AGH University of Science and Technology
Faculty of Physics and Applied Computer Science
Field of Study: Applied Computer Science

Fundamentals of Data Science

Project

Analysis and Classification of Fashion MNIST Data

---

author
Przemysław Ryś

Krakow, 26 January 2024

1 Introduction

The Fashion MNIST dataset is essentially a curated collection of grayscale images representing various clothing items and accessories. Unlike the original MNIST dataset, which consists of handwritten digits, Fashion MNIST offers a more diverse and challenging set of images to work with. The data structure of image information is usually represented in the form of pixels (this is not the case, for example, in vector graphics). Pixels are the basic units of data in digital images and constitute individual points on the screen. Each pixel contains information about the color and/or brightness of a given point in the image. Each image in the data set is a grayscale representation of 28x28 pixels, for this reason the characterization of a pixel comes down to specifying only one number. Where the limits of this scale are the value 0 meaning white, and the value 255 meaning black. Each of them represents clothing items such as shirts, dresses, trousers, shoes, bags, and more.

The dataset is split into two main subsets: a training set containing 60,000 examples and a test set with 10,000 examples. This division allows to train and evaluate machine learning models effectively. The images in Fashion MNIST are associated with one of 10 possible classes, making it a multi-class classification problem.

Each image is composed of 28 pixels in height and 28 pixels in width, resulting in a total of 784 pixels. Each pixel is assigned a single value representing its lightness or darkness, with higher values indicating darker shades, ranging from 0 to 255 as integers. Both the training and test datasets consist of 785 columns. The first column contains the class labels, representing the type of clothing item (see Table 1 for label meanings). The subsequent columns hold the pixel-values associated with each image.

It is widely used as a benchmark dataset in the field of computer vision and pattern recognition. It serves as a standard testbed for evaluating the performance of various machine learning algorithms, including classification, clustering, and dimensionality reduction techniques. This dataset provides a convenient environment for exploring and experimenting with different machine learning techniques without the need for extensive data preprocessing or computational resources.

Zalando Fashion MNIST Repository can be found here.

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

Tab. 1: Description of Labels

2 Import of necessary libraries and data

The Python code above imports the necessary libraries for data analysis and machine learning tasks. It includes libraries such as Pandas for data manipulation, NumPy for numerical computations, Matplotlib for data visualization, and scikit-learn for machine learning algorithms.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

Listing 1: Code for importing libraries

Additionally, the following code segment imports the Fashion MNIST dataset from CSV files and separates the features (X) from the labels (y):

```
# Import training and test data from CSV files
df_train = pd.read_csv('fashion-mnist_train.csv')
df_test = pd.read_csv('fashion-mnist_test.csv')

# Split data into features (X) and labels (y)
X_train_csv, y_train_csv = df_train.drop('label', axis=1).values,
   df_train['label'].values
X_test_csv, y_test_csv = df_test.drop('label', axis=1).values,
   df_test['label'].values
```

Listing 2: Code for importing and preparing the Fashion MNIST dataset

3 Summary of the Data

This code segment imports the Fashion MNIST training and test datasets from CSV files and prepares them for further analysis:

```
# Import training and test data from CSV files
df_train = pd.read_csv('fashion-mnist_train.csv')
df_test = pd.read_csv('fashion-mnist_test.csv')

# Split data into features (X) and labels (y)
X_train_csv, y_train_csv = df_train.drop('label', axis=1).values,
   df_train['label'].values
X_test_csv, y_test_csv = df_test.drop('label', axis=1).values,
   df_test['label'].values
```

Listing 3: Code importing and preparing Fashion MNIST dataset

The `pd.read_csv` function is used to read the Fashion MNIST training and test datasets from CSV files named 'fashion-mnist_train.csv' and 'fashion-mnist_test.csv', respectively. Next, the datasets are split into features (X) and labels (y). For the training dataset (`df_train`), the `drop` function is applied along the 'label' column axis to separate the features (`X_train_csv`) from the labels (`y_train_csv`). Similarly, for the test dataset (`df_test`), the features are stored in `X_test_csv`, and the labels are stored in `y_test_csv`. A visualization of the dataset is provided in Figure 2. This visualization depicts the entire dataset with cluster assignments, representing the distribution of clothing item images in a reduced feature space. Each point on the plot corresponds to an image sample, with colors indicating different cluster assignments. This visualization aids in understanding the clustering tendencies within the dataset, facilitating the initial exploration and analysis.

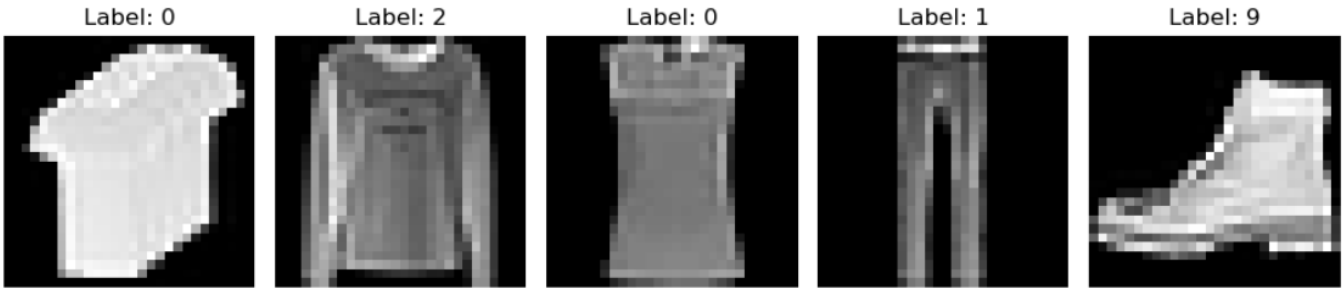

Fig. 1: Sample of the Fashion MNIST dataset with cluster assignments

```
# Calculate accuracy for training predictions
accuracy_train = accuracy_score(y_train_csv, y_pred_train)

# Display accuracy of the training set only
print(f"Accuracy on Training Set: {accuracy_train * 100:.2f}%")
```

Listing 4: Code calculating accuracy for training predictions

The consistency between the testing data and the training data is 10.07%.

4 Reduce Data Dimensionality

The following code snippet demonstrates the application of Principal Component Analysis (PCA) to reduce the dimensionality of the Fashion MNIST dataset. PCA is a fundamental technique used in machine learning and data analysis for reducing the dimensionality of high-dimensional datasets while preserving most of their variance.

In simple terms, PCA works by transforming the original features of a dataset into a new set of orthogonal features, called principal components. These principal components are ordered in terms of the amount of variance they capture in the data, with the first component capturing the most variance, the second component capturing the second most, and so on. By retaining only a subset of these principal components, PCA can effectively reduce the dimensionality of the dataset while retaining most of its information.

Initially, the dataset is limited to 10,000 examples using the train_test_split function. This step is crucial for effectively managing computational resources, especially when dealing with large datasets.

PCA is then applied to reduce the dimensionality of the dataset to 2 components. This transformation allows the retention of essential information while reducing redundancy, facilitating subsequent analysis tasks and improving model performance.

The PCA model is fitted to the training subset (X_train), and the transformation is applied to both the training and test datasets (X_train_reduced and X_test_reduced). This ensures consistency in the dimensionality reduction process across different subsets of the data.

Finally, the shapes of the reduced datasets are printed to confirm successful dimensionality reduction. The output indicates that both the training and test datasets now have 2 features instead of the original number of features.

The following Python code demonstrates the dimensionality reduction process using PCA:

```
# Perform dimensionality reduction
pca = PCA(n_components=2)
X_train_reduced = pca.fit_transform(X_train)
X_test_reduced = pca.transform(X_test)
```

Listing 5: Code performing dimensionality reduction using PCA

5 Reduce Data Dimensionality

This code generates a scatter plot where each class of data is represented by points of different colors. The x and y axes represent the first two principal components obtained through dimensionality reduction using PCA. Each class of data is distinctly marked on the plot, and the legend identifies individual classes. This visualization allows us to observe how the data is dispersed in the new two-dimensional space after applying PCA.

```
# Plot the reduced data
plt.figure(figsize=(10, 6))
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap='
    viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Fashion MNIST Dataset with PCA Dimensionality Reduction')
plt.colorbar(label='Class')
plt.show()
```

Listing 6: Code for visualizing the reduced dataset

Fig. 2: Plot the reduced data

This code aims to visualize unique classes within a reduced dataset using scatter plots. It begins by identifying the unique classes present in the dataset. Then, it determines the layout of subplots based on the number of unique classes, allocating three columns per row. Subsequently, the code iterates over each unique class, selecting data points corresponding to that class from the reduced feature space. For each class, a scatter plot is generated, depicting the data points in a two-dimensional space defined by the first two principal components. Each subplot represents a different class, with data points colored according to their class label. Finally, the generated scatter plot is saved as an image file named 'scatter_plot_pca.png', and displayed using Matplotlib's show function.

```
# Find unique classes in the data
classes = np.unique(y_train)

# Calculate number of rows needed for subplots
num_rows = (len(classes) + 2) // 3  # Adding 2 for rounding up to the
    next integer

# Create subplots for each class in three columns
fig, axes = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5 *
   num_rows))

# Iterate over unique classes and create scatter plots for each class
for i, cls in enumerate(classes):
    # Get indices of samples for the given class
    indices = np.where(y_train == cls)[0]
    # Select data for the given class
    X_class = X_train_pca[indices]
    # Calculate subplot indices
    row = i // 3
    col = i % 3
    # Create scatter plot for the given class
    axes[row, col].scatter(X_class[:, 0], X_class[:, 1], cmap='
   viridis', label=f'Class {cls}')
    axes[row, col].set_title(f'Class {cls}')
    axes[row, col].set_xlabel('Principal Component 1')
    axes[row, col].set_ylabel('Principal Component 2')
    axes[row, col].legend()

plt.tight_layout()

# Save the plot to a file
plt.savefig('scatter_plot_classes_pca.png')

plt.show()
```

Listing 7: Code for visualizing unique classes in the reduced dataset

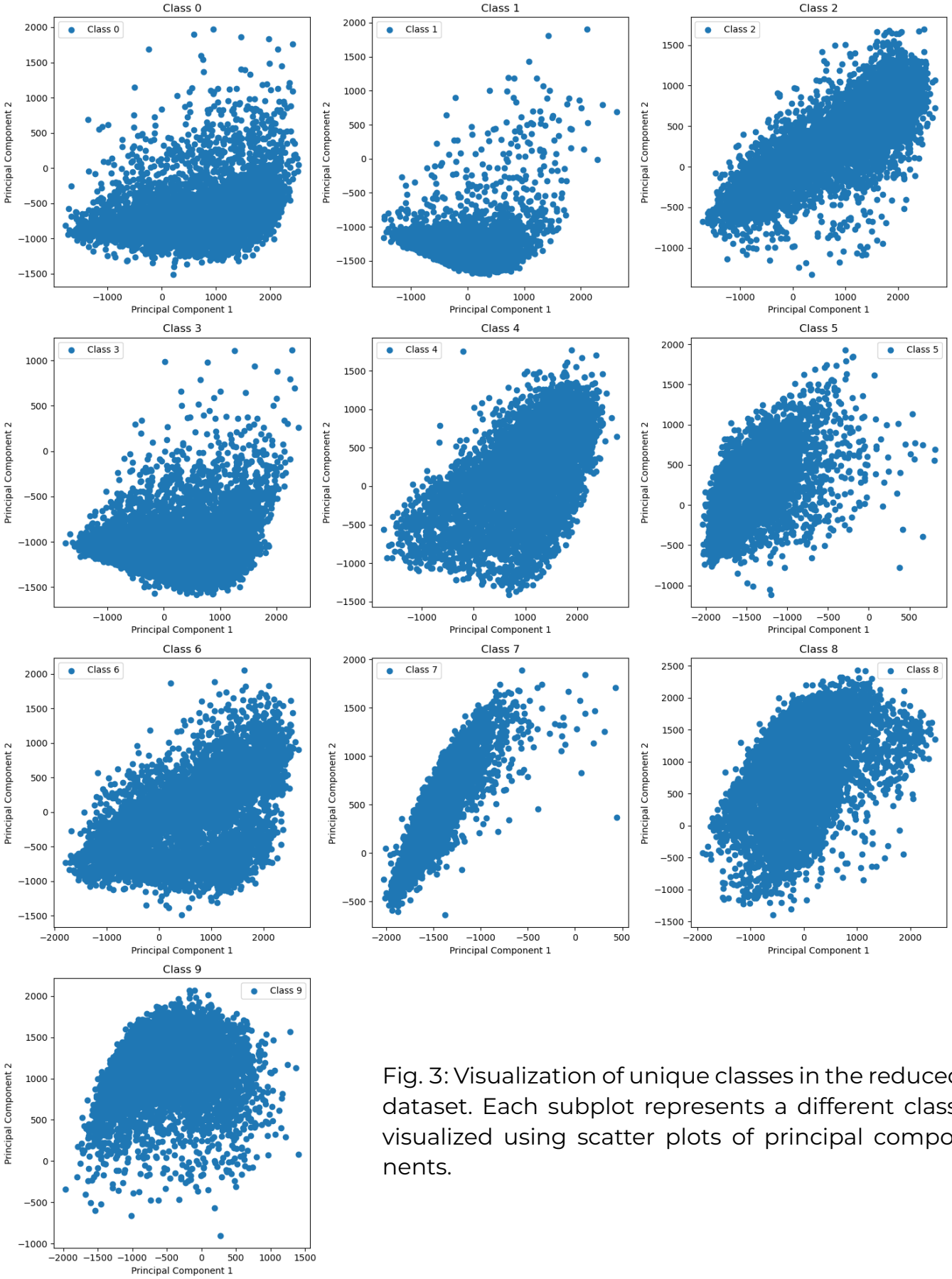

Fig. 3: Visualization of unique classes in the reduced dataset. Each subplot represents a different class, visualized using scatter plots of principal components.

6 Clustering and Evaluation

The provided code segment clusters the dataset using the KMeans algorithm with 10 clusters and evaluates the clustering results using the Adjusted Rand Index (ARI).

```
# Cluster the dataset (and evaluate clustering results with
   classification labels)
kmeans = KMeans(n_clusters=10)

# Fit KMeans to the reduced dataset
kmeans.fit(X_train_pca)

# Predict cluster labels for the data points
y_pred_kmeans = kmeans.predict(X_train_pca)

# Evaluate clustering results using Adjusted Rand Index (ARI)
ari_kmeans = adjusted_rand_score(y_train, y_pred_kmeans)
print("Adjusted Rand Index for KMeans:", ari_kmeans)
```

Listing 8: Code for calculating Adjusted Rand Index for KMeans clustering

The KMeans clustering achieved an Adjusted Rand Index of approximately 0.36, indicating moderate similarity between the actual labels and the predicted clusters.

The Adjusted Rand Index (ARI) is a measure of similarity between two clusterings, considering random chance. It ranges from -1 to 1, where:

- ARI close to 1 indicates perfect clustering agreement,
- ARI close to 0 indicates random cluster assignments,
- ARI close to -1 indicates extremely poor clustering agreement.

The obtained ARI value, approximately 0.36, suggests moderate similarity between the actual labels and the clusters predicted by the KMeans algorithm. However, the interpretation of ARI value depends on the context and requires further analysis.

7 Dataset Splitting

```
# Split the dataset into training and testing subsets
X_train_split, X_test_split, y_train_split, y_test_split =
    train_test_split(X_train_csv, y_train_csv, test_size=0.2,
    random_state=42)
```

Listing 9: Code splitting the dataset into training and testing subsets

In this section, the data is split into training and testing sets using the `train_test_split` function from the `sklearn.model_selection` module. This split is a crucial step in machine learning as it allows assessing the model's performance on data it has not seen before.

The `train_test_split` function takes four arguments: the feature data (`X_train_csv`), the class labels (`y_train_csv`), the size of the testing data (in this case, 20%), and the `random_state` value, which controls the randomness of the data split.

After executing this code, the variables `X_train_split` and `X_test_split` contain the training and testing features, respectively, while `y_train_split` and `y_test_split` contain their corresponding class labels.

Splitting the data into training and testing sets enables evaluating the model's performance on data that was not used during the training process. This is crucial to ensure that the model can generalize to new, unseen data and avoids overfitting.

8 Classification with k-Nearest Neighbors (k-NN) and Evaluation

In this section, we employ the k-Nearest Neighbors (k-NN) algorithm for classification and evaluate its accuracy.

```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_pca, y_train_subset)
y_pred_knn = knn_classifier.predict(X_test_pca)
accuracy_knn = accuracy_score(y_test_csv, y_pred_knn)
print("Accuracy of k-NN Classifier:", accuracy_knn)
```

Listing 10: Code training a k-NN classifier and evaluating its performance

The code snippet above initializes a k-NN classifier with 5 neighbors and trains it on the reduced training dataset (`X_train_pca`). Subsequently, the classifier predicts the class labels for the test dataset (`X_test_pca`) and computes its accuracy using the true class labels (`y_test_csv`).

The output provides the accuracy of the k-NN classifier, representing the proportion of correctly classified instances in the test dataset.

9 Summary

The analysis of the Fashion MNIST dataset involved several key steps, each aimed at exploring, preprocessing, clustering, and classifying the data to gain insights and build predictive models. This summary provides an overview of the methods employed and the findings obtained throughout the analysis. The Fashion MNIST dataset comprises 60,000 training examples and 10,000 test examples, making it suitable for various computer vision and pattern recognition tasks. A comprehensive exploration of the dataset revealed important statistical measures, providing insights into the distribution of features and classes. Principal Component Analysis (PCA) was utilized to reduce the dimensionality of the dataset while preserving essential features, facilitating visualization and enhancing computational efficiency. Clustering analysis using the KMeans algorithm identified natural groupings within the dataset, with the Adjusted Rand Index (ARI) indicating moderate similarity between actual labels and predicted clusters. The dataset was split into training and testing subsets to evaluate classification models accurately, preventing overfitting and providing reliable performance estimates. A k-Nearest Neighbors (k-NN) classifier achieved an accuracy of approximately 83.52%, demonstrating the effectiveness of the algorithm in predicting class labels based on nearest neighbors in the feature space. Through a systematic analysis, valuable insights into the dataset's characteristics, clustering tendencies, and classification performance were obtained, paving the way for future research and applications in computer vision and pattern recognition. It encapsulates the key findings and methodologies employed throughout the analysis, providing a comprehensive overview of the exploration, preprocessing, clustering, and classification of the Fashion MNIST dataset.