Cluster analysis using KMeans and Gaussian Mixture Models

Teja Akasapu
50314294
tejaakas@buffalo.edu

Abstract

To aim of this project is to cluster the images of given Fashion-MNIST data set and identify any given image as belonging to one of the clusters. For achieving this, first I have used K-Means clustering algorithm, then built an Auto-encoder to condense dimensionality of the data set and implemented the K-Means using Sklearn library. Finally I have used a Auto-encoder based Gaussian mixture model clustering model on the condensed data set. The developed models have been validated and tested successfully for accuracy and confusion matrix has been drawn.

1 Introduction

In this project, we are using clustering algorithms - KMeans and Gaussian Mixture models to find the cluster to which given image belongs.

1.1 Clustering

Clustering is a Machine Learning technique that involves the grouping of data points. Given a set of data points, we can use a clustering algorithm to classify each data point into a specific group. In theory, data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features. Clustering is a method of unsupervised learning and is a common technique for statistical data analysis used in many fields.

1.2 KMeans clustering

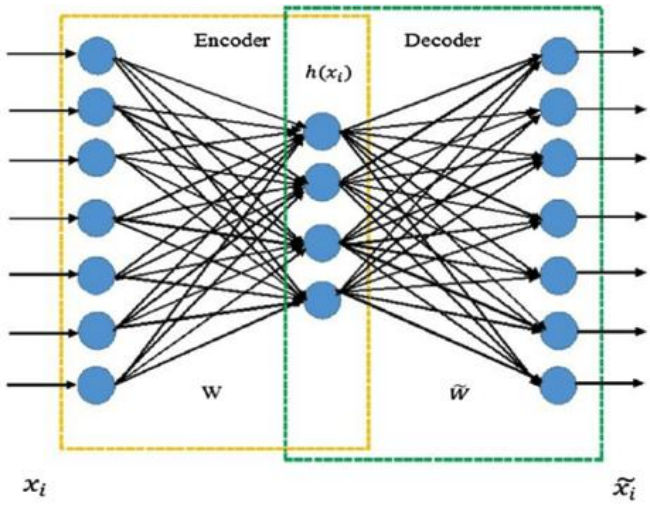
K-Means clustering is an unsupervised learning algorithm that, as the name hints, finds a fixed number (*k*) of clusters in a set of data. A *cluster* is a group of data points that are grouped together due to similarities in their features. When using a K-Means algorithm, a cluster is defined by a *centroid*, which is a point (either imaginary or real) at the center of a cluster.
K-Means starts by randomly defining *k* centroids. From there, it works in iterative (repetitive) steps to perform two tasks:

1. Assign each data point to the closest corresponding centroid, using the standard Euclidean distance. In layman's terms: the straight-line distance between the data point and the centroid.
2. For each centroid, calculate the mean of the values of all the points belonging to it. The mean value becomes the new value of the centroid.

Once step 2 is complete, all of the centroids have new values that correspond to the means of all of their corresponding points. These new points are put through steps one and two producing yet another set of centroid values. This process is repeated over and over until there is no change in the centroid values, meaning that they have been accurately grouped.

1.3 Auto-Encoder

Autoencoders map the data they are fed to a lower dimensional space by combining the data's most important features. It *encodes* the original data into a more compact representation. It also decides how the data is combined, hence the *auto* in Autoencoder.



This 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 2 Figure 2: AutoEncoder 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.

1.4 Auto-Encoder with KMeans clustering:

The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum-ofsquares criterion:

$$\text{Xn i=0 min } \mu j \in C \ (||xi - \mu j||2)$$

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 highdimensional 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.

1.5 Gaussian Mixture Models:

Gaussian Mixture Models (GMMs) assume that there are a certain number of Gaussian distributions, and each of these distributions represent a cluster. Hence, a Gaussian Mixture Model tends to group the data points belonging to a single distribution together.

Gaussian Mixture Models are probabilistic models and use the soft clustering approach for distributing the points in different clusters.

1.6 Confusion Matrix:

A confusion matrix is a summary of prediction results on a classification problem.
The number of correct and incorrect predictions are summarized with count values and broken down by each class.

1.7 Non-Linear Activation Functions:

These are used to introduce non-linearity in an artificial neural network. It allows us to model a class label or score that varies non-linearly with independent variables

1.7.1 Relu Function:

Rectified linear unit(RELU) has the output 0 if its input is less than or equal to 0, otherwise, its output is equal to its input. It is widely used in convolutional neural networks. It is also superior to the sigmoid and tanh activation function, as it does not suffer from the vanishing gradient problem. Thus, it allows for faster and effective training of deep neural architectures.

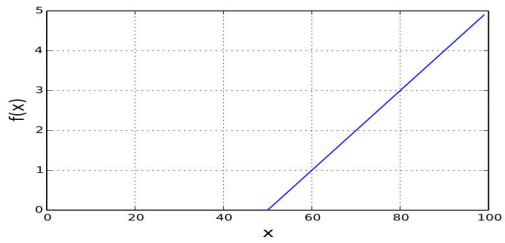


Figure : Relu Function Graph

$$a_j^i = f(x_j^i) = \max(0, x_j^i)$$

Figure : Relu Function Equation

1.7.2 Sigmoid Function:

To achieve binary outcome, logistic regression algorithm uses Sigmoid/ Logit function which squeezes the output graph into a binary class of '0' and '1'. The formula of sigmoid function is as follows:

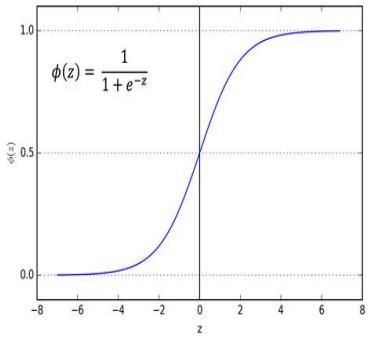


Figure: Graph for Sigmoid function

2 Dataset

The model is trained using a Zalando's article images of clothing and accessories named 'fashion_MNIST'. It contains 70,000 examples of which 60,000 are for training and 10,000 are for test and validation.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.

Table 1: Labels for Fashion-MNIST dataset

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

3 Preprocessing

Each pixel value has a value between 0 and 255 which represents the grey scale value of the corresponding pixel. To make our calculations easier and reduce the gap between values of each pixel, we do pre-processing.

3.1 Normalization

It is better to scale our features to a common set of values as it helps finding the parameters for our hypothesis functions. If scaling is not done, the graph for our cost function would be skinny and vertically long which makes the task of minimizing the cost function tougher.

For each pixel value x which is in the range (0-255), we use the following formula to get the scaled value of it:

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

In our case average(x)=0

max(x)=255; min(x)=0

Figure 2: Formula for mean normalization

The mean value i.e., average(x) is not being considered by me. So it's value is 0. The max and min values are '0' and '255' respectively. So we are diving each pixel value in an image with 255. By doing this, all the features will have values in the range (0, 1), which the calculations easier and faster.

4 Architecture:

4.1 Data reshaping and normalizing:

Images stored as NumPy arrays are 2-D arrays. But, the KMeans algorithm provided by scikit-learn takes 1-D arrays as input, so the images have to be reshaped. For a fact, clustering algorithms almost always use 1-D data. Since the Fashion-MNIST data contains images that are 28*28 pixels, the reshaped 1-D data array should be of length 784. Also, for the purpose of Normalization, to bring the dataset values to the range [0,1], each pixel was divided by 255.

4.2 Kmeans on Fashion-Mnist data set:

I have used the Kmeans cluster algorithm provided by SKlearns package wherein the number of clusters I've used is 10 and the initialization method is 'kmeans++'. Maximum iterations for each run is set to 250. two functions have been defined to map the labels to the cluster IDs:

The 'infer_Cluster_Labels' function maps the most related label with each cluster in KMeans model, and returns a dictionary of clusters assigned to each label.

The "infer_Data_Labels" function determines the label for each array, depending on the cluster it has been assigned to, and returns the predicted labels for each array.

The number of epochs were set to 250 initially but as early stopping was used, model has been stopped after 19th epoch. Early stopping has been used for preventing overfitting to the given data set. Activation functions Relu, softmax, sigmoid have been used. After training the neurons, we obtain the optimal weights and bias. We then test the model by using the test data. Using this data I calculated the accuracy using the SKlearns metric.accuracy_score function.

Final accuracy of 55.32 has been observed.

4.3 Auto-encoder with KMeans clustering:

A convolutional auto encoder network has been used initially. It has 2 encoder levels and 2 decoder levels. To compile the model, Stochastic Gradient Descent optimizer has been used. For this project purpose, SGD has been used. But the loss is more when compared to Adam optimizer.

Due to this reason, I have shifted to a Dense layered Auto-encoder, in which the hidden layers are as shown below:

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_9 (Dense)              (None, 2000)              1570000
_________________________________________________________________
dense_10 (Dense)             (None, 1000)              2001000
_________________________________________________________________
dense_11 (Dense)             (None, 500)               500500
_________________________________________________________________
dense_12 (Dense)             (None, 10)                5010
_________________________________________________________________
dense_13 (Dense)             (None, 500)               5500
_________________________________________________________________
dense_14 (Dense)             (None, 1000)              501000
_________________________________________________________________
dense_15 (Dense)             (None, 2000)              2002000
_________________________________________________________________
dense_16 (Dense)             (None, 784)               1568784
=================================================================
Total params: 8,153,794
Trainable params: 8,153,794
Non-trainable params: 0
```

Figure: Dense layers used for Auto-encoder

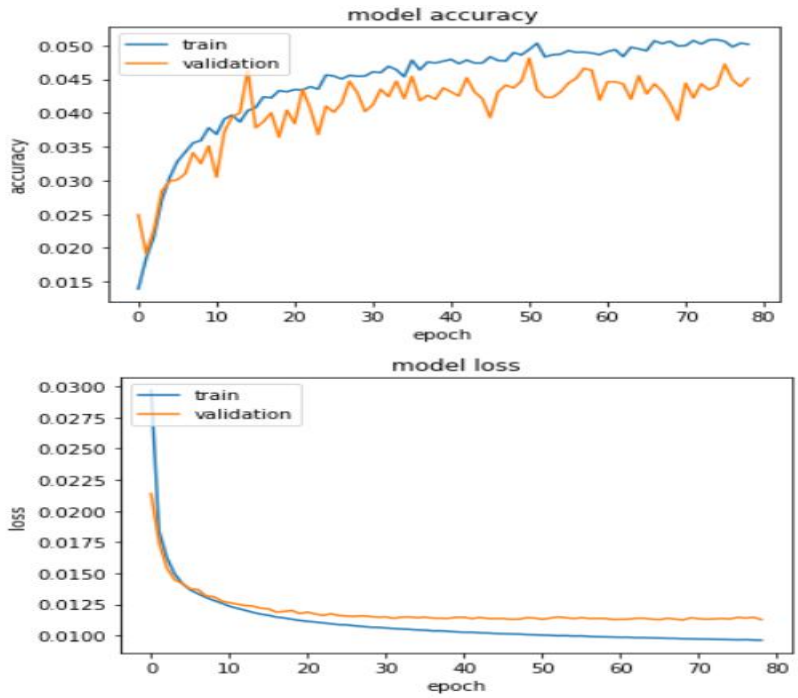


Figure: Model accuracy Vs Number of epochs; Training loss and Validation Loss vs Number of epochs

Now on the condensed data set, Kmeans clustering algorithm has been modeled and used. The number of epochs were set to 200 initially but as early stopping was used, model has been stopped after 79th epoch.

Final accuracy of 60.79 has been observed.

4.4 Auto-encoder with Gaussian Mixture Models:

Similar to the above algorithm, on the condensed data set, Gaussian Mixture Models clustering algorithm has been modeled and used. The number of epochs were set to 300 and tolerance of tolerance of e^-10 was used.

Final accuracy of 62.23 has been observed

5 Results:

5.1 Kmeans on Fashion-Mnist data set:

```
[ ] kmeans_acc = metrics.accuracy_score(train_y, predicted_Y)
    print(kmeans_acc)

    0.5532666666666667
```

Figure: Accuracy for KMeans clustering

5.2 Auto-encoder with KMeans clustering:

```
print(kmeans_acc)

0.6079166666666667
```

Figure: Accuracy for Auto-encoder with KMeans clustering

```
print(cm)
```

```
[[ 806 4699    0   15   40    8   72  268   72   20]
 [ 291   44    0   41 5529    1   20   30   29   15]
 [  69   37    0 2195   11    2  770 2846   57   13]
 [5366  188    0    7   94    3   99  226   14    3]
 [ 839   13    0 1939    8    1 2266  898   33    3]
 [   3    5  781    5    1 1395   15   43   72 3680]
 [ 600 1349    0  466   10    5  537 2868  155   10]
 [   0    0 1400    1    0   35    1    0   13 4550]
 [ 197   19   42   17    9   35 2466  237 2870  108]
 [   0    1 2836    2    0 3096    0    0   12   53]]
```

Figure: Confusion Matrix for Auto-encoder with KMeans clustering

5.3 Auto-encoder with Gaussian Mixture Models:

```
print(gmm_acc)
```

```
0.6223666666666666
```

Figure: Accuracy for Auto-encoder with Gaussian Mixture Models

```
print(cm)
```

```
[[4243    0   24  704    0  254    0    0  775    0]
 [   7 5348    5  277    0  328    0    8   27    0]
 [  16    0 3715 1468    0  395    0    0  406    0]
 [ 177    8    7 5442    0  122    0    0  244    0]
 [  10    1 3689 1808    0  315    0    0  177    0]
 [   0    0    0    4    0 3286    0 2584    7  119]
 [1015    0 1570 2122    0  310    0    0  983    0]
 [   0    0    0    0    0  185    0 4502    2 1311]
 [   1    0   15   38    0  646    0   21 5275    4]
 [   0    0    0    1    0  387    0   79    2 5531]]
```

Figure: Confusion Matrix for Auto-encoder with Gaussian Mixture Models

6 Conclusion:

Three variations of clustering algorithms have been implementedFashion MNIST dataset. The normal KMeans algorithm has given accuracy of 55.32%. The accuracy of KMeans on the encoded images is found to be 60.79% and that of the Gaussian Mixture Model on the encoded images is found to be 62.23%.

7 References:

https://www.analyticsvidhya.com/blog/2019/10/gaussian-mixture-models-clustering/

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html

https://www.datacamp.com/community/tutorials/autoencoder-classifier-python

https://www.datacamp.com/community/tutorials/k-means-clustering-python

https://towardsdatascience.com/clustering-based-unsupervised-learning-8d705298ae51