Fundamentals of Data Science Project Report

2023/2024

1. Summary of the data.

The data utilised in this project was downloaded from kaggle.com. It is Fashion-MNIST — a dataset of Zalando's article images. It is split beforehand into a training set of 60000 items and a test set that consists of 10000 items, 70000 items in total. Each item is a 28x28 grayscale image and is associated with a label from 10 classes.

Each image is stored as 28*28 = 784 pixels, and each pixel value is between 0 and 255, indicating its lightness, the higher the number is, the darker the pixel is.

In other words, each data item is 784-dimensional.

In the dataset, the first column is a label of the article, and the rest is the image described by the pixels.

Each row is a separate image, so there is 70000*784 = 54 880 000 data cells in total.

The labels are as following:

0 — a T-shirt/Top

1 — Trouser

2 — Pullover

3 — Dress

4 — Coat

5 — Sandal

6 — Shirt

7 — Sneaker

8 — Bag

9 — Ankle boot.

Data was analysed with code from https://github.com/Areczek00/FoDS-Project —> projekt.py

2.Reducing data dimensionality

It is both inefficient and unnecessary to analyse all 784 dimensions of given data, so the first step in this project is reducing the data dimensionality.

At first, I used Principle Component Analysis to project the data to a lower dimensional space. A scree plot based on explained variance ratio was constructed to determine how much reduction can be done.

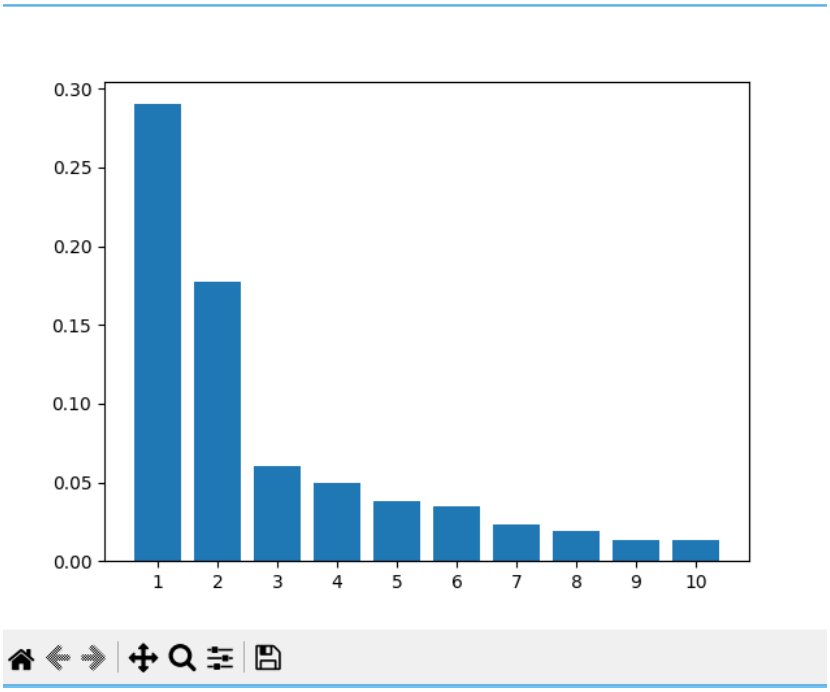

Fig. 1: Scree plot, explained variance ratio

The biggest drop is between second and third dimension, so, at least for presentational purpose, the data was reduced from 784 to 2 dimensions.

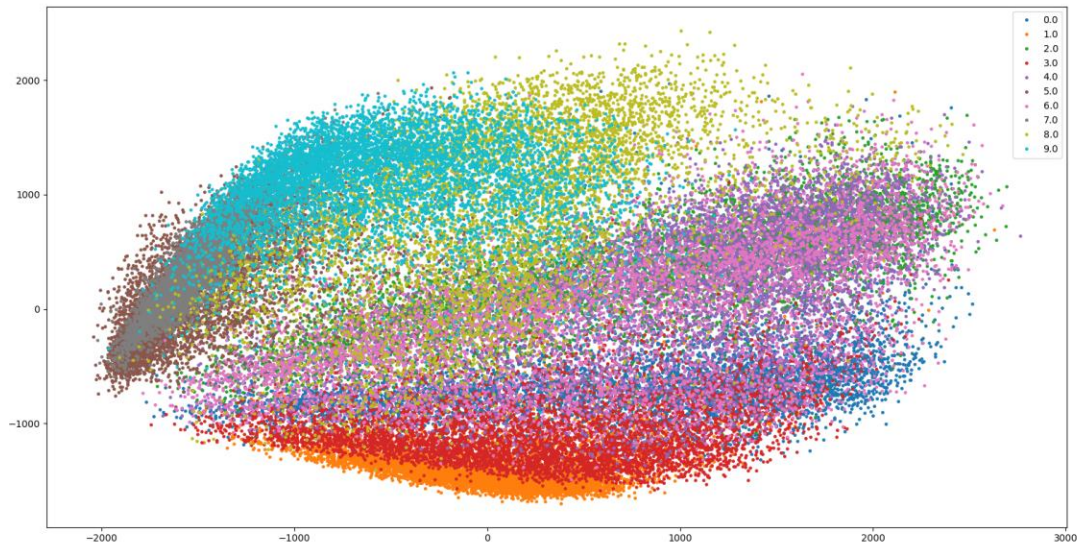

Fig. 2: 2-dimensional data with labels.

In clustering and classification however, it was possible and efficient to use data of higher dimensions. To determine those an optimal number of dimensions in clustering, rand index, scree plot, and trial-and-error method were used.

3. Clustering

There are various methods of clustering a dataset. At first, I tried to use Agglomerative Clustering, the method we got to know during the labs. However, this algorithm requires quite a lot of memory, and the program failed to allocate it, throwing an error message:

```
Unable to allocate 13.4 GiB for an array with shape (1799910001,) and data type
float64.
```

Therefore, other methods needed to be used.

First, I used KMeans on 2-dimensional data. However, the results were quite unsatisfactory.

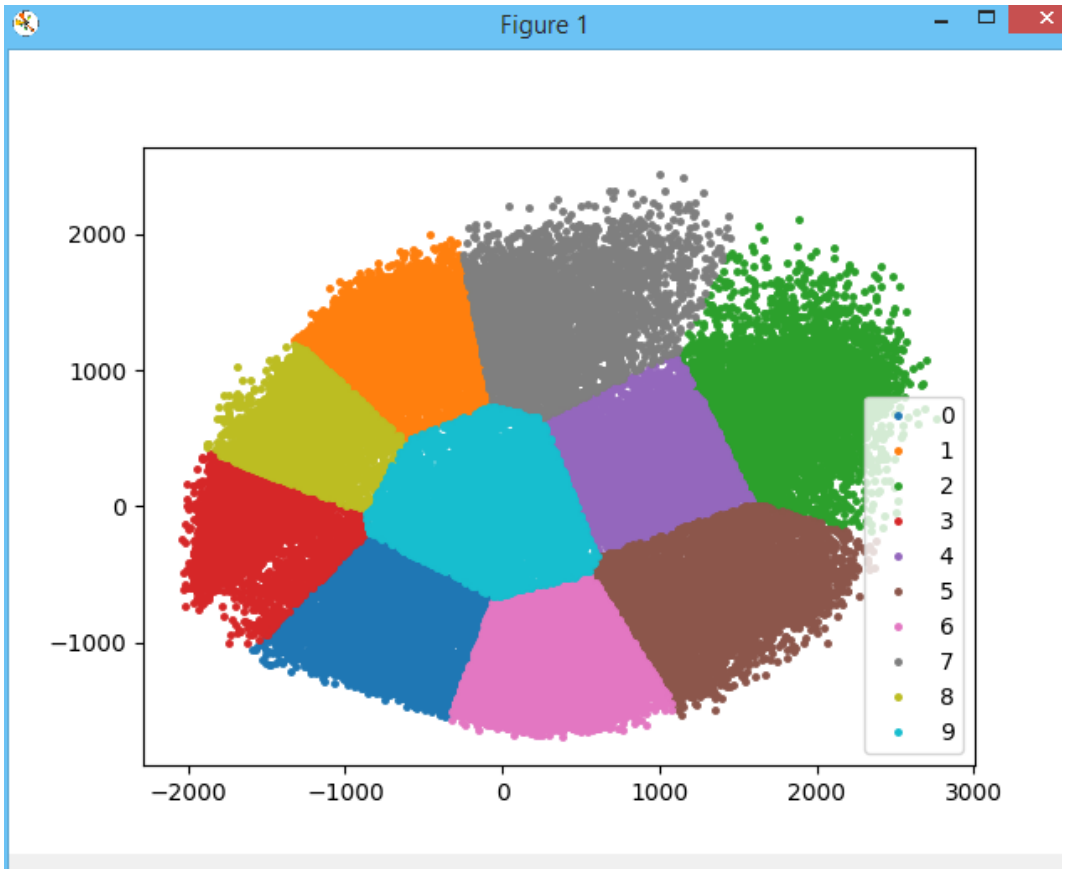

Fig. 3: 2-dimensional dustering, K-Means.

Just in case, a different method was used: Mini Batch K-Means, but to no avail.

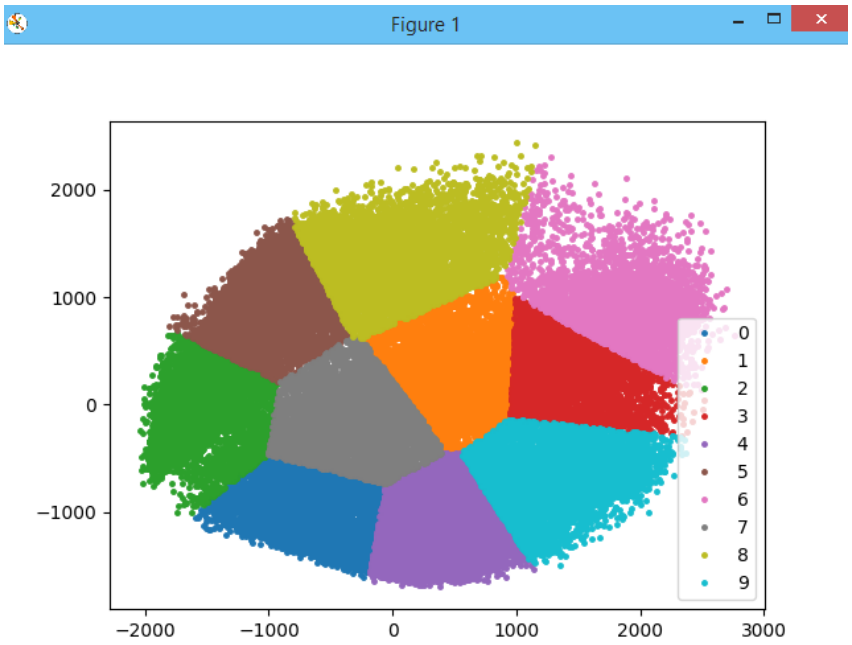

Fig. 4: 2-dimensional Mini Batch K-Means

A decision was made to heighten the number of dimensions. It wasn't quite possible to pinpoint the exact number of dimensions this clustering should have, so to evaluate the result of each clustering, rand index, which uses the original labels, was used.

dim	2	3	4	5	6	7	8	9	10	11	12	13
KMeans	0.867443307388456 4	0.870734316683055 8	0.868335181989153	0.875013483558059 3	0.875457897076062 4	0.875526964893859 4	0.881621062812891 33	0.875168467807796 8	0.880420746456885 4	0.881027276565720	0.870969802830047 2	0.889073349555826
MiniBatch	0.866849170263948 8	0.869667450390200 95	0.874896342161258 2	0.878965396645499 7	0.879054109790718 7	0.873636940615677	0.872019908665144 4	0.879969458379861 8	0.870398049411934 7	0.877414958582643 1	0.872817721962032 7	0.882108086245881 9

I noticed that heightening dimensions doesn't necessarily improve the quality of clustering. Finally, I decided to use 8-dimensions for clustering, and show its results in 2D.

Using trial-and-error, I determined optimal starting values for random number generation for centroid initialization:

MiniBatch K-Means Clustering, 8-dimension, random state 1, rand index is: **0.8850895770485063.**

K-Means Clustering, 8-dimension, random state 0, rand index is: **0.8806545136863392**.

Fig. 5: K-Means Cluster Centres

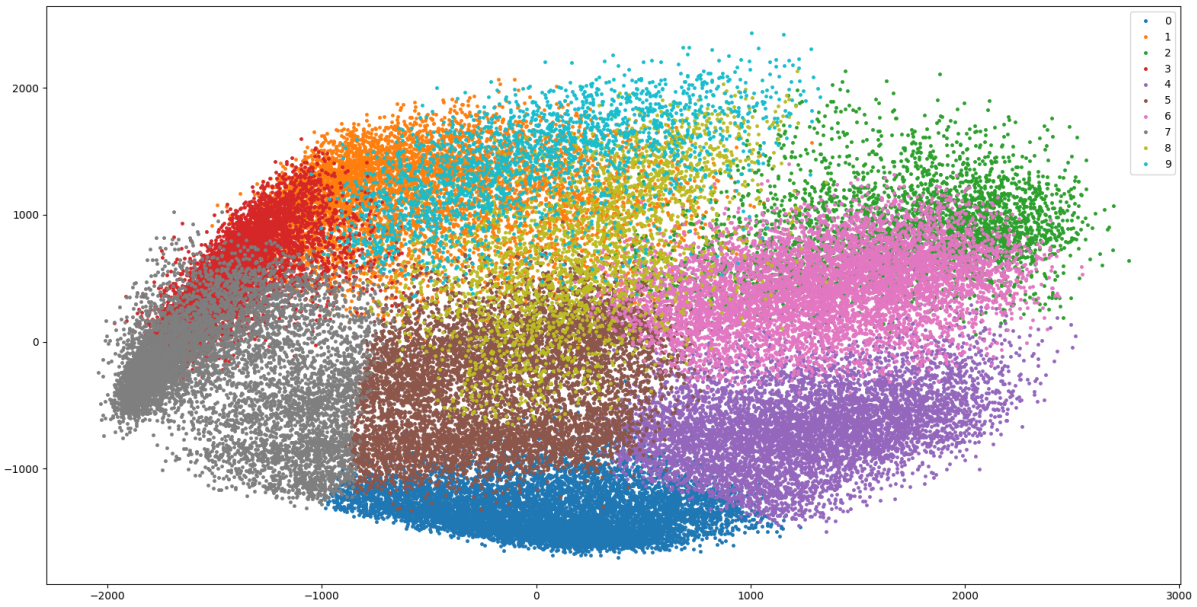

Fig. 6: 8-dimensional K-Means clustering

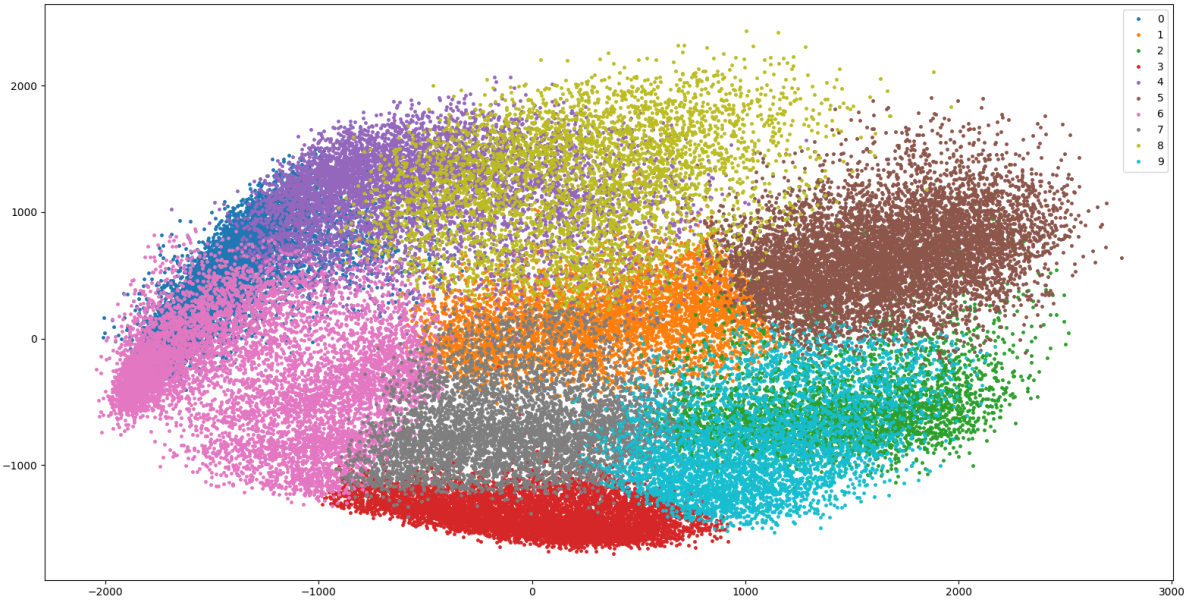

Fig. 7: 8-dimensional Mini Batch K-Means

To make clustering more diverse and interesting, I also employed other methods, such as Bisecting K-Means, Birch, DBSCAN, OPTICS, Spectral clustering.

BisectingKMeans K-Means Clustering, 8-dimension, with bisecting strategy chosen as *largest cluster*, rand index is: **0.8699555848153024**

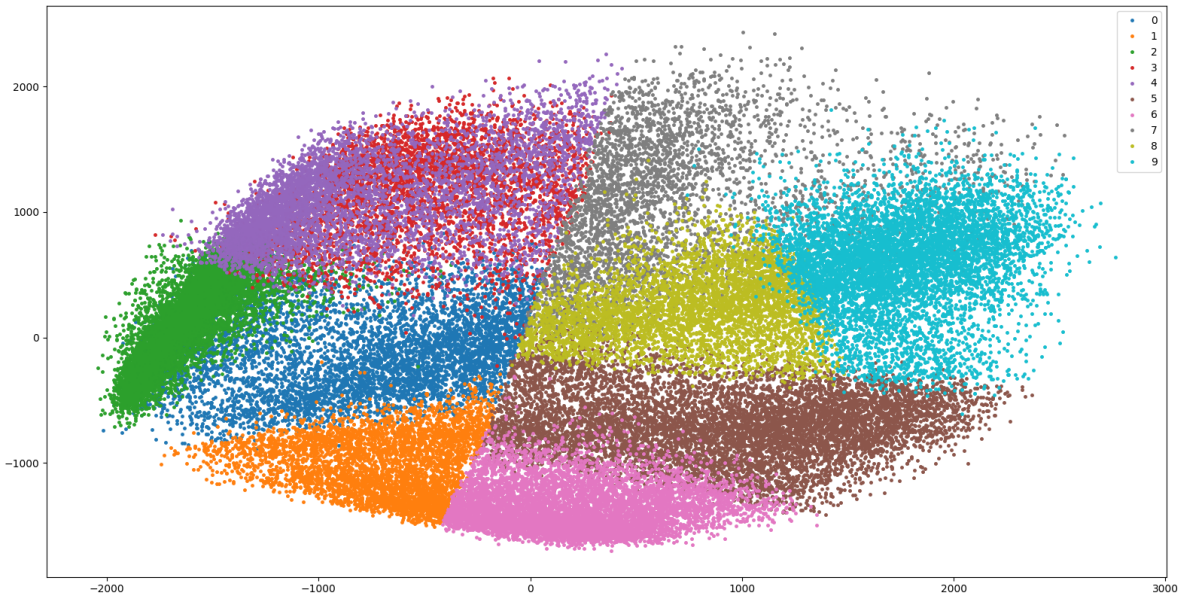

Fig. 8: 8-dimensional Bisectional K-Means

Birch clustering either uses Agglomerative Clustering (which results in a memory error) or creates so much subclusters (with *n_clusters=None* parameter) that it's impossible to decipher and hard to plot. It resulted in rand index equal to **0.9000150241392912,** so the biggest so far.

Therefore, I tried providing a sklearn.cluster model (K-Means) to the final step of Birch algorithm, which treats the subclusters from the leaves of the tree data structure as new samples of cluster centroids.

Birch algorithm takes some time to compute and is not really that much effective. Combining Birch and KMeans, the rand index result is **0.8806723262054368**.

Fig. 9: 8-dimensional Birch+KMeans clustering

DBSCAN is a density-based clustering method. Unfortunately, it did not work as intended. Rand index result for 8-dimensional DBSCAN was 0.09998499974999583.

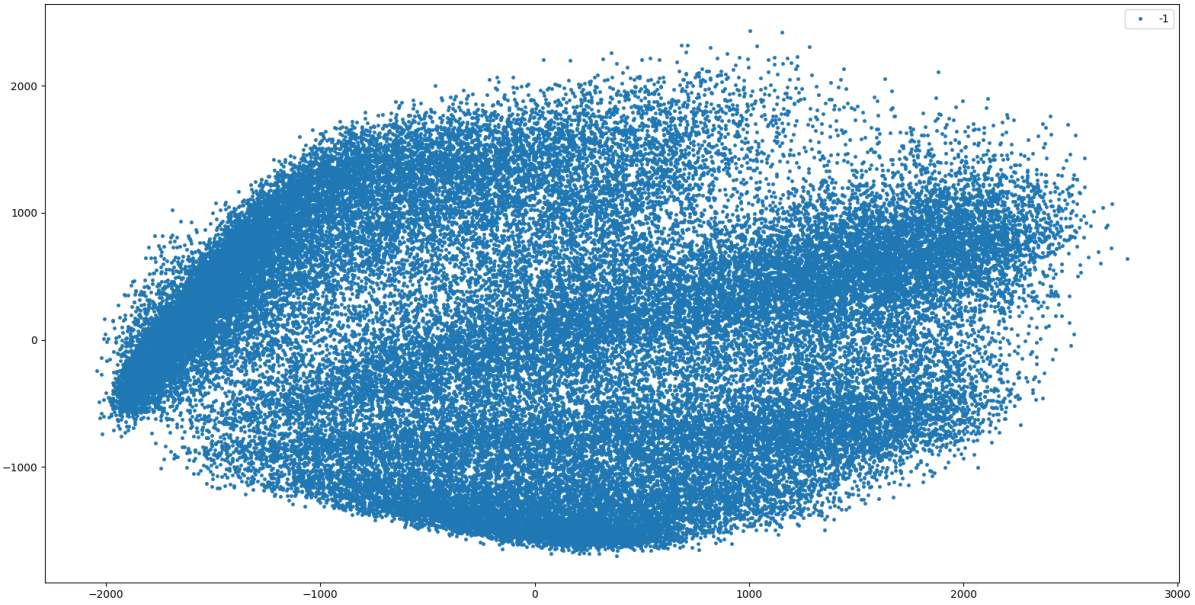

Fig. 10: 8-dimensional DBSCAN clustering

I tried using another density-based clustering — OPTICS, but due to its computing complexity, it could not be processed.

Using Spectral Clustering also ended in failure:

```
Unable to allocate 26.8 GiB for an array with shape (60000, 60000) and data
type float64.
```

In the end, the best clustering result was obtained by 8-dimensional Mini Batch K-Means.

4. Classification

There was no need to split the dataset, since it has been split already. If there had been such a need, KFold sklearn function could have been used.

As a test set, I used data stored in *fashion-mnist_test.csv* file. As a training set — *fashion-mnist_train* was used.

In classification, KNeighborsClassifier was used, since it is fast, efficient and consumes little memory.

I performed classification without dimensionality reduction, since the algorithm is so fast, and dimensional reduction really hurts the training quality.

The accuracy of classification was calculated to be **0.8589.**

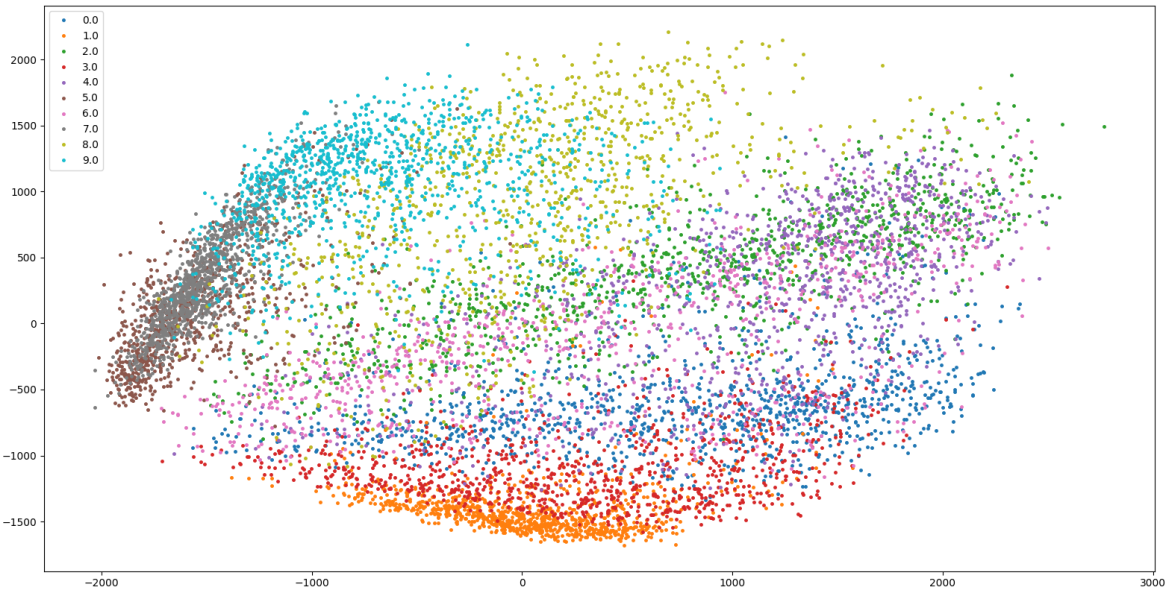

Fig. 11: Full-dimensional classification of test dataset.

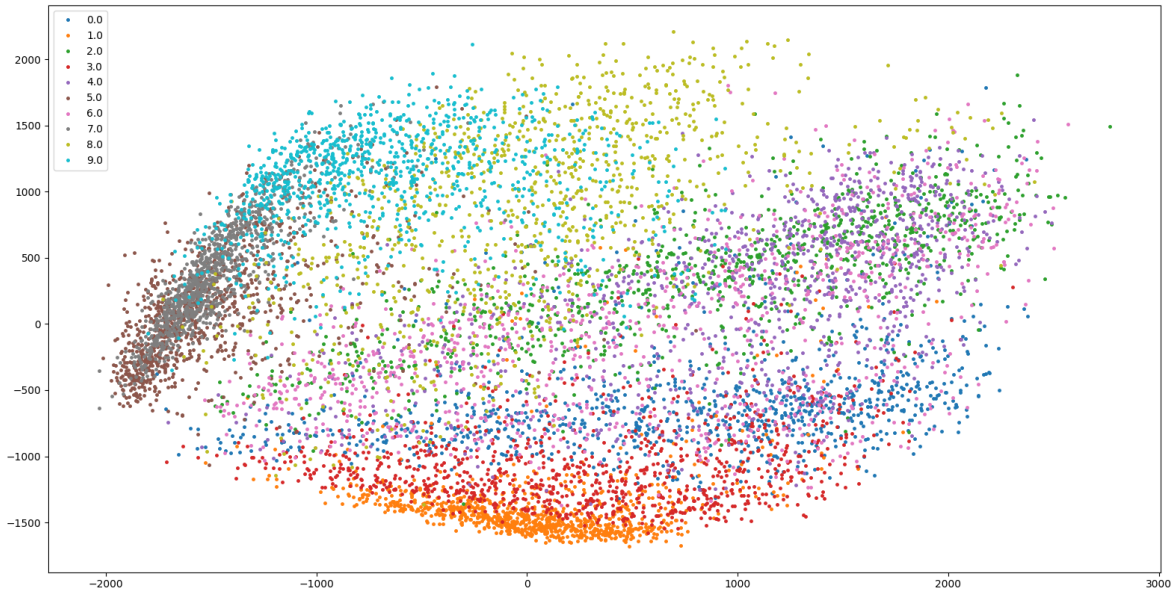

Fig. 12: Full-dimensional classification of test data set, true labels

The accuracy is quite satisfying, there are of course some mismatches here and there.

```
[[874   1  16  11   5   0  86   2   5   0]
 [  4 965   7  14   1   0   9   0   0   0]
 [ 18   0 808  14  98   0  61   0   1   0]
 [ 36  11  14 882  32   0  25   0   0   0]
 [  4   0  99  26 799   0  70   0   2   0]
 [  1   0   1   1   0 816   8  98   4  71]
 [202   1 114  16  74   0 583   0  10   0]
 [  0   0   0   0   0   1   0 945   0  54]
 [  2   1  14   2   6   1  13   5 953   3]
 [  0   0   0   0   0   2   0  34   0 964]]
```

Fig. 13: Confusion Matrix

5. AI Comparison

Surprisingly, the AI (ChatGPT) is not that good at classifying and clustering data. It cannot analyse data by itself (only can give code that we can execute locally), and even if it gives us

some code, it often contains errors or things that will, for example, lead to memory error due to allocating too much memory.

- Summary of data

  It is not this bad, it uses a ready function in pandas library:
  Summary of the training data:
  <class 'pandas.core.frame.DataFrame'>
  RangeIndex: 60000 entries, 0 to 59999
  Columns: 785 entries, label to pixel784
  dtypes: int64(785)
  memory usage: 359.3 MB
  None

- Reduce data dimensionality

  It applies PCA, but it seems to give a rather random value of desired dimensions (it gave me 50 n_components and told me that we can adjust it based on my requirements)

- Visualise the reduced data set

  At first sight, it seems more professional and beautiful of a plot, but it is actually flawed. The colors are very similar to each other, so it's hard to distinguish one from another, and moreover, for some reason labels 2, 5 and 8 are missing. It also gave wrong code the first time, and I had to ask to correct it.

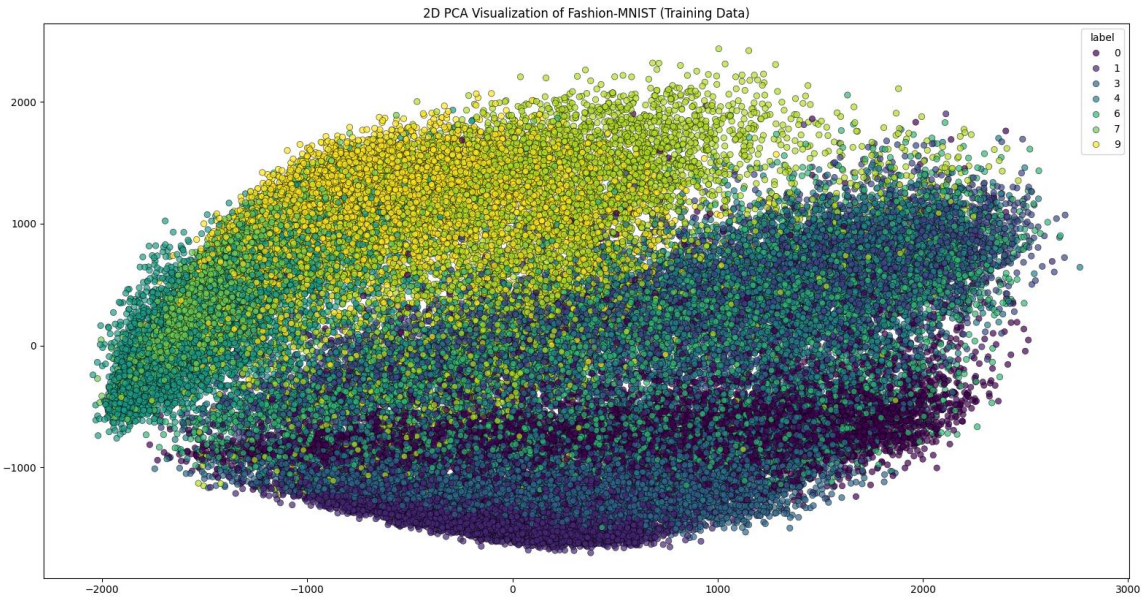

Fig. 14: ChatGPT's visualisation

- Clustering

  Chat-GPT used KMeans, and a visualised confusion matrix to evaluate the result.

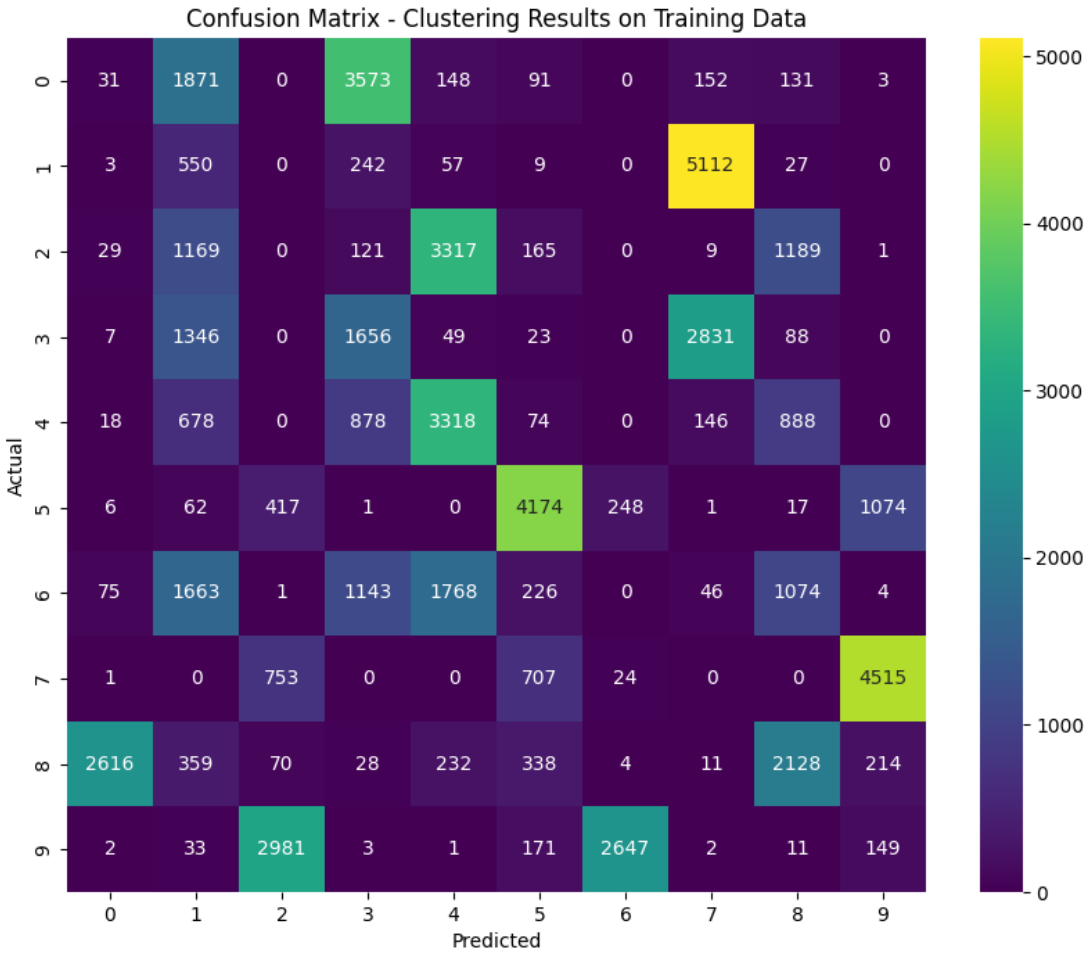

Fig. 15: Chat GPT's confusion matrix

I additionally asked the chat to visualise the clusters.

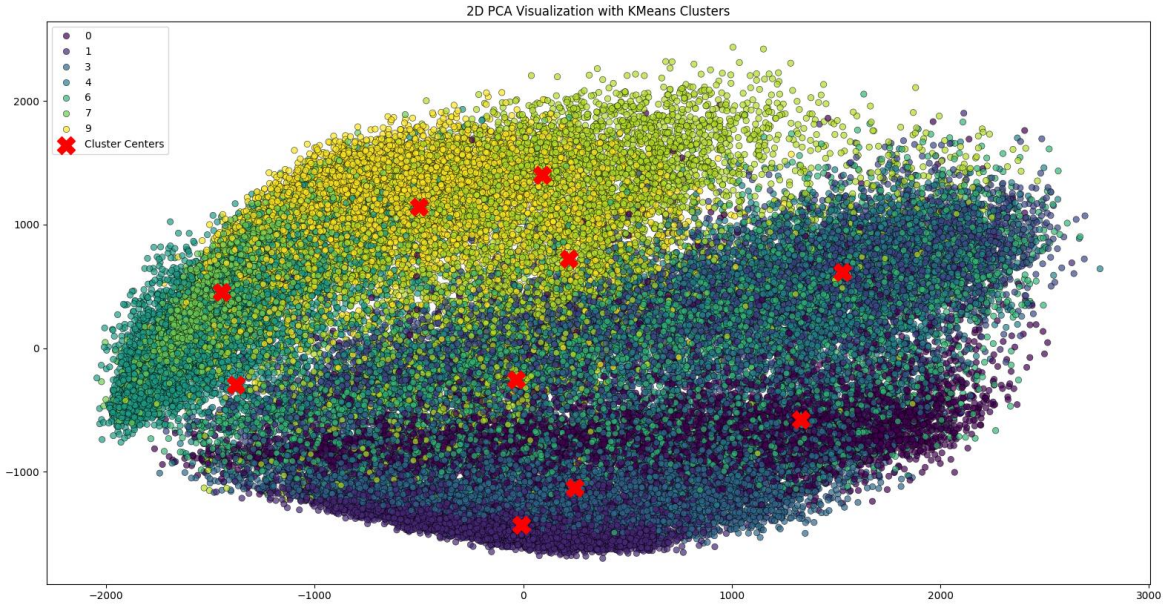

Fig. 16: Chat GPT's clustering

Chat GPT did not bother to use rand index, so it is a bit hard to compare the results to our clustering methods, especially with flawed visualisation like that.

- Splitting data, classification

  Even though ChatGPT acknowledged the existence of previously split test dataset, it still generated new splits, though only using the training dataset.

  To classify the data, it used Logistic Regression classifier.
  It did not give any code so as to plot classification results, but instead generated a ready classification report, shown below.

Classification Report on Validation Data:

	precision	recall	f1-score	support
0	0.78	0.81	0.79	1232
1	0.95	0.94	0.95	1174
2	0.72	0.70	0.71	1200
3	0.79	0.84	0.81	1242
4	0.69	0.74	0.71	1185
5	0.86	0.89	0.87	1141

6	0.59	0.48	0.53	1243
7	0.86	0.86	0.86	1224
8	0.94	0.93	0.93	1149
9	0.92	0.93	0.93	1210
accuracy			**0.81**	12000
macro avg	0.81	0.81	0.81	12000
weighted avg	0.81	0.81	0.81	12000

It seems that the classification accuracy was a bit worse than ours (0.8589).

However, classification turned out to be much faster, using the method that the Chat suggested. It seems that classification was being done on data with reduced dimensions. After raising the number of dimensions to 100, the accuracy also raised to 0.85. Raising the number of dimensions further did not result in increase of the accuracy, however. It still is really fast.

All in all, ChatGPT is not that great a tool, if you do not know what you are doing.

If you know however, it may be a bit annoying to deal with, but it does its job.

Sources:

Data:

https://www.kaggle.com/datasets/zalando-research/fashionmnist

Library:

https://scikit-learn.org/stable/index.html

Code:

https://github.com/Areczek00/FoDS-Project