Clustering of Fashion Apparels in their condensed form using AutoEncoder, K-Means and Gaussian Mixture Model

Vineeth Thayanithi
University at Buffalo
vthayani@buffalo.edu

Abstract

Unsupervised learning is a sub category of machine learning in which the data the is to be trained on contains no target variables. In this project we will try to classify images by using various classification strategies using unsupervised learning. The images that are being used are initially at a higher dimensional space, we then try to bring them into a lower dimensional space and classify them using their condensed image forms.

1 Introduction

1.1 Unsupervised Learning

Unsupervised Learning is a machine learning algorithm that obtains patterns from the data that is being provided without the use of any outcomes of the given dataset. Unsupervised Learning cannot be directly used for regression and classification since the dataset has no known results making the algorithm to be virtually impossible to learn from it. However, unsupervised learning can be used in places where the underlying structure of the data is to be known. Unsupervised learning aims to uncover the previously unknown patterns of the given dataset however, these patterns are vague approximations and a supervised learning model can achieve more things. There are several applications for a unsupervised learning and they include:

1.1.1 Clustering

Clustering is a grouping operation in which the data in a specific are more similar to the other data that are present in the same group than to the data that are present in different groups.Clustering is not a specific algorithm and is more than a general task. There exists many algorithms for clustering a group of data.

1.1.2 Anomaly detection

Anomaly detection is used for finding out flaws that exists in the given dataset. This is useful in pinpointing fraudulent transactions, discovering faulty pieces of hardware, or identifying an outlier caused by a human error during data entry.

1.1.3 Association Mining

Association mining tries to establish relationships between the items that are present in the dataset. This can be used for finding items that generally occur together. One of the real time applications of this kind of unsupervised learning algorithm is basket analysis, because it allows analysts to discover goods often purchased at the same time and develop more effective marketing and merchandising strategies.

1.1.4 Latent variable models

These unsupervised algorithms are used for preprocessing the dataset. The latent variable model can take a dataset with a high dimensional space and then reduce it to another equivalent set of lower dimensions.

1.2 AutoEncoder

An auto encoder is a artificial neural network that is typically used for encoding the given set of data,usually in a compressed form. Apart from learning the compressed representation, the reconstruction of the data from the compressed form is also learnt. Autoencoders are effectively used for solving many applied problems, from face recognition to acquiring the semantic meaning for the words. An auto encoder is made up of 2 parts, an encoder part that maps input to the latent representation and a decoder part that maps takes the latent representation of data and then maps it back again to the initial input. Autoencoders have been very famous for a very long time with the oldest application of it dating back to the 1980s.
The simplest architecture of auto encoder includes 3 layers of a neural network with one hidden layer. The output of the hidden layer is the latent representation of the data given. An autoencoder consists of two parts, the encoder and the decoder, which can be defined as transitions and , such that:

$$\phi : \mathcal{X} \rightarrow \mathcal{F} \tag{1}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X} \tag{2}$$

The mapping of the data to the latent representation can be represented as:

$$h = \sigma(Wx + b) \tag{3}$$

Here σ is the activation function such as Rectified Linear Unit, Sigmoid etc, W is the weights of the hidden layer and b is the bias variable.

The mapping of the latent representation back to the data can be represented as:

$$x = \sigma(Wh + b) \tag{4}$$

Here σ is the activation function such as Rectified Linear Unit, Sigmoid etc, W is the weights of the hidden layer, h is the latent representation of the data and b is the bias variable. The Weights , activation function and the bias from equation 4 are unrelated to that of equation 3. Auto encoders are trained to minimise the loss between the reconstructed data and the input data. A loss function such as mean squared error can be used for this purpose.

$$\mathcal{L}(\mathbf{x}, \mathbf{x'}) = \|\mathbf{x} - \mathbf{x'}\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W'}(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b'})\|^2 \tag{5}$$

Although this architecture has only one hidden layer, auto encoders can be extended with multiple hidden layers. These kinds of deep auto encoders have the following advantages over the architecture with one hidden layer:
1. Depth can yield better compression ratios compares to shallow or linear autoencoders.
2. Depth can exponentially reduce the amount of data required to learn some function.
3. Depth can also exponentially reduce the computational cost for representing some functions.

1.3 K-Means Clustering

K-Means is a prototype based clustering method in which the each cluster is represented using a prototype, typically the centroid of the similar data points. Although K-Means is really efficient at clustering spherical clusters, the number of clusters must be defined before k means algorithm is run. A wrong value for k can result in poor clustering performance.
The K-Means algorithm can be generally described as the following steps:

1. Randomly pick k centroids from the sample points as initial cluster centers.
2. Assign each sample to the nearest centroid μ (j), j belongs to 1, . . . , k.
3. Move the centroids to the center of the samples that were assigned to it.
4. Repeat steps 2 and 3 until the cluster assignments do not change or a user-defined tolerance or maximum number of iterations is reached.

We use Sum of Squared Errors between data points to find similarity between data.

1.4 Gaussian Mixture Model

The Gaussian Mixture Model is similar to the K-Means algorithm in a way that both K-Means and GMM performs clustering on a set of data. However, the GMM takes into consideration of co-variances. The co-variance refers to the shape of the bell curve. In two dimensions, the co-variance determines the shape of the ellipse that formed by the GMM rather than the circle formed by using K-Means. Secondly, K-Means performs hard clustering whereas GMM performs soft clustering i.e, GMM returns the probability with which a given data could lie in each of the possible clusters. The Gaussian Mixture Model has two steps, The Expectation step and the maximization step. The expectation predicts what the given sample looks like. The maximization step optimizes the log likelyhood of the samples.

2 Dataset

We would be using the Fashion-MNIST data set for implementing the neural net from scratch, dense neural net using keras and convolutional neural net using keras. The Fashion-MNIST is a replacement dataset to the conventional MNIST.The MNIST consists of pictures of size 28x28 which contains handwritten numerical values from 0-9. Although, the MNIST is the most popular dataset for starting with neural networks, there also exists several issues with the MNIST. The MNIST dataset is claimed to be easy in a way that it can convultional networks can reach an accuracy of 99.7% and classic machine learning strategies producing an accuracy of 97%. The MNISt dataset is also overused and is not efficient in solving computer vision problems. Hence we move on to the Fashion-MNIST. The Fashion-MNIST has a similar collection with 60000 examples in the training set and 10000 in the test set.This dataset contains images of size 28x28 in grayscale of c The Fashion - MNIST has 10 classes in total.

3 Pre-Processing

3.1 Baseline K-Means

We use the load_mnist method from the mnist_reader.py for loading the train and the test sets. The parameter kind in the load_mnist funtions specifies whether the data is a train or a test set, train and t10k specifies the train and the test sets in the kind parameter. Once the dataset is imported, we would have to normalize the dataset. We could achieve normalization by dividing the datasets by 255(Maximum RGB value - Minimum RGB value). Normalization is required for neural networks to have a good performance. Normalization helps reduce the differences in the inputs given to the neural networks.

3.2 Categorization of latent codes using auto encoders with K-Means

Importing and normalization of the dataset is the same as it was for task1. To fit into the neural network, the dataset is reshaped in the shape of (60000,28,28,1) and (10000,28,28,1). From the training set, 20 percent is used for validation.

3.3 Categorization of latent codes using auto encoders with GMM

All the preprocessing tasks are the same as task 2.

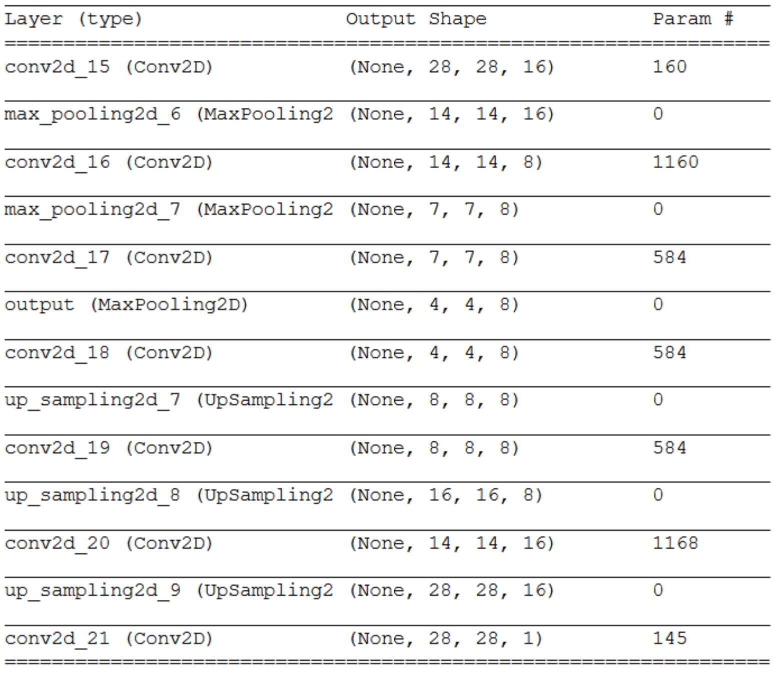
4 Architecture

4.1 Baseline K-Means

In task 1, we assume that we do not know the number of clusters that are present in the dataset. Hence we try to predict the number of clusters with K-Means by using the elbow method, i.e., we iterate through differnet values of K running K means for each K on the training dataset. We obtain the inertias of each of these iterations and plot a graph of inertias against K. Once we do this we would observe a steep decline in the inertias at one point in the graph, we then consider that to be our K and continue with the rest of the task. The model with the predict K is then again fit on the training dataset. Then using the trained clustering model, the labels of the test set are predicted. The predicted labels may need not necessarily be the same as that of original labels. Hence in order to construct the confusion matrix, we perform linear assignment of the randomly ordered confusion matrix that we obtained.

4.2 Categorization of latent codes using auto encoders with K-Means

In task 2, we use an auto encoder to shrink down the input features. This would result in a better clustering performance. For the construction of the auto encoder, the architecture as shown below is used.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 7, 7, 8)           584
_________________________________________________________________
output (MaxPooling2D)        (None, 4, 4, 8)           0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 4, 4, 8)           584
_________________________________________________________________
up_sampling2d_7 (UpSampling2 (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d_8 (UpSampling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_9 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 28, 28, 1)         145
=================================================================
```

Figure 1: Architecture of Auto Encoder

The architecture of the autoencoder consists of 13 layers in total, 6 layers for encoding and 7 layers for decoding. Since the input dataset are images we use 3 convolutional layers with 3 max pooling layers to downsample the outputs in between each of the layers. The output of the encoding stage is an array of size 4,4,8. This is passed as the input to the decoder phase. The decoder consists of the same architecture as that of the encoder but in the reverse order. The max pooling layers in the encoder are replaced with upsampling layers in the decoder. All the layers except the last layer of the autoencoder, uses relu as its activation funtion and the last layer uses sigmoid. The auto encoder model is compiled with Adam optimizer and mean squared error as its optimization function. The model is trained for 100 epochs with 20 percent of the training set being the validation set. After training the auto encoder, we use the outputs from the last stage of the encoding phase to create a

new model just for encoding. We use this to obtain the encoded format of the input image.
This encoded format is used for classification similar to that done in task1.

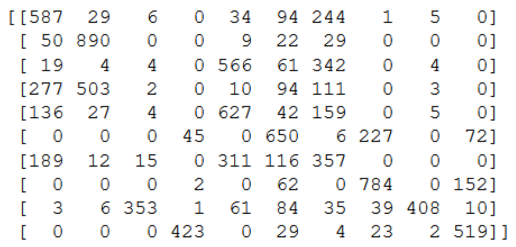
4.3 Categorization of latent codes using auto encoders with GMM

The task 3 follows the same work flow as that of task 2, We would just have to change how the encoded images are clusters. In task 2, we relied on K-Means Algorithm for clustering the condensed Images. In task 3, we would move on to the soft clustering algorithm Gaussian Mixture Model that has been introduced and explained previously. We use Sklearn's GaussianMixture method to cluster the output from the encoding layer.

5 Results

5.1 Baseline K-Means

From the elbow method graph that we obtained and with the prior knowledge of the dataset, we could approximate the number of clusters as 10. We then fit the train set and then predict the test set. The accuracy obtained from this was 48.26. The confusion matrix obtained is as below

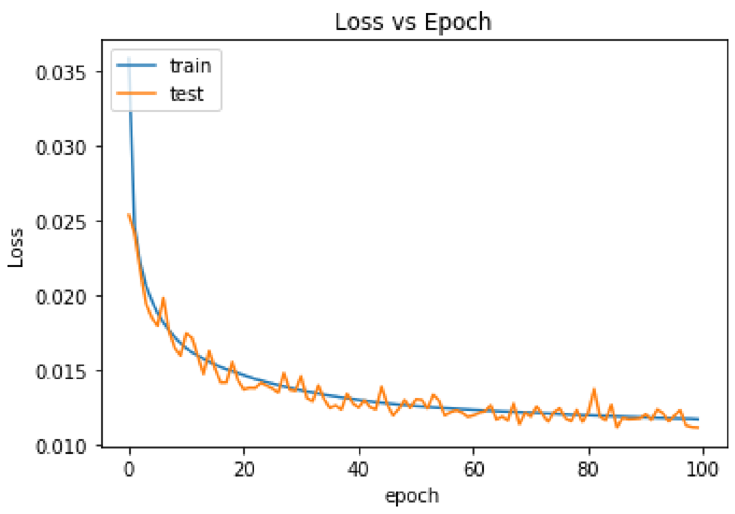


```
[[587  29   6   0  34  94 244   1   5   0]
 [ 50 890   0   0   9  22  29   0   0   0]
 [ 19   4   4   0 566  61 342   0   4   0]
 [277 503   2   0  10  94 111   0   3   0]
 [136  27   4   0 627  42 159   0   5   0]
 [  0   0   0  45   0 650   6 227   0  72]
 [189  12  15   0 311 116 357   0   0   0]
 [  0   0   0   2   0  62   0 784   0 152]
 [  3   6 353   1  61  84  35  39 408  10]
 [  0   0   0 423   0  29   4  23   2 519]]
```

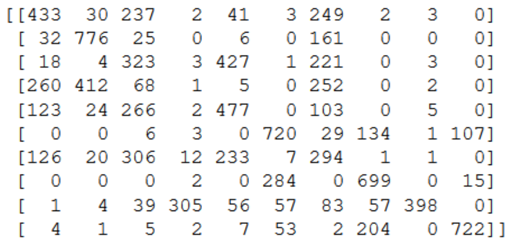
Figure 2: Confusion Matrix for K-Means Basis

5.2 Categorization of latent codes using auto encoders with K-Means

In task 2, the auto encoder is trained using the training and 20 percent of the training set is used as the validation. The loss obtained from training the auto encoder after 100 epochs is 0.0117 with a validation loss of 0.0112. The figure below is the loss vs epochs graph for the auto encoder.



The accuracy of the K-Means clustering algorithm by passing the condensed test set was observed to be 48.43. The confusion matrix for obtained is as below

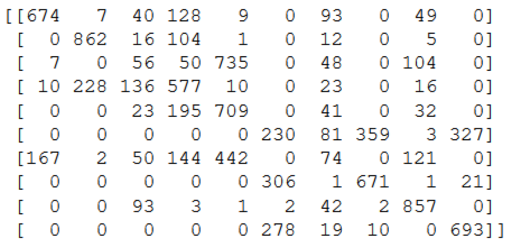


```
[[433  30 237   2  41   3 249   2   3   0]
 [ 32 776  25   0   6   0 161   0   0   0]
 [ 18   4 323   3 427   1 221   0   3   0]
 [260 412  68   1   5   0 252   0   2   0]
 [123  24 266   2 477   0 103   0   5   0]
 [  0   0   6   3   0 720  29 134   1 107]
 [126  20 306  12 233   7 294   1   1   0]
 [  0   0   0   2   0 284   0 699   0  15]
 [  1   4  39 305  56  57  83  57 398   0]
 [  4   1   5   2   7  53   2 204   0 722]]
```

Figure 3: Confusion Matrix for Auto Encoder based K-Means Clustering

5.3 Categorization of latent codes using auto encoders with Gaussian Mixture Model

In task 3, the auto encoder is trained using the training and 20 percent of the training set is used as the validation. The loss obtained from training the auto encoder after 100 epochs is 0.0117 with a validation loss of 0.0112. The accuracy of the Gaussian Mixture Model algorithm by passing the condensed test set was observed to be 54.03. The confusion matrix for obtained is as below



```
[[674   7  40 128   9   0  93   0  49   0]
 [  0 862  16 104   1   0  12   0   5   0]
 [  7   0  56  50 735   0  48   0 104   0]
 [ 10 228 136 577  10   0  23   0  16   0]
 [  0   0  23 195 709   0  41   0  32   0]
 [  0   0   0   0   0 230  81 359   3 327]
 [167   2  50 144 442   0  74   0 121   0]
 [  0   0   0   0   0 306   1 671   1  21]
 [  0   0  93   3   1   2  42   2 857   0]
 [  0   0   0   0   0 278  19  10   0 693]]
```

Figure 4: Confusion Matrix for Auto Encoder based GMM Clustering

6 Conclusion

We have successfully trained an unsupervised learning model along with an auto encoder. From the results, we observe that these models perform better when the dimensionality of the input data is lower but not too low for the results to deteriorate. The accuracy of task 1,task 2 and task 3 are 48.26% , 48.43% and 54.03% respectively. Hence, we can conclude that the best performance is obtained using the Gaussian Mixture Method for this task.

References

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

[2] https://ramhiser.com/post/2018-05-14-autoencoders-with-keras/

[3] https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html

[4] https://smorbieu.gitlab.io/accuracy-from-classification-to-clustering-evaluation/

[5] https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e

[6] https://towardsdatascience.com/kmeans-clustering-for-classification-74b992405d0a

[7] https://en.wikipedia.org/wiki/Autoencoder