=confusion_matrix(y_test,y_pred)
metrics.normalized_mutual_info_score(y_test, y_pred,average_method='geometric')
```

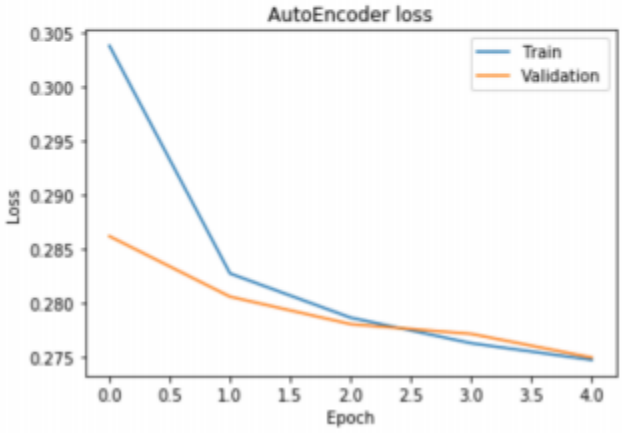
```
0.5127289114258798
```

Note: Accuracy is at the bottom of the above snippet.

Part 2: K-means using Auto-Encoders

Note: Due to less computation power the number of epochs are less but the accuracy seems to be pretty good.

```
Epoch 1/5
60000/60000 [==============================] - 779s 13ms/step - loss: 0.3038 - val_loss: 0.2862
Epoch 2/5
60000/60000 [==============================] - 778s 13ms/step - loss: 0.2827 - val_loss: 0.2806
Epoch 3/5
60000/60000 [==============================] - 744s 12ms/step - loss: 0.2787 - val_loss: 0.2781
Epoch 4/5
60000/60000 [==============================] - 744s 12ms/step - loss: 0.2763 - val_loss: 0.2772
Epoch 5/5
60000/60000 [==============================] - 612s 10ms/step - loss: 0.2747 - val_loss: 0.2750
```




```
[[577  39   2 257   4 103   0   0  14   4]
 [  1  13   0  24   0  82   0   0 880   0]
 [  4 636   0 337   4  18   0   0   1   0]
 [ 16  21   0 115   1 433   0   0 413   1]
 [  0 687   0 142   3 155   0   0   9   4]
 [  0   0 524  24   1   0 212 239   0   0]
 [121 349   1 395  19 107   0   0   5   3]
 [  0   0 781   0   0   0   3 216   0   0]
 [  1  55  23  50 421  29   0   1   9 411]
 [  0   0  15   0   1   3 415 566   0   0]]
```

Accuracy: 56.64%

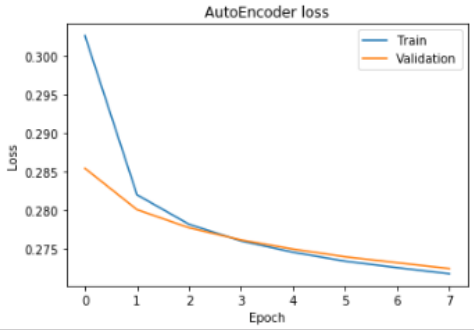
Part 3: GMM using Auto-Encoders:

Note: In the code to avoid the optimizer error, try writing lr as learning_rate if keras version is higher that 2.2.3

Note: Due to less computation power the number of epochs are less but the accuracy seems to be pretty good.

```
WARNING:tensorflow:From C:\Users\ADMIN\Anaconda3\lib\site-packages\keras\optimizers.py:744: The name tf.train.Optimizer is depr
ecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\ADMIN\Anaconda3\lib\site-packages\tensorflow\python\ops\nn_impl.py:180: add_dispatch_support.<
locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 60000 samples, validate on 10000 samples
Epoch 1/8
60000/60000 [==============================] - 437s 7ms/step - loss: 0.3026 - val_loss: 0.2854
Epoch 2/8
60000/60000 [==============================] - 419s 7ms/step - loss: 0.2820 - val_loss: 0.2801
Epoch 3/8
60000/60000 [==============================] - 309s 5ms/step - loss: 0.2782 - val_loss: 0.2778
Epoch 4/8
60000/60000 [==============================] - 211s 4ms/step - loss: 0.2760 - val_loss: 0.2762
Epoch 5/8
60000/60000 [==============================] - 212s 4ms/step - loss: 0.2746 - val_loss: 0.2750
Epoch 6/8
60000/60000 [==============================] - 215s 4ms/step - loss: 0.2734 - val_loss: 0.2740
Epoch 7/8
60000/60000 [==============================] - 211s 4ms/step - loss: 0.2726 - val_loss: 0.2732
Epoch 8/8
60000/60000 [==============================] - 213s 4ms/step - loss: 0.2718 - val_loss: 0.2725
```




```
[[ 23   0  16 164   0 656   6   1   0 134]
 [  1   0 877  32   0   0   1   0   0  89]
 [765   0   0 186   0   6   9   0   0  34]
 [ 14   0 321  58   0  15   1   1   0 590]
 [674   0   2 149   0   0   2   0   0 173]
 [  0 194   0   7 483   0   7   0 309   0]
 [473   0   3 198   0 183  15   3   0 125]
 [  0 204   0   1 787   0   4   0   4   0]
 [  3   1   0  56   9   0 451 442   0  38]
 [  0 556   0   6  24   0   0   0 414   0]]
```

```
Accuracy:  0.585739242006252
```

CONCLUSION:

In this project we learnt to apply K-means clustering algorithm in to the Unsupervised data of MNIST. The Gaussian Mixing Model were also studied with the application of the Auto encoders. The Keras and Sk-learns libraries are super useful tools in the machine learning domain.I implemented autoncoders with K-means and Autoencoders with GMM and got an accuracy of 56.09% and 55.27% respectively. For increasing the accuracy the kernel size and weight also play a role hence the kernel size should be considered as important parameter for accuracy.

Part 1 Accuracy: 51.2 %

Part 2 Accuracy: 56.69%

Part 3 Accuracy: 58.57%