Exploring Clustering with Fashion MNIST Clothing Dataset

Manisha Biswas
mbiswas2@buffalo.edu

Abstract

This paper presents the development and the simulation of a machine learning model to perform cluster analysis using unsupervised learning. Cluster analysis is one of the unsupervised machine learning techniques which doesn't require labeled data. For obtaining a reasonable model, here we are using the Fashion-MNIST dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. The first part of this work has been dedicated to pre-process the data. Fashion MNIST dataset is downloaded and processed into a Numpy array that contains the feature vectors and a Numpy array that contains the labels. For the implementation of the model, the training dataset has been partitioned in the following fashion: 80% for the training phase, and 20% for the validation phase. Next, K-Mean algorithm has been used to cluster the Fashion MNIST dataset into 10 clusters and accuracy has been reported as 50.53%. Here the implementation of the model has been divided into three parts: Implementation of KMeans algorithm to cluster original data space of Fashion-MNIST dataset, building an Auto-Encoder based K-Means clustering model to cluster the condensed representation of the unlabeled fashion MNIST dataset and building an Auto-Encoder based Gaussian Mixture Model clustering model to also cluster the condensed representation of the unlabeled fashion MNIST dataset. The accuracy statistics for each model is as follows: for K-Mean based clustering model gives an accuracy of 50.53%, for Auto-Encoder based K-Means clustering model gives an accuracy of 48% and Auto-Encoder based Gaussian Mixture Model clustering gives an accuracy of 55.94%.

1. Introduction

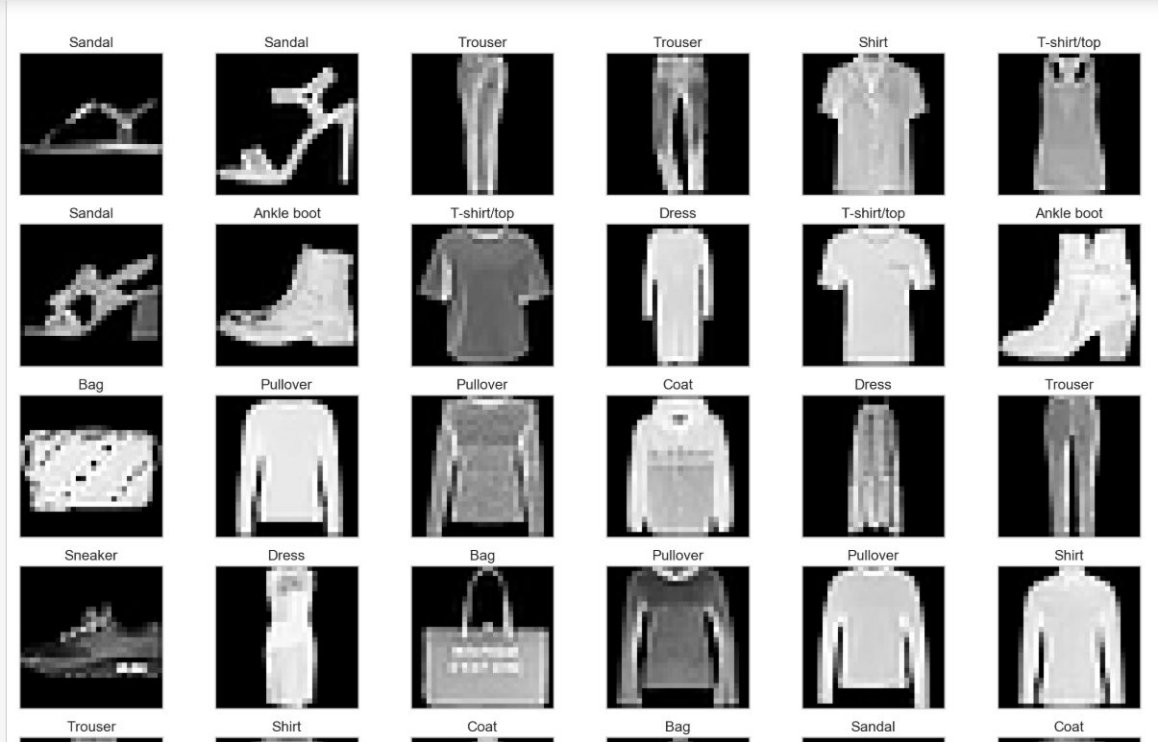
We live in the age of Instagram, YouTube, and Twitter. Images and video (a sequence of images) dominate the way millennials and other weirdos consume information.Having models that understand what images show can be crucial for understanding your emotional state,location, interests and social group.

Images of clothes are great factors in driving the clothing industry today and trends found in the current fashion styles drive the online clothing economy. This study attempts to perform cluster analysis on fashion MNIST dataset using unsupervised learning.

This paper presents three different approaches, namely K-Mean based clustering model, Auto-Encoder based K-Mean clustering model , Auto-Encoder based GMM clustering model as part of the cluster analysis on fashion MNIST dataset.

2. Dataset Definition

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. 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. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.



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.

Each training and test example are assigned to one of the labels as shown in table 1.

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

Table 1: Labels for Fashion-MNIST dataset

3. Data Preprocessing

We are using Jupyter notebook to work on this dataset. We will first import the necessary packages and load the train set and test set which are given in two separate datasets. We can visualize the dataset as visualization of data is an imperative aspect of data science. It helps to understand data and also to explain the data to another person.

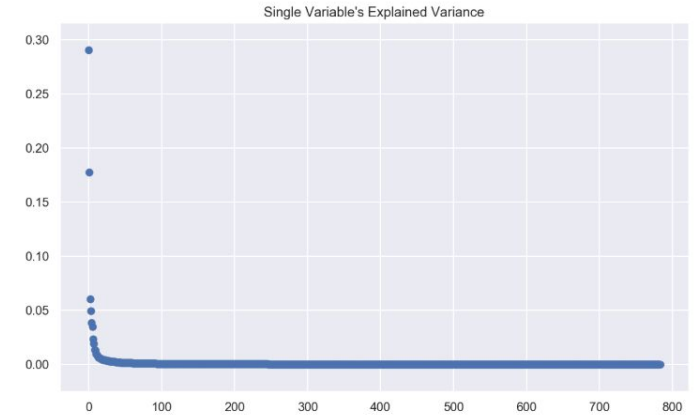
We further split the train set in train and validation set. The validation set will be 20% from the original train set, therefore the split will be train/validation of 0.8/0.2.

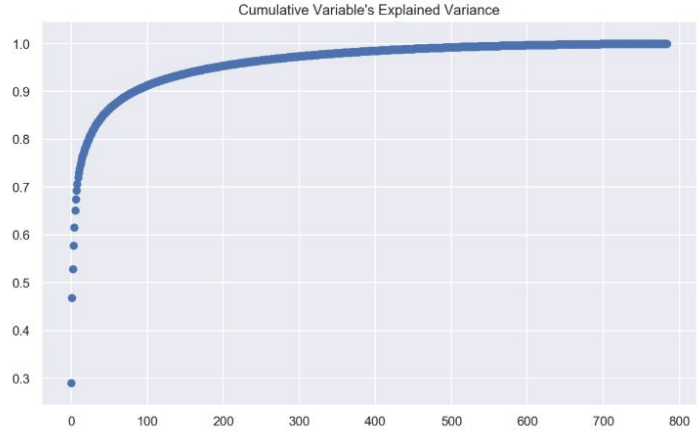
- **Training data** — used for training the model
- **Validation data** — used for tuning the hyperparameters and evaluate the models
- **Test data** — used to test the model after the model has gone through initial vetting by the validation set.

Now, We will perform Principal Component Analysis which is a simplest example of dimensionality reduction because as a pre-processing technique, dimensionality reduction makes

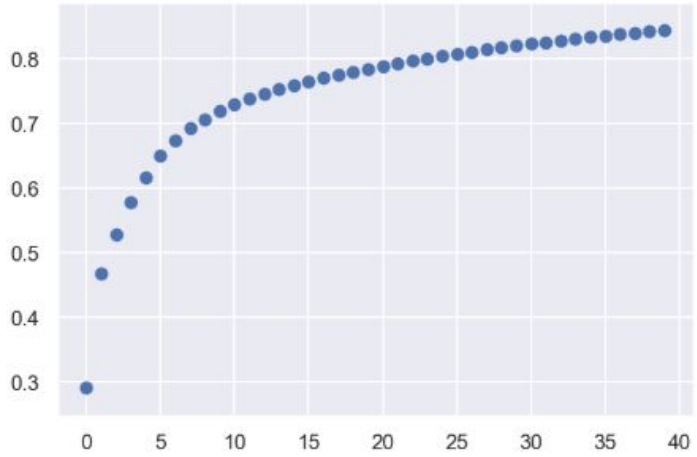
the most sense when it is used to redesign a dataset with particularly many artificially defined features. Image datasets like Fashion MNIST, for example, are a perfect target for PCA because they are entirely artificial: pixel values are simply not particularly meaningful to us as casual observers; whereas "does this look more like a shoe or a pair of pants" *is*.

This paper demonstrates the power of PCA by picking top 40 PCA components. When using PCA, it's a good idea to pick a decomposition with a reasonably small number of variables by looking for a "cutoff" in the effectiveness of the model. To do that, we can plot the variance ratio for each of the components of the PCA as follows

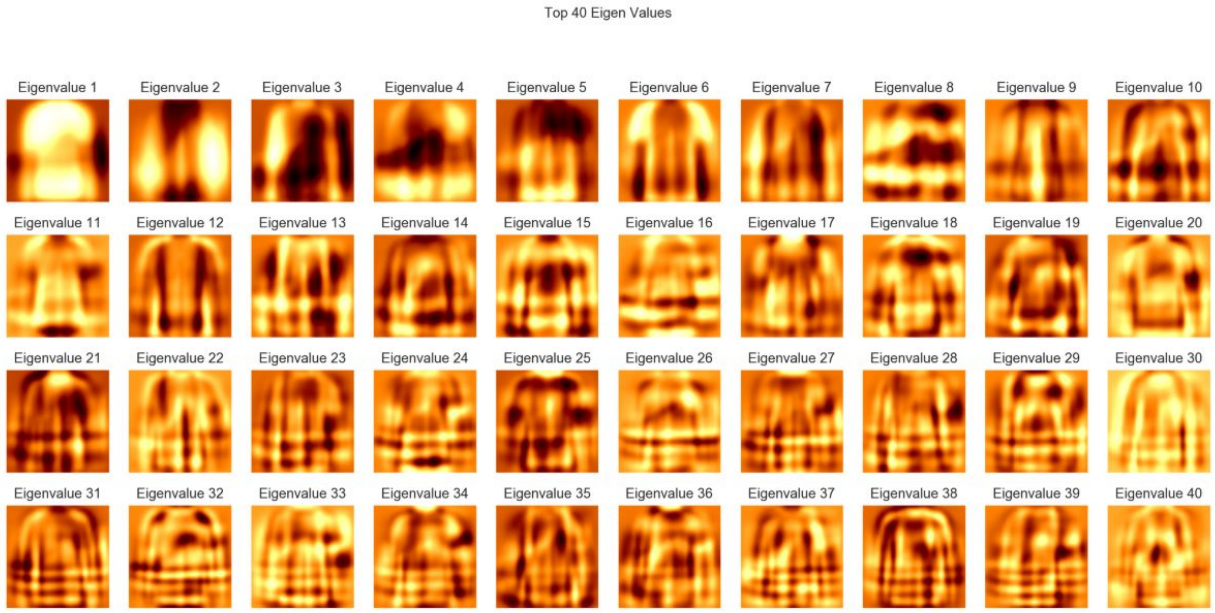




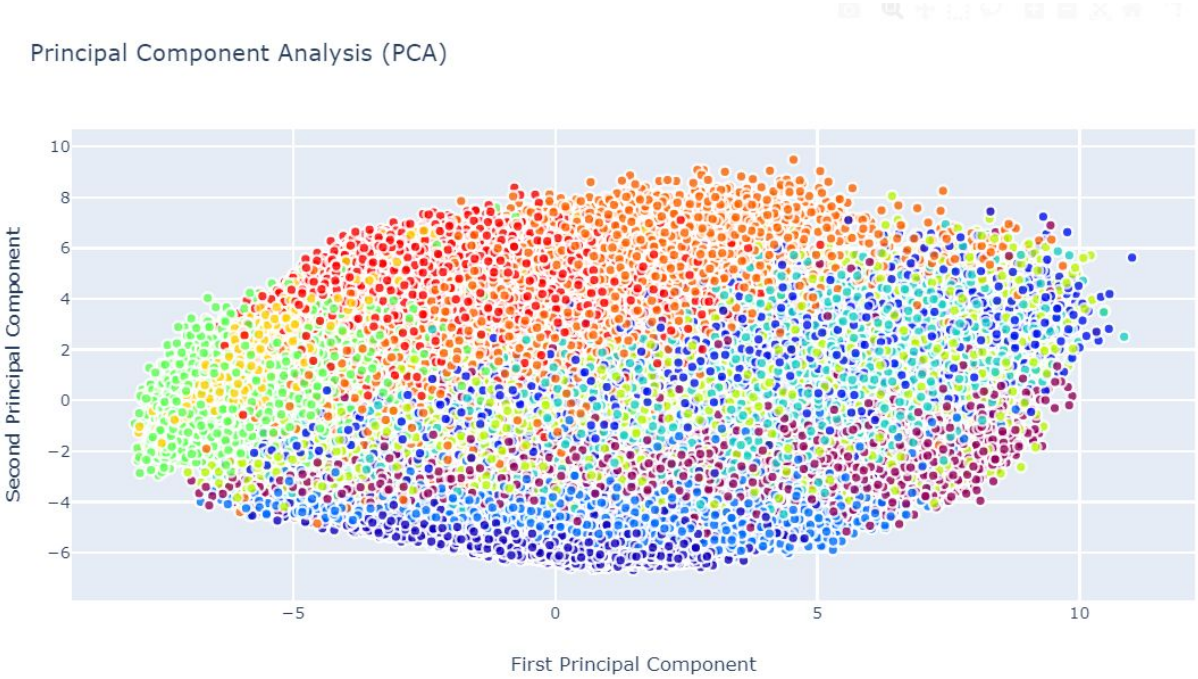
By looking at this plot, it seems like the first few components add a lot of information about the variables to the model, but every component after that is not particularly worthwhile.



The reduced data representation has been computed when projecting only on to the top 40 eigenvectors. Below is the visualization of 40 principal components that describe the largest variations in the dataset.



The below plot shows 10 distinguishable clusters. This factoid tells us that the observations in the dataset can be grouped.



4. Model Architecture

- **K-Mean based clustering model:**

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variances, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below).

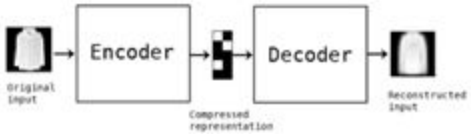
$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

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; note that they are not, in general, points from set of samples, although they live in the same space.

The function K-Means applies K-Means clustering to the train data with the number of classes as the number of clusters to be made and creates labels both for train and test data. The parameter output controls how do we want to use these new labels, ‘add’ will add the labels as a feature in the dataset and ‘replace’ will use the labels instead of the train and test dataset to train our classification model in our case it is Random Forest Classifier.

Autoencoder:

"Autoencoding" 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 below:



To build an autoencoder, we need three things: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of our 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.

- **Auto-Encoder with K-Means Clustering:**

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

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

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 high dimensional spaces, Euclidean distances tend to become inflated. 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.

- **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 that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the Gaussian it mostly probably belongs to using the GaussianMixture.predict method.

5. Results

The results obtained in this paper were very accurate with possibility of being better by usage of different techniques, methods of data pre-processing and including more data into the training. We discuss a few such methods below along with the results we obtained by applying them.

1. **K-Mean based clustering model:**

We apply K-Means clustering to train the data with the number of classes (i.e.10) as the number of clusters to be made and creates labels both for train and test data. The parameter flag controls how do we want to use these new labels, ‘add’ will add the labels as a feature in the dataset and ‘replace’ will use the labels instead of the train and test dataset to train our classification model which is Random Forest Classifier.

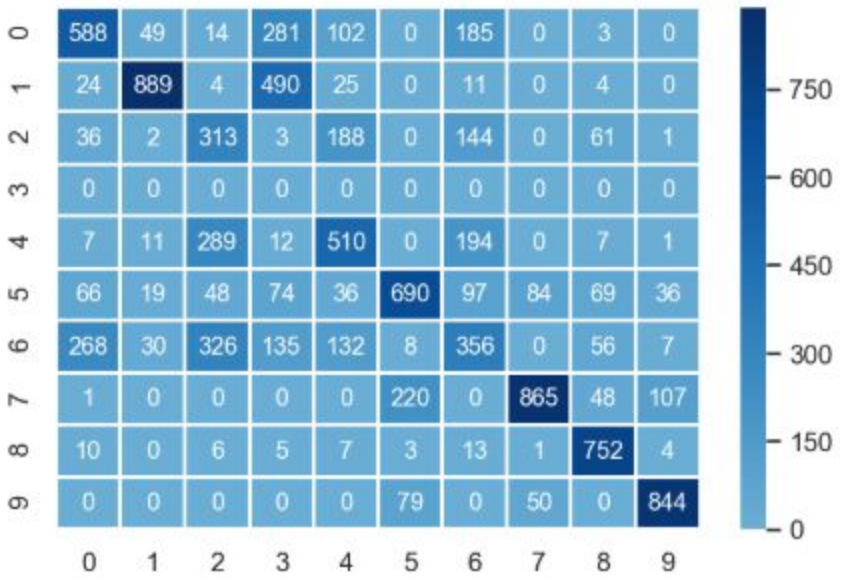
Initially, only clusters found by KMeans are used to train a classification model. These clusters alone give a model with an accuracy of 58.07%.

Confusion Matrix

To check the accuracy of test dataset we introduce a concept of confusion matrix which we import using the confusion_matrix method of sklearn metrics class. The confusion matrix is a way of tabulating the number of misclassifications, i.e., the number of predicted classes which

ended up in a wrong classification bin based on the true classes. It is a table with four different combinations of predicted and actual values.There are two confusion matrices in our case.

The confusion matrix by replacing the data with the clusters is as follows:

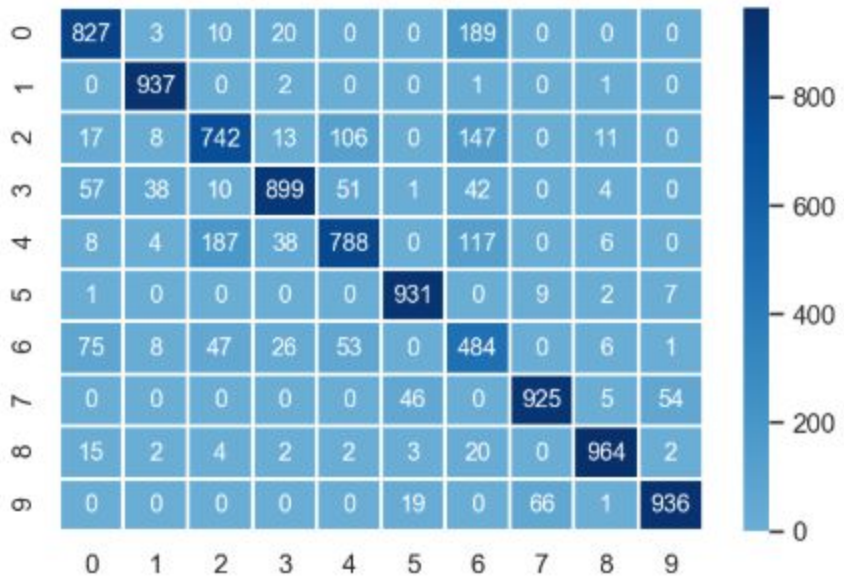


We will use the Confusion Matrix to calculate the value of Accuracy, Precision ,Recall and F1-score.

	precision	recall	f1-score	support
0	0.59	0.48	0.53	1222
1	0.89	0.61	0.73	1447
2	0.31	0.42	0.36	748
3	0.00	0.00	0.00	0
4	0.51	0.49	0.50	1031
5	0.69	0.57	0.62	1219
6	0.36	0.27	0.31	1318
7	0.86	0.70	0.77	1241
8	0.75	0.94	0.84	801
9	0.84	0.87	0.86	973
accuracy			0.58	10000
macro avg	0.58	0.53	0.55	10000
weighted avg	0.66	0.58	0.61	10000

Then we "add" the clusters as a feature(column) and train the same Random Forest model which gives an accuracy of 83.33% and the results show an improvement over our previous model.

The confusion matrix by adding the predicted clusters is as follows:



We will use the Confusion Matrix to calculate the value of Accuracy, Precision ,Recall and F1-score.

	precision	recall	f1-score	support
0	0.83	0.79	0.81	1049
1	0.94	1.00	0.97	941
2	0.74	0.71	0.73	1044
3	0.90	0.82	0.86	1102
4	0.79	0.69	0.73	1148
5	0.93	0.98	0.95	950
6	0.48	0.69	0.57	700
7	0.93	0.90	0.91	1030
8	0.96	0.95	0.96	1014
9	0.94	0.92	0.93	1022
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000
weighted avg	0.85	0.84	0.85	10000

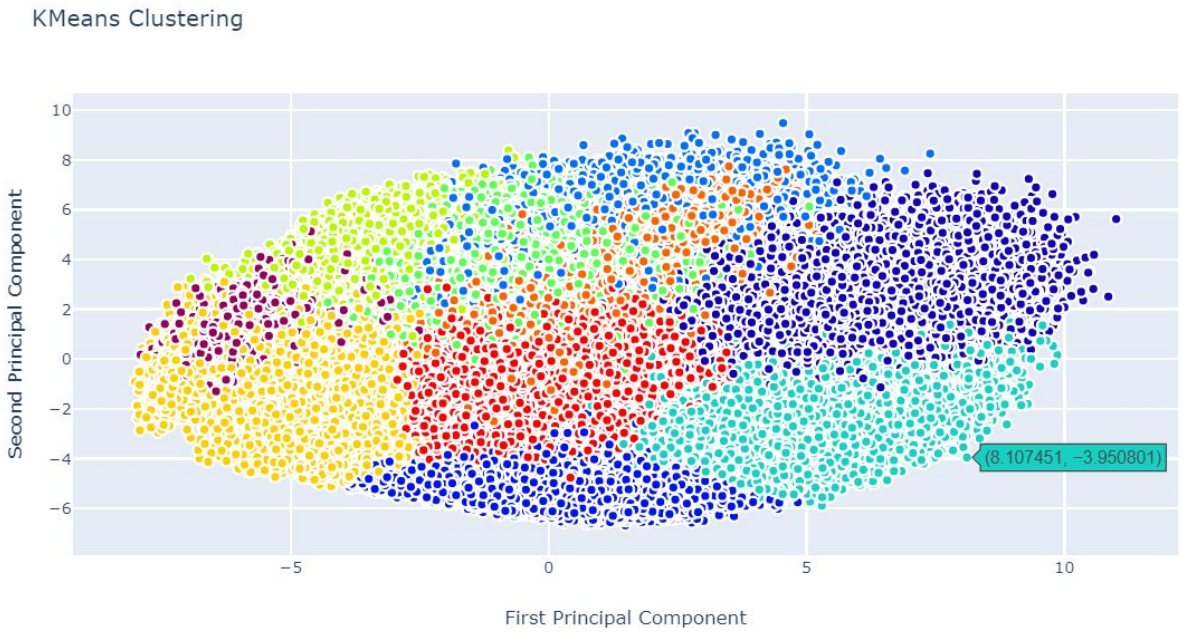
Wall time: 49.1 s

The overall performance is as follows:

Adjusted Mutual Info Score	
Training data	50.37 %
Test data	50.53 %

Accuracy	
After adding the predicted clusters	84.33 %
After replacing the data with the clusters	58.07 %

Once clusters and their mean is stable, all data items are then known to be grouped into their relevant clusters. Below we can see that 10 clusters have been formed.



2. Auto-Encoder based K-mean clustering model:

The autoencoder has 5 Convolution blocks, each block has a convolution layer followed by a batch normalization layer and a Dropout layer.Max-pooling layer is used in the first and second convolution blocks.

- The first convolution block will have 32 filters of size 3 x 3, followed by a max-pooling layer and a dropout layer.
- The second block will have 16 filters of size 3 x 3, followed by another max-pooling layer and a dropout layer
- The third block of encoder will have 16 filters of size 3 x 3 followed by upsampling layer.
- The fourth block of encoder will have 32 filters of size 3 x 3 followed by upsampling layer.
- The final block of encoder will have 1 filters of size 3 x 3 which will reconstruct back the input having a single channel.

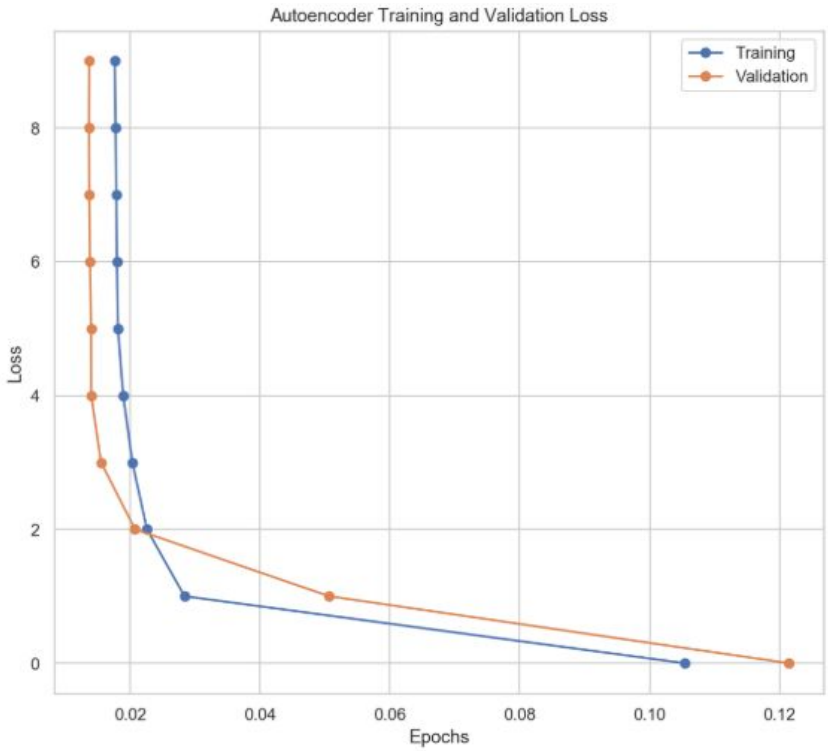
The max-pooling layer will downsample the input by two times each time you use it, while the upsampling layer will upsample the input by two times each time it is used. Let's visualize the layers that we created in the above step by using the summary function. This will show a number of parameters (weights and biases) in each layer and also the total parameters in our model.

```
model.summary()
```

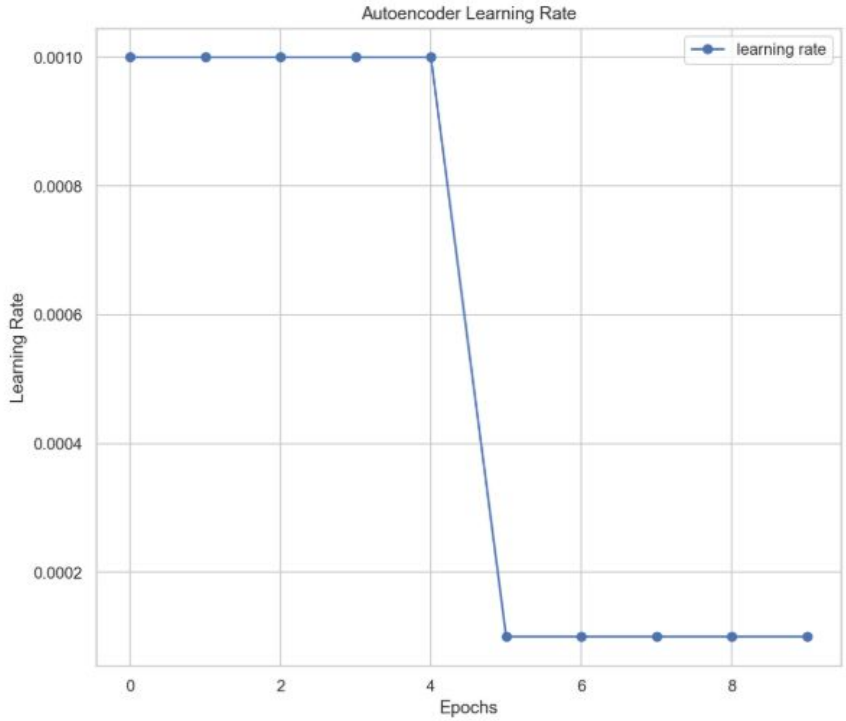
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
batch_normalization_4 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 16)        4624
_________________________________________________________________
batch_normalization_5 (Batch (None, 14, 14, 16)        64
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 7, 7, 16)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 7, 7, 16)          2320
_________________________________________________________________
batch_normalization_6 (Batch (None, 7, 7, 16)          64
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 14, 14, 16)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 32)        4640
_________________________________________________________________
batch_normalization_7 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 28, 28, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 1)         289
=================================================================
Total params: 12,577
Trainable params: 12,385
Non-trainable params: 192
_________________________________________________________________
```

Now we need to train the model with Keras's fit() function. The model trains for 10 epochs. The fit() function will return a history object. By storing the result of this function we can use it later to plot the loss function plot between training and validation which will help us to analyze the model's performance visually.

We record the Autoencoder Training and Validation loss as follows:



The plot for Autoencoder Learning rate is as follows:

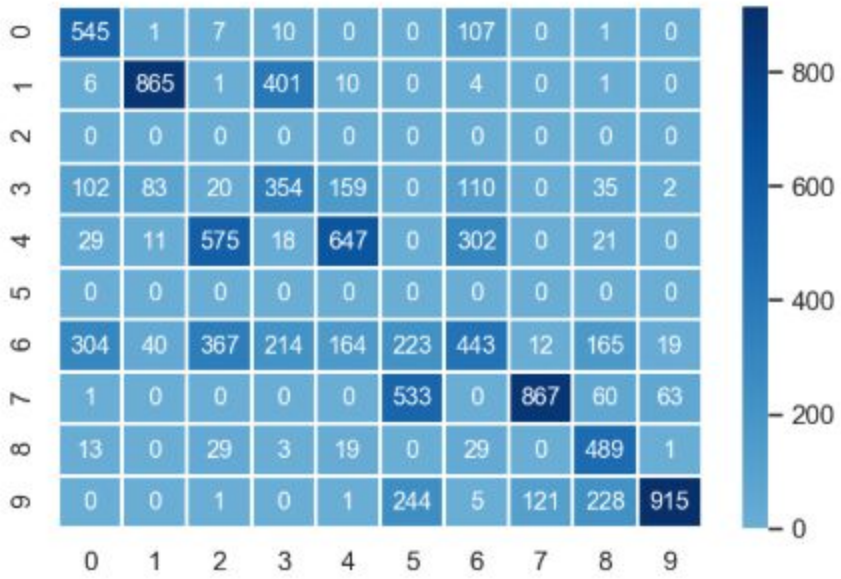


Then we apply the K-Means algorithm to the encoded train data with the number of classes as the number of clusters to be made and creates labels both for train and test data. The parameter flag controls how do we want to use these new labels, ‘add’ will add the labels as a feature in the

dataset and ‘replace’ will use the labels instead of the train and test dataset to train our Random Forest classification model.

Initially, only clusters found by KMeans are used to train a classification model. These clusters alone gives a model with an accuracy of 51.25%.

The confusion matrix by replacing the data with the clusters is as follows:



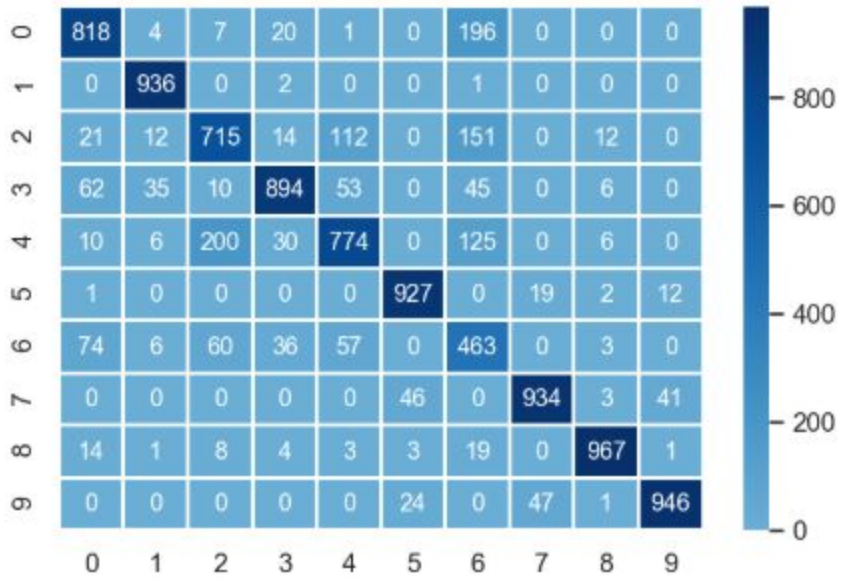
We will use the Confusion Matrix to calculate the value of Accuracy, Precision ,Recall and F1-score .

	precision	recall	f1-score	support
0	0.55	0.81	0.65	671
1	0.86	0.67	0.76	1288
2	0.00	0.00	0.00	0
3	0.35	0.41	0.38	865
4	0.65	0.40	0.50	1603
5	0.00	0.00	0.00	0
6	0.44	0.23	0.30	1951
7	0.87	0.57	0.69	1524
8	0.49	0.84	0.62	583
9	0.92	0.60	0.73	1515
accuracy			0.51	10000
macro avg	0.51	0.45	0.46	10000
weighted avg	0.67	0.51	0.56	10000

```
Wall time: 1.78 s
```

Then we "add" the clusters as a feature(column) and train the same Random Forest model which gives an accuracy of 83.74% and the results show an improvement over our previous model.

The confusion matrix by adding the predicted clusters is as follows:



We will use the Confusion Matrix to calculate the value of Accuracy, Precision ,Recall and F1-score .

	precision	recall	f1-score	support
0	0.82	0.78	0.80	1046
1	0.94	1.00	0.97	939
2	0.71	0.69	0.70	1037
3	0.89	0.81	0.85	1105
4	0.77	0.67	0.72	1151
5	0.93	0.96	0.95	961
6	0.46	0.66	0.55	699
7	0.93	0.91	0.92	1024
8	0.97	0.95	0.96	1020
9	0.95	0.93	0.94	1018
accuracy			0.84	10000
macro avg	0.84	0.84	0.83	10000
weighted avg	0.85	0.84	0.84	10000

The overall performance is as follows:

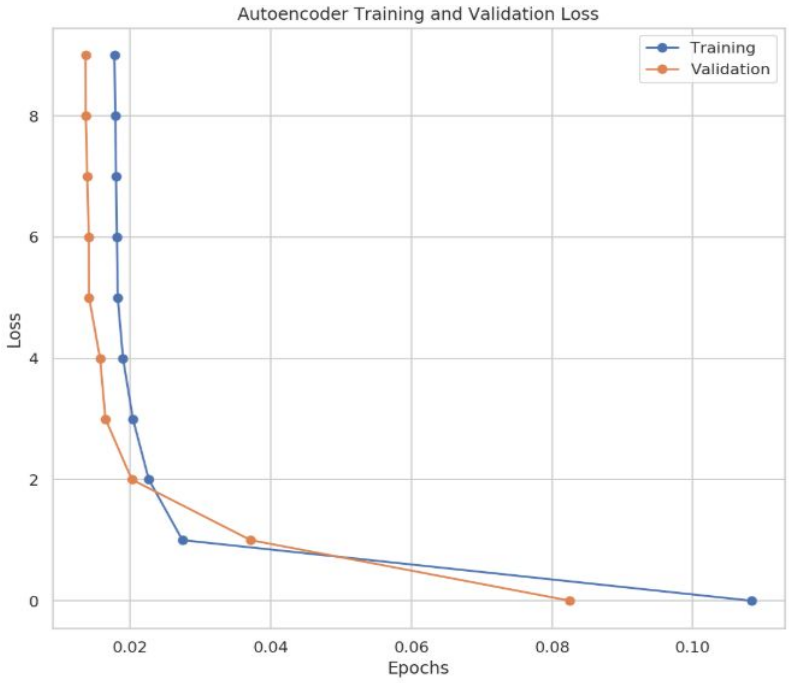
Adjusted Mutual Info Score	
Training data	48.0%
Test data	48.0 %

Accuracy	
After adding the predicted clusters	83.74 %
After replacing the data with the clusters	51.25 %

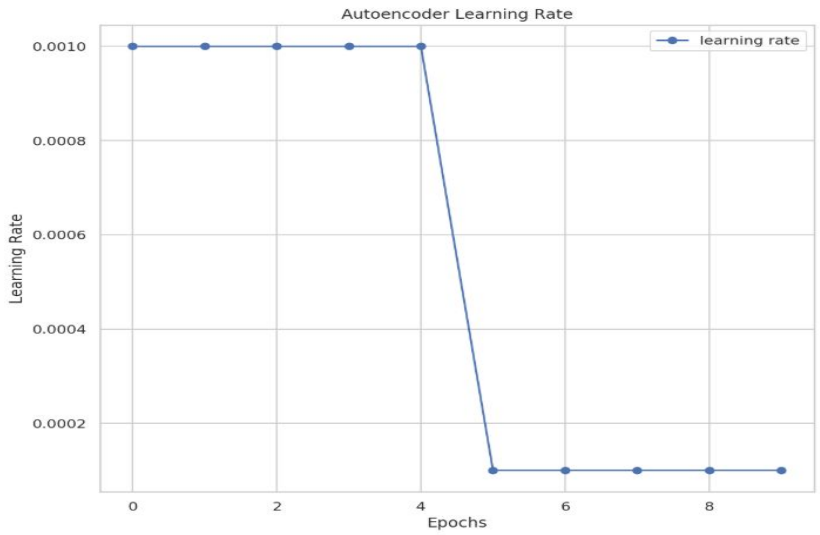
3. Auto-Encoder based GMM clustering model:

The autoencoder model architecture is same as above we used in K-mean clustering model with 5 convolutional block. We again use the fit() function to train the model for 10 epochs which return a history object and using that we plot the training and validation loss.

We record the Autoencoder Training and Validation loss as follows:



The plot for Autoencoder Learning rate is as follows:

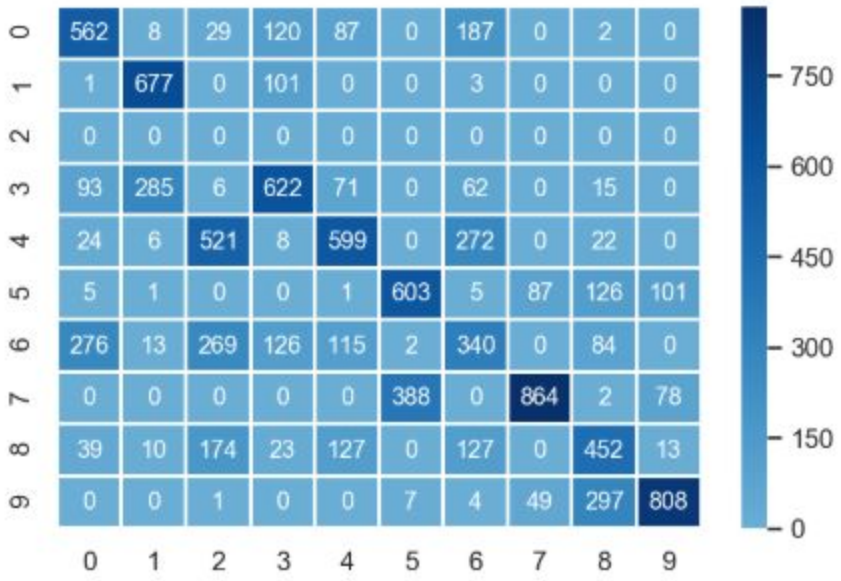


Then we apply the GMM algorithm to the encoded train data with the number of classes(i.e. 10) as the number of clusters to be made and creates labels both for train and test data. The

parameter flag controls how do we want to use these new labels, ‘add’ will add the labels as a feature in the dataset and ‘replace’ will use the labels instead of the train and test dataset to train our Random Forest classification model.

Initially, only clusters found by GMM are used to train a classification model. These clusters alone gives a model with an accuracy of 55.27%.

The confusion matrix by replacing the data with the clusters is as follows:

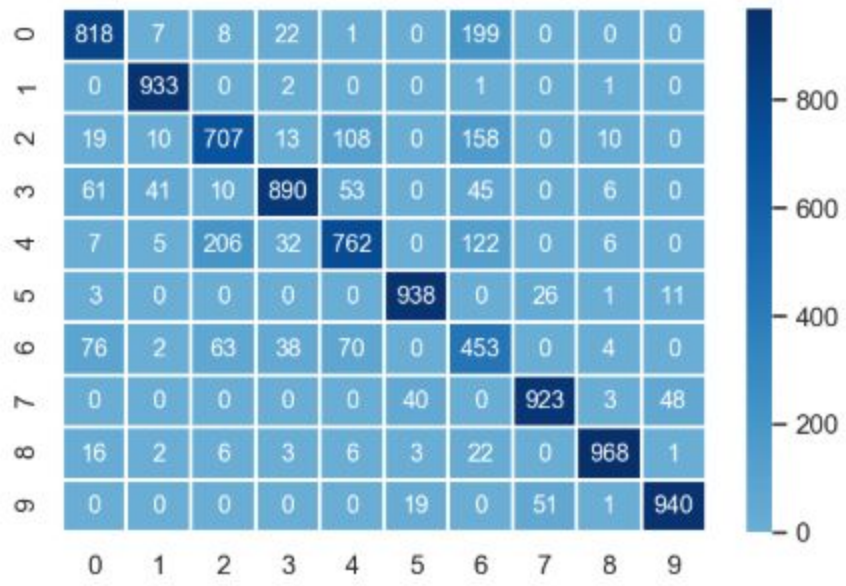


We will use the Confusion Matrix to calculate the value of Accuracy, Precision ,Recall and F1-score.

	precision	recall	f1-score	support
0	0.56	0.56	0.56	995
1	0.68	0.87	0.76	782
2	0.00	0.00	0.00	0
3	0.62	0.54	0.58	1154
4	0.60	0.41	0.49	1452
5	0.60	0.65	0.63	929
6	0.34	0.28	0.31	1225
7	0.86	0.65	0.74	1332
8	0.45	0.47	0.46	965
9	0.81	0.69	0.75	1166
accuracy			0.55	10000
macro avg	0.55	0.51	0.53	10000
weighted avg	0.62	0.55	0.58	10000

Then we "add" the clusters as a feature(column) and train the same Random Forest model which gives an accuracy of 83.38% and the results show an improvement over our previous model.

The confusion matrix by adding the predicted clusters is as follows:



We will use the Confusion Matrix to calculate the value of Accuracy, Precision ,Recall and F1-score.

	precision	recall	f1-score	support
0	0.82	0.78	0.80	1055
1	0.93	1.00	0.96	937
2	0.71	0.69	0.70	1025
3	0.89	0.80	0.85	1106
4	0.76	0.67	0.71	1140
5	0.94	0.96	0.95	979
6	0.45	0.64	0.53	706
7	0.92	0.91	0.92	1014
8	0.97	0.94	0.96	1027
9	0.94	0.93	0.93	1011
accuracy			0.83	10000
macro avg	0.83	0.83	0.83	10000
weighted avg	0.84	0.83	0.84	10000

The overall performance is as follows:

Adjusted Mutual Info Score	
Training data	56.55 %
Test data	55.94 %

Accuracy	
After adding the predicted clusters	83.38 %
After replacing the data with the clusters	58.35 %

6. Conclusion

This paper presents the development and simulation of a machine learning model to perform cluster analysis using unsupervised learning on Fashion-MNIST, a fashion product images dataset.

Clustering apart from being an unsupervised machine learning can also be used to create clusters as features to improve classification models. On their own they aren't enough for classification as the results show. But when used as features they improve model accuracy.

7. Acknowledgments

We are extremely grateful to Professor Sargur Srihari and Mihir Chauhan for teaching all the necessary concepts related to Logistic Regression and helping in this project throughout.

8.References

- https://www.datacamp.com/community/tutorials/autoencoder-classifier-python
- https://ramhiser.com/post/2018-05-14-autoencoders-with-keras/
- https://towardsdatascience.com/kmeans-clustering-for-classification-74b992405d0
- https://gurus.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/
- https://buildmedia.readthedocs.org/media/pdf/umap-learn/latest/umap-learn.pdf
- https://medium.com/machine-learning-algorithms-from-scratch/k-means-clustering-from-scratch-in-python-1675d38eee42