CLUSTER ANALYSIS USING UNSUPERVISED LEARNING TECHNIQUES

Sriparna Chakraborty
Department of Computer Science
University of Buffalo
Buffalo, NY 14214
sriparna@buffalo.edu

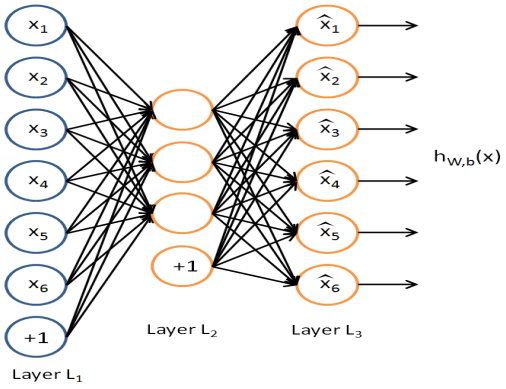
ABSTRACT

The purpose of this experiment is to perform cluster analysis on fashion MNIST dataset using unsupervised learning. Cluster analysis is one of the unsupervised machine learning techniques which does not require labeled data. I have used the Fashion-MNIST dataset of Zalando's article images for training, validation and testing. The dataset contains 60,000 examples of training data and 10,000 examples of testing data. The clustering accuracies have been measured for all the techniques from the confusion matrices built for the models. The training and validation losses have been plotted against the number of epochs while training the auto-encoder. The models have been built using Scikit-learn and Keras libraries.

1 INTRODUCTION

Unsupervised learning is a group of machine learning algorithms and approaches that work with a kind of "no-ground-truth" data. It helps find previously unknown patterns in data set without pre-existing labels. The only requirement to be called an unsupervised learning method is to learn a new feature space that captures the characteristics of the original space while maximising some objective function. Two of the main methods used in unsupervised learning are principal component and cluster analysis.

Cluster analysis is used in unsupervised learning to group, or segment, datasets with shared attributes in order to extrapolate algorithmic relationships. It is a branch of machine learning that groups the data that has not been labelled, classified or categorized. Instead of responding to feedback, cluster analysis identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data. This approach helps detect anomalous data points that do not fit into either group.



The first model only uses K-Means for performing the cluster analysis task. The K-Means 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.

In this experiment, we have performed the following three tasks based on Clustering analysis algorithms:

1. Using K-Means algorithm to cluster original data space of Fashion-MNIST dataset using Sklearns library.
2. To build an Auto-Encoder based K-Means clustering model to cluster the condensed representation of the unlabelled fashion MNIST dataset using Keras and Sklearns library.
3. To build an Auto-Encoder based Gaussian Mixture Model clustering model to cluster the condensed representation of the unlabelled fashion MNIST dataset using Keras and Sklearns library.

The accuracy, precision and recall have been presented in the form of a classification report for each model. The graph for Training loss and Validation loss have been plotted against each epoch while training the Auto-encoder.

2 DATASET

For training and testing of our classifiers, we have used the Fashion-MNIST dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. 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 below), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

Each training and test example is assigned to one of the labels as shown below:

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

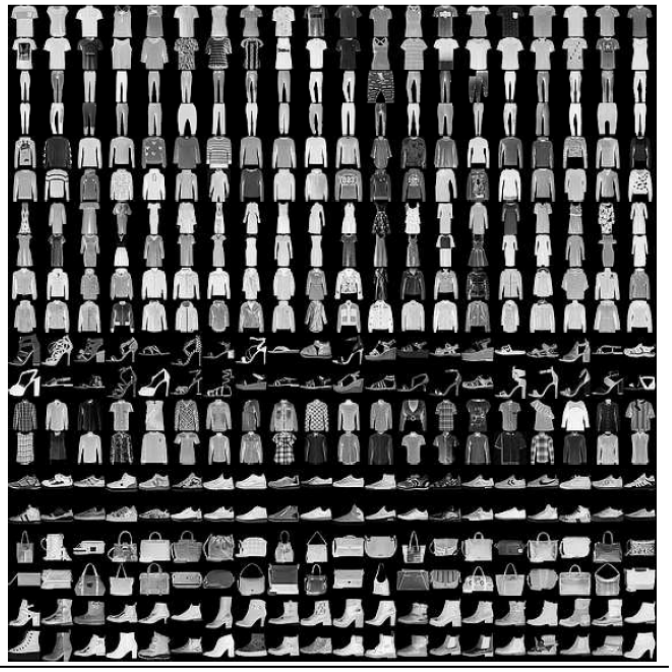
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 2: Example of how the data looks like

Each of the training datasets X_train (features) and Y_train (labels) are of dimensions 60000x784 and 60000x10 respectively and each of the testing datasets X_test and Y_test is of dimensions 10000x784 and 10000x10 respectively. The label datasets have been encoded using **one-hot encoding** to make it more expressive and efficient for proper prediction purposes.

The dataset is loaded by importing mnist_reader package in python and keras.datasets.fashion_mnist in Keras framework.

3 PREPROCESSING

Data preprocessing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn. Therefore, it is extremely important that we preprocess our data before feeding it into our model.

The preprocessing steps that I performed in this experiment on the dataset are as follows:

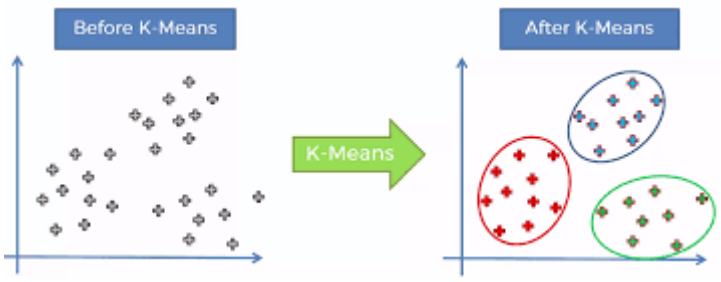
1. The feature datasets X_train and X_test have been normalized by dividing each value by 255 as our input dataset consists of images only with a pixel range from 0 to 255. So it is highly advisable to normalize each largely varying pixel value to a standard value so that its mean and standard deviation are uniform.
2. The training and the testing data have been rescaled with the maximum pixel value of the training and the testing data.
3. The training and the testing data have been scaled to range between 0 and 1.

4 ARCHITECTURE

TASK 1:

The first task is to cluster the original data space of the Fashion-MNIST dataset into 10 clusters using K-means algorithm. For this, I have used Sklearn library functions and Keras utilities. The sklearn.cluster module has KMeans class that is used to create an object of that class and then perform the necessary activities.

The loss function used in this task is the Sum of Squared Error distance between the cluster points and the centroids of each cluster.



After training the dataset, K-means is applied on the testing dataset to get the predicted output. Accuracy is calculated using the normalized_mutual_info_score of sklearns.metrics.cluster library. Initially the accuracy is found to be in the range 50-60 %. To improve this, I have applied a classifier on the clustered dataset and am able to improve the accuracy to be >80%.

TASK 2:

Using an Auto-encoder: "Auto-encoding" 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. Auto-encoder uses compression and decompression functions which are implemented with neural networks as shown in the figure below. Or in other words, auto-encoders help in reducing the dimensionality of high dimensional datasets.

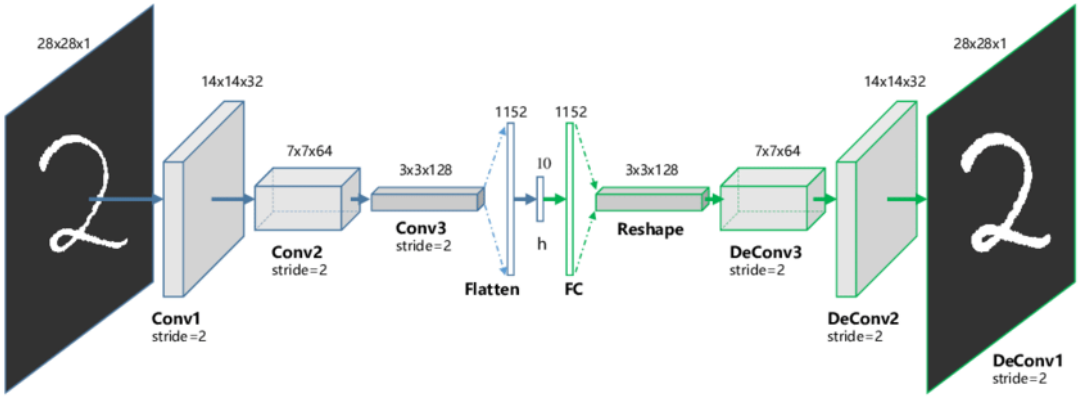


Figure showing how an auto-encoder model works on a handwritten digit dataset

To build an auto-encoder, three components are required:

1. An encoding function

2. A decoding function
3. A distance function between the amount of information loss between the compressed representation of the 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 optimized to minimize the reconstruction loss, using Stochastic Gradient Descent.

2 a. **Auto-encoder with K-means clustering**:

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

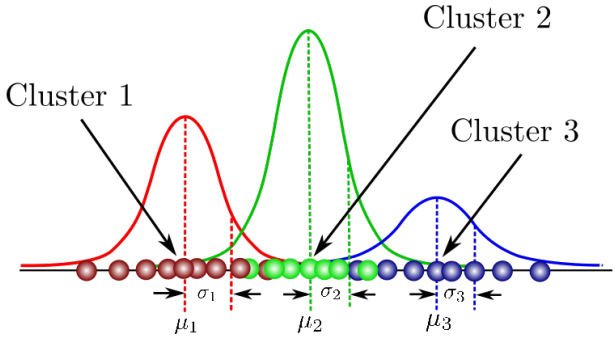
$$\sum_{i=0}^{n} \min_{\mu_j \epsilon C}(||x_i - \mu_j||\text{^}2)$$

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 high- dimensional 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.

The auto-encoder is first trained on the training dataset for 50 epochs and then the encoding layer has been extracted and K-means is applied on the encoded training data. After training the dataset, K-means is applied on the encoded testing dataset to get the predicted encoded output. The same has been used to calculate the accuracy. Accuracy is calculated using the normalized_mutual_info_score of sklearns.metrics.cluster library. Initially the accuracy is found to be in the range 50-60 %. To improve this, I have applied a classifier on the clustered dataset and am able to improve the accuracy to be >80%.

2 b. **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 Gaussian Mixture 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. A GaussianMixture.fit method is provided in sklearn.mixture module that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belong to using the GaussianMixture.predict method.

The auto-encoder is first trained on the training dataset for 50 epochs and then the encoding layer has been extracted and GMM is applied on the encoded training data. After training the dataset, GMM is applied on the encoded testing dataset to get the predicted encoded output. The same has been used to calculate the accuracy. Accuracy is calculated using the normalized_mutual_info_score of sklearns.metrics.cluster library. Initially the accuracy is found to be in the range 50-60 %. To improve this, I have applied a classifier on the clustered dataset and am able to improve the accuracy to be >80%.

N.B.: Initially a simple auto-encoder was implemented for classifying the data but after applying convolutional auto-encoder, the accuracy was found to be better. For the encoder model, I have used a CNN with 3 hidden layers and the decoder with 4 hidden layers.

CONFUSION MATRIX:

A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).

Here confusion matrix is built for multiclass classifiers with a size of 10x10. For each class (0-9), the maximum value of the cell corresponding to that particular class is the TP value of that class.

The four metrics of the confusion matrix are:

a. TP/ True Positive: Actual output is positive and Predicted is positive
b. TN/ True Negative: Actual output is negative and Predicted is negative
c. FP/ False Positive: Actual output is negative and Predicted is positive
d. FN/ False Negative: Actual output is positive and Predicted is negative

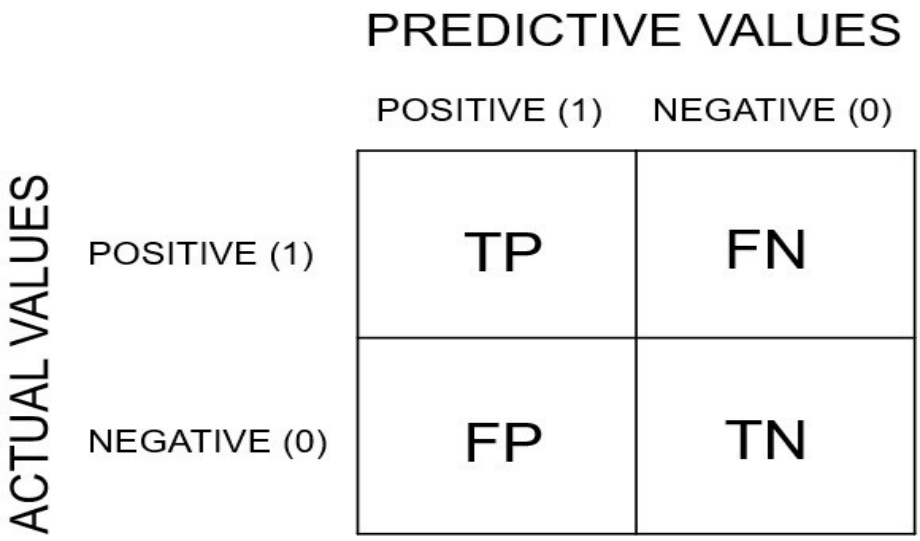


Figure 4: Confusion Matrix structure

The resultant values Accuracy, Precision and Recall are calculated from the confusion matrix values as follows:

$$\text{ACCURACY} = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

$$\text{PRECISION} = \frac{TP}{(TP+FP)}$$

$$\text{RECALL} = \frac{TP}{(TP+FN)}$$

5 RESULTS

TASK1:

Accuracy:

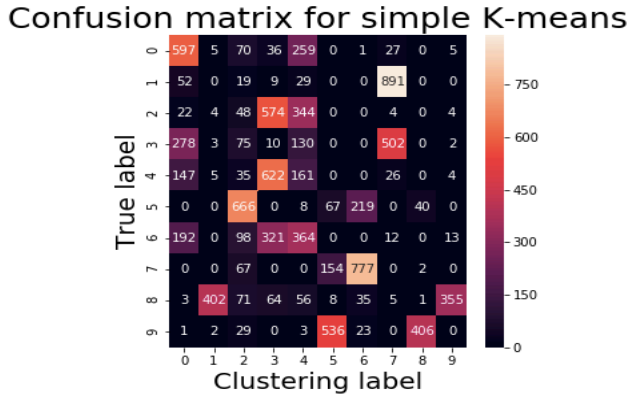
CLASSIFICATION REPORT

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1000
1	0.61	0.88	0.72	1000
2	0.02	0.02	0.02	1000
3	0.00	0.00	0.00	1000
4	0.38	0.66	0.48	1000
5	0.14	0.25	0.18	1000
6	0.15	0.22	0.18	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	0.13	0.20	0.16	10000
weighted avg	0.13	0.20	0.16	10000

TESTING ACCURACY: 0.5186884775727859

Confusion Matrix:

```
SSE for #cluster =  1 is 4092975.6596677564
SSE for #cluster =  2 is 3233032.0676284176
SSE for #cluster =  3 is 2766659.5236280463
SSE for #cluster =  4 is 2505225.873973058
SSE for #cluster =  5 is 2352605.66958883
SSE for #cluster =  6 is 2202642.5083292704
SSE for #cluster =  7 is 2099373.5016917875
SSE for #cluster =  8 is 2027691.0161877794
SSE for #cluster =  9 is 1963211.0586392875
SSE for #cluster =  10 is 1906652.6329259197
```

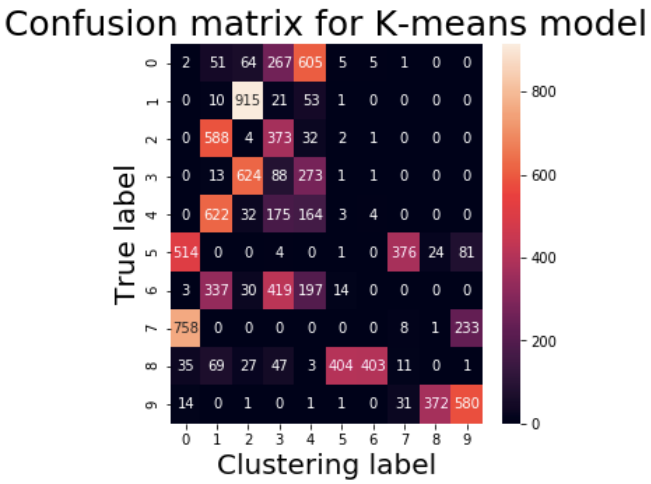


TASK 2a:

Accuracy:

```
TESTING ACCURACY: 0.5474918410142511
CLASSIFICATION REPORT
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.01      0.00      0.01      1000
           1       0.56      0.91      0.69      1000
           2       0.00      0.00      0.00      1000
           3       0.21      0.30      0.25      1000
           4       0.00      0.00      0.00      1000
           5       0.01      0.00      0.00      1000
           6       0.00      0.00      0.00      1000
           7       0.00      0.00      0.00      1000
           8       0.02      0.01      0.01      1000
           9       0.00      0.00      0.00      1000

    accuracy                           0.12     10000
   macro avg       0.08      0.12      0.10     10000
weighted avg       0.08      0.12      0.10     10000
```

Confusion Matrix:

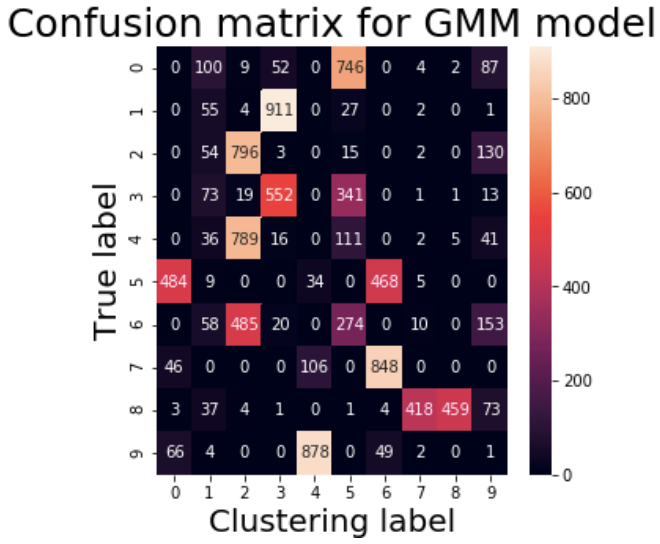


TASK 2b:

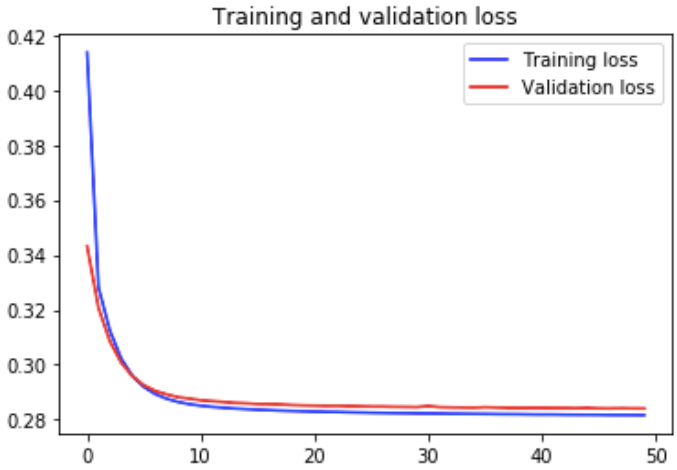
Accuracy:

```
TESTING ACCURACY: 0.5651704900348504
CLASSIFICATION REPORT
------------------------------------------------------
              precision    recall  f1-score   support

           0       0.21      0.08      0.12      1000
           1       0.00      0.00      0.00      1000
           2       0.00      0.00      0.00      1000
           3       0.00      0.00      0.00      1000
           4       0.20      0.21      0.20      1000
           5       0.00      0.00      0.00      1000
           6       0.20      0.18      0.19      1000
           7       0.00      0.00      0.00      1000
           8       0.98      0.46      0.62      1000
           9       0.64      0.70      0.67      1000

    accuracy                           0.16     10000
   macro avg       0.22      0.16      0.18     10000
weighted avg       0.22      0.16      0.18     10000
```

Confusion Matrix:



The training loss and validation loss have been plotted against the epochs while training the auto-encoder.



6 CONCLUSION

We have used the Fashion-MNST dataset to model our clustering analysis problem using unsupervised learning techniques. There were three tasks to perform: to implement a simple K-means clustering algorithm on the MNIST data space, the second one was to construct and train an auto-encoder and implement K-means and GMM using the same. The models are evaluated on the basis of the 'accuracy' metric and the accuracy has been found to be different for each of the models. Accuracy depends on many factors: how well the hyperparameters are tuned, the learning rate and the number of epochs we are using to train our model. For the first task, I have used 7 epochs to build the ten clusters and for every iteration I have measured the SSE (Sum of Squared Error) as a tuning parameter. The low the value of SSE, the more accurate the clusters are. For the next two tasks, I have used 50 epochs to train the auto-encoder and then K-means and GMM have been implemented on the encoded layer to get the output. A Logistic Regression classifier has been used on the resultant datasets after clustering so that the accuracy is improved. Before using the classifier, accuracy was attained in the range of 50-60% and after using the classifier, it improved to >80%. This can be improved if I train my model with more epochs over a period of time.

ACKNOWLEDGEMENTS:

I am extremely grateful to Prof. Sargur Srihari to have introduced us to the concepts and intricacies of Unsupervised learning algorithms, their applications and the content of the slides structured by him has contributed much to the detailed construction of the report. The other sources of information have been listed below:

[1] https://medium.com/

[2] https://towardsdatascience.com/

[3] https://www.wikipedia.org