Fundamentals of Data Science

Project report

AGH University of Science and Technology Kraków
Faculty of Physics and Applied Computer Science

Adam Młyńczak

All of the code used to perform given tasks is in the GitHub repository.

1. Summary of the data

The Fashion MNIST dataset is database of fashion images from a dataset of Zalando. The original dataset contains over 70 thousand images of fashion products from 10 categories (I have put a label from a dataset in brackets):

1) T-shirt/top (0),
2) Trouser (1),
3) Pullover (2),
4) Dress (3),
5) Coat (4),
6) Sandal (5),
7) Shirt (6),
8) Sneaker (7),
9) Bag (8),
10) Ankle boot (9).

Every category listed has about the same amount of images (7k per category). Every one picture is a 28 pixels high and 28 pixels wide image in grayscale. Each example is associated with label listed above. Looking into the single image, every pixel has a value connected with lightness or darkness of this pixel, on a scale from 0 to 255. The 0 means the pixel is white and 255 means the pixel id black.

Each row represents separate image and each row contains the same amount of data. The first column is class label (0-9, same as label in brackets on the list of categories above) and the remaining 784 columns (28 times 28 – height and width of single image) represents this darkness I wrote about in paragraph above.

If we want to access single pixel in picture, we can use a simple equation:

$$x = 28 * i + j,$$

where i is number between 0 and 27 representing row and j is integer representing column in the same range.

The format of this data is in my opinion really deliberate. Now of course we can just paste an image to AI Model and it will analyse any part of this shot with a specific description of it, but when the dataset was made it was not that easy. We had to preprocess this data and convert it into the form that computer/model would understand. And what is better representation of data for computers than just numbers?

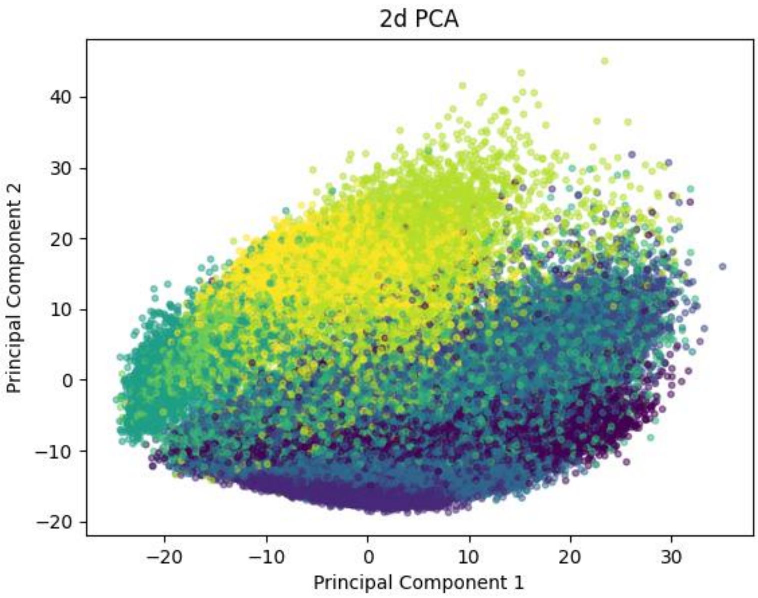
2. Reducing data dimensionality

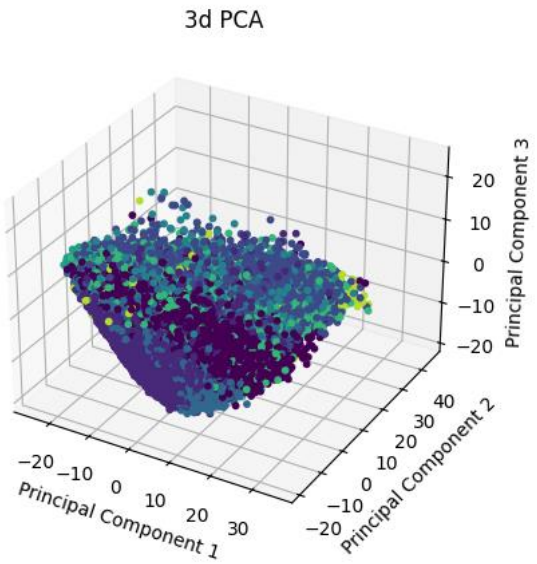
To reduce the data dimensionality I used a PCA function from a *sklearn.decomosition* library. This function uses, as the name indicates, Principal Component Analysis is which we obtain eigenvectors for a matrix to reduce the mentioned dimensionality. In python code I set a number of components to 2 and 3, because its easy to visualize this and what it indicates its easier to understand the reduced data. The code that contains described operation is in the */solutions* directory under the name *reduce_dimen.py*.

3. Visualize the reduced dataset

The reduced dataset is visualized using matplotlib in */solutions* in file *visualisation.py*. The colours of points at those plots are associated with first column of data – the label of item. As we can see, the same-coloured dots are in the same area (the number of those dots even make a impression of some kind of gradient), which means that it works.

Here are the visualised and reduced to smaller number of dimensions data:



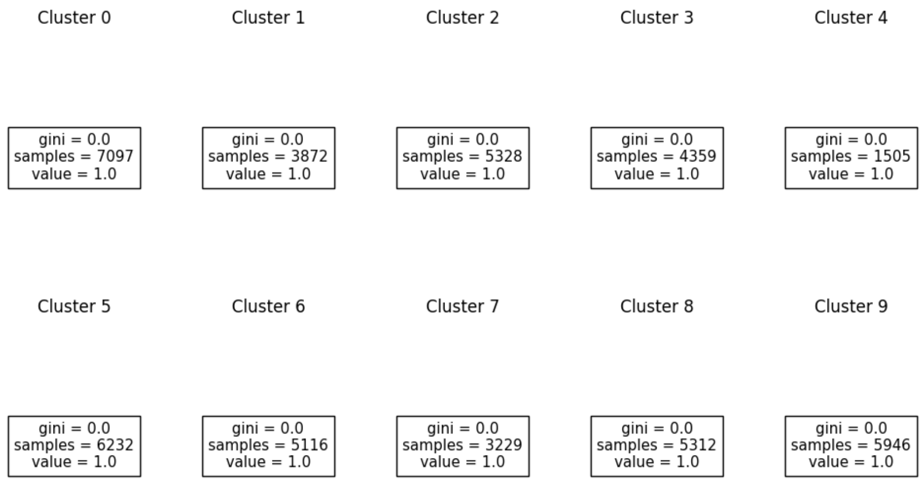


4. Clustering the data

As we know, clustering is nothing else than dividing the dataset info groups – clusters to group those dataset, to accumulate the similar objects into one "place". To achieve this I used a KMeans lib from *sklearn.cluster* (to locate those neighbours). Additionally I create a Decision Tree Classifier to portrait what has happened, with Classifier Accuracy for every Cluster. At the end I also wanted to see the confusion matrix to evaluate the performance of the clustering model. The code is provided in */solutions* by the name *clustering.py*. The accuracy for clusters and confusion matrix looks like this:

```
Cluster 0 - Classifier Accuracy: 1.0
Cluster 1 - Classifier Accuracy: 1.0
Cluster 2 - Classifier Accuracy: 1.0
Cluster 3 - Classifier Accuracy: 1.0
Cluster 4 - Classifier Accuracy: 1.0
Cluster 5 - Classifier Accuracy: 1.0
Cluster 6 - Classifier Accuracy: 1.0
Cluster 7 - Classifier Accuracy: 1.0
Cluster 8 - Classifier Accuracy: 1.0
Cluster 9 - Classifier Accuracy: 1.0
Confusion matrix:
[[1083    6 2795   88   23  381  369   11   16 1228]
 [4280    0  178    5    5   35   50    1    0 1446]
 [  28   14  302  143   60 2833 1698   32   49  841]
 [2800    0 1424   22    0   30  110    1    2 1611]
 [ 276    3 1074   75   30 2741 1273   13   10  505]
 [   1  483    0 3146    2    1   52   63 2204   48]
 [ 378   24  871  216   81 1596 1443   27   65 1299]
 [   0  882    0 1649    1    0    3   48 3417    0]
 [  25  721   17   71 1417  145 1163 1641  359  441]
 [   1 2707    0   34  263   29  234 2200  518   14]]
```

At the first glance it looks perfect with Accuracy for every cluster at the same level of 1.0, but when we look at the Confusion Matrix its not that good. Of course most of the big numbers are on the diagonal or near it, but there are some values that indicates that the clustering was not that good. Maybe the implementation of clustering provided by myself is not that good nor accurate.
The Decisions for each cluster looks like this:



As we can see, the clusters are not evenly placed in terms of number of samples, which again, indicates the problem with clustering – or problem with method used for this dataset specifically.

5. Split the dataset into training and testing

To split the dataset I used function train_test_split from *sklearn.model_selection*. The code is provided in */solutions* under the name *split.py*.

6. Perform classification and evaluate its results

To complete this task I used four different methods. For each I performed a test with accuracy and classification report at the end. The code with this task is in the */solutions* directory under the name *classification.py.*

1) K-Nearest Neighbours

In this situation, the program looks for a nearest neighbour – the data with most similarities and indicates that the missing information will be the same as the neighbour.

The results are as follows:

```
KNeighborsClassifier:

Accuracy: 0.85

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.86      0.81      1202
           1       0.99      0.97      0.98      1219
           2       0.76      0.80      0.78      1205
           3       0.89      0.88      0.88      1184
           4       0.76      0.77      0.77      1202
           5       0.99      0.83      0.90      1211
           6       0.65      0.57      0.61      1218
           7       0.87      0.95      0.91      1159
           8       0.98      0.92      0.95      1197
           9       0.90      0.97      0.93      1203

    accuracy                           0.85     12000
   macro avg       0.86      0.85      0.85     12000
weighted avg       0.85      0.85      0.85     12000
```

The accuracy is at fine level, as we can see.

2) Random Forest Classification

In this case, the way is to create a forest and the trees are "voting" which information should we put in the missing information. The results of this classification:

```
RnadomForestClassifier:

Accuracy: 0.88

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      1202
           1       1.00      0.97      0.98      1219
           2       0.79      0.82      0.80      1205
           3       0.87      0.91      0.89      1184
           4       0.77      0.83      0.80      1202
           5       0.97      0.96      0.97      1211
           6       0.75      0.60      0.67      1218
           7       0.94      0.94      0.94      1159
           8       0.96      0.97      0.96      1197
           9       0.95      0.96      0.95      1203

    accuracy                           0.88     12000
   macro avg       0.88      0.88      0.88     12000
weighted avg       0.88      0.88      0.88     12000
```

As we can see, the results are similar to the previous test.

3) Logistic Regression

In this scenario, the key is to count the probability on behalf of the previous results, with the linear combination using weights as indicator. The accuracy and report for this classification:

```
LogisticRegression:

Accuracy: 0.85

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.82      0.80      1202
           1       0.98      0.97      0.97      1219
           2       0.76      0.74      0.75      1205
           3       0.85      0.88      0.86      1184
           4       0.74      0.78      0.76      1202
           5       0.93      0.94      0.93      1211
           6       0.65      0.60      0.62      1218
           7       0.91      0.92      0.91      1159
           8       0.95      0.92      0.94      1197
           9       0.94      0.95      0.94      1203

    accuracy                           0.85     12000
   macro avg       0.85      0.85      0.85     12000
weighted avg       0.85      0.85      0.85     12000
```

4) **Gaussian Naïve Bayes classifier**

In this method, it's the similar to previous method, but there we are counting the probability for each possibility and choose the highest. Here are the results:

```
GaussianNB:

Accuracy: 0.57

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.59      0.69      1202
           1       0.56      0.96      0.71      1219
           2       0.59      0.31      0.40      1205
           3       0.42      0.44      0.43      1184
           4       0.36      0.76      0.49      1202
           5       0.92      0.24      0.38      1211
           6       0.39      0.05      0.08      1218
           7       0.48      0.98      0.64      1159
           8       0.85      0.71      0.77      1197
           9       0.92      0.65      0.76      1203

    accuracy                           0.57     12000
   macro avg       0.63      0.57      0.54     12000
weighted avg       0.63      0.57      0.54     12000
```

It this case the accuracy is much lower, probably because the assumption that features in dataset are conditionally independent, where in scenario with images – its really hard to be independent from your neighbouring pixel.

7. ChatGPT part

As was asked in the project description, I enquire ChatGPT to complete the same tasks. The code it provided is in main directory under the name *ChatGPT_all.py*. Here are similarities/differences:

7.1. Download the dataset

As AI is unable to download the dataset (or is unable to download in the free version), it has downloaded the dataset using *tensorflow.keras.dataset*. In the beginning I downloaded the dataset from Kaggle website, but after this discovery I changed it also in my code, because its just easier way.

7.2. Summary of the data

In this task ChatGPT did what was logic for it – just inform about the data shape and listed the labels. It is probably the only information it needed to work on this dataset, so I understand this move.

```
Shape of training data: (60000, 28, 28)
Unique labels in training data: [0 1 2 3 4 5 6 7 8 9]
```

7.3. Reduce data dimensionality

In this task, ChatGPT use two different ways to obtain the wanted result. One is the same as I did, the PCA, and the other is using t-SNE (t-distributed Stochastic Neighbour Embedding). In assumptions it's not defined by random probability and concerned only about retaining the variance of neighbour points. In some ways it is better method to reduce the dimensionality, but for a simple dataset like fashion-MNIST it is not that important – there are other, more complicated sets of information, where this method will be much better than PCA.

7.4. Cluster the dataset

The AI model uses the same thing I did – KMeans, to cluster the dataset. Nothing interesting. In addition it uses confusion matrix to evaluate the clustering results. The matrix looks like this:

```
Confusion Matrix:
 [[1083    6 2795   88   23  381  369   11   16 1228]
 [4280    0  178    5    5   35   50    1    0 1446]
 [  28   14  302  143   60 2833 1698   32   49  841]
 [2800    0 1424   22    0   30  110    1    2 1611]
 [ 276    3 1074   75   30 2741 1273   13   10  505]
 [   1  483    0 3146    2    1   52   63 2204   48]
 [ 378   24  871  216   81 1596 1443   27   65 1299]
 [   0  882    0 1649    1    0    3   48 3417    0]
 [  25  721   17   71 1417  145 1163 1641  359  441]
 [   1 2707    0   34  263   29  234 2200  518   14]]
```

It looks almost the same, if not the same. It just because we used the same method, I guess.

7.5. Split the dataset into training and testing

Same thing as in clustering, Chat used the same function as me to do it.

7.6. Perform classification and evaluate its result

In this task ChatGPT used the RandomForrestClassifier to evaluate. The accuracy is as follows:

```
Classification Accuracy: 0.5031666666666667
```

This looks really low compared to what happened in my classification using the same method, but we used different parameters for the *RandomForestClassifier* function and I think the main difference comes from this.

8. Summary

In this project I performed actions to learn as much as possible from a single dataset. Of course, for every single task in this project we can use multiple solutions – from downloading dataset in different way to evaluating classification results. Data Science is really big and still growing field of study and this was only a drop in the ocean, showing what we can do on one of the not that much complicated datasets.