# Fashion MNIST Dataset Analysis

Fundamentals of Data Science Final project

Author: Andrzej Świętek

# Table of Figures



# Table of Contents



# Index of Tables

# Introduction

The Fashion MNIST dataset consists of 60,000 training samples and 10,000 test samples, each with 784 features representing pixel values (28x28 images) and one label indicating the class of the fashion item.

In order to identify the coordinates of a pixel in an image, consider expressing the pixel index, denoted as x as $x := i * 28 + j$ where $i, j \in [0, 27]$ . This representation indicates that the pixel is positioned at the intersection of the i-th row and the j-th column within a matrix of size 28 x 28.
For instance, when referring to pixel 31, it signifies the pixel's location in the fourth column from the left and the second row from the top, maintaining a clear access.

The dataset contains 10 classes, each associated with a specific fashion category.

| Label | Category |
|---|---|
| 0 | T-shirt / top |
| 1 | Trousers |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneakers |
| 8 | Bag |
| 9 | Ankle boot |

*Table 1: Classes labels*

# Brief dataset summary

The analysis yielded intriguing results through the computation of the average image for each clothing class. This method provides valuable insights into the common pixel areas shared among items within the same class, highlighting the typical variations within a class and distinctions between classes.

Notably, T-shirts, pullovers, coats, and shirts exhibit similarities, suggesting potential challenges in clustering or potential inaccuracies since they share significant common area on the average images.

Conversely, it is evident that bags, dresses, and sandals display significant differences both within their respective classes and compared to other types of clothing. This dissimilarity assures that classification challenges may be minimized, even though the commonality in shoe soles forms a recognizable pattern across the three types of shoes. The unique pixels specific to each clothing piece contribute significantly to the classification process

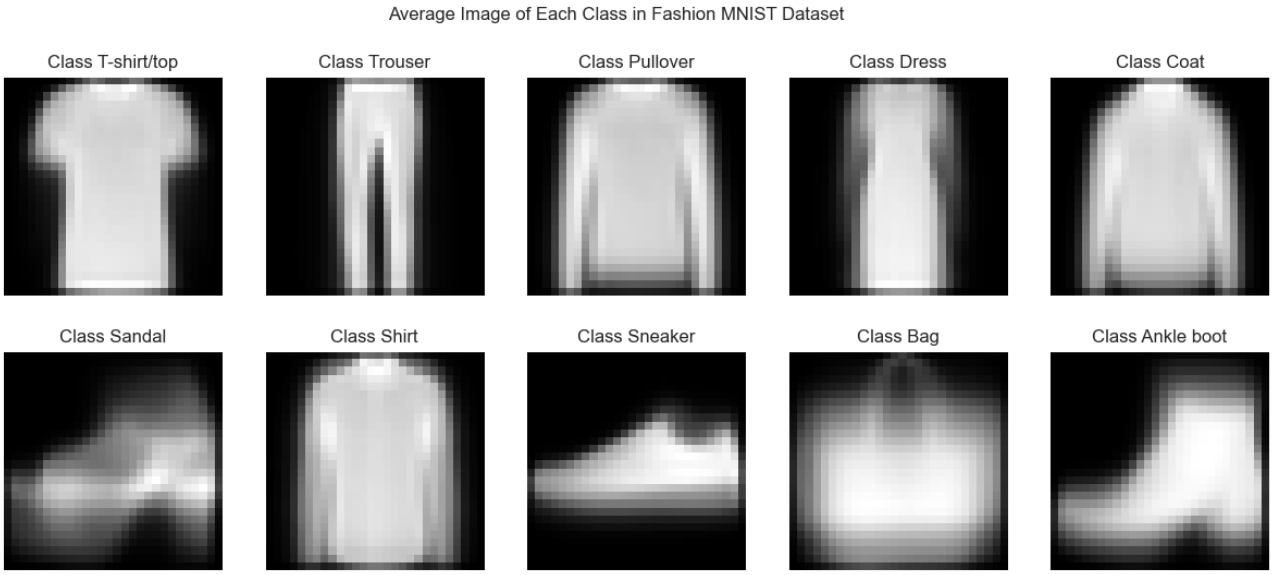



*Figure 1: Average clothes images*

# Classes distribution

The dataset is well-balanced, containing an equal number of elements (6000) for each class. This balance ensures that every piece of clothing is equally represented in the dataset, with each class having the same number of rows.

This information is particularly valuable as it indicates that randomly selecting an element from a given class is equally probable as selecting from any other class.

Therefore, when dealing with random selection of elements, no additional steps need to be taken to account for class imbalances.

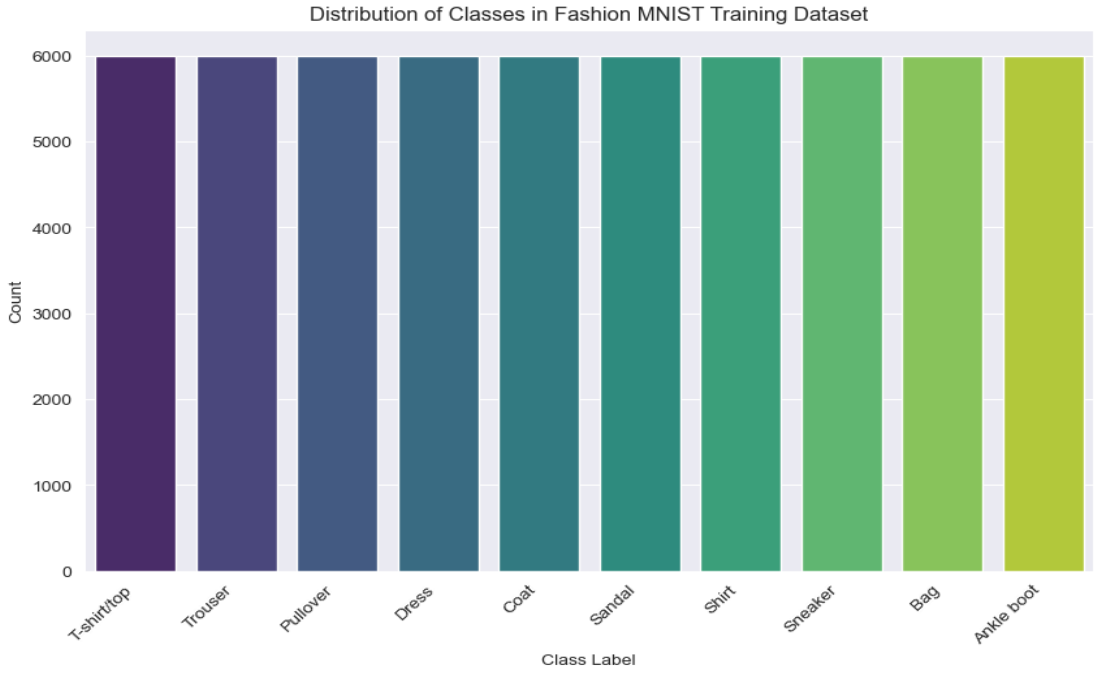



*Figure 2: Chart presenting equal distribution of each class in the dataset*

# Data Scaling and dimensionality reduction

In the customary practice of dataset analysis, it is advisable to employ scaling and normalization techniques. To achieve this, I partitioned the dataset into two components: X and Y. Here, X represents the feature matrix, excluding the label columns (numeric label and corresponding label name), while Y solely comprises the label feature column.

The normalization and scaling procedures were executed using the sklearn library, incorporating the StandardScaler object. The data was first fitted to this scaler, and subsequently, the normalize function was applied.

Normalization formula:

$$X_{normalized} = \frac{X - \mu}{\sigma}$$

Scaling: Scaling involves transforming the data to a standardized range, ensuring that each feature contributes proportionally to the model. It helps prevent certain features from dominating due to differences in their scales, thereby enhancing the upcoming models' performance.

Scaling formula:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

The result of these operations can be seen under the section "Data Scaling & Normalization" in fashion-mnist.ipynb file.

As a next step of analyzing provided dataset I took dimensionality reduction which is meant to reduce the number of input variables or features in a dataset while preserving its essential information. The primary goal is to simplify the dataset, making it more manageable and computationally efficient, without losing critical patterns or trends.

First of all, it is crucial to find out how many dimensions the dataset can be reduced to while still retaining 99.7% of the overall information regarding regarding to $3\sigma$ rule – here applied with $2\sigma$ equal to doubled standard divination retaining 95% of information. In order to accomplish this, I utilized PCA from sklearn library with the n_components parameter set to 'mle' and calculated the cumulative sum of its explained variance.

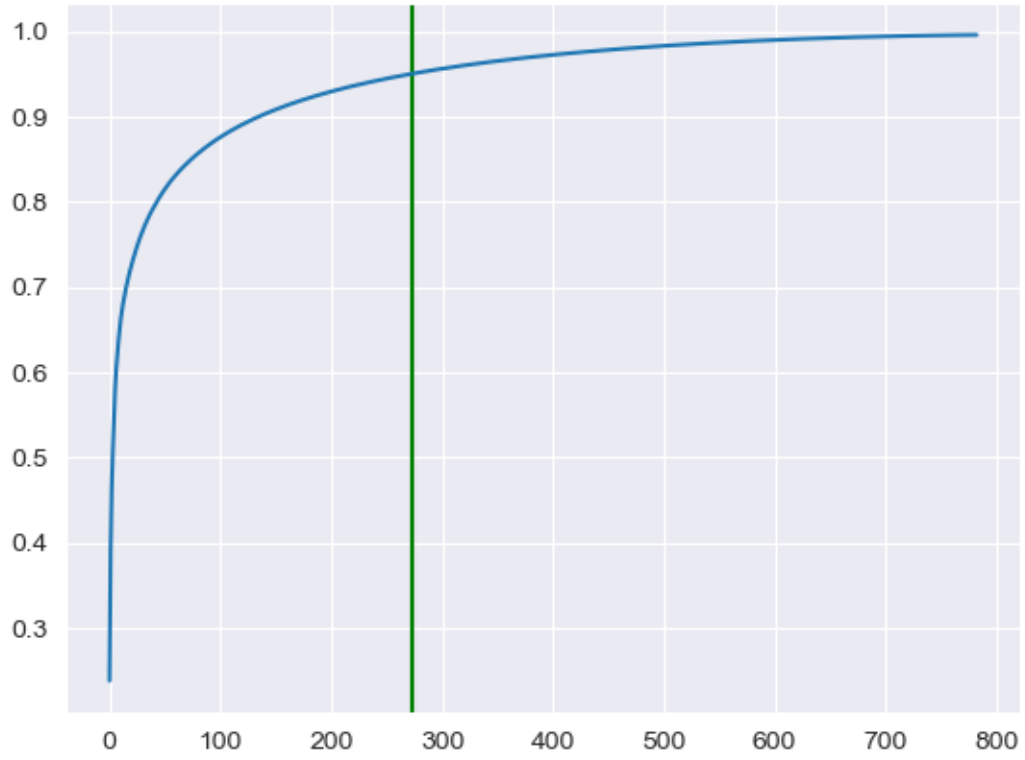



*Figure 3: Explained Variance Cumulative Sum Chart*

The outcome of the analysis produced 273 features that must remain in order to keep 95% of all information. The cumulative sum of explained variance grows rapidly as more components are considered up to the point od around 300 where slope becomes much more flat.

As a next step, I employed Principal Component Analysis (PCA) and t-SNE to reduce the dimensionality of the Fashion MNIST dataset to 2 and 3 dimensions. This reduction facilitated visual representation, offering insights into the proximity of different classes and their shared spatial relationships. This approach not only aids in comprehending the dataset's structure but also provides a meaningful visualization of class distributions within the reduced feature space.

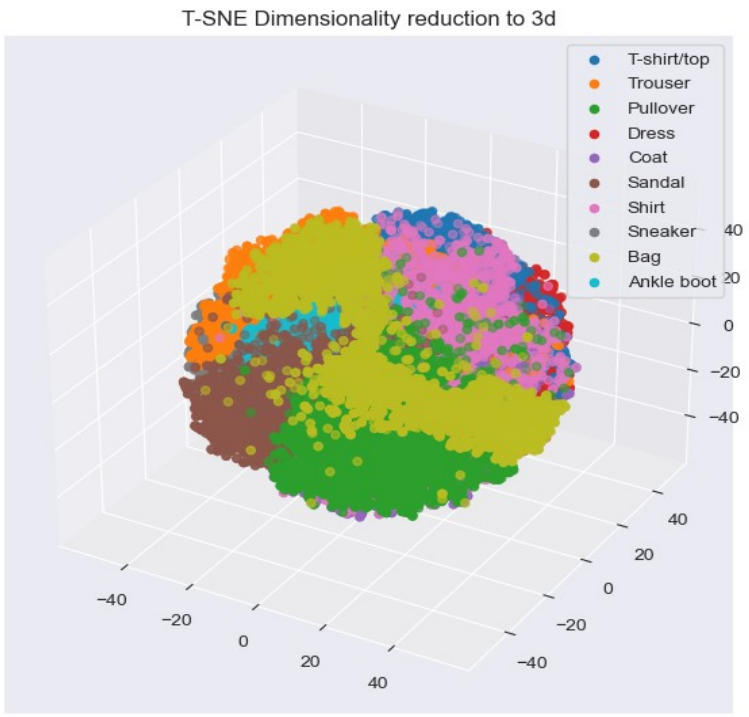



*Figure 5: T-SNE 3D*

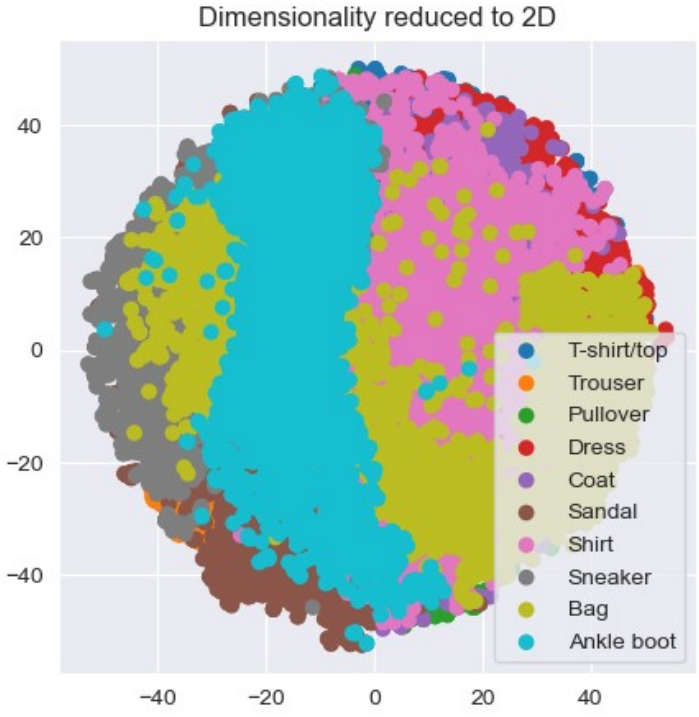



*Figure 4: T-SNE 2D*

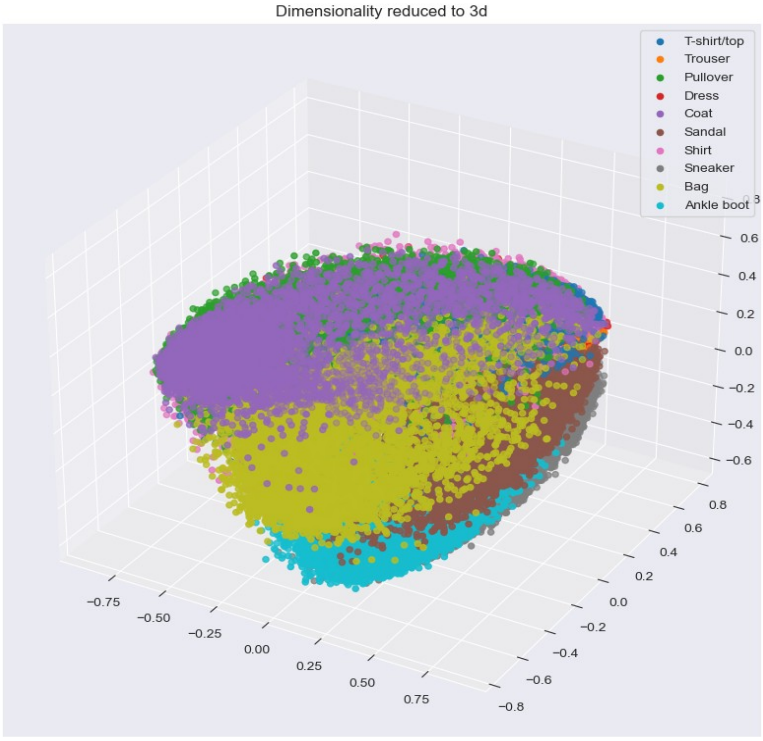



*Figure 6: PCA 3D*


*Figure 7: PCA 2D*

The differences in results between PCA (Principal Component Analysis) and t-SNE (t-distributed Stochastic Neighbor Embedding) dimensionality reduction techniques can be attributed to their underlying algorithms and objectives.

1. PCA (Principal Component Analysis):

Objective: PCA aims to maximize the variance along the principal components, capturing the most significant information in the dataset.

How it works: PCA identifies orthogonal axes (principal components) in the data space and projects the data onto these components. The first few principal components retain the most variance in the data.

Resulting Differences: PCA is effective for linear relationships and is particularly useful for capturing global structure, but it might not preserve local relationships well.

2. t-SNE (t-distributed Stochastic Neighbor Embedding):

Objective: t-SNE is designed to emphasize the local relationships between data points, preserving the pairwise similarities as much as possible.

How it works: t-SNE constructs probability distributions over pairs of high-dimensional points and seeks to minimize the divergence between these distributions in the high-dimensional and low-dimensional spaces. It focuses on retaining relative distances between neighboring points.

Resulting Differences: t-SNE is effective for visualizing clusters and preserving the local structure of the data. It tends to group similar instances together in the lower-dimensional space.

As an interpretation of these results, we can conclude that elements belonging to the same class are mostly concentrated around a common center for each class, with a few exceptions represented by outliers. Another notable observation is the overlapping of certain classes, making it clear to which group of clothes they belong.

# Clustering

In case of data set so dense like that it is expected for clustering algorithms to encounter difficulties with aggregating to the same cluster elements of the same class however clusters and the percentage of included cloth types in them represent certain relation ship between the appearance of those types.

Firstly I used 3 different approaches of getting right clusters: Agglomerate Clustering, Kmeans, DBSCAN

Agglomerative Clustering operates by progressively combining the most akin data points or clusters until a designated number of clusters is achieved, and its computational cost is heightened in contrast to alternative methods. This is particularly pronounced in my case, given the comprehensive utilization of the entire dataset rather than just a subset, and the execution across all linkage methods for the Euclidean metric.

A dendrogram in clustering visually represents the hierarchical relationships between data points or clusters in a dataset.

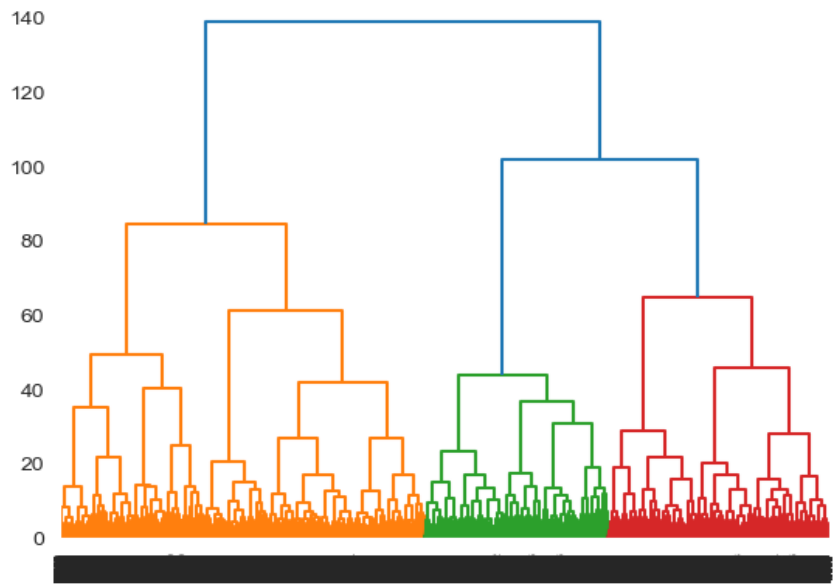



*Figure 8: Clusters Dendrogram*

After that I used Kmeans algorithm with various parameters however the best one is when used scaled dataset with reduced dimensionalities. KMeans clustering works by iteratively assigning data points to the nearest cluster center and then updating the cluster centers based on the mean of the assigned points. This process repeats until convergence, resulting in clusters that minimize the within-cluster sum of squared distances.

Eventually I counted what classes of clothes are assigned to given cluster. When compared with previous prediction I found that my initial predictions were largely accurate..

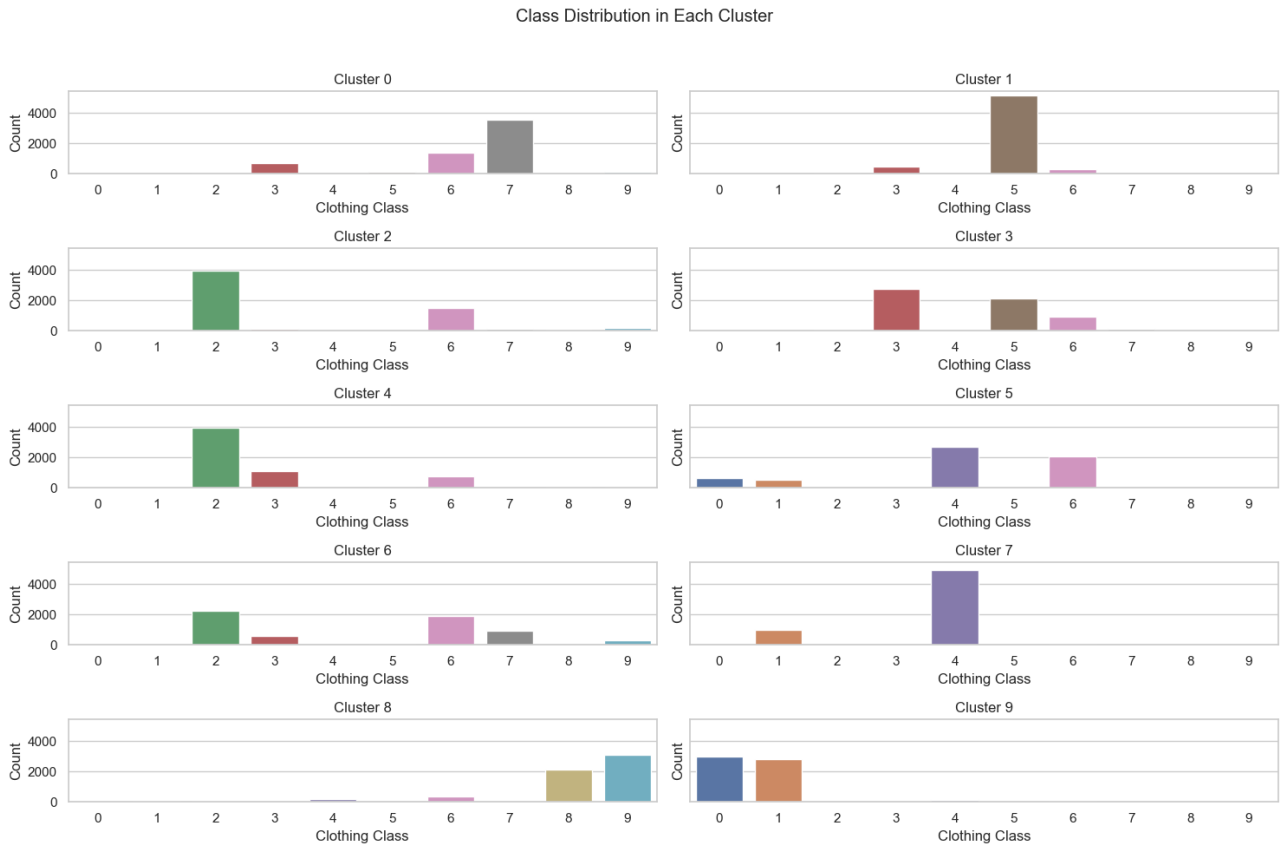



*Figure 9: Clusters*

In the end we receive disappointing Adjusted Rand Score equal to 0.3623

Event though clustering has rather small chance of getting cluster class-accurate, because of geometrically collecting pixels sequences  in attempt to get the closer result I employed simple genetic algorithms in order to find right parameters for DBSCAN – so that it matched number of cluster to number of classes. However the results were not to successful because of probably incorrect parameters or because due to fitness function converging to slow. The whole reason for that search is the fact that DBSCAN found 32 clusters but most of elements were assigned to first 2 as section “DBSCAN CLUSTERING With Reduced dimensionality” in Jupyter Notebook presents.

Unsatisfied with the results and suspicious of previous methods of clustering spherical manner I tried also other types of clustering that do not involve spherical cluster growth such as: Gaussian Mixture Models (GMM), Affinity Propagation and BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies).

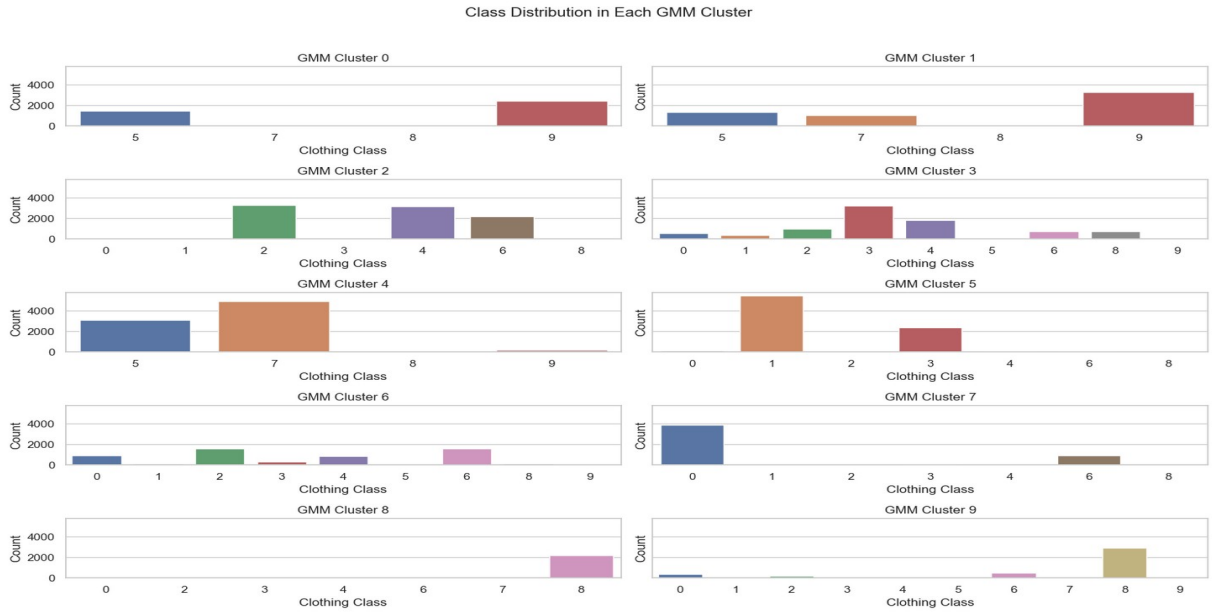



*Figure 10: GMM*

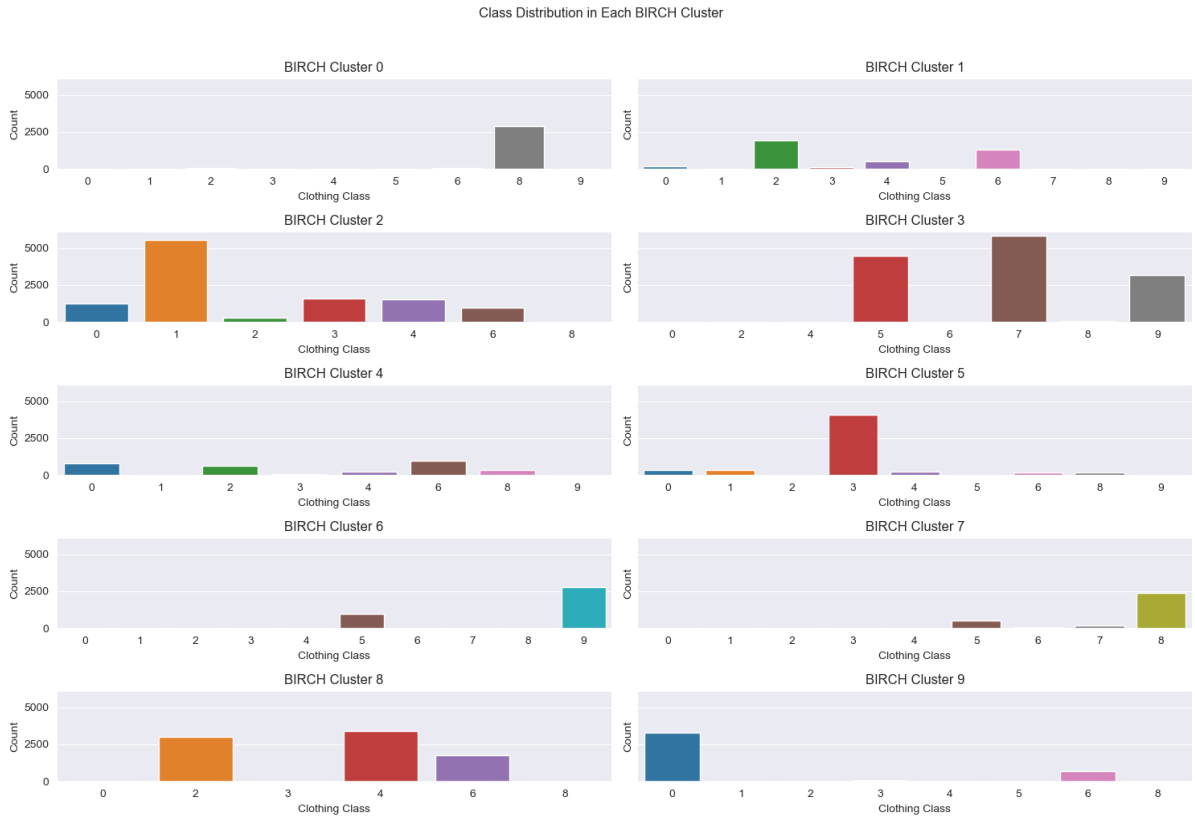



*Figure 11: Birch*

# Classification

Fashion-classification.ipynb

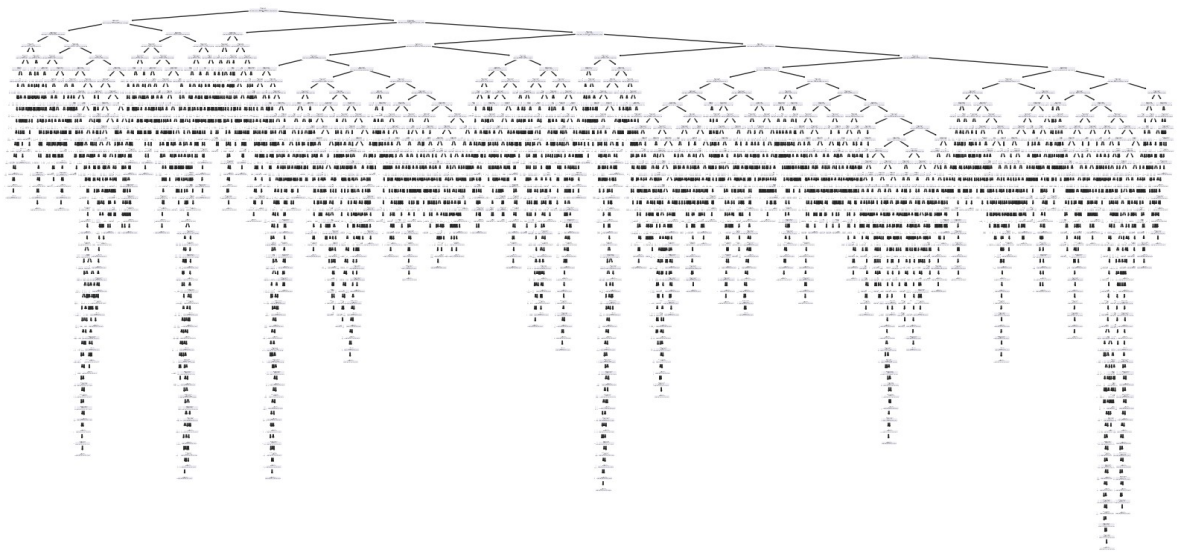
In all classification techniques I use parameter n_job=-1 that is responsible for using 100% of CPU which makes required calculations in much shorter time.

My first approach to classification was utilizing K Neighbors Classifier along with K Folds.

| $N^{th}$ Fold | Score |
|---|---|
| 0 | 0.863 |
| 1 | 0.864 |
| 2 | 0.862 |
| 3 | 0.857 |
| 4 | 0.860 |

*Table 2: KNN Scores*

Subsequently I trained decision tree on full dataset however the results where much lower than KNN Classifier: 0.79; Whereas it is impossible to see actual conditions in the tree we can see the over all shape.



As always when dealing with Decision Tree a next step is almost always Random Forest algorithm which is expected to perform better.

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. It introduces randomness by training each tree on a subset of the data and using a random subset of features for making split decisions, reducing overfitting and improving overall predictive performance.

Utilizing this method I received .88 accuracy which is substantially better that just a single tree.

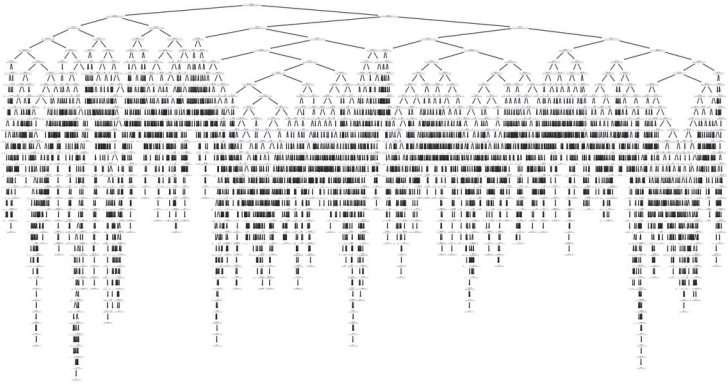


*Figure 12: Random Forest one of the trees*

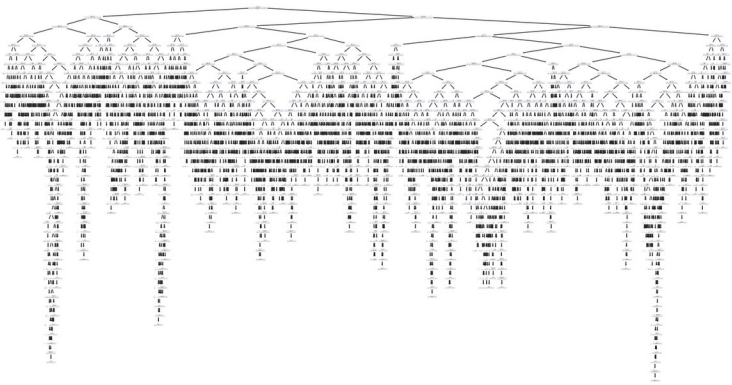


*Figure 13: Random Forest other of the trees*

Having plotted some of the trees I noticed some similarities in a few areas however the characteristic longest branches are placed in different places.

As for the last classification algorithm I used XGboost that works in in a slightly different way for 2 previous algorithms. XGBoost (Extreme Gradient Boosting) is a powerful ensemble learning algorithm that sequentially builds a series of weak learners, usually decision trees, and combines their predictions to create a strong predictive model. It minimizes a loss function by iteratively adding trees, with each subsequent tree focusing on the errors of the previous ones, resulting in a robust and accurate model.

With this method the classification score was by far the best exceeding .90;

In my quest to enhance the classification model, I explored training a convolutional neural network (CNN) using Keras. Initial attempts yielded a decent accuracy of 0.81 on the test set. Subsequently, I employed Keras Tuner to fine-tune model parameters, achieving an improved accuracy of 0.8665 with a batch size of 32 and 10 epochs.

Wheres this result is quite satisfying my trained model is yet another great source of information. I striped it of its last layer and fitted data though it. As a result of that I have received image embeddings based on which NN is classifying given image. Subsequently

I had conducted basic analysis in order to observe the main differences between Neural Network and classical approach.

I followed my exact steps of dealing with dataset. I reduced dimensionality using t-SNE to 3 and 2  and then plotted results.

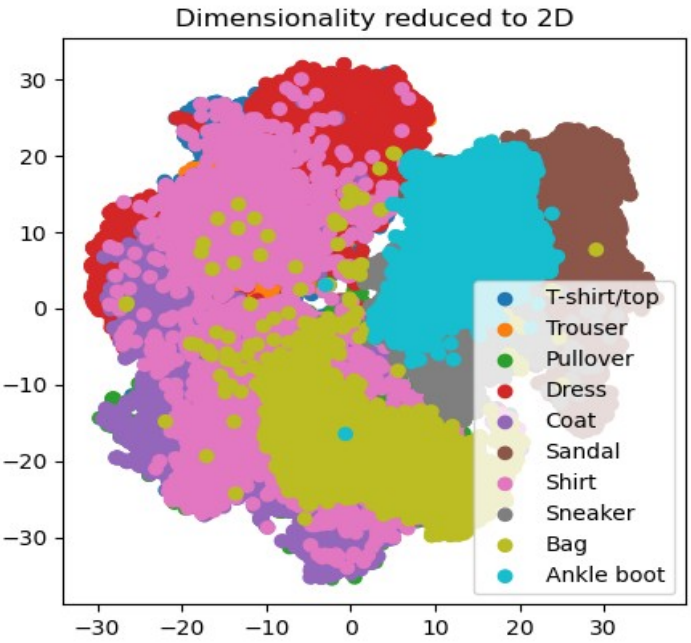



*Figure 14: NN Embeddings 2D*

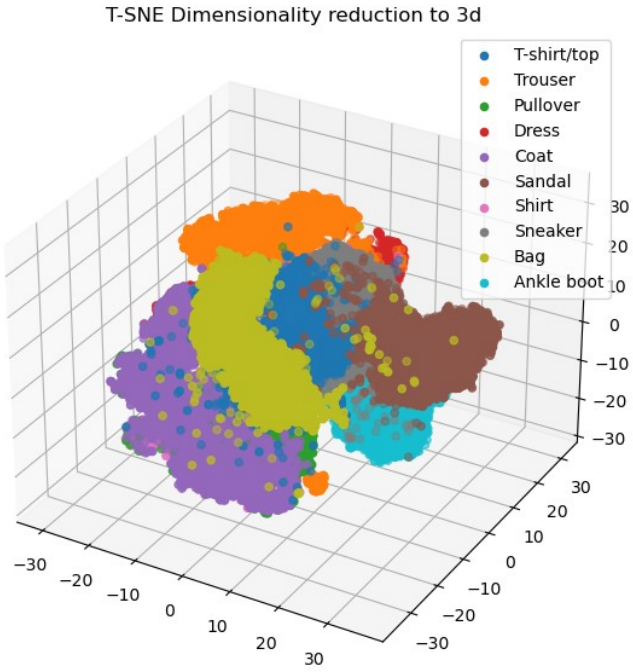



*Figure 15: NN embeddings 3D*

At the very end I clustered embeddings using Kmeans. The result are much more “class accurate” than default approached.

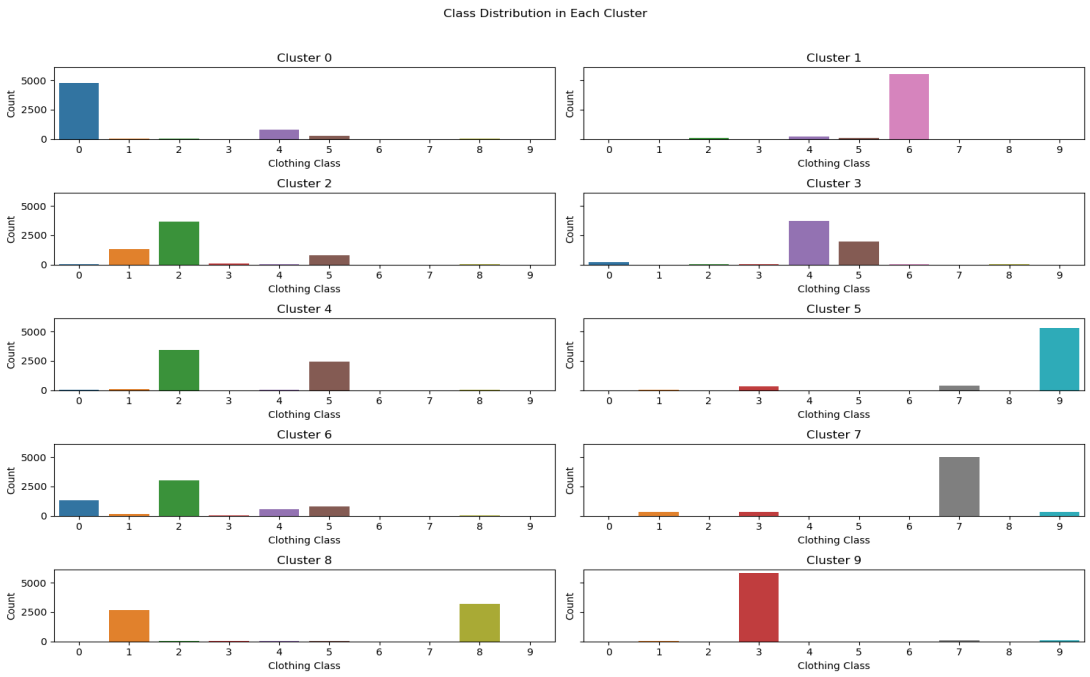



*Figure 16: NN embeddings clustering with Kmeans*

# GPT

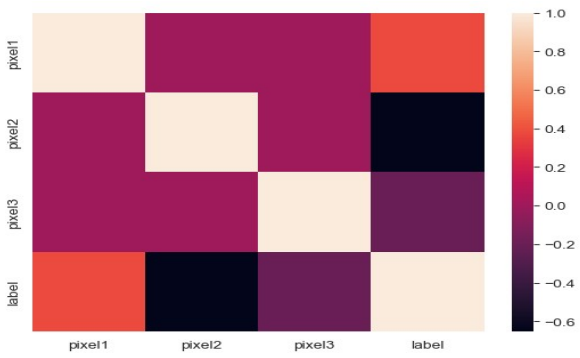
Throughout the project, I followed my own ideas and knowledge, cross-referencing them with the results and methods suggested by ChatGPT. It proved quite helpful in some areas, while in others, it struggled to provide useful insights.

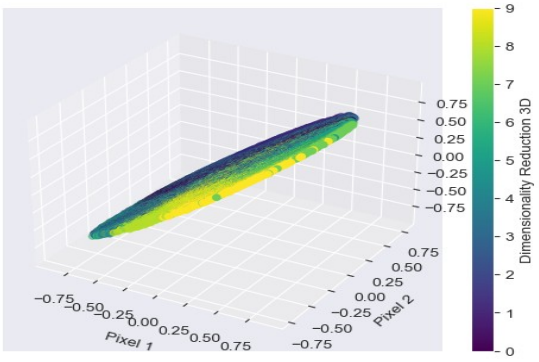
## 1. Brief data summary

ChatGPT's contributions in this area were not particularly creative. It suggested methods involving Pandas objects and recommended comparing the class distribution across the entire dataset. Essentially, it reiterated similar steps to what we covered in the initial laboratories.

## 2. Scaling and dimensionality reduction.

During the dimensionality reduction using PCA, ChatGPT offered a valuable suggestion to plot a heatmap of pixels. However, when it came to actual plotting, it omitted grouping the dataset by label, leading to results that were not aligned with expectations. This aspect required correction on my part.




When asking it for code snippets for t-SNE and PCA it gave basic implementation from sklearn  documentation.

Additionally it mentioned LDA algorithm that did not appear on the lectures which is a supervised technique that seeks to maximize the separation between classes while minimizing the variance within each class.

## 3. Clustering

This time, ChatGPT proved highly beneficial by not only offering a diverse set of clustering algorithms but also providing concise explanations of their mechanisms. One noteworthy instance was when I sought insights into non-spherical algorithms, and ChatGPT's response included valuable information that seamlessly found its way into my project.

Suggestions: DBSCAN, Aglommerative Clustering, Density Peak Clustering and Affinity Propagation – the last one was computationally to complex to conduct. Cluster plots also were terrible.

Also when asked it could not find right parameters for dbscan.

## 4. Classification

Regarding classification chat gpt recommend multiple algorithms such as:

Logistic Regression, Decision Trees, Random Forests, SVM, k-NN, LDA and Neural Networks. Also I didn't remember how to use xgboost and so it provided decent implementation – that I modified though.

It was also helpful debugging my KerasTuner for Neural Network – after 2 attempts it found the issue and suggested solution – it was hard to spot because it was a minor detail.

## Summary

In the majority of instances, ChatGPT proved to be a valuable resource for my project. However, its effectiveness necessitated continuous validation to ensure the accuracy of results. This required my vigilance and a thorough understanding of the tasks at hand. Initially, it was less proficient in assisting with plotting until I provided specific directions, particularly in cases requiring result grouping during clustering. Despite these challenges, ChatGPT proved highly beneficial, especially in the context of debugging, ultimately resulting in significant time savings.