Project 3: Cluster Analysis using unsupervised learning

UBIT Name: Prashi Khurana
UBIT Number: 50316796
prashikh@buffalo.edu

Abstract

In this project, we will learn how to train and perform 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.

1 Introduction

In this project we are focusing on how to perform cluster analysis on unsupervised learning data. There are 3 parts to this: a. Using K-means clustering algorithm b. Using an auto-encoder to reduce the dimensionality and then use that for K-means c. Using an auto-encoder based GMM.

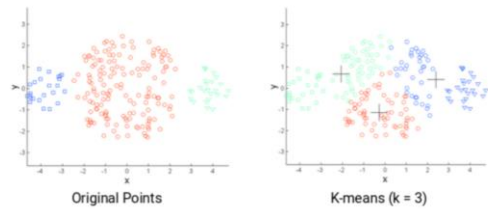
1.1 What is Unsupervised Data ?

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. [1]

2 Understanding the underlying algorithms:

2.1 K-means algorithm:

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into 'K' pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.[2]



[3]

2.2 GMM:

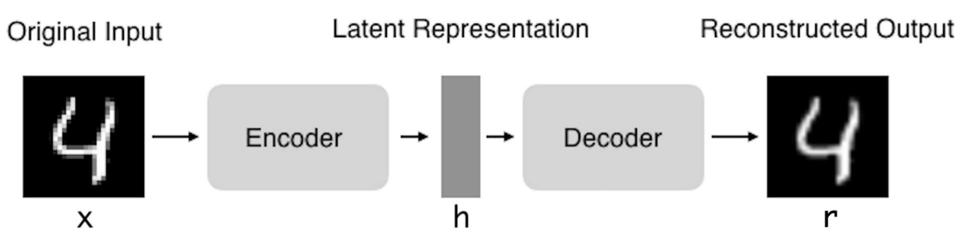
In real life, many datasets can be modeled by Gaussian Distribution (Univariate or Multivariate). So it is quite natural and intuitive to assume that the clusters come from different Gaussian Distributions. Or in other words, it is tried to model the dataset as a mixture of several Gaussian Distributions. This is the core idea of this model. [6]

2.3 Auto Encoder:

Let us first try to understand what are auto-encoders.

Autoencoders (AE) are neural networks that aims to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation. This kind of network is composed of two parts :

Encoder: This is the part of the network that compresses the input into a latent-space representation. It can be represented by an encoding function h=f(x).
Decoder: This part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function r=g(h).



What is the benefit of using auto encoders ?

Autoencoders are learned automatically from data examples. It means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input and that it does not require any new engineering, only the appropriate training data. [4]

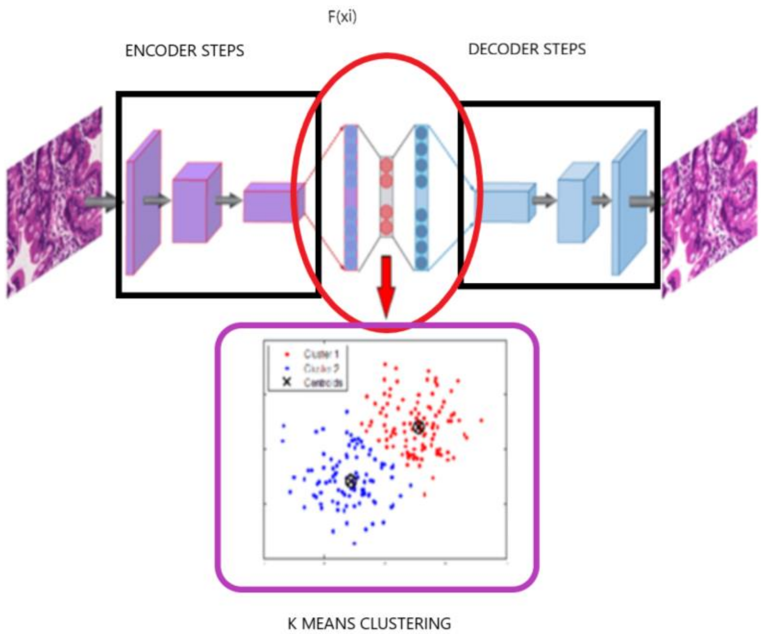
The main benefit of auto encoders are they can be used for dimensionality reduction and denoising of images.

2.3.1 Auto Encoders with KMeans:

KMeans is a clustering algorithm which is used to create clusters. But if the input has too many dimensions KMeans will get confused and would not be able to give a very good output. So we use an Auto-Encoder to reduce the dimensionality of the input and then pass it through KMeans.

Steps for the above algorithm:

1. Encode the input image
2. Get F(xi)
3. Do KMeans Clustering for the F(xi)



[5]

2.3.2 Auto Encoder and GMM:

The same as above we will use Auto Encoder but we will now use the GMM clustering algorithm.

3 Dataset Definition

The dataset is *Fashion-MNIST clothing images* . A few instances of data set:

Training Input size -> (60000, 784)
Training Output -> (60000,1)
Test Input size -> (10000, 784)
Test Output size->(10000,1)

4. Pre-processing

4.1 Preprocessing the data

The data which is generally used for training models might be inconsistent, incomplete and needs pre-processing. In our project, the following pre-processing functions were required:

4.1.1 Normalize the data

Normalization is an important part of the pre-processing of the data with machine learning. Normalization is the process of getting all the features of the of the instance in a common range. Gradient descent converges much faster when the features are normalized. The equation of normalization is X_train = X_train/255

5 Implementing the underlying algorithms in on our dataset

Now we will learn how to implement these models on our dataset.

5.1 KMeans

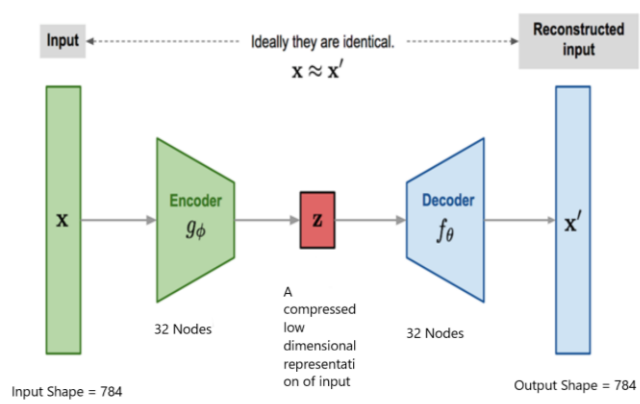
Since we have an output of 10 classes, we will be using 10 clusters. Here if we use say more than 10 clusters we will get a better accuracy but at what cost?

Here we find a useful method called the ‘elbow’ method where we find a graph for the SSE(sum of squared error) vs the number of clusters, and where we get the elbow point we use those number of clusters as the best clusters. But since the input size here has 10 clusters, we will use 10 clusters.

5.2 Auto Encoder:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
```

Architecture of Auto Encoder:



The function f(xi) will compress an image of 784 size to 32. As we can see in the output of the code we are getting a size of : EncodedImagesSHape (10000, 32), which is then passed on to the clustering algorithm.

5.2.1 Auto Encoder with KMeans

We will use our auto encoder with K means. This means we will train our model and then use the encoded layer as an input to the KMeans. As mentioned in section 2.3.1 we will then use this f(xi) -> (encoded layer from the autoencoder) as the input for the KMeans.

5.2.2 Auto Encoder with GMM

We will use our auto encoder with GMM. This means we will train our model and then use the encoded layer as an input to the GMM. As mentioned in section 2.3.2 we will then use this f(xi) -> (encoded layer from the autoencoder) as the input for the GMM.

6 Actual Implementation

6.1 KMeans

6.1.1 KMeans with Normalized Mutual Info Score:

1. Pre-process the data (3)
2. Assign a KMeans model with the cluster size as 10
3. Fit the KMeans algo with the data X
4. Predict the clusters with KMeans.predict
5. Find the normalized_mutual_info_score of the KMeans Model

6.1.2 KMeans with Accuracy:

1. Pre-process the data (3)
2. Assign a KMeans model with the cluster size as 10
3. Fit the KMeans algo with the data X
4. Use infer_cluster_labels to Associates most probable label with each cluster in KMeans model. Infer_Cluster_Labels from the KMeans ,Y
5. Use infer_data_labels Determines label for each array, depending on the cluster it has been assigned to.
6. Predict the X_clusters from kmeans.predict(X)
7. Get the predicted labels as predicted_labels = infer_data_labels(X_clusters, cluster_labels)and this returns predicted labels for each array
8. Get the accuracy score, accuracy_score from the KMeans.[8]

6.2 Auto-Encoder

1. Create a model as mentioned in the section 5.2.
2. encoding_dim = 32
3. input_img = Input(shape=(784,))
4. encoded = Dense(encoding_dim,activation='selu',kernel_regularizer=regularizers.l2(0.01))(input_img)
5. decoded = Dense(784, activation='sigmoid')(encoded)
6. autoencoder = Model(input_img,decoded)
7. encoder = Model(input_img,encoded)
8. encoded_input = Input(shape=(encoding_dim,))
9. decoder_layer = autoencoder.layers[-1]
10. decoder = Model(encoded_input, decoder_layer(encoded_input))
11. autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
12. autoencoder.fit(x_train,x_train, epochs=60, batch_size=2056, shuffle=True, validation_data=(x_test,x_test))
13. autoencoder.summary()
14. encoded_images = encoder.predict(x_test)

6.3 Auto Encoder with KMeans

1. Once the auto-encoder is set(as in section 6.2), pass the f(xi) to the KMeans algorithm (as in section 6.1)

6.4 Auto Encoder with GMM

1. Once the auto-encoder is set(as in section 6.2), pass the f(xi) to the GMM algorithm
2. clf = GmmMml()
3. clf.fit(encoded_image)
4. clf.predict(encoded_image)
5. Once you predict the encoded_images we can get the normalized_mutual_info_score

7 Results

7.1 Understanding Normalized Mutual Info Score:

Normalized Mutual Information (NMI) is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation). In this function, mutual information is normalized by some generalized mean of H(labels_true) and H(labels_pred)), defined by the average_method. This measure is not adjusted for chance. Therefore adjusted_mutual_info_score might be preferred. 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.[7]

Now we need to know the difference between accuracy and NMIS.

If we need to calculate the accuracy we need to know the labels assigned to the clusters. NMIS takes that into account so for our case, we can do with NMIS instead of the accuracy.

7.1.1 Understanding the results

	Accuracy	NMIS
Kmeans	0.56	0.5123
Auto Encoder with Kmeans		0.5323
Auto Encoder with GMM		0.5152

7.2 Confusion Matrix

7.2.1 KMeans with NMIS:

```
[[ 29   0 587   6 244   1   5  94   0  34]
 [890   0  50   0  29   0   0  22   0   9]
 [  4   0  19   4 342   0   4  61   0 566]
 [503   0 277   2 111   0   3  94   0  10]
 [ 27   0 136   4 159   0   5  42   0 627]
 [  0  45   0   0   6 227   0 650  72   0]
 [ 12   0 189  15 358   0   0 115   0 311]
 [  0   2   0   0   0 784   0  62 152   0]
 [  6   1   3 353  35  39 408  84  10  61]
 [  0 423   0   0   4  23   2  29 519   0]]
```

```
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.47      0.58      0.52      1000
           1       0.02      0.02      0.02      1000
           2       0.27      0.34      0.30      1000
           3       0.34      0.50      0.41      1000
           4       0.00      0.00      0.00      1000
           5       0.00      0.00      0.00      1000
           6       0.19      0.31      0.24      1000
           7       0.20      0.15      0.17      1000
           8       0.96      0.41      0.57      1000
           9       0.02      0.02      0.02      1000

    accuracy                           0.23     10000
   macro avg       0.25      0.23      0.22     10000
```

```
weighted avg       0.25      0.23      0.22     10000
```

7.2.2 KMeans with Accuracy:

```
 [[588  24  36   0   7  66 268   1  10   0]
 [ 49 889   2   0  11  19  30   0   0   0]
 [ 14   4 313   0 289  48 326   0   6   0]
 [281 490   3   0  12  74 135   0   5   0]
 [102  25 188   0 510  36 132   0   7   0]
 [  0   0   0   0   0 690   8 220   3  79]
 [185  11 144   0 194  97 356   0  13   0]
 [  0   0   0   0   0  84   0 865   1  50]
 [  3   4  61   0   7  69  56  48 752   0]
 [  0   0   1   0   1  36   7 107   4 844]]

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.47      0.59      0.52      1000
           1       0.61      0.89      0.72      1000
           2       0.00      0.00      0.00      1000
           3       0.00      0.00      0.00      1000
           4       0.39      0.63      0.48      1000
           5       0.52      0.65      0.58      1000
           6       0.28      0.36      0.31      1000
           7       0.73      0.79      0.76      1000
           8       0.94      0.76      0.84      1000
           9       0.77      0.94      0.85      1000

    accuracy                           0.56     10000
   macro avg       0.47      0.56      0.51     10000
weighted avg       0.47      0.56      0.51     10000
```

7.2.3 Auto Encoder with KMeans with NMIS:

```
[[  5   4   6   0   0  28 258   2 651  46]
 [  1   0  10   0   0 900  36   0  51   2]
 [  3   3 410   0   0   4 309   0  17 254]
 [  2   1  14   0   0 538 181   0 260   4]
 [  2   5 605   0   0  28 137   0 119 104]
 [  2   0   0 206 134   0 111 547   0   0]
 [ 11   2 263   0   0  12 370   3 216 123]
 [  1   0   0 165   2   0   0 832   0   0]
 [360 392   7   9   0   5 108  45   5  69]
 [  1   0   1 565 400   0   8  23   2   0]]

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       0.17      0.24      0.20      1000
           1       0.01      0.01      0.01      1000
           2       0.00      0.00      0.00      1000
           3       0.34      0.49      0.40      1000
           4       0.00      0.00      0.00      1000
           5       0.39      0.58      0.46      1000
           6       0.00      0.00      0.00      1000
```

	precision	recall	f1-score	support
7	0.00	0.00	0.00	1000
8	0.92	0.41	0.56	1000
9	0.00	0.00	0.00	1000
accuracy			0.17	10000
macro avg	0.18	0.17	0.16	10000
weighted avg	0.18	0.17	0.16	10000

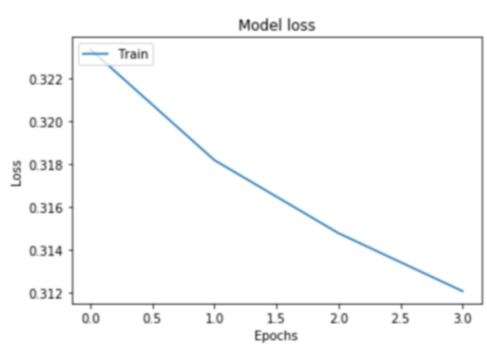
7.2.4 Auto Encoder with GMM with NMIS:

```
[[ 26 822  82  59  11   0   0   0   0   0]
 [  1  55  16 926   2   0   0   0   0   0]
 [122  76 775   2  25   0   0   0   0   0]
 [ 13 382  19 584   2   0   0   0   0   0]
 [ 57 191 718  16  18   0   0   0   0   0]
 [402   1   0   0  51 546   0   0   0   0]
 [131 334 497  15  23   0   0   0   0   0]
 [  5   0   0   0  13 982   0   0   0   0]
 [110  69  67   0 741  13   0   0   0   0]
 [ 67   0   0   0 627 306   0   0   0   0]]
```

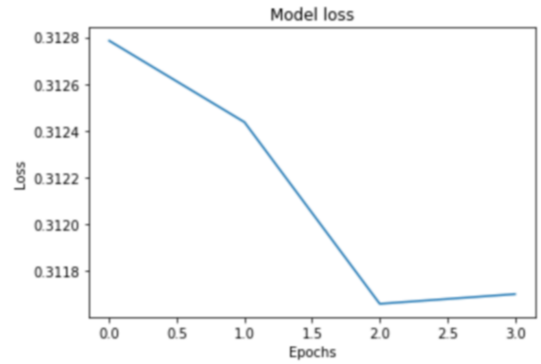
CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.03	0.03	0.03	1000
1	0.03	0.06	0.04	1000
2	0.36	0.78	0.49	1000
3	0.36	0.58	0.45	1000
4	0.01	0.02	0.01	1000
5	0.30	0.55	0.38	1000
6	0.00	0.00	0.00	1000
7	0.00	0.00	0.00	1000
8	0.00	0.00	0.00	1000
9	0.00	0.00	0.00	1000
accuracy			0.20	10000
macro avg	0.11	0.20	0.14	10000
weighted avg	0.11	0.20	0.14	10000

7.3 Graphs for Training Dataset Loss:



7.4 Graphs for Validation Dataset Loss



8 Conclusion

Once we have trained the logistic regression model, we need to understand the results of the model.

- In this project, we are trying to see the difference of applying different clustering algorithms
- Initially we will use the KMeans Clustering algorithm to see the accuracy of the input size which is around 56%, while the NIMS is around 51%. We can use the NMIS score instead of the accuracy as it performs the same operation.
- Auto encoder is used to reduce the dimensionality of the input
- After passing through the encoder we get a reduced dimensionality and then use KMeans, we can train them faster.
- We also see that the validation and training loss decrease as we increase the epochs.

References

[1] https://www.geeksforgeeks.org/supervised-unsupervised-learning/

[2]https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

[3] https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

[4] https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

[5] https://www.researchgate.net/figure/Structure-of-clustering-model-with-autoencoder-and-K-means-combination_fig2_332368916

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

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

[8]https://github.com/xoraus/K-Means-Clustering-for-Imagery-Analysis/blob/master/KMeans%20Clustering%20for%20Imagery%20Analysis%20(Jupyter%20Notebook).ipynb