Cluster analysis on fashion MNIST dataset using unsupervised learning

Meghana Vasudeva
Person#: 50290586
University at Buffalo,
The State University of New York Buffalo,
New York 14260
mvasudev@buffalo.edu

Abstract

In this project I have performed the task of cluster analysis on fashion MNIST dataset using unsupervised learning. Cluster analysis is one of the unsupervised machine learning technique which doesn't require labeled data. The project task is to cluster the images and to identify the cluster as one of the clothes. The dataset to be worked on was the Fashion-MNIST which is that of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. The task was divided into 3 parts.

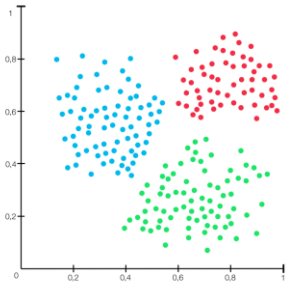
1. Use KMeans algorithm to cluster original data space of Fashion-MNIST dataset using Sklearns library.

2. Build an Auto-Encoder based K-Means clustering model to cluster the condensed representation of the unlabeled fashion MNIST dataset using Keras and Sklearns library.

3. Build an Auto-Encoder based Gaussian Mixture Model clustering model to cluster the condensed representation of the unlabeled fashion MNIST dataset using Keras and Sklearns library.

1. Introduction

Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group be similar to one another and different from the objects in other groups. The greater the similarity within a group and the greater the difference between groups, the better or more distinct the clustering.



Clustering for the sake of understanding classes, or conceptually meaningful groups of objects that share common characteristics, play an important role in how people analyze and describe the world. Human beings are skilled at dividing objects into groups and assigning particular objects to these groups. Many real world examples for clustering can be seen business, information retrieval and many more.

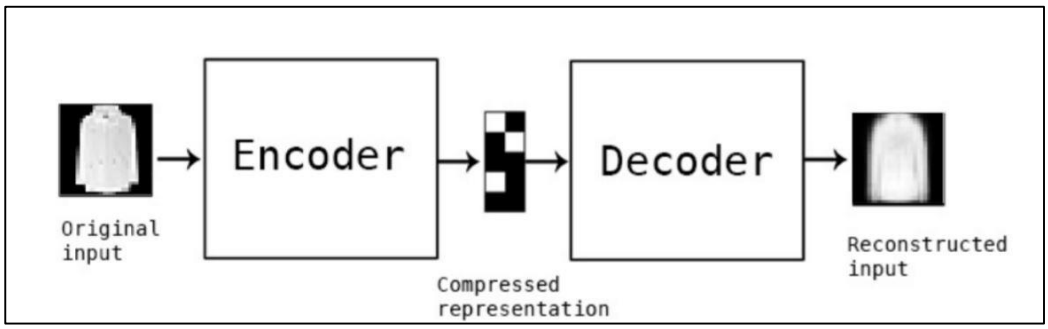
Architecture of K-means and Autoencoders

K-Means

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. 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.

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. 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 2 AutoEncoder your data and the decompressed representation.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 optimizing to minimize the reconstruction loss, using Stochastic Gradient Descent.



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. 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. 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.

Auto-Encoder with GMM Clustering

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.

2. Dataset

Fashion MNIST Clothing Classification

The Fashion-MNIST dataset is proposed as a more challenging replacement dataset for the MNIST dataset. It is a dataset comprised of training set of 60,000 examples and a test set of 10,000 examples of small square 28×28-pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. 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. Fashion-MNIST is intended to serve as a direct drop-in replacement for the original MNIST dataset to benchmark machine learning algorithms, as it shares the same image size and the structure of training and testing splits. Each training and test example is assigned to one of the following labels with example images in the dataset:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

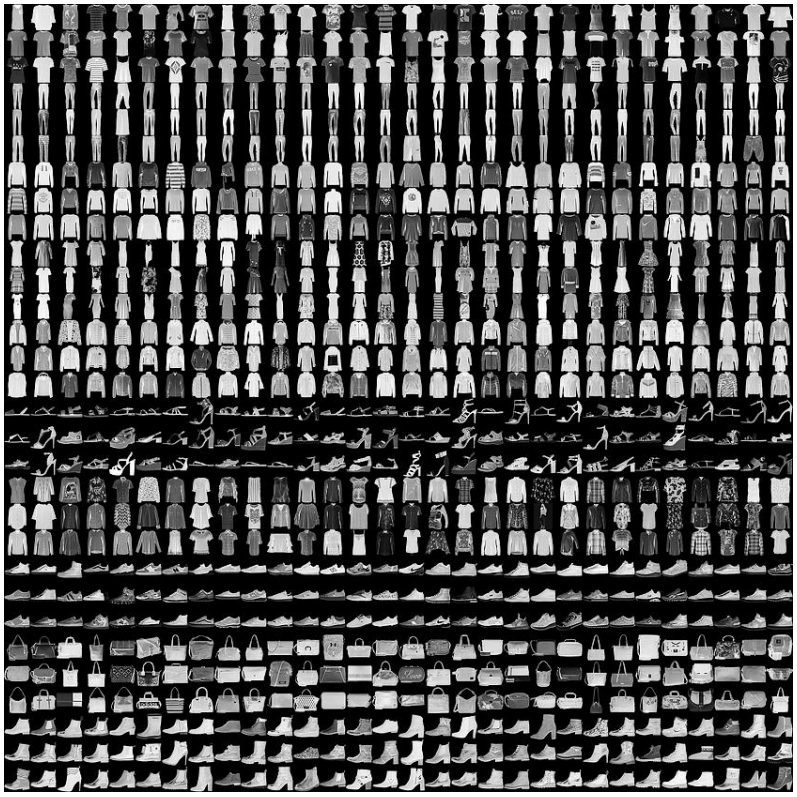
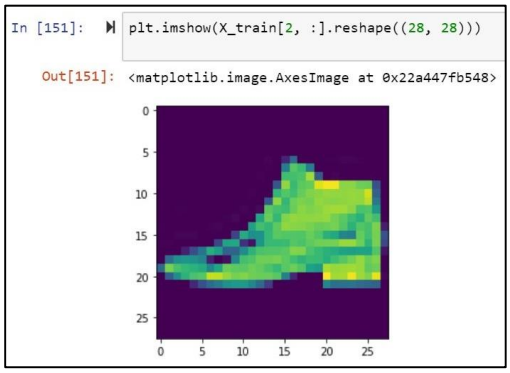


Fig: Example images in Fashion-MNIST data

Below are the files contained in the Fashion-MNIST dataset.

Name	Description	Number of Examples	Size
train-images-idx3-ubyte.gz	Training set images	60, 000	25 MBytes
train-labels-idx1-ubyte.gz	Training set labels	60, 000	140 Bytes
t10k-images-idx3-ubyte.gz	Test set images	10, 000	4.2 MBytes
t10k-labels-idx1-ubyte.gz	Test set labels	10, 000	92 Bytes

Let us have a look at one instance (an article image), say at index 2, of the training dataset. So, we see that image at index (instance no.) 2 is an Ankle boot.



3. Setting up Data and Preprocessing

Importing the libraries for Task 1,2,3. In the first task I have used sklearn library for KMeans clustering and for the second and third tasks I have used keras libraries to model the autoencodes.

```
import sys
import sklearn
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.mixture import GaussianMixture as GMM
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from tensorflow import keras
from keras.datasets import fashion_mnist
from keras.layers import Input, Dense
from keras.models import Model
import matplotlib.pyplot as plt
```

Load the data and split the data into training and testing set of data. Then we normalize the data dimensions so that they are of the same scale. MNIST contains images that are 28 by 28 pixels; as a result, they will have a length of 784 once we reshape them into a 1-dimensional array

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

print('Training Data: {}'.format(x_train.shape))
print('Training Labels: {}'.format(y_train.shape))
print('Testing Data: {}'.format(x_test.shape))
print('Testing Labels: {}'.format(y_test.shape))
```

```
Training Data: (60000, 28, 28)
Training Labels: (60000,)
Testing Data: (10000, 28, 28)
Testing Labels: (10000,)
```

```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(x_train.shape)
print(x_test.shape)
```

```
(60000, 784)
(10000, 784)
```

4. Implementation

Task 1: KMeans algorithm to cluster original data space of Fashion-MNIST dataset using Sklearns library.

The MNIST dataset contains images of the 10 different types of clothes. Because of this, let's start by setting the number of clusters to 10, one for each type of clothing.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=10,random_state = 5)
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=10, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=5, tol=0.0001, verbose=0)
```

K-means clustering is an unsupervised machine learning method. The labels assigned by our KMeans algorithm refer to the cluster each array was assigned to, not the actual target integer. To fix this, I have defined a few functions that will predict which integer corresponds to each cluster.

```
# test the infer_cluster_labels() and infer_data_labels() functions
cluster_labels = infer_cluster_labels(kmeans, Y)
X_clusters = kmeans.predict(X)
predicted_labels = infer_data_labels(X_clusters, cluster_labels)
print(predicted_labels[:20])
print(Y[:20])
```

```
[9 0 5 6 1 4 7 4 5 5 0 9 7 7 7 9 1 0 4 5]
[9 0 0 3 0 2 7 2 5 5 0 9 5 5 7 9 1 0 6 4]
```

```
from sklearn import metrics
# function to calculate all the metrics
def calculate_metrics(estimator, data, labels):

    # Calculate and print metrics
    print('Number of Clusters: {}'.format(estimator.n_clusters))
    print('Inertia: {}'.format(estimator.inertia_))
    print('Homogeneity: {}'.format(metrics.homogeneity_score(labels, estimator.labels_)))
```

```
clusters = [10, 16, 36, 64, 144, 256]
# test different numbers of clusters
for n_clusters in clusters:
    estimator = KMeans(n_clusters = n_clusters)
    estimator.fit(X)

    # print cluster metrics
    calculate_metrics(estimator, X, Y)

    # determine predicted labels
    cluster_labels = infer_cluster_labels(estimator, Y)
    predicted_Y = infer_data_labels(estimator.labels_, cluster_labels)

    # calculate and print accuracy
    print('Accuracy: {}\n'.format(metrics.accuracy_score(Y, predicted_Y)))
```

With the functions defined above, the accuracy of our algorithms can be determined. Since I am using this clustering algorithm for classification, accuracy is ultimately the most important metric; however, there are other metrics out there that can be applied directly to the clusters themselves, regardless of the associated labels. Two of these metrics that we will use are inertia and homogeneity. Furthermore, earlier I made the assumption that K = 10 was the appropriate number of clusters; but, this might not be the case. By fitting the K-means clustering algorithm with several different values of K, I evaluated the performance using our metrics.

Task 2: Build an Auto-Encoder based K-Means clustering model to cluster the condensed representation of the unlabeled fashion MNIST dataset using Keras and Sklearns library

First, encoders and decoders need to be built by creating encoding and decoding models.
Next, train the autoencoder to reconstruct MNIST fashion our model to use a per-pixel binary crossentropy loss, and the Adadelta optimizer.

```
from keras.layers import Input, Dense
from keras.models import Model

# this is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)


# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
```

```
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)

# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy',metrics=['accuracy'])
```

Then the encoder is trained for 50 epochs. After 50 epochs, we can visualize the reconstructed inputs and the encoded representations using Matplotlib. Following this we run the same Kmeans model on the autoencoder model and find out the accuracy of the model.

```
autoencoder_train= autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 4s 66us/step - loss: 0.0456 - acc: 0.5021 - val_loss: 0.0260 - val_acc: 0.
4999
Epoch 2/50
60000/60000 [==============================] - 4s 64us/step - loss: 0.0199 - acc: 0.5021 - val_loss: 0.0157 - val_acc: 0.
4999
```

Task 3: Build an Auto-Encoder based Gaussian Mixture Model clustering model to cluster the condensed representation of the unlabeled fashion MNIST dataset using Keras and Sklearns library.

A Gaussian mixture model (GMM) is used to find a mixture of multi-dimensional Gaussian probability distributions that best models any input dataset. GMMs can be used for finding clusters in the same manner as *k*-means:

```
from sklearn.mixture import GaussianMixture as GMM
n_components = np.arange(50, 210, 10)
gmm = GMM(n_components=10)
gmm_fit = gmm.fit(encoded_imgs)
gmm_perdict= gmm.fit_predict(encoded_imgs)
```

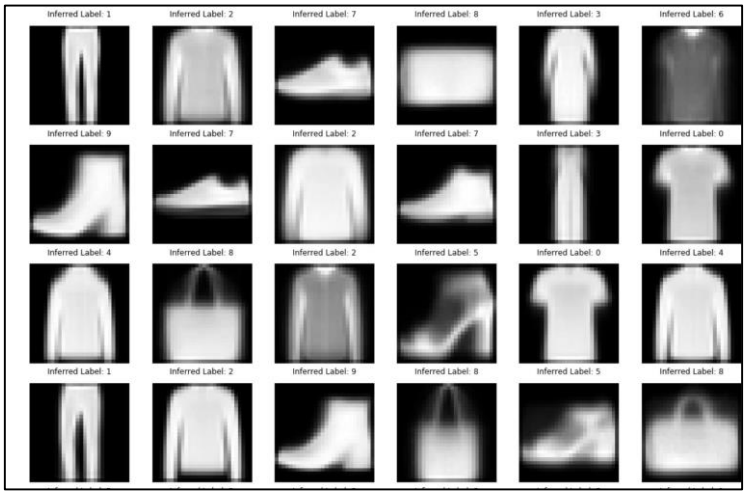
5. Analysis and Results

Task 1: Kmeans

I made the assumption that K = 10 and got an accuracy of 57% but as I was trying to fit the K-means clustering algorithm with several different values of K. As the clusters increased the accuracy started becoming better. This is the following result of the outcome. The second image is that of the inferred labeling which is almost correctly labeled.

Cluster K = 10
Accuracy = 57.4%

```
Number of Clusters: 10
Inertia: 1914601.1988981345
Homogeneity: 0.5038895400894868
Accuracy: 0.5747833333333333

Number of Clusters: 16
Inertia: 1686143.4985453724
Homogeneity: 0.6014211452237251
Accuracy: 0.6580833333333334

Number of Clusters: 36
Inertia: 1428962.286020338
Homogeneity: 0.650821734779208
Accuracy: 0.6868666666666666

Number of Clusters: 64
Inertia: 1298158.6929245037
Homogeneity: 0.6848119244064973
Accuracy: 0.7226166666666667

Number of Clusters: 144
Inertia: 1146155.6338063045
Homogeneity: 0.7266649450145769
Accuracy: 0.7580333333333333

Number of Clusters: 256
Inertia: 1053537.451999085
Homogeneity: 0.7546211434409001
Accuracy: 0.783
```



Task 2: Autoencoder with Kmeans

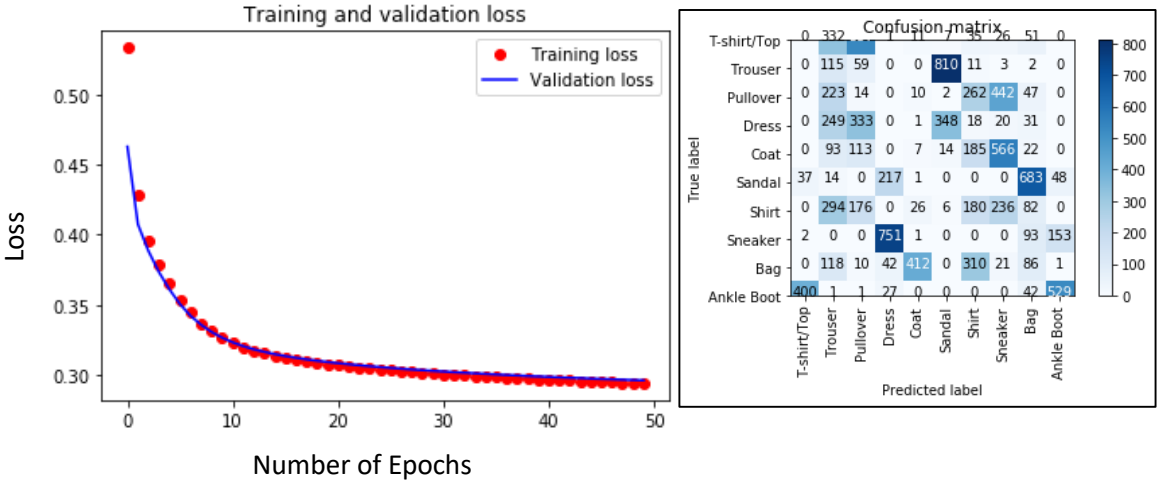
Ran the autoencoder for 50 epochs and got a loss of 0.29 and accuracy to be 50.56% later ran the kmeans model on it to get an accuracy of 55.3%

Cluster K = 10
Accuracy = 55.3%

```
Epoch 50/50
60000/60000 [==============================] - 6s 92us/step - loss: 0.2930 - acc: 0.5078 - val_loss: 0.2950 - val_acc: 0.
5056
```

```
Number of Clusters: 10
Inertia: 1906652.5028672658
Homogeneity: 0.5004518286383872
Accuracy: 0.5537
```

Below is the graph for the loss vs epoch for training and validation set. We can notice that as the accuracy increases the loss reduces. The confusion matrix is given below too.



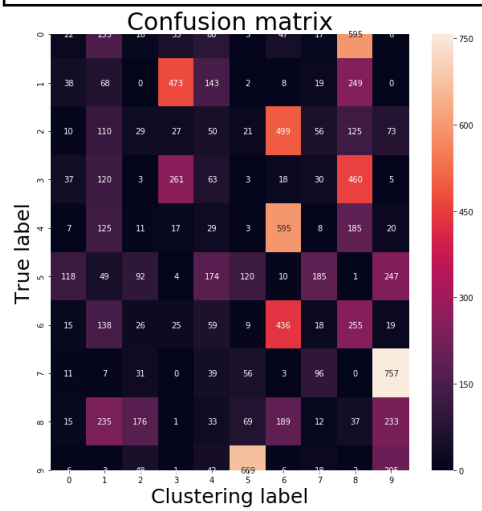
Task 3: Autoencoder with GMM

A Gaussian mixture model (GMM) 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 kmeans: The confusion matrix is also given below.

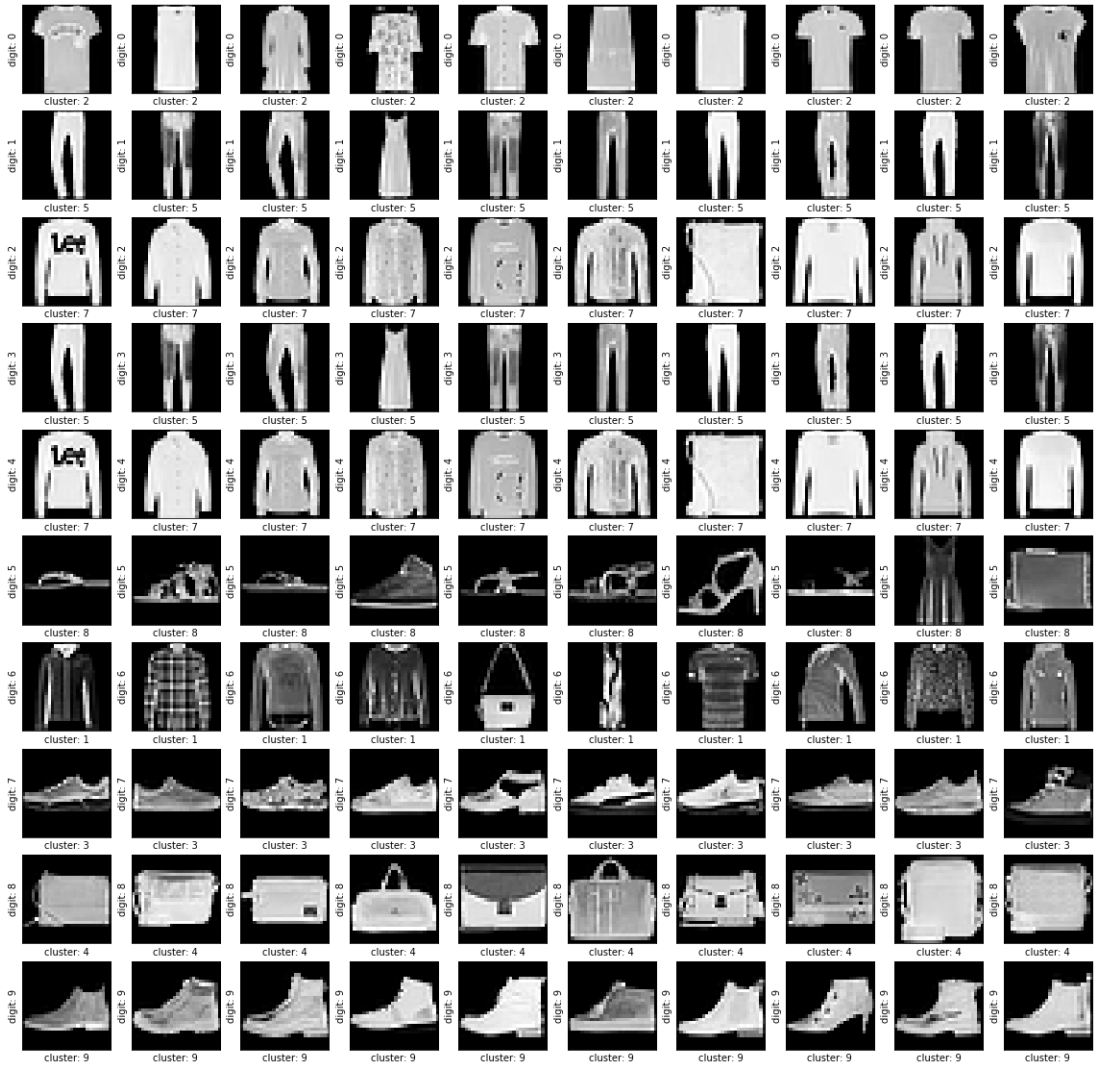
Cluster K = 10
Accuracy = 51.4%

```
print('Accuracy: {}\n'.format(metrics.accuracy_score(y_test, gmm_perdict)))
```

Accuracy: 0.5143



This was the analysis of the clustering for finding out different kinds of clothes and labeling them into the 10 categories.



6. Conclusion

Clustering apart from being an unsupervised machine learning can also be used to create clusters as features to improve classification models. On their own they aren't enough for classification as the results show. But when used as features they improve model accuracy. We can see that Kmeans is one of the best ways to classify an unsupervised cluster in the results with the highest accuracy of 57%.

References

Fashion MNIST dataset: https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/
Kmeans : https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
Autoencoders: https://blog.keras.io/building-autoencoders-in-keras.html
GMM: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html
Project Description:https://ublearns.buffalo.edu/bbcswebdav/pid-5219574-dt-content-rid-26904568_1/courses/2199_23170_COMB/iml_project3%282%29.pdf