Zalando Clothing Classificaton

December 31, 2022

Authors:

Kristian Graven Hansen (krgh@itu.dk)

Lukas Sarka (lsar@itu.dk)

Mikkel Geisler (mgei@itu.dk)

1 Introduction

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum. Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum. Lorem Ipsum is simply dummy text of the printing and typesetting industry.

2 Data and Preprocessing

2.1 Dataset

The Fashion-MNIST is a dataset of Zalando article images, consisting of a training set of 60,000 samples and 10,000 test samples. The dataset used here is a small portion of the original dataset, 10,000 training and 5,000 test samples. Each sample is a 28x28 pixels gray-scale image with a label indicating the type of clothing item associated with each.

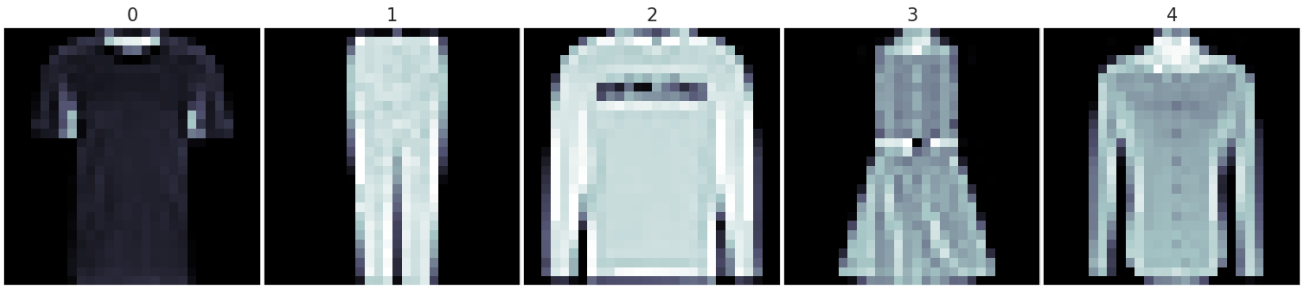

Figure 1: One sample from each class (reconstructing images from pixels)

2.2 Naming Conventions

The exploration of samples within each class gave rise to more appropriate names. Class 0 became T-shirt etc. Below is the naming conventions, which will be used throughout the report for better readability.

Label	0	1	2	3	4
Type of clothing	T-shirt/Top	Trousers	Pullover	Dress	Shirt

Table 1: Mapping from class-labels to clothing type

2.3 Data Cleaning

The dataset provided was already in a cleaned state, which was verified by checking for missing values, and checking that the pixel values were no greater than 255 and no smaller than 0.

2.4 Preprocessing

The pixels values were in the range $[0, 255]$. This range was normalized to $[0, 1]$ by dividing each pixel by 255. This was done to improve training time, and because neural networks prefers to work with small numbers.

2.5 Class Distribution

Whether or not a machine learning model can learn to predict classes well depends to a high degree on how those classes are distributed within our training and training dataset. Both the datasets are extremely balanced, with the test being fully balanced (1000 of each class). This is illustrated on the plots below

Figure 2: Distribution of clothing items in our training and test dataset

3 Exploratory Data Analysis

One simple yet useful way to explore a dataset is to visualize the feature distribution for each class. In our situation this is simply not possible and not very insightful as we have too many features. To be precise $728 = 28 \times 28$ features/pixels. This doesn't mean the dataset can't be visualized, which we will discuss in the next section.

3.1 Principal Component Analysis

As mentioned before due to our large dataset it is hard to visualize feature distributions, this is where principal component analysis or PCA can be used instead. Principal Component Analysis (PCA) is

a dimensionality reduction technique used to reduce the number of features in a dataset, while still preserving the most important information. It does this by transforming variables into a new set of variables, called principal components, which are uncorrelated from each other and explain the maximum amount of variance in the data. Below we explore the relationships between the first 3 principal components. These 3 principal components represent the direction of most variance in the original data

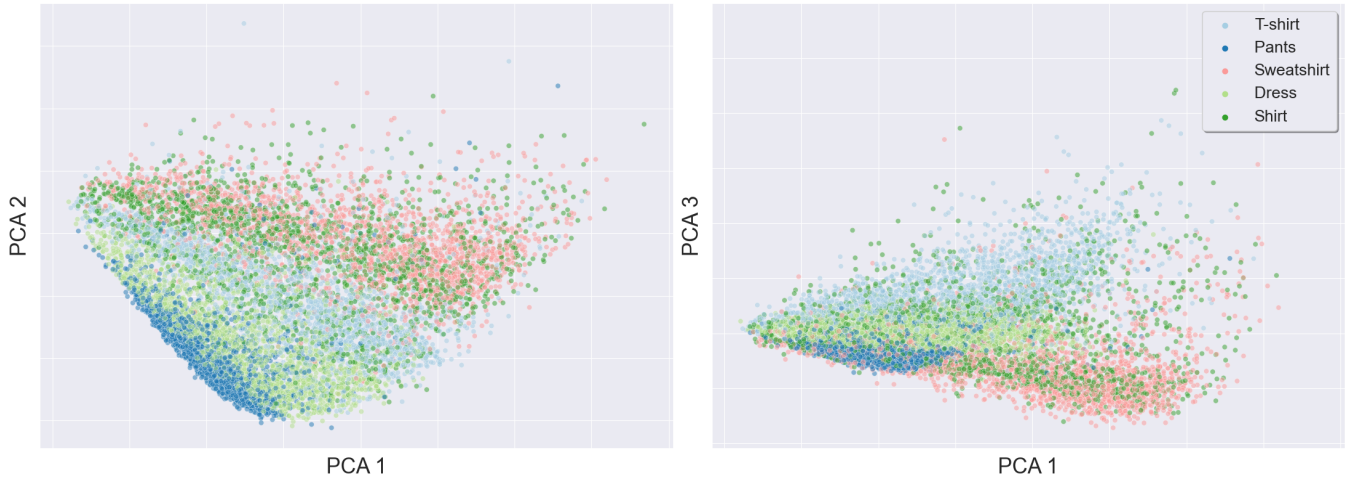

Figure 3: Visualizing relationships between first 3 PCA's

4 Decision Tree

In this section we will discuss the implementation and results of the `DecisionTreeClassifier`. A decision tree classifier is a supervised machine learning model that can be used for regressing or classification. It works by building a tree-like structure, where at each internal node, the algorithm considers all the available features and chooses the split that maximizes the purity of the resulting splits. This process is repeated until a certain stop condition is reached, such as a maximum depth or no further improvement in purity. Decision tree classifiers are popular because they are easy to understand and interpret due to their tree-like structure, and they can handle different data types and types of decision boundaries.

4.1 Implementation

The implementation was done in python using two classes:

1. `Node()`
2. `DecisionTreeClassifier()`

4.1.1 Node

The `Node()` class has the most responsibility of the two. It holds the data and passes it down the tree constantly splitting itself into more nodes with the `_split()` method. This is done by finding

the best split with the `_get_best_split()` method based on the impurity measure specified in the `DecisionTreeClassifier()`. The best split refers to the feature and cutoff-value that leads to the highest gain in purity, meaning the more the classes are seperated in the nodes the better. After finding the best split, it can then create two new nodes, `self.left` and `self.right` if all split criteria are fulfilled. These criteria implemented are max_depth and min_samples_split which and are discussed further in the `DecisionTreeClassifier()` implementation. This process then happens recursively in the `fit()` method in the `DecisionTreeClassifier()`, and is how the tree-structure is build.

4.1.2 DecisionTreeClassifier

The `DecisionTreeClassifier()` class is the classifier interface which has the `fit()` and `predict()` methods and some 'less' important methods `get_depth()` and `get_n_leaves()`, that can be called after the model has been fitted. Our implementation allow for specification of the following 4 parameters.

1. `max_depth` – specify maximum depth to which the tree can grow
2. `min_samples_split` – specify minimum number of samples in a node before it can be split
3. `criterion` – split based on either gini or entropy
4. `random_state` – set random seed for reproducible results

4.2 Hyperparameter