Table of Contents

Question 1

The Fashion MNIST dataset consists of Zalando's article images. There are 60000 training and 10000 test 28 x 28 greyscale images, each pixel value ranging from 0 to 255 indicating the lightness or darkness of the pixel. The datasets have 785 columns, the first corresponding to the label (0-9 for 10 classes) and the rest representing the 784 (28 x 28) pixel values. The following steps are performed to load and process the dataset. (from part 1 of the project)

1. The dataset is downloaded from the source https://www.kaggle.com/datasets/zalando-research/fashionmnist?resource=download and stored in the location /content/drive/MyDrive/UMD/DATA 604/fashion_mnist_data/
2. The training and test datasets are loaded from the respective CSV files as dataframes and combined into a single dataframe.
3. The dataframe is then sorted by the labels of the images to obtain images of each class sequentially. Now, the dataframe contains 70000 images with 7000 images in each class.

Generally, data is represented using the canonical Orthonormal Basis (ONB), consisting of orthonormal components like (1, 0, . . . , 0), (0, 1, 0, . . . , 0), . . . , (0, . . . , 0, 1). At times, there exist bases that are more suitable for certain data sets. Hence, it is important to organize the dataset especially when there is repetitive information or lower information captured by redundant features. There are several methods for Data Organization, one being Principal Component Analysis or PCA, also known as Karhunen–Loève transform (KLT) or the Hotelling transform. PCA provides a data-dependent representation of the feature space.

X consists of 70000 observations in the dataset with all the 784 features. Hence, m = 70000 and

$$X = [x_1, ..., x_m]^T$$

The PCA functionality in Python also mean-centers the data that being one of the assumptions We assume that our observed data X is obtained by linear transformation W for p<m unknown variables $z = [z_1, ..., z_p]$. Hence,

$$X = Wy$$

W represents a change in the coordinate system or basis. Hene, rows, and columns of W are orthonormal to each other.

$$Id_p = W^TW$$

WW^T need not be an identity matrix.

Firstly, we mean-center the dataset X. C_{ZZ} is the covariance matrix of Z.

$$C_{ZZ} = E\{ZZ^T\}$$

We assume that C_{ZZ} is a diagonal matrix, and hence, the latent variables are uncorrelated.
C_{XX} is the covariance matrix of observed data X. Hence,

$$C_{XX} = E\{XX^T\} = E\{WZZ^TW^T\} = WE\{ZZ^T\}W^T = WC_{ZZ}W^T$$

$$\Rightarrow \quad C_{ZZ} = W^TC_{XX}W$$

Also, $C_{XX} = XX^T/(n-1)$
Using eigenvalue decomposition, we can say that

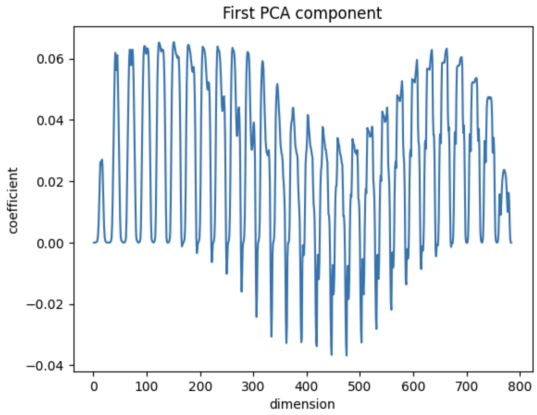
$$C_{xx} = V \Lambda V^T$$

Hence,

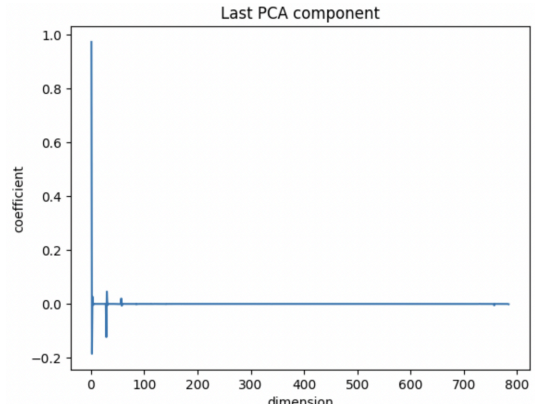
$$C_{ZZ} = W^T V \Lambda V^T W$$

With the assumption that p latent variables are uncorrelated, only p<m eigenvalues of Λ are non-zero. For maximum variance with p principal components, $W = VId_{m\times p}$
The variance captured by a principal component z_i can be given by $\sigma_i^2 = \lambda_i$, where λ_i is the ith eigenvalue of the covariance matrix C_{XX}.

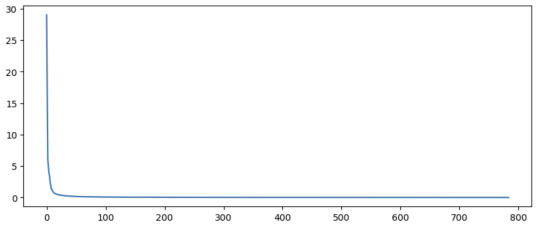
In this case, the PCA function from Python's sklearn library is fit on the given dataset to perform the above steps. The principal components are evaluated as columns of a 784 x 784 matrix. The principal components are returned in a decreasing order of variance explained by those components. Hence, the first principal component captures the maximum variance (1285715.238) and the last principal component captures the least variance (0.006) in the dataset. The first principal component can be written as a function v(i) = v_i for i from 1 to 784, where v_i is the coefficient of the first principal component across the ith dimension. It can be visualized as follows:



Similarly, the last principal component can be visualized as a function as follows:



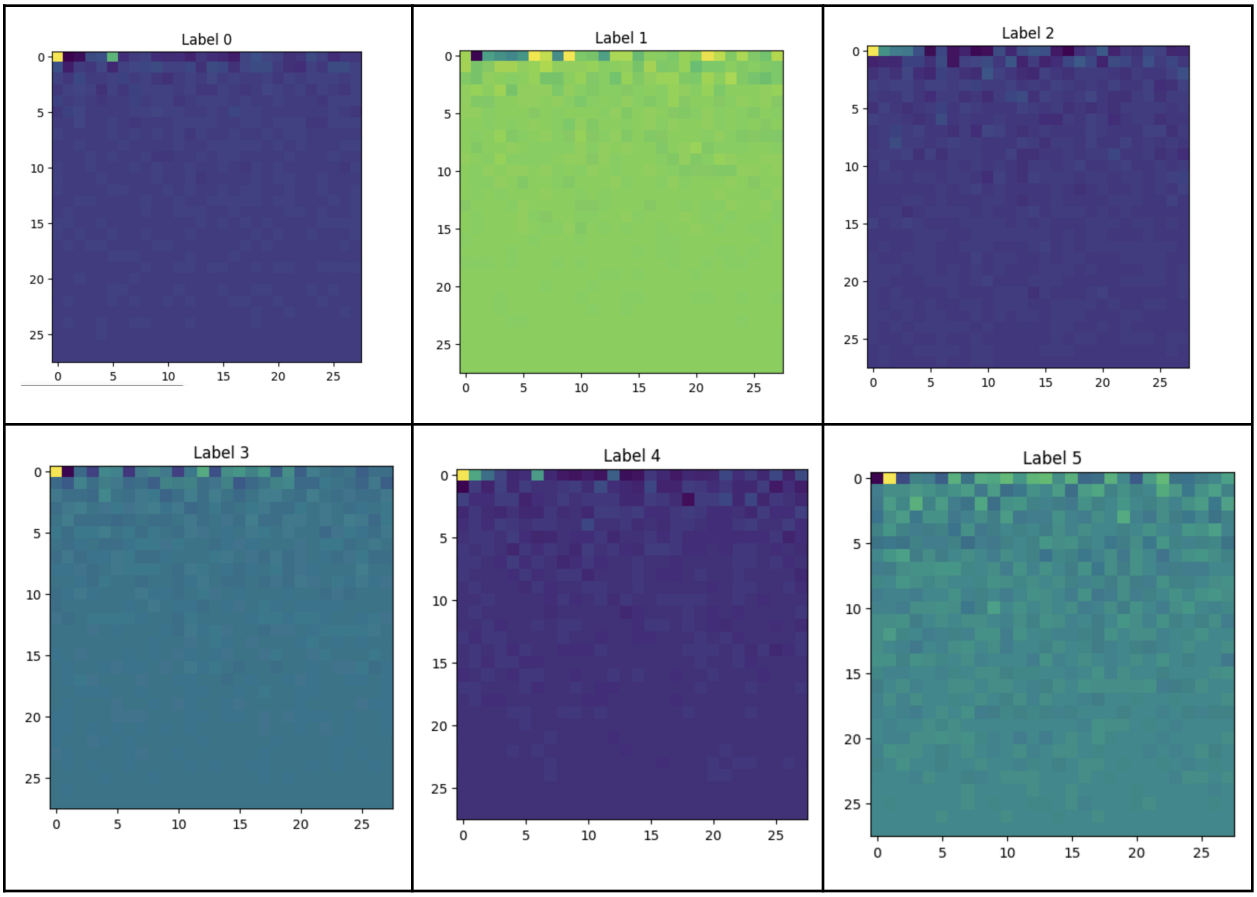
The percentage of explained variance captured by each of the 784 principal components is as follows:

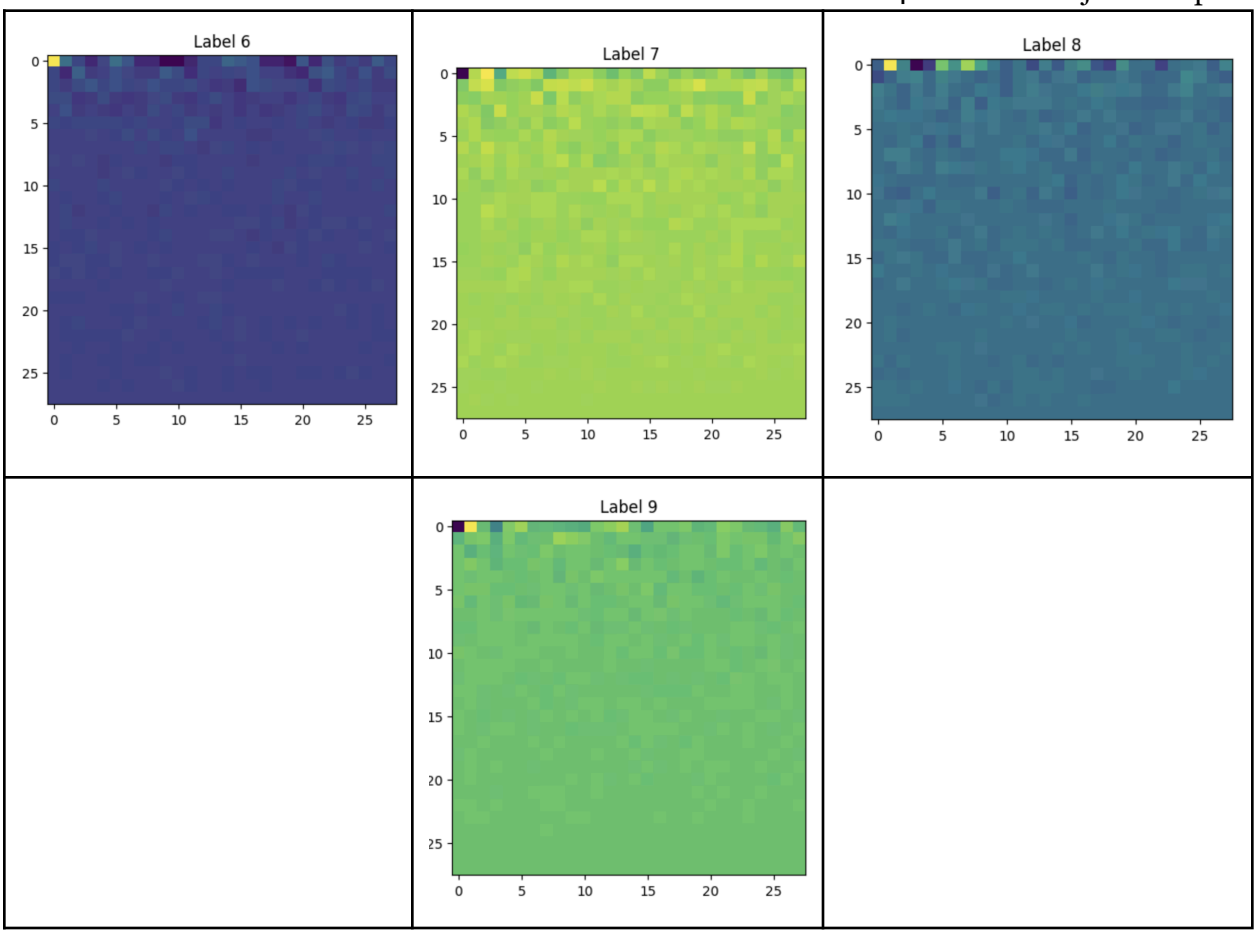


Question 2

Furthermore, the data X is now transformed to be represented on the basis of these principal components. Here, we consider PCA as more of a data organization tool than a dimensionality reduction technique. Hence, we preserve all 784 dimensions of the dataset.

Each observation can be reshaped to form a 28 x 28 image. The representations of the first observations for each digit class can be visualized as follows:



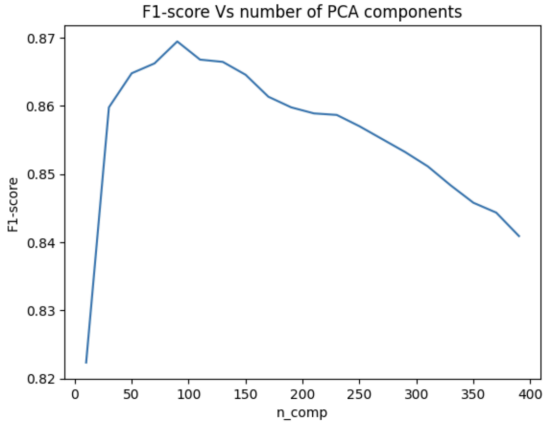


Question 3

The transformed dataset is now treated as our new dataset, now we apply the brute force K-Nearest Neighbours algorithm for optimal parameters and methods obtained in part 1 of the project. The following steps are followed for different numbers of initial principal components (the rest of the feature dimensions are neglected) ranging from 10 to 400 at intervals of 20.

1. The split_train_test_2(df, N) splits the dataframe into training and testing datasets by randomly choosing N samples from each class for training and the remaining ones for testing. It uses the 'stratify' option to avoid class imbalance.
2. The train_knn_l1(X_train, y_train) function creates a KNN model with k = 20 using the brute force algorithm and "l1" metric. It fits the model on the given training set and returns the model. It also computed the time required for training the model.
3. The test_knn(knn_model, X_test, y_test) function uses the fitted KNN model to predict the class label for the test dataset. It returns the predicted labels as well as computes the time required for testing.
4. A KNN model is fitted on the training set obtained for N = 6100 (optimal N obtained in the last question) and used for predictions on the test dataset.

A plot of the f1-score vs. the number of principal components is plotted:



It is noticed that the f1 score is maximum when the number of components is 90.

Since the experiment interval was 20, we further investigate the range of the number of principal components ranging from 90 to 110, at an interval of 2. It is observed that the f1 score is highest for the number of principal components is 100. The percentage of variance (or information) captured by the first 100 principal components is 91.235 %.

The steps performed while conducting the experiment to fit and predict a KNN model with optimal parameters (obtained in the 1st part of the project) are now performed for the number of principal components = 100.

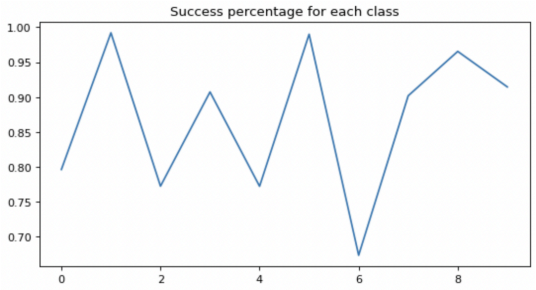
The following observations and conclusions are made from the given experiment:

1. An F1-score and accuracy score (global success rate) of 0.8679 is obtained on the test dataset.
2. The success rates for individual classes are evaluated on the KNN model trained by calculating the normalized diagonal elements of the confusion matrix on the predictions and real test data labels. The success rates for individual classes are as follows:

Class	Success Rate (%)
0	79.64
1	99.2
2	77.25
3	90.73
4	77.24
5	98.99
6	67.37
7	90.19
8	96.54

Class	Success Rate (%)
9	91.46

The success percentages for each class can be visualized as follows:



Class with the label 1 has the highest success rate (99.2 %) and class with label 6 has the lowest success rate (67.37 %). The global success rate is 100 * accuracy score = 86.79 %.

Question 4

The following conclusions have been made in the given experiment in comparison to part 1 of the project:

1. The accuracy and f1 score are higher after applying Principal Component Analysis, though the difference is not very significant (<1%).
2. The time required for predicting using the KNN model on the test dataset has reduced significantly from 566.628 seconds to 87.051 seconds (84.64 %). This is because performing dimensionality reduction reduces the number of features (d) and hence, the time required for prediction with the KNN model (O(knd)).
3. The classes with the highest and lowest success rates remain the same (1 and 6 respectively) and their success rates do not vary much in comparison to Part 1.

Conclusively, though PCA doesn't improve the performance metric significantly in this case, it helps achieve similar accuracy and f1 score with less number of features. This dimensionality reduction helps to keep only the relevant features/information in the dataset and get rid of the redundant features capturing lower variance/information. It also helps to save computational overhead and hence, the time required for the classification model.

Code

The Python code for all the experiments discussed above is attached below: