IMAGE CLASSIFICATION USING UNSUPERVISED LEARNING METHODS

Uneeth Akula
50317480
Dept. of Computer Science
University at Buffalo
Buffalo, NY 14221
uneethak@buffalo.edu

Abstract

There are many different technique and models to solve the problem of image classification. It is important for us to fully understand the principles behind each model and its performance based on the dataset. The purpose of this project is to gain a deeper understanding of different unsupervised classification models, and how they perform on the Fashion-MNIST dataset. Cluster analysis is one of the unsupervised machine learning technique which doesn't require labeled data. First, KMeans algorithm was used to cluster original data space of Fashion-MNIST model using Sklearns library. Then, a convolutional autoencoder was used to condense the representation of the unlabeled data and then the KMeans clustering was performed in this space. Finally, the Gaussian Mixture Model was used to perform clustering on the condensed representation of the dataset using Tensorflow and Keras with various activation functions to boost the performance of the models. .

1 Introduction

1.1 Clustering

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

K-means is a partition or centroid-based clustering method.

Gaussian Mixture Model is a probabilistic distribution model

Mixture models make use of latent variables to model different parameters for different groups (or clusters) of data points. A Gaussian Mixture Model (GMM) is a mixture of Gaussians with K components.

1.2 Dense Layers:

A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected. The layer has a weight matrix W, a bias vector b, and the activations of previous layer a. The following is the docstring of class Dense from the keras documentation:

output = activation(dot(input, kernel) + bias)

where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer

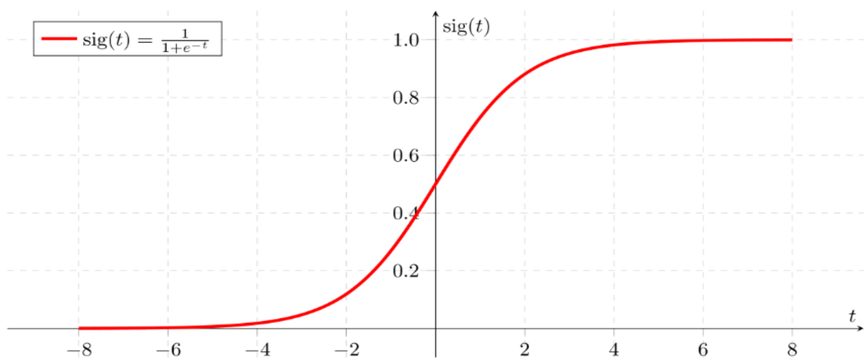
1.2.1 Sigmoid Activation function

The logistic regression model uses the Logistic function to limit the output of a linear equation between 0 and 1.

The logistic function is defined as:

$$g(z) = \frac{1}{1 + e^{-z}}$$

For large positive values of z, the sigmoid function should be close to 1, while for large negative values of z, the sigmoid function will be close to 0.

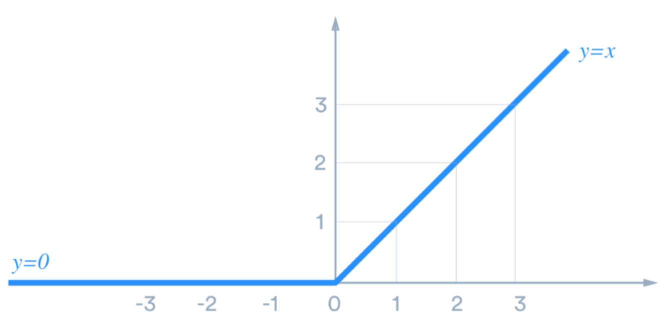


The hypothesis function for logistic regression:

$$h_\theta(\mathbf{x}) = g(\theta^\top \mathbf{x})$$

1.2.2 ReLU Activation function

ReLU stands for rectified linear unit, and is a type of activation function. Mathematically, it is defined as *y = max (0, x)*. Visually, it looks like the following:



ReLU Graph

ReLU is linear (identity) for all positive values, and zero for all negative values. This means that:

1.3 Keras and Tensorflow:

Keras is an open-source, high-level neural networks API, which is written in Python and is used for training deep learning models. TensorFlow is an open-source library, which is used for Machine Learning applications such as Neural Networks. Keras is capable of running on TensorFlow and also supports Convolution Neural Networks.

1.4 Auto Encoder:

An autoencoder is an unsupervised machine learning algorithm that takes an image as input and tries to reconstruct it using fewer number of bits from the bottleneck also known as latent space. The image is majorly compressed at the bottleneck. The compression in autoencoders is achieved by training the network for a period of time and as it learns it tries to best represent the input image at the bottleneck. The general image compression algorithms like JPEG and JPEG lossless compression techniques compress the images without the need for any kind of training and do fairly well in compressing the images.

In this project I used a DeepNet Auto-Encoder. A deep autoencoder is composed of two, symmetrical deep-belief networks that typically have four or five shallow layers representing the encoding half of the net, and second set of four or five layers that make up the decoding half.

1.5 Gaussian Mixture Model:

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. Expectation-Maximization (EM) is a statistical algorithm for finding the right model parameters. We typically use EM when the data has missing values, or in other words, when the data is incomplete.

These missing variables are called **latent variables**. We consider the target (or cluster number) to be unknown when we're working on an unsupervised learning problem.

Broadly, the Expectation-Maximization algorithm has two steps:

E-step: In this step, the available data is used to estimate (guess) the values of the missing variables

M-step: Based on the estimated values generated in the E-step, the complete data is used to update the parameters

Expectation-Maximization is the base of many algorithms, including Gaussian Mixture Models.

1.6 Hyper Parameters

Number of Layers, Number of hidden nodes in each layer, parameters in the Kmeans and GMM functions are the hyper parameters which are tuned in order to increase the accuracy. I used 3 hidden layers,1st with 2000, 2nd with 1000 and 3rd with 500 nodes. After training the model with the train data set, the accuracy and confusion matrix for the test data is calculated.

2 Dataset

The Fashion-MNIST is a dataset of Zalando's article images. It consists of 60,000 training examples and 10,000 testing examples. It shares the same image size and structure for the training and testing splits. There are 10 different classes and each example is a 28x28 greyscale image which is associated with one of the class. In the dataset, each example is an image which has 28 pixels as rows and 28 pixels as columns, which constitute to a total of 784 pixels in total. Each pixel has a pixel-value for it. This pixel-value is an integer which ranges from 0 to 255, where the number implies the darkness of the pixel. A low pixel-value indicates that the pixel is light and a high pixel-value indicates that the pixel is dark. This means that each row is a separate image and the Dataset consists of 785 columns, where the first column is the class label and the remaining columns are the pixel-values associated with the image. Each example of the Dataset is associated with one of the following 10 class labels:

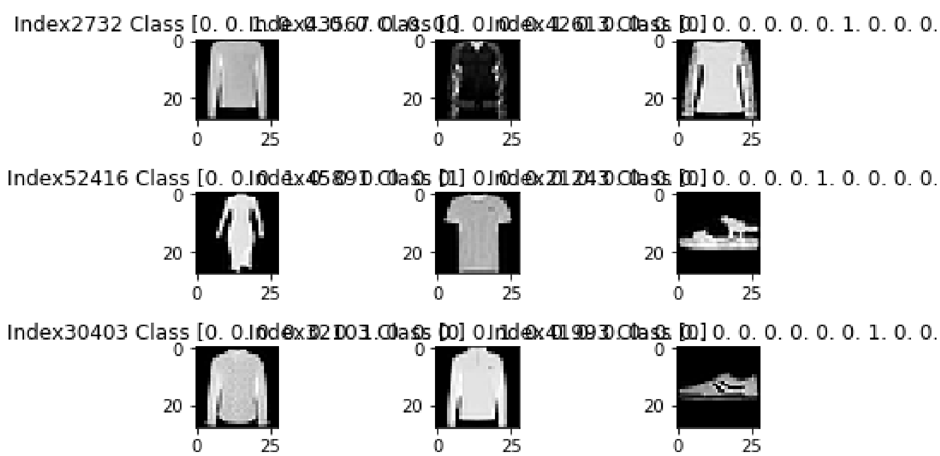
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image. To locate a pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27. The pixel is located on row i and column j of a 28 x 28 matrix.

2.1 Labels:

Each example in training data and test data belongs to one of the below 10 classes, also can be said as assigned to one of the below 10 labels.

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

Labels in the Fashion MNIST dataset



Classes in dataset

3 Work:

3.1 Data Pre-processing:

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.

3.2 Regularization:

I used **Early Stopping Regularization** to prevent overfitting with patience = 5. So, when there is an increase in the loss value consecutively for 5 iterations, it stops running.

3.3 KMeans Clustering without Dimensionality Reduction

To design our Machine Learning model, we use a clustering algorithm KMeans. The data must be preprocessed before training the network. We first split the input data into 2 parts. Training data and Testing data. Thus, we get four numpy arrays – Y_test, X_train, X_test, Y_train.
The pixel values in every image fall in the range of 0 to 255, depending on the presence of light. We need to scale these values before feeding them to the network. Hence, we divide each of these values by 255 to bring them to a range of (0,1). Both training and test set are preprocessed the same way.

3.4 DeepNet AutoEncoder + KMeans/GMM

In this task, we use an open source neural network library Keras and TensorFlow. We first split the input data into 2 parts. Training data and Testing data. Thus, we get four numpy arrays – X_test, X_train, y_test, y_train. The pixel values in every image fall in the range of 0 to 255, depending on the presence of light. We need to scale these values before feeding them to the network. Hence we divide each of these values by 255 to bring them to a range of (0,1). Both training and test set are preprocessed the same way. We flatten the image from 28x28 to 784 for easier processing.

3.5 Training Auto Encoder

Now, an encoder network has been defined. It has 3 encoder levels and 3 decoder levels. To compile the model and decrease the loss, I used Adam optimizer. The network is now fitted on the training set and validated on the validation set. For validation set, the training set has been divided. Now, the main objective of the auto-encoder is to reduce the dimensions of the input images to help with clustering. Since the images have 28*28=784 dimensions, clustering would be difficult. So, once the dimensions are reduced, the computation would become easy.

3.6 Taking the Encoder Output:

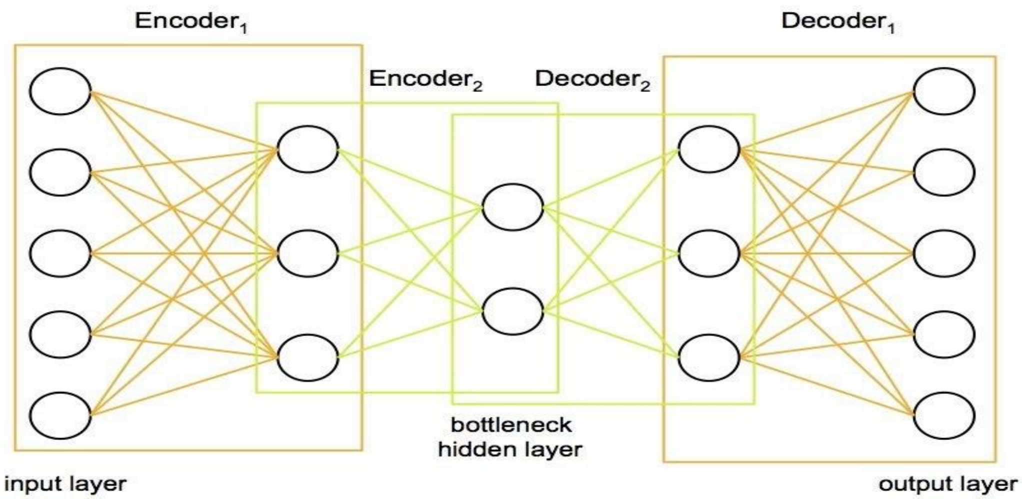
By training the auto-encoder, the model has now learned to compress each image into latent floating-point values. Now, the test set is passed through the network and the output (encoded images) is taken from the encoder, and the KMeans algorithm is applied on this output to generate the cluster centroids.

3.7 Applying Gaussian Mixture Model:

Now, on the same encoded images, Gaussian Mixture Model is applied to generate the cluster centroids. Normally, if GMM was to be applied on the dataset with 784 dimensions, it would require heavy computation, and the performance would also be low.

4. Architecture of the Model

4.1 Auto-Encoder



Sigmoid and ReLU activation functions were used. ReLU has a less computation when compared to other activation functions such as sigmoid and tanh, which makes it a better choice for deep neural networks.

4.2 Deep Auto-Encoders:

The auto encoder is a symmetric model that compresses the image and decompresses it. The auto encoder can be of different types like he Convolutional network, Deep Auto-Encoder which used dense layers to build the encoder and decoder. A deep autoencoder is composed of two, symmetrical deep-belief networks that typically have four or five shallow layers representing the encoding half of the net, and second set of four or five layers that make up the decoding half.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 784)	0
dense_17 (Dense)	(None, 2000)	1570000
dense_18 (Dense)	(None, 1000)	2001000
dense_19 (Dense)	(None, 500)	500500
dense_20 (Dense)	(None, 10)	5010
dense_21 (Dense)	(None, 500)	5500
dense_22 (Dense)	(None, 1000)	501000
dense_23 (Dense)	(None, 2000)	2002000
dense_24 (Dense)	(None, 784)	1568784

Total params: 8,153,794
Trainable params: 8,153,794
Non-trainable params: 0

The first 4 dense layers in the figure are the encoder part that gives us the compressed images and the remaining 4 layers are the decoder part.

4.3 Adam Optimization

Stochastic gradient descent (often abbreviated as SGD) is an iterative method for optimizing an objective function with suitable smoothness properties. It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data).

The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

5 Results

5.1 K-Means Clustering without Auto Encoder

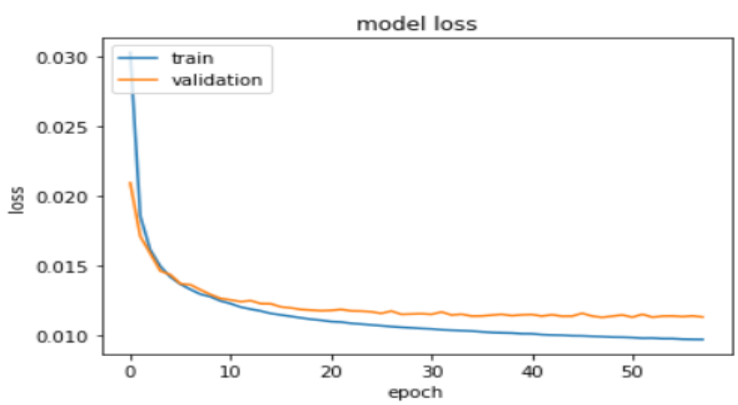
With n_clusters=10

```
kmeans_acc = metrics.accuracy_score(y_train, predicted_y)
print("Accuracy:",kmeans_acc*100,"%")
```

```
Accuracy: 55.34 %
```

5.2 Training Auto-Encoder:

Training and Validation Loss vs No. of iterations:



5.3 K-Means Clustering with Auto Encoder

Accuracy

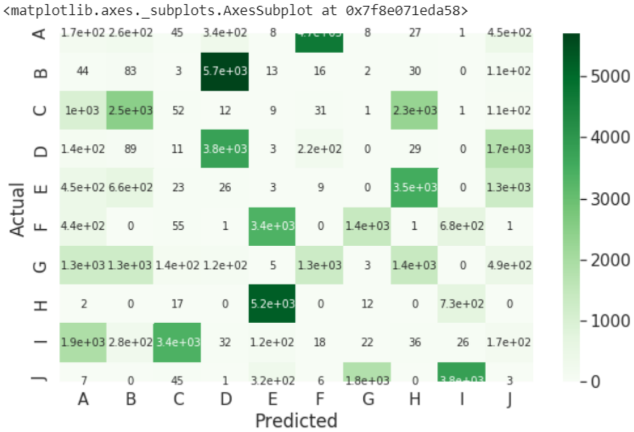
```
print("Accuracy",kmeans_acc*100,'%')
```

```
Accuracy 57.13 %
```

Confusion Matrix

```
[[ 168  262   45  343    8 4687    8   27    1  451]
 [  44   83    3 5697   13   16    2   30    0  112]
 [1002 2479   52   12    9   31    1 2303    1  110]
 [ 137   89   11 3767    3  225    0   29    0 1739]
 [ 453  658   23   26    3    9    0 3519    0 1309]
 [ 442    0   55    1 3381    0 1443    1  676    1]
 [1268 1262  139  116    5 1310    3 1411    0  486]
 [   2    0   17    0 5236    0   12    0  733    0]
 [1909  280 3392   32  116   18   22   36   26  169]
 [   7    0   45    1  318    6 1803    0 3817    3]]
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e071eda58>
```



5.4 Gaussian Mixture Model Clustering without Auto Encoder

Accuracy=60.8%

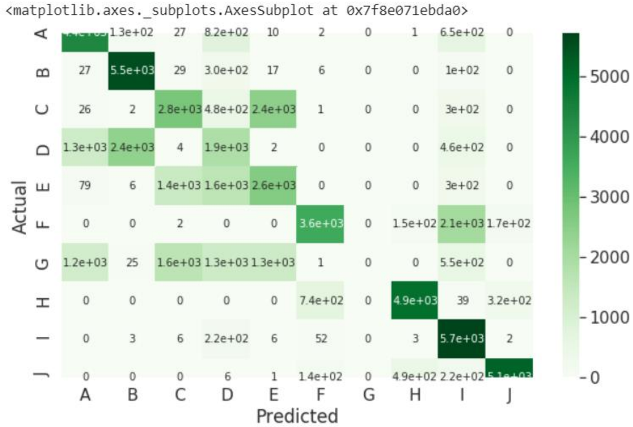
```
print("Accuracy",gm_acc*100,'%')
```

```
Accuracy 60.795 %
```

Confusion Matrix

```
[[4364  132   27  816   10    2    0    1  648    0]
 [  27 5513   29  305   17    6    0    0  103    0]
 [  26    2 2755  480 2441    1    0    0  295    0]
 [1264 2396    4 1875    2    0    0    0  459    0]
 [  79    6 1389 1628 2599    0    0    0  299    0]
 [   0    0    2    0    0 3608    0  149 2075  166]
 [1170   25 1649 1298 1311    1    0    0  546    0]
 [   0    0    0    0    0  735    0 4910   39  316]
 [   0    3    6  215    6   52    0    3 5713    2]
 [   0    0    0    6    1  145    0  488  220 5140]]
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e071ebda0>
```



6 Conclusion

Successfully trained the models using the given Fashion MNIST data and tested the model. As we can clearly observe, the auto encoder with convolution layer with pooling and dropout increases accuracy and the image is reconstructed close to the original one. Thus, by efficiently representing the high dimensionality data in lesser number of dimensions with great accuracy helps to attain the accuracy in clustering using KMeans and GMM. The Baseline K-Means Acccuracy has been found to be 55.34%. The accuracy of KMeans on the encoded images is found to be 57.13%. The accuracy of the Gaussian Mixture Model on the encoded images is found to be 60.8%. Further by tuning hyper parameters in an efficient way gives us optimal accuracy and desired results.

7 References

[1]https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/K-Means_Clustering.pdf K-Means Clustering

[2] https://www.datacamp.com/community/tutorials/autoencoder-keras-tutorial Implementing Autoencoders in Keras.

[3]https://www.researchgate.net/figure/Stacked-autoencoders-architecture_fig21_319524552 Architecture of AutoEncoder

[4] https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping// Early Stopping to Halt training of NN at right Time.

[5]https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html Pandas.DataFrame Pandas

[6]https://docs.scipy.org/doc/numpy/reference/ Numpy

[7] https://www.tensorflow.org/install/pip Installing Tensorflow

[8] https://inmachineswetrust.com/posts/deep-learning-setup/ Installing Keras

[9] https://skymind.ai/wiki/deep-autoencoder Deep AutoEncoders