Fundamentals of Data Science
Report

Serhii Vynohradov

December 16, 2024

1 Introduction

This report is about the Fundamentals of Data Science course project. The goal of the project was to process and analyze a given dataset by addressing the following tasks and questions:

- Do the summary of the data and discuss it.
- Reduce data dimensionality
- Visualize the reduced dataset
- Cluster the dataset (and evaluate clustering results with classification labels). Do we have some outliers here?
- Split the dataset into training and testing.
- Perform classification and evaluate its result.

The code was written in **Google Colab**, and is provided as an attached **.ipynb** notebook.
In this project, I tried to use most of the algorithms shown in labs/lectures and approach the problem with different initial settings. Most machine learning functions are imported from **Scikit-learn** library.

2 Dataset Overview

My work on the **FODS** project started from finding *Fashion-MNIST* dataset. This dataset was made by **Zalando Fashion**. It includes 60,000 different clothing pieces,each represented as 28×28 pixel image. In **.csv** file, columns or, as we should call them, features represent each of 784 pixels, while rows represent cases or records. The first column is the label number, which we will drop and use for reference later. The labels are numbered from 0 to 9:

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

Reshaped and plotted via *matplotlib* set looks like this:

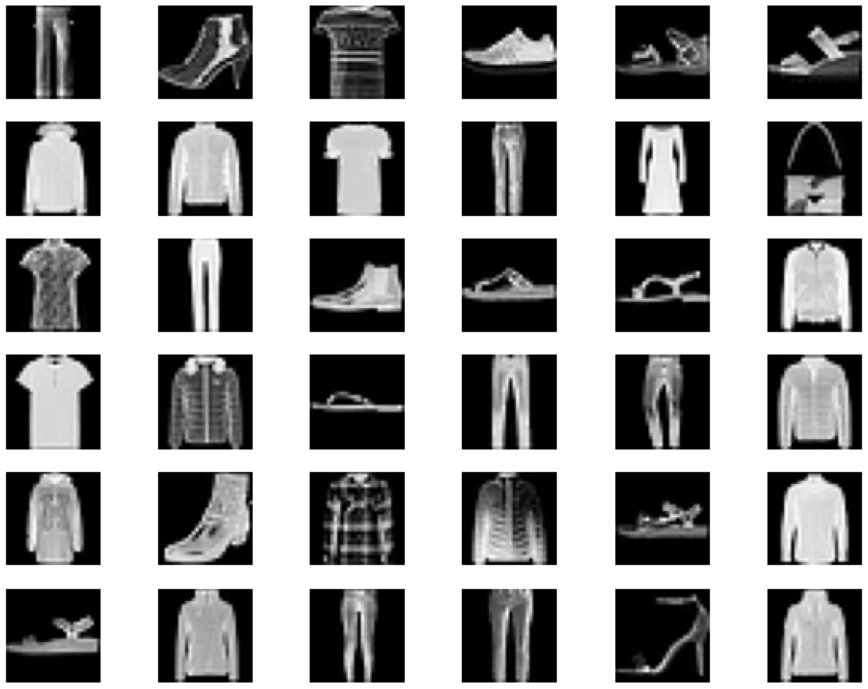


Figure 1: First 36 items represented graphically.

Unfortunately, some of the computations we are doing might be time-consuming. For that reason, we resampled the dataset, shrinking it from 60,000 to only 30,000. It does affect accuracy, but that's the price we have to pay. However, the effect is mostly just a few percent.
Now, let's sum up what we know about the data and do some basic calculations:

- Range: data ranges from 0 to 255
- Type: int values
- Mode(dominant value): 0
- Median: 0.0
- Arithmetic mean: 72.788
- Variance: 8091.641
- Standard deviation: 89.9535

We can see that the dominant value is 0, which affects our mean and deviation. That is because half of every image is darkness - the background.

3 Reduce data dimensionality

Before we do anything with the data, we should reduce dimensionality. 784 features that we have now will definitely affect us with the 'curse of dimensionality'. I chose two algorithms which we used in labs. That's **PCA** and **t-SNE**. Some perform better or worse in different scenarios. First of all, I normalized the data using *MinMaxScaler* to be in $[0, 1]$ range.

3.1 PCA

As an input for PCA algorithm we should pass number of dimensions or wanted variance. Ideally, we want to keep variance above 90%. I decided to keep 95%, which left us with almost 180 features. 3 times less, which is amazing. For only 90% it leaves 80 components, however it doesn't affect the time that much. Here's a graph to show how much variance each principal component has:

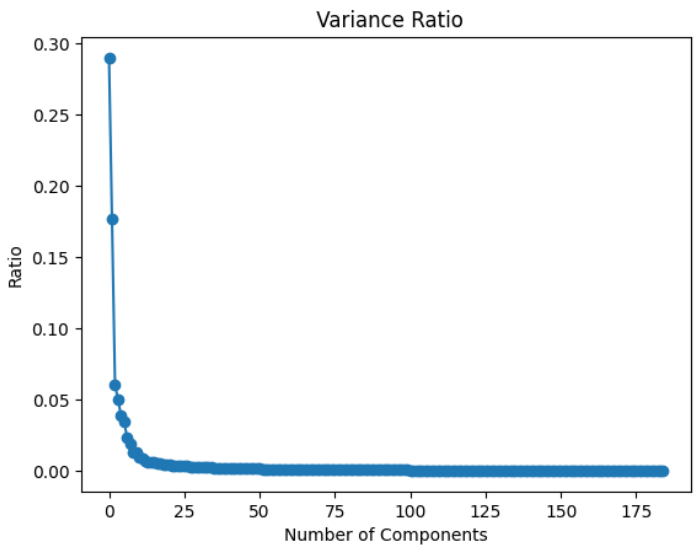


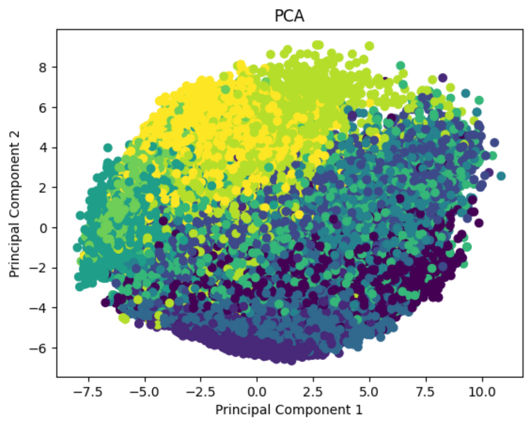
Figure 2: Graph showing variance ratio of PCA principal components.

3.2 t-SNE

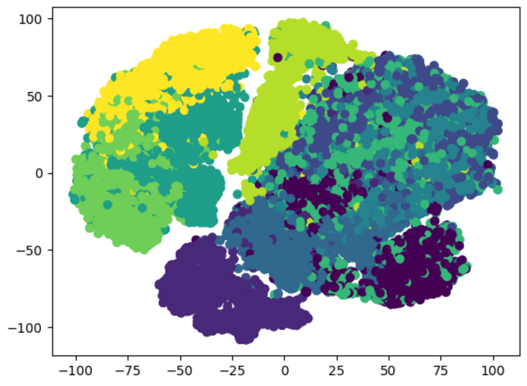
As an input for the t-SNE algorithm, we should pass the number of dimensions. After some testing, it turned out that 2 is a better option, and it reduces the time of computation by a bit, which is still pretty high.

4 Visualize the reduced dataset

Below is the demonstration of scatter plots colored using the label array:



(a) PCA generated scatter plot.



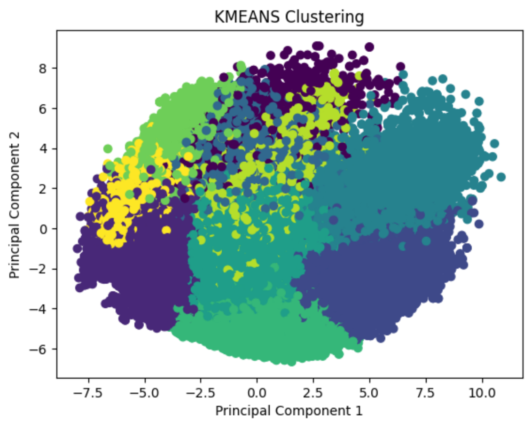
(b) t-SNE generated scatter plot.

5 Cluster the dataset

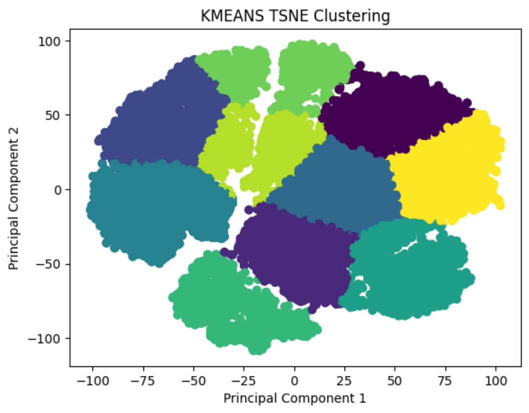
For a clustering problem, I chose 3 algorithms that we used in the labs. I tried each of them on both of our reduced datasets, to see which one works better.

5.1 KMeans

I decided to start with the simplest: KMeans algorithm. By passing it the number of clusters we want, $n = 10$, and setting init positions into automatic mode, we got the next results:



(a) KMeans over data reduced with PCA.



(b) KMeans over data reduced with t-SNE.

To calculate the efficiency, we use rand index:

$$PCA : 0.874 \mid \text{Rand Index} \mid t - SNE : \; 0.89$$

After seeing the result, I was surprised by the algorithm's efficiency. However, after plotting the first 100 records, we can see the reason for such a result:

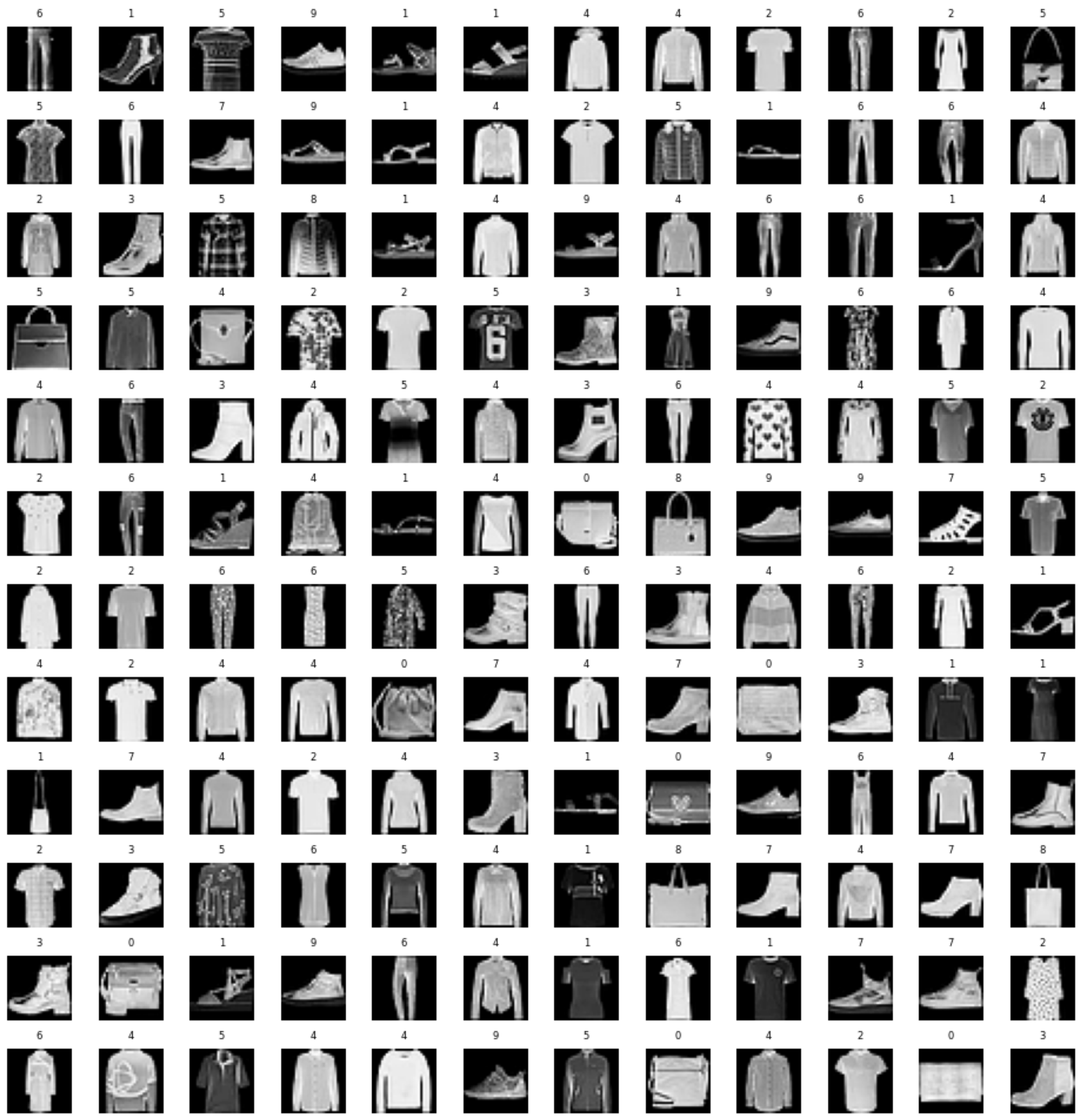


Figure 5: Image with their cluster number, achieved by KMeans over data reduced with PCA.

While at first glance everything might seem as it should be, after a closer look we can think of some pattern: coats are being mistaken with pullovers, bags are separated into clusters 4 and 5, probably depending on how 'square' they are and the position of their handle and so on. However, even with such mistakes, we can admit that simplification of the categories such as top/bottom/boots and bags would work much better. We can support this theory by one observation: despite the even number of items in given labels, number of elements in clusters obtained by KMeans varies slightly: from 1100 in the smallest to almost 5000 in the biggest one.

We can also try to see the cluster center elements:

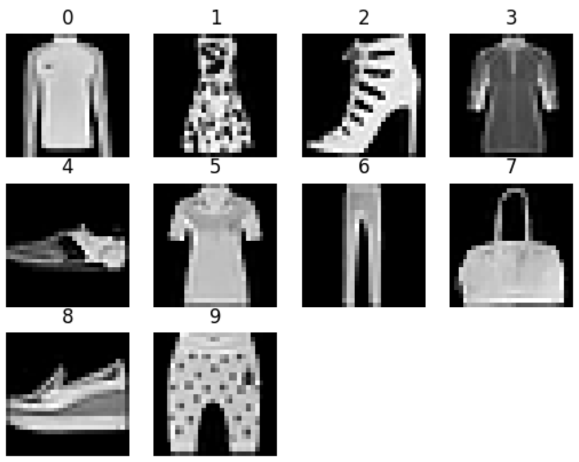


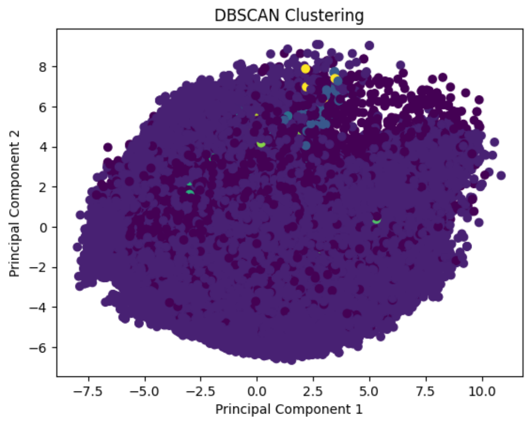
Figure 6: Cluster center locations obtained by KMeans over data reduced with t-SNE.

The confusion raised by rand_index comes from the way it's calculated. Objects in comparable sets are indeed in the different clusters; however, for us, it is important in which one.
To solve this issue, we will use *ARI* which stands for *Adjusted Rand Index* and *AMI* - *Adjusted Mutual Information Score*. Calculated score:

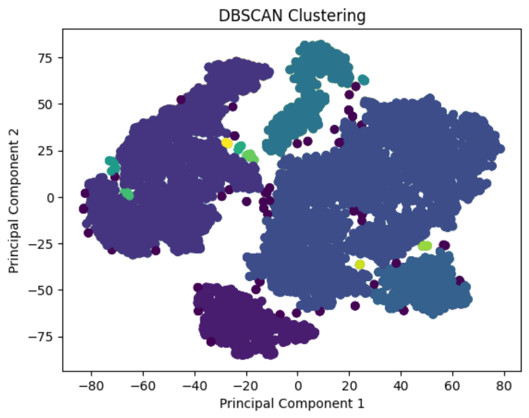
$PCA: 0.52$	AMI	$t - SNE:$ 0.55
$PCA: 0.36$	ARI	$t - SNE:$ 0.38

5.2 DBSCAN

Next algorithm is DBSCAN.



(a) DBSCAN over data reduced with PCA.

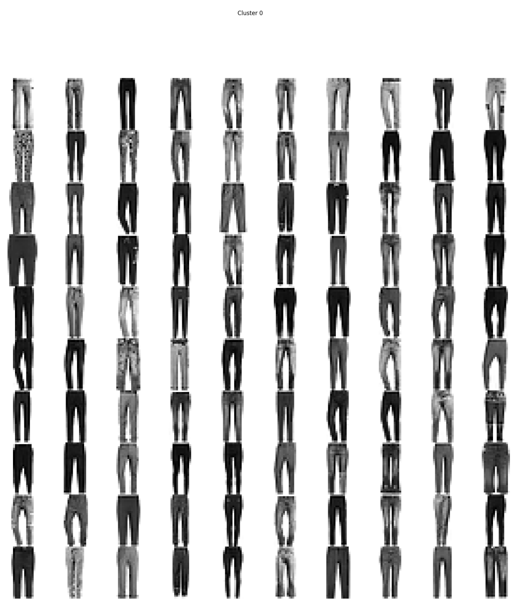


(b) DBSCAN over data reduced with t-SNE.

$PCA: 0.33$	Rand Index	$t - SNE:$ 0.76
$PCA: 0.04$	AMI	$t - SNE:$ 0.6
$PCA: 0.01$	ARI	$t - SNE:$ 0.31

With a lot of time spent on tuning PCA's tragic result,trying different distances *eps* and minimal sample numbers, this is the best I managed to get. We can see that almost every component is assigned to 1 cluster. At first, I thought it was a result of the 'curse of dimensionality', since we have more than 100 features. However, changing PCA mode to 50, 20, or 10 principal components did not improve the situation.
Result for t-SNE is taken from the lower amount of $n = 10,000$, where I experimented and managed

to get quite nice results, unlike PCA.
But, why do I say that the result is good despite low *AMI* and *ARI* scores? For that information, we should look into clusters directly:



(a) Records assigned to cluster 0.

(b) Records assigned to cluster 1.

Above is a plot that shows us the first 100 records of cluster 0 and 1. We can see 100% accuracy with both pants and shoes. This was a problem I previously pointed out to with KMeans algorithm. This is related to the unsupervised nature of the clustering problem, which helps us to 'discover' the data, since we can't directly tell the algorithm the difference between shirt and pullover.
To be fair, this exact algorithm has not performed that well on the rest, sometimes mistaking shirts for bags, and it's the only algorithm that discovered outliers. Let's actually see them:

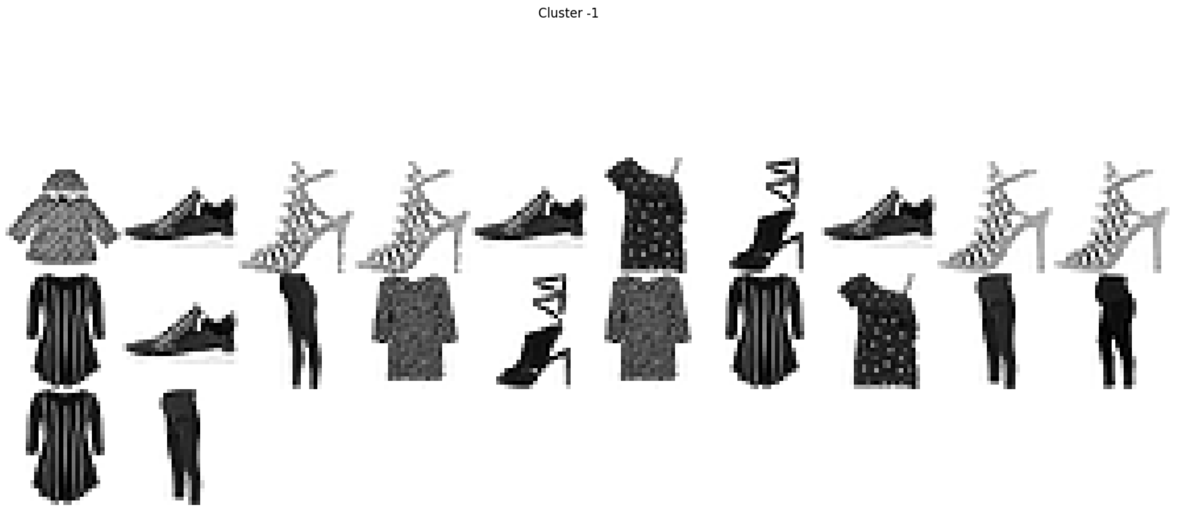
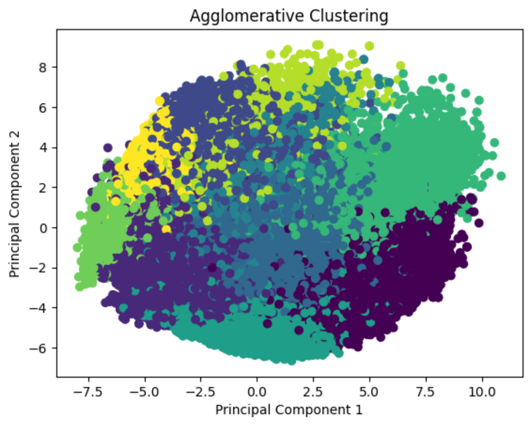


Figure 9: Outliers uncovered by DBSCAN.

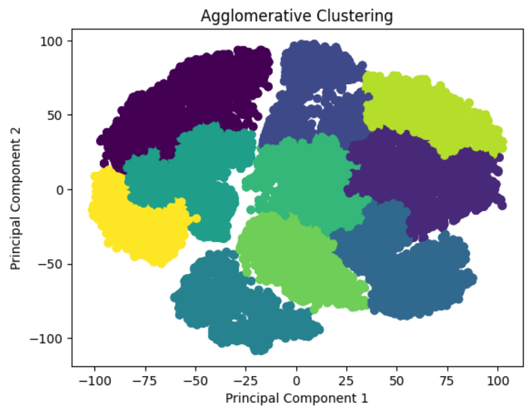
We observe that there were issues with pants being placed at an unusual angle and the heels.

5.3 Agglomerative Clustering

The last tested algorithm was hierarchical clustering in an agglomerative implementation.



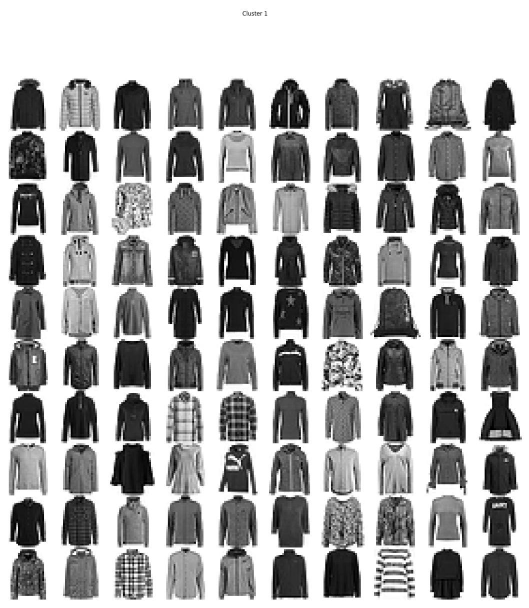
(a) Agglomerative Clustering over data reduced with PCA.



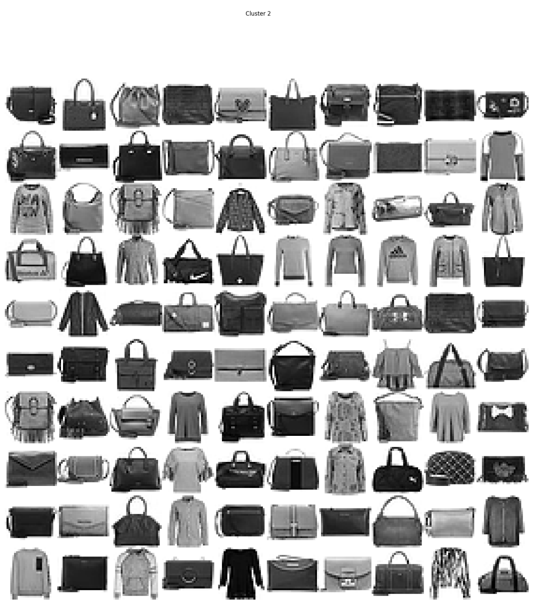
(b) Agglomerative Clustering over data reduced with t-SNE.

$PCA: 0.88$	Rand Index	$t-SNE:\ 0.9$
$PCA: 0.53$	AMI	$t-SNE:\ 0.59$
$PCA: 0.367$	ARI	$t-SNE:\ 0.455$

Version that has been operating over PCA reduced array did similar to KMeans algorithm. However, the one that operated on t-SNE did the best out of all tested algorithms according to ARI score. It even had quite good accuracy while distinguishing the top clothing items, and unlike DBSCAN it has mistaken bags for shirts quite rarely:



(a) Records assigned to cluster 1.



(b) Records assigned to cluster 2.

Maybe, if we shorten our labels to only 5 or 6,by combining categories, this algorithm would work with efficiency close to actually trained classifiers...

5.4 Summarization

Overall we can notice that t-SNE did better than PCA. However, PCA algorithm was incomparably faster to compute. For $n = 30,000$ it took us 1 second for PCA and 9 minutes for t-SNE. Knowing both algorithm properties, we might assume that our dataset likely contains non-linear relationships.

6 Split the dataset into training and testing

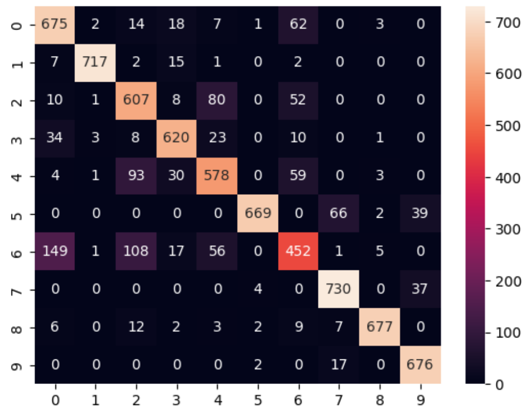
Using **Scikit-learn** data splitter we split our data into 4 parts. 2 for training, 2 for testing.

7 Perform classification and evaluate its result

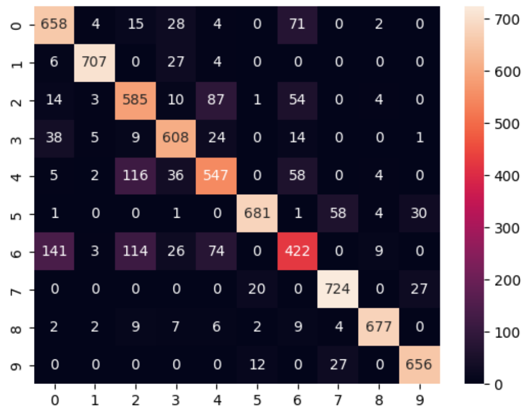
In this section, I tried to test most of the classifiers that we talked about in labs/lectures and represent their scores with a confusion matrix. Similar to the previous section, for almost every classifier tested, the data was prepared using PCA and t-SNE to compare their results.

7.1 KNeighborsClassifier

First, I tested KNeighborsClassifier with parameter $n = 5$:



(a) Confusion matrix for KNeighborsClassifier over data reduced with PCA.



(b) Confusion matrix for KNeighborsClassifier over data reduced with t-SNE.

$PCA : 0.853$	Score $n = 30,000$ samples	$t - SNE :\ 0.835$
$PCA : 0.825$	Score $n = 10,000$ samples	$t - SNE :\ 0.804$

KNeighborsClassifier did quite well with over 85% accuracy, having problems only with a few categories. Changing the number of neighbors n may not significantly improve the score. I gathered data for the number of samples $n = 10,000$ as well, so we can see a small improvement 3%.

7.2 Decision Tree

Next, I tested the decision tree with a depth of 4; data was previously reduced with t-SNE:

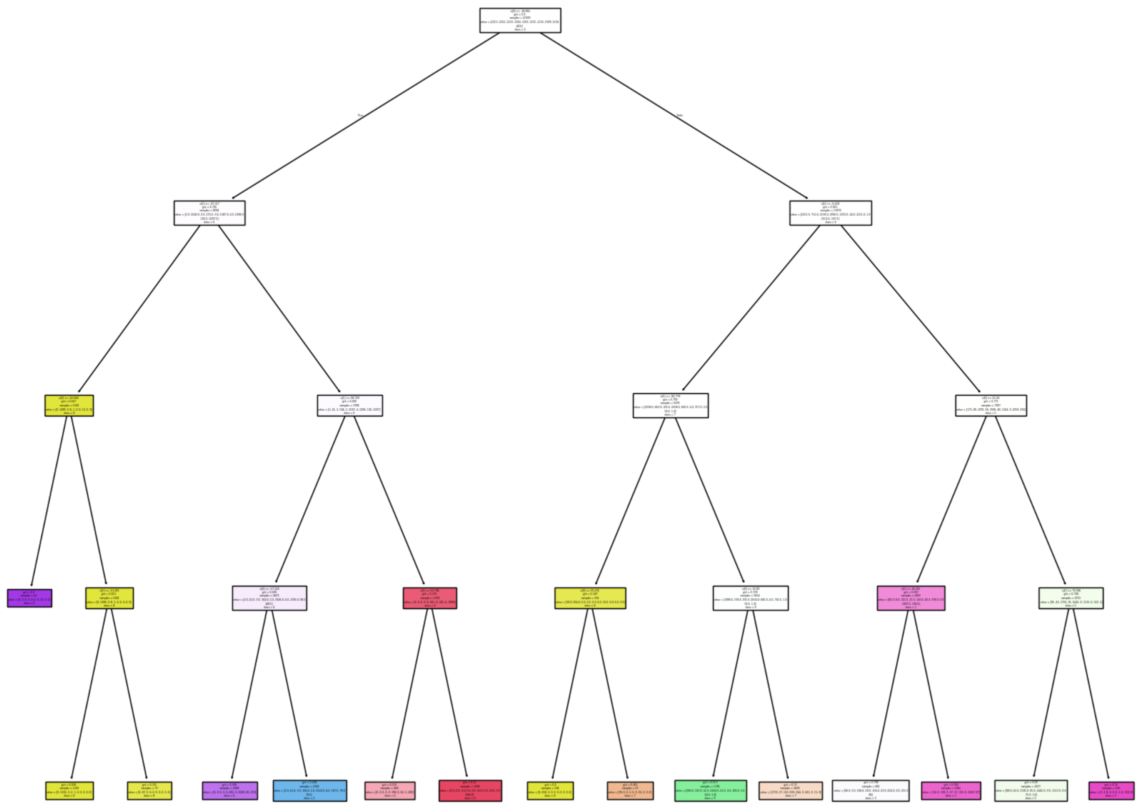


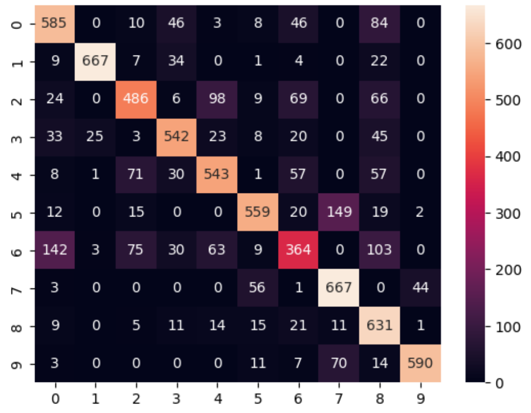
Figure 13: Decision Tree over data reduced with t-SNE.

Score $n = 30,000$ samples	$t - SNE:$ 0.62
Score $n = 30,000$ samples	$PCA:$ 0.55
Score $n = 10,000$ samples, depth 3	$PCA:$ 0.471

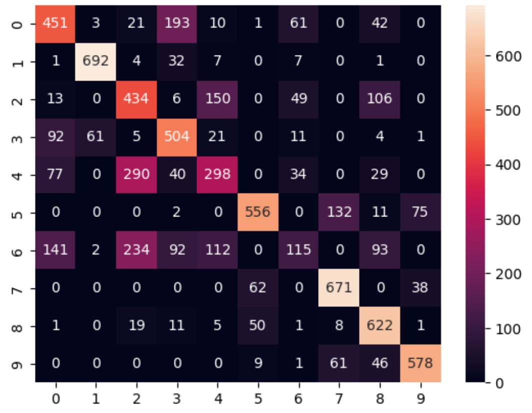
Decision tree left us with average classification results.

7.3 Gaussian Naive Bayes

Then we tested Naive Bayes classifier in Gaussian implementation:



(a) Confusion matrix for Gaussian Naive Bayes over data reduced with PCA.



(b) Confusion matrix for Gaussian Naive Bayes over data reduced with t-SNE.

$PCA : 0.75$	Score $n = 30,000$ samples	$t - SNE$: 0.66
$PCA : 0.73$	Score $n = 10,000$ samples	$t - SNE$: 0.67

We can notice a strange result here: more data for the t-SNE version did not result in an accuracy improvement.

7.4 Support vector machine

Next, I tried the support vector machine algorithm:

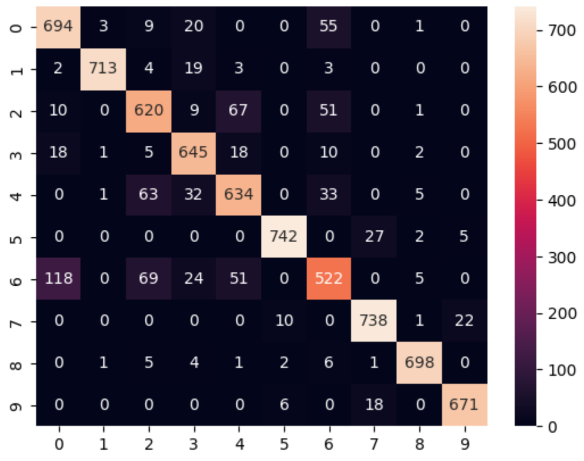


Figure 15: Confusion matrix for SVM over data reduced with PCA.

$PCA : 0.89$	Score $n = 30,000$ samples	$t - SNE$: 0.7388
PCA : 0.863	Score $n = 10,000$ samples	

SVM algorithm shows the best results among classifier algorithms. Its confusion matrix shows us the error it often makes: mistaking shirts for t-shirts/tops and pullovers.

7.5 Summarization

After testing most of the popular classifiers on our particular dataset, we can say that the best were: SVM and KNeighborsClassifier. All the classifiers, as we can see from their confusion matrix, as well as clustering algorithms, had a problem telling apart shirts from other upper body clothing. Data reduced by PCA, unlike before, was overall better than t-SNE.

8 ChatGPT opinion

After asking ChatGPT how they would do the task, I got this answer:

1. **Download and Summarize the Data:**
   - Load the Fashion-MNIST dataset using libraries like TensorFlow or PyTorch.
   - Summarize the dataset by examining shapes, class distributions, and displaying sample images.
2. **Reduce Data Dimensionality:**
   - Flatten the images and normalize pixel values to a range of 0–1.
   - Use dimensionality reduction techniques like **PCA** (e.g., reduce to 2 components) or **t-SNE**.
3. **Visualize the Reduced Dataset:**
   - Create a scatter plot of the reduced dataset (e.g., PCA components).
   - Color-code points based on their labels to observe patterns.
4. **Cluster the Dataset:**
   - Apply clustering algorithms like **K-Means** or **DBSCAN**.
   - Evaluate clustering results using metrics such as **ARI (Adjusted Rand Index)** or **NMI (Normalized Mutual Information)**.
   - Analyze potential outliers identified through clustering.
5. **Split the Dataset:**
   - Use the provided train-test split in the Fashion-MNIST dataset.
6. **Perform Classification:**
   - Train a classifier like Logistic Regression, Random Forest, or a simple Neural Network.
   - Evaluate classification performance using accuracy, precision, recall, F1-score, and a confusion matrix.

Let's analyze the response: Parts 1-3 I did according to their advice, the only difference is, I tested both t-SNE and PCA instead of using a single technique.

4. I used Hierarchical Clustering as an additional method, which turned out to be quite good with the best ARI score.
5. In my program, I used **Scikit-learn function** to split the set. ChatGPT's advice is better since it saves more data.
6. ChatGPT presented the methods which I haven't tried.
   I measured classification performance with accuracy and confusion matrix only, not calculating ROC score, which we can try to see if it changes the ranking result.

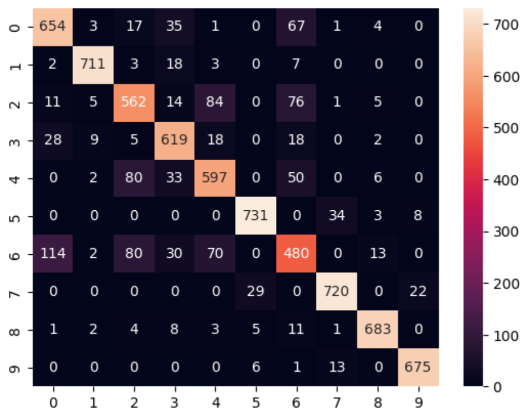
8.1 Evaluating ROC

$PCA : 0.972$	Score KNeighborsClassifier	$t - SNE :$ 0.964
$PCA : 0.87$	Score DecisionTree	$t - SNE :$ 0.923
$PCA : 0.95$	Score Gaussian Naive Bayes	$t - SNE :$ 0.944
$PCA : 0.992$	Score SVM	$t - SNE :$ 0.959

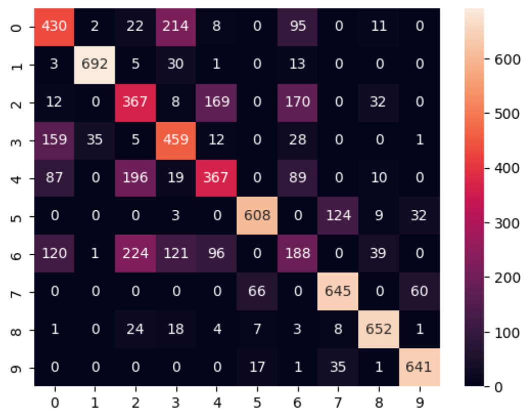
The ROC score is quite high for all of the classifiers. It lets us evaluate classification results better than the accuracy I used. The 'ranking' still remains the same: SVM goes first, then KNeighborsClassifier.

8.2 Logistic Regression

I wanted to check if Logistic Regression would perform better than the methods I've tried, and compare it to SVM in particular. This is the result I got:



(a) Confusion matrix for Logistic Regression over data reduced with PCA.



(b) Confusion matrix for Logistic Regression over data reduced with t-SNE.

$PCA : 0.857$	Accuracy Score	$t - SNE :$ 0.67
$PCA : 0.985$	ROC Score	$t - SNE :$ 0.946

These results are surprisingly good, and the version with PCA reduced dimensions performed almost as well as SVM.