IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Revealing Fashion's Hidden Patterns: Leveraging Principal Component Analysis (PCA) for Fashion MNIST Analysis

KHADKA PILOT[1*] AND POUDEL KSHITIZ[2*]

[1]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: pilot.076bct025@tcioe.edu.np)
[2]Institute of Engineering, Thapathali Campus, Thapathali, Nepal (e-mail: kshitizpoudel18@gmail.com)

Corresponding author: Khadka Pilot (e-mail: pilot.076bct025@tcioe.edu.np).

* Authors contributed equally

ABSTRACT The exponential growth of data generation poses a significant challenge. Managing and extracting useful information from vast amount of data is crucial across various domains, including business, industry, medicine and more.However, both humans and computers are overwhelmed by the sheer volume and intricacy of the data. Given the limitations in resources and the recognition that not all features are equally significant, feature selection plays a vital role in the data processing pipeline. Principal component analysis(PCA) is one of the dimensionality reduction technique which helps to identify the representation of raw data with fewer components.This allows for a more focused analysis, avoiding unnecessary computational burden. Here, We study the application of PCA on random dataset and Iris dataset. Leveraging the insights obtained, we applied it to Fashion-MNIST dataset with much higher dimensions. For validating the obtained results, we compared classification report and reconstruction quality of the datasets.

INDEX TERMS Feature selection, PCA

I. INTRODUCTION

THE digital revolution has brought about a staggering consequence: the immense accumulation of data. Our online activities, such as search queries and clicks, are meticulously logged by search engines like Google and Bing. Even beyond the internet, our actions are tracked through governmental records, electronic health records, financial transactions, and more. These diverse sources collectively form a data footprint of our lives. [1]

In the face of this vast sea of information, techniques for constructing data mining models and performing feature selection have become imperative. One such technique is Principal Component Analysis (PCA), which addresses the challenge of handling high-dimensional data by transforming it into a lower-dimensional representation. Through the process of projecting the data onto principal components (PCs), PCA aims to retain the essential trends and patterns present in the dataset. This projection is carefully optimized to minimize the distance between the original data and its transformed representation. By employing PCA, the complexity of the data is effectively reduced while preserving a substantial amount of meaningful information and the underlying structure of the dataset. This enables for more efficient data analysis and allows for discovery of valuable insights. In essence, PCA serves as a powerful tool for navigating and extracting knowledge from the vast amount of data that the digital age has bestowed upon us. [2]

II. METHODOLOGY

A. THEORY

Consider we have m number of points $\{x_1, x_2, \ldots, x_m\}$. If we were to apply lossy compression to these points, we would store them in a way that requires less storage space while retaining most of the precision or information. One approach to achieve this is by representing the points in a lower-dimensional space using Principal Component Analysis (PCA). [3]

For each point $x_i \in \mathbb{R}^n$, we need to find a code vector $c_i \in \mathbb{R}^l$ such that the resulting space has a lower dimension, $l < n$. To do this, we define an encoding function $f(x_i) = c_i$, which produces the code for the input x_i. Additionally, we need a decoding function $g(c_i)$ that can reconstruct x_i such that $x_i \approx g(f(x_i))$.

To generate the code vector c_i, one approach is to minimize the distance between x_i and its reconstruction $g(c)$.This distance can be measured through a norm, which quantifies the difference between two vectors. For any input point x and its reconstruction $g(c^*)$, their distance can be measured through the norm by:

$$c^* = \arg\min_c \|x - g(c)\|_2 \tag{1}$$

Upon simplification,

$$c = D^T x \tag{2}$$

where $D \in \mathbb{R}^{n \times l}$ is a matrix that defines the decoding. To encode x using matrix operations, we define the encoding function as $f(x) = D^T x$, where matrix D represents the eigenvectors. For decoding (reconstruction), we have $r(x) = g(f(x)) = DD^T x$. The optimal value of D is given by the 'l' eigenvectors of XX^T.

Let $A = X.X^T$, then the eigenvector v of the resultant matrix A is a non-zero vector that alters only the scale of A.

$$Av = \lambda v \tag{3}$$

This scalar λ is known as the eigenvalue of the corresponding eigenvector.

Suppose matrix A has n eigenvectors, then they can be represented in the form of the matrix $V_{1 \times n}$. Then, the eigen decomposition of A is given by:

$$A = V \operatorname{diag}(\lambda) V^T \tag{4}$$

The decomposition of matrix into eigen values and eigen vectors is useful as it provides information about properties of matrix.

B. SYSTEM BLOCK DIAGRAM

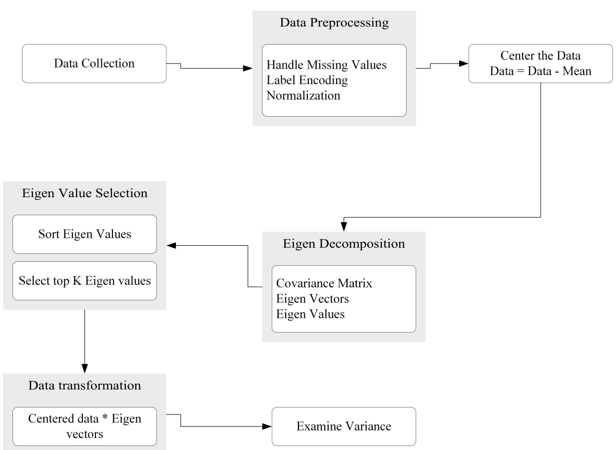

FIGURE 1. System Block Diagram

C. INSTRUMENTATION

The following libraries were instrumental in completion of this project :

- **Matplotlib**: It is a popular data visualization library for Python. The scatter method from the Matplotlib library was utilized to visualize the relationship between two variables.
- **NumPy**: Numpy provides tools for handling large, multi-dimensional arrays, along with a wide range of mathematical functions. Various NumPy methods were used in the analysis, including np.dot() for matrix multiplication, np.transpose() for transposing a matrix, and np.linalg.eig() for calculating eigenvalues and eigenvectors.
- **Scikit-learn**: Scikit-learn features algorithms like regression, classification, etc. The module load iris was used to load the iris dataset. mean squared error was used to calculate the reconstruction loss in the Fashion MNIST dataset.
- **Pandas**: It is a library for data manipulation and analysis. It was used for data loading of Iris dataset. Furthermore, data selection was used to filter the target column.
- **Tensorflow**: It is an open source library for machine learning. It provides tools to build, train and deploy machine learning models. It was used to create a simple neural network which was then trained on both lower dimensional dataset and the full dataset.

III. WORKING PRINCIPLE

A. DATASET COLLECTION

1) Generation of 20 data points

Let's consider a set of 20 2-D data points randomly sampled from the standard normal distribution. To help visualize these data points, a scatter plot is created (as shown in Figure 1), where each point represents one of the 20 data points.

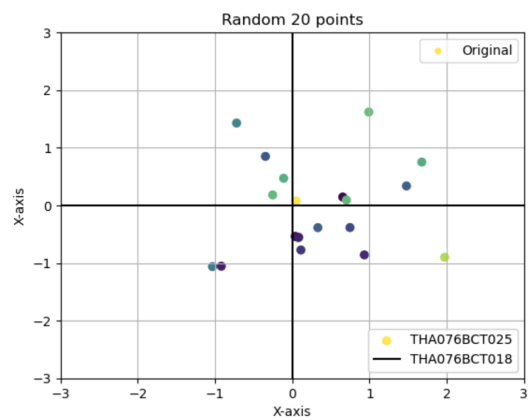

FIGURE 2. Scatter plot of 20 data points

To visualize the density distribution of these points, we can employ Kernel Density Estimation. This method estimates the underlying probability density function of the data by smoothing out the individual data points. By applying this technique, a contour plot (as shown in Figure 2) is generated.

When the data points are multiplied by a matrix taken from a uniform distribution over [0,1), the points are transformed.

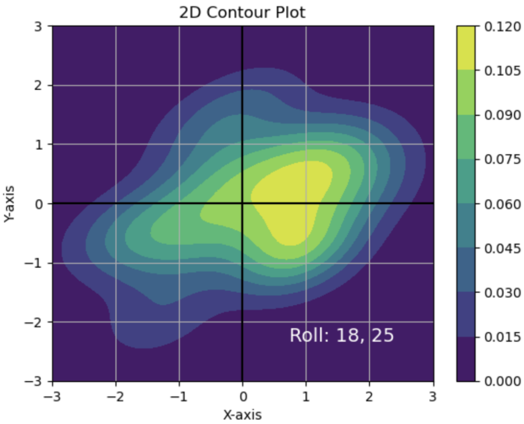

FIGURE 3. 2D contour plot of the Original data

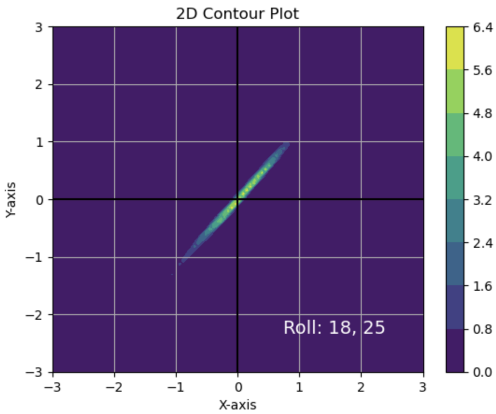

FIGURE 4. 2D contour plot of the Transformed data

This occurs because 2D points sampled from the normal distribution lie around the origin. They spread out from the origin in circular manner. When a matrix taken from uniform distribution transforms such space or the circle for reference, it shears it into a ellipse. This is why the data points look aligned. The eigenvectors v_i with eigenvalues λ_i of the matrix scale the space in the direction of v_i by a factor of λ_i as shown in Figure (5).

To observe scaling or rotation with shearing or stretching, the matrix should scale or rotate the data. A matrix that scales the data is given by:

$$\begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} \tag{5}$$

A rotation matrix is a special case where the scaling factor is always 1. In this case, we have $a = d$ and $b = -c$. The matrix takes the form:

$$\begin{bmatrix} a & -c \\ c & a \end{bmatrix} \tag{6}$$

Since the elements are drawn from a uniform distribution, any specific relationship between the elements, such as $b = -c$ for rotation, would have a probability of zero.

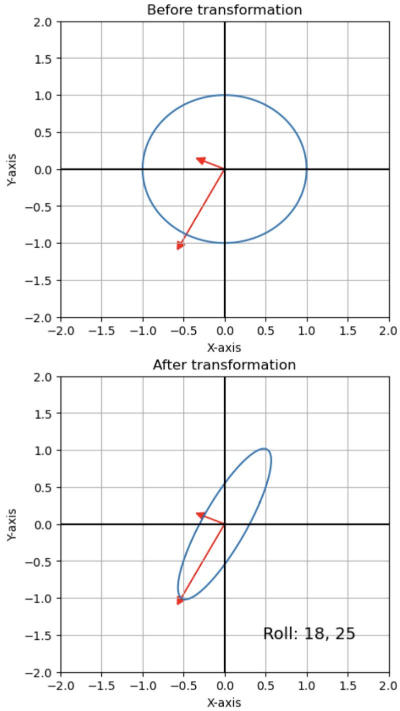

FIGURE 5. Transformation of the data due to matrix

Additionally, the probability of obtaining exactly zero from `np.random.rand()` is also very close zero.

2) Iris Dataset

Iris dataset is a widely used dataset in the field of machine learning often used as a beginner's dataset for practicing various ML techniques. It was first introduced by Ronald Fischer in his 1936 paper " The use of multiple measurements in taxonomic problems". [5] The dataset consists of 150 samples, with each sample representing an individual Iris flower.

It is available in the popular machine learning library, scikit-learn (sklearn), as a built-in dataset. To load the Iris dataset using scikit-learn, `load_iris()` function from the `sklearn.datasets` module was used.

3) Fashion MNIST Dataset

The Fashion MNIST dataset is a popular benchmark dataset in the field of machine learning and computer vision due to its simplicity and real-world image classification problem. [6] The thumbnail of 70,000 unique products were taken to create the dataset. Furthermore, those products comes from different groups: men, women, children and neutral.

TABLE 1. Files contained in the Fashion-MNIST dataset

Name	Description	# Examples	Size
train-images-idx3-ubyte.gz	Training set images	60,000	25 MBytes
train-labels-idx1-ubyte.gz	Training set labels	60,000	140 Bytes
t10k-images-idx3-ubyte.gz	Test set images	10,000	4.2 MBytes
t10k-labels-idx1-ubyte.gz	Test set labels	10,000	92 Bytes

The dataset was accessed using the `tf.keras.datasets` module.

B. WORKING PRINCIPLE OF PCA

1) Dataset preparation

Consider a dataset $X = [x_{ij}]_{m \times n}$ having m number of instances with n number of features. For x_{ij}, i represents the data and j represents the feature. For the sake of simplicity, let us take 2-dimensional data. [4]

2) Centering the data

Calculating and subtracting the mean of the data values to remove the influence of the mean value from the data. By centering the data, any bias that may be present in the original variables is removed. This is important because PCA is primarily concerned with capturing the directions of maximum variance in the data.

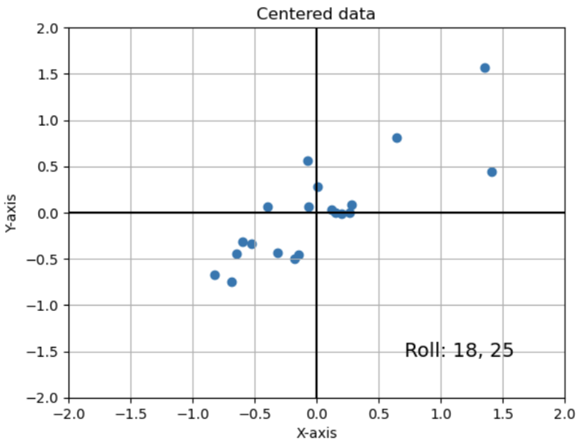

FIGURE 6. Scatter plot of Centered data

3) Calculation of Covariance matrix

The Covariance, which measures the relationships between variables, is calculated by utilizing the formula:

$$S_{ab} = \frac{1}{m-1} \sum_{i=1}^{m} (x_{ia} - \bar{x}_a)(x_{ib} - \bar{x}_b) \tag{7}$$

where S_{ab} is the covariance of the feature 'a' and 'b'.

However, when the data is centered (i.e., the means of all variables are subtracted), the above equation can be simplified to:

$$S_{ab} = \frac{1}{m-1} \sum_{i=1}^{m} x_{ia}.x_{ib} \tag{8}$$

Alternatively, the covariance matrix can be expressed in matrix form as:

$$S_{ab} = \frac{1}{m-1} \mathbf{X}^T \mathbf{X} \tag{9}$$

4) Calculation of Eigenvalues, Eigenvector from the covariance matrix

Once the Covariance matrix is calculated, the eigenvalues and eigenvectors can be calculated using the equation:

$$\det(S_{ab} - \lambda I) = 0 \tag{10}$$

$$(S_{ab} - \lambda I)E = 0 \tag{11}$$

I is the identity matrix of size $n \times n$, and E represents the eigenvectors.

The eigenvector of the dataset is plotted in scatter plot as follows.

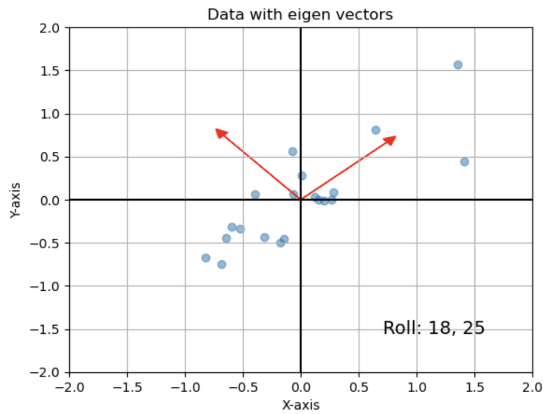

FIGURE 7. Scatter plot of Data along with its eigen vectors

5) Finding Feature score

Feature scores represent the importance or weight of each feature in the principal components generated by PCA. These scores are obtained by multiplying the standardized values of each feature with the corresponding eigenvector. Higher feature scores indicate greater relevance and influence in capturing the overall variance of the dataset.

$$\Lambda_{i,j} = x_{ij} \cdot E_j \tag{12}$$

where $\Lambda_{i,j}$ is the feature score for the i-th feature and j-th principal component,

6) Obtaining New data

Using the above equations, for the 20 data points, eigen vectors was calculated to be :

$$\begin{bmatrix} 0.74516502 & -0.66688012 \\ 0.66688012 & 0.74516502 \end{bmatrix}$$

The new data is obtained by multiplying the centered data with the selected top eigenvectors.

$$\text{PC scores} = \text{Row vector} \times \text{Row zero mean data}$$

Where, PC scores is the new data matrix.

Since only the first principal component was selected, 1D data was obtained. The scatterplot of the data can be seen in figure (8).

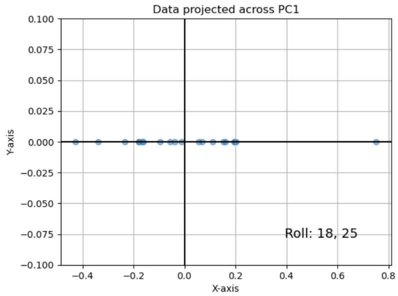

FIGURE 8. Scatter plot of Data in lower dimension

IV. RESULT AND ANALYSIS

A. IRIS DATASET

The iris dataset originally contains 4 features or dimensions, namely sepal length, sepal width, petal length and petal width all in centimetres (cm). It contains 3 target labels 0, 1, 2 corresponding to flower species 'setosa' 'versicolor' 'virginica'. Each species has 50 instances making total of 150 instances.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

FIGURE 9. Sample from Iris Dataset

Hence the data matrix size is (150 x 4 making the covariance matrix size (4 x4) which measures covariance of a feature with every other feature and there are 150 target labels.

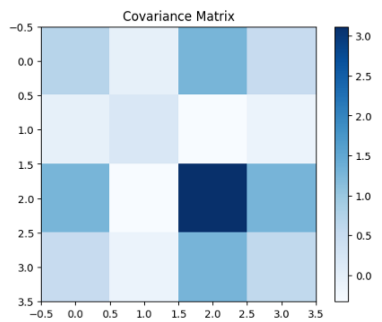

FIGURE 10. Covariance matrix of original data

If we now calculate the eigenvectors of covariance matrix and project our data onto those vectors we get transformed data in 4 dimensions. The covariance matrix of the corresponding data matrix has all off-diagonal elements close to 0.

The very dark square at the top left indicates that the data has now been transformed in such a way that the variance is very high in the direction of principal axis while other

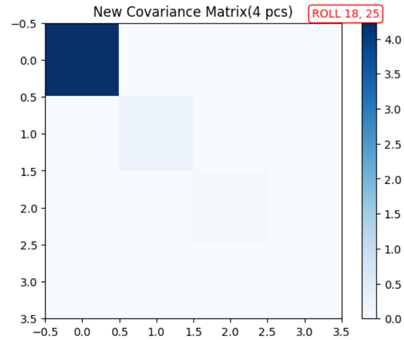

FIGURE 11. Covariance matrix after transformation

variances are relatively low. The off diagonal elements being close to zero mean that the 4 principal components are uncorrelated.

Our goal is to not use all the components but use lesser components for dimensionality reduction.

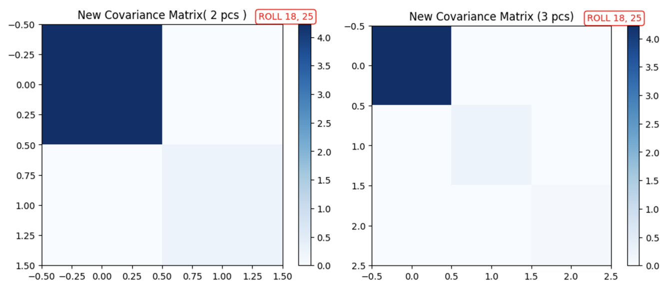

FIGURE 12. Covariance matrix with different PCs

The 1x1 covariance matrix has only 1 element which is 4.2284171.

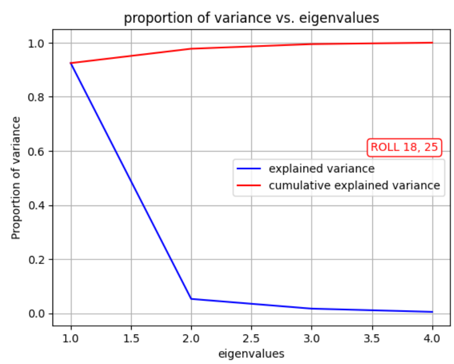

FIGURE 13. Proportion of variance vs Eigen values

The above graph plots explained variance vs. Total eigenvalues selected and suggests that 2 or even 1 eigenvectors are sufficient for us to represent the data.

If we do not use the top principal components but instead use the worst components to represent the data then the data

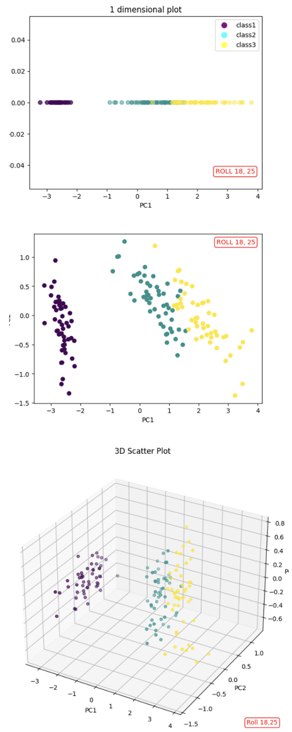

FIGURE 14. Iris data after dimensionality reduction

points cannot be separated by decision boundary in the space.

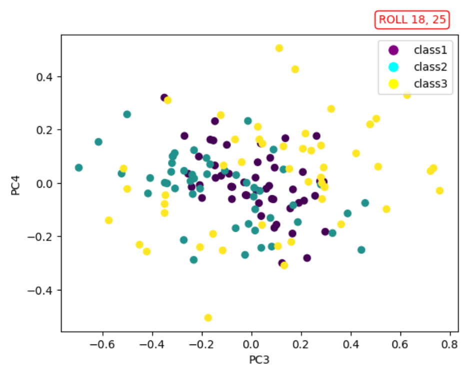

FIGURE 15. Iris dataset after choosing two worst principal component

We can reconstruct the original data of size (150×4) from the projected data of size $(150 \times k)$ with k less than 4 with some loss by the formula:

$$\text{Reconstructed data} := \text{PC scores} \times \text{eigenvectors}^T + \text{mean} \tag{13}$$

The dimension of Reconstructed Data is $(150, 4)$, PC scores is $(150, k)$, transpose of eigenvectors is $(k, 4)$, and mean is $(1, 4)$. The value of k denotes how many eigenvectors we initially selected as basis vectors to apply PCA. This is equivalent to resolving back to n dimensions from k dimensions.

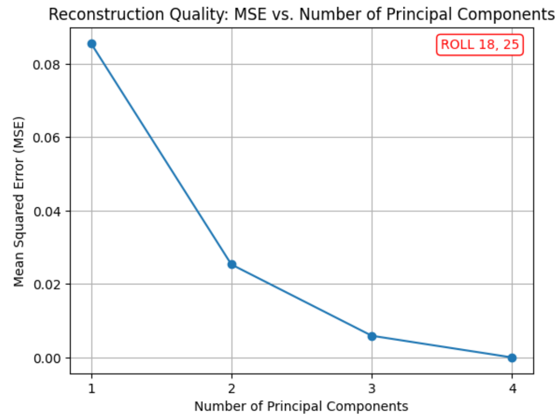

FIGURE 16. MSE vs Number of PCs

The above plot shows that the more components we use the better the reconstructed data matches to the original data. When we use all 4 components to reconstruct the data the error must be zero as shown in the plot.

B. FASHION MNIST DATASET

The fashion-MNIST data consist of images of size (28 x28) of different clothing items. The size of Training images is 60000 and test dataset is 10000. There are 10 classes of items: 'T-shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag' and 'Ankle boot'. Training dataset has 6000 items of each class.

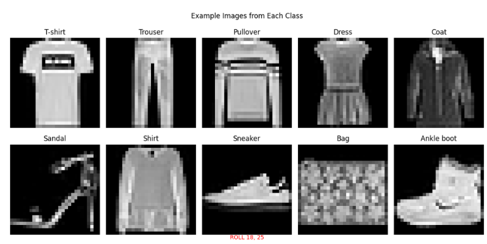

FIGURE 17. Fashion MNIST dataset

The first step to PCA is to subtract mean of every feature.

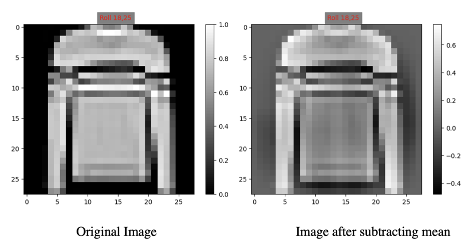

FIGURE 18. Sample image after centering

Similarly with iris dataset, we calculate the 784 eigenvalues and eigenvectors.

The calculated eigenvectors act as a new set of basis vectors in the 784 dimensional space. Out of those vectors the

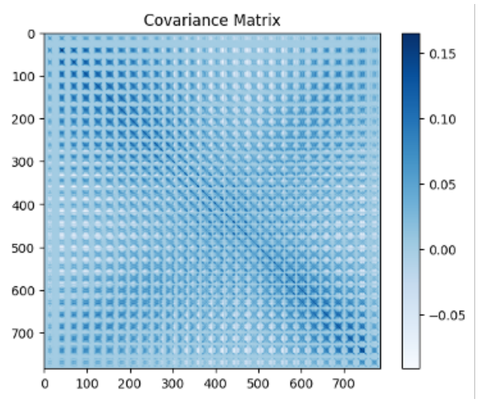

FIGURE 19. Covariance Matrix of Fashion MNIST

top few vectors can be regarded as principal components. These vectors point in the direction of maximum variance and are extremely important when representing data because data points can be projected onto these vectors with minimal information loss.

These vectors can also be considered eigen-images which are a set of representative images that capture the main patterns or variations present in a given dataset.

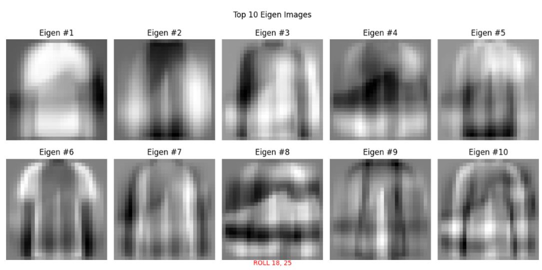

FIGURE 20. Top 10 Principal Components of dataset

The proportion of variance of first 5 eigenvalues are 0.29039, 0.17755, 0.06019, 0.049574 and 0.03847.

Unlike the iris dataset where only the proportion of variance of first eigen value was large this data has few significant eigenvalues. Which means that few components are not enough to represent the data well.

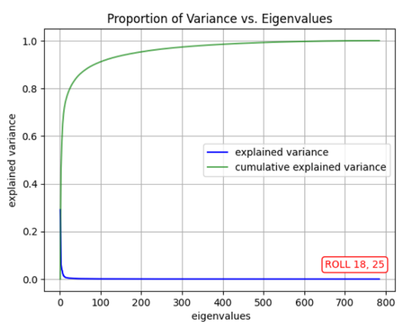

FIGURE 21. Proportion of Variance vs Eigenvalues for Fashion MNIST

Just like with the iris dataset reconstruction was performed from the data represented in different number of principal components.

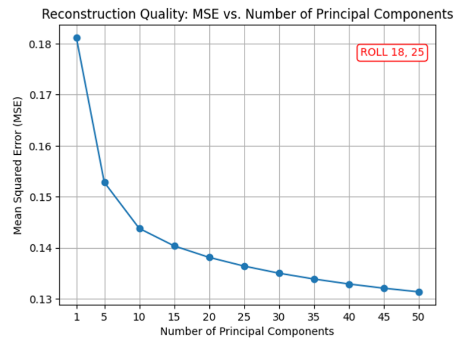

FIGURE 22. MSE vs Number of PCs for Fashion MNIST

We can consider only a particular class (pullover) and compare the reconstruction done from different number different number of principal components.

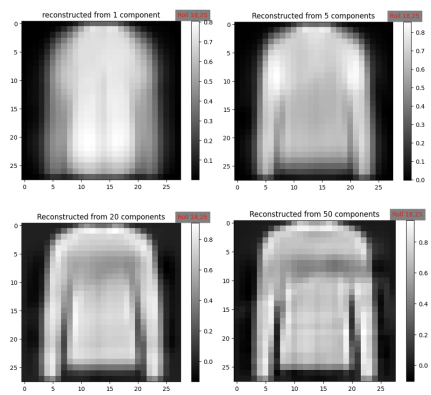

FIGURE 23. Reconstruction using different number of PCs

The dataset in 25 principal components was trained with a feed forward neural network.

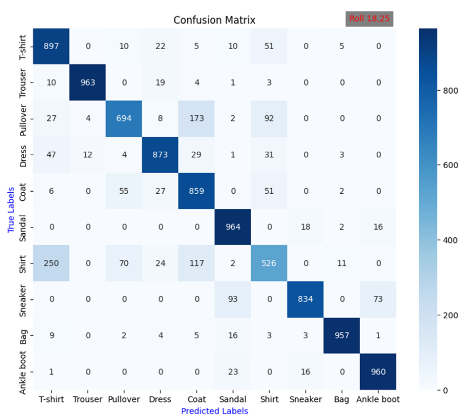

FIGURE 24. Confusion matrix for classification

V. DISCUSSION

A. IRIS DATASET

After the dimensionality reduction was performed using different numbers of components, the data points in the new space were clearly distinguishable even in fewer dimensions. The data obtained with components having the least eigenvalues could not be visibly separated with any decision boundary, meaning that those components could not describe the data well. One observation was that eigenvalues of the initial covariance matrix appeared along the diagonal of the covariance matrix of the new data after applying PCA.

To understand this, let's review the steps of PCA. The covariance matrix of the initial data matrix is given by:

$$S_x = \frac{1}{n-1} XX^T \tag{14}$$

We aim to find a transformation matrix, denoted as P, such that when we apply the transformation $Y = PX$ on the original data, the covariance matrix S_y is diagonalized. In other words, we want to diagonalize the covariance matrix S_y, which can be expressed as:

$$S_y = \frac{1}{n-1} YY^T \quad \text{(diagonalized)} \tag{15}$$

Substituting $Y = PX$, we get:

$$S_y = \frac{1}{n-1} PAP^T \tag{16}$$

Where $A = XX^T$ is the original covariance matrix. Now, from the eigen decomposition, we can write $A = VDV^T$, where D represents a diagonal matrix with eigenvalues.

Now, we select $P = V^T$ such that:

$$S_y = \frac{1}{n-1} V^T AV \tag{17}$$

Due to the orthogonality of eigenvectors, we have $V^T \cdot V = V \cdot V^T = I$. So, we can simplify the expression to:

$$S_y = \frac{1}{n-1} V^T VDV^T V \tag{18}$$

Simplifying further, we get:

$$S_y = \frac{1}{n-1} V^T DV \tag{19}$$

Since D is a diagonal matrix with eigenvalues, the resulting expression becomes:

$$S_y = \frac{1}{n-1} D \tag{20}$$

Thus, the matrix D consisting of eigenvalues is obtained.

Along the diagonal the first element of diagonal represent the principal component with highest variance. To validate that the data obtained after applying PCA was good. 2D and 3D plots showed that the data can be clearly distinguished if best components are used and cannot be distinguished if components with lower eigenvalues were used. Similarly, the reconstruction of data from the reduced form showed that more the number of components are used lesser is the reconstruction error.

B. FASHION MNIST DATASET

The fashion MNIST data contains more number of dimensions as each pixel of the 28*28 size image can be considered a feature and thus a dimension. Due to higher number of dimensions it was difficult to visualize the results even after applying PCA . At least 10 dimensions were necessary according to the explained variance. The top principal components represented the eigen images which means all the other images in the dataset can be taken as combination of eigen images just like a point in 3d space can be represented by x, y and z components. The results of explained variance and reconstruction loss were similar to the iris dataset and consistent with the principle.

Unlike the iris dataset visualization couldn't be performed so a dense feed forward model was trained on the new reduced dataset which consisted only 25 dimensions. The results were as good as training on the original form of dataset. The model could easily classify different images to the correct class.

TABLE 2. Result with Reduced Dimension (25 PCs)

Class	Precision	Recall	F1-Score	Support
0	0.76	0.86	0.81	1000
1	0.98	0.96	0.97	1000
2	0.81	0.72	0.76	1000
3	0.83	0.91	0.87	1000
4	0.74	0.85	0.79	1000
5	0.91	0.94	0.92	1000
6	0.71	0.50	0.59	1000
7	0.88	0.96	0.92	1000
8	0.96	0.95	0.95	1000
9	0.97	0.90	0.93	1000
Accuracy			0.86	10000
Macro Avg	0.85	0.86	0.85	10000
Weighted Avg	0.85	0.86	0.85	10000

TABLE 3. Result with Full Dataset (784 Dims)

Class	Precision	Recall	F1-Score	Support
0	0.84	0.74	0.79	1000
1	0.99	0.94	0.96	1000
2	0.76	0.67	0.71	1000
3	0.90	0.81	0.85	1000
4	0.77	0.59	0.67	1000
5	0.99	0.88	0.93	1000
6	0.46	0.76	0.58	1000
7	0.85	0.99	0.92	1000
8	0.96	0.94	0.95	1000
9	0.94	0.90	0.92	1000
Accuracy			0.82	10000
Macro Avg	0.85	0.82	0.83	10000
Weighted Avg	0.85	0.82	0.83	10000

VI. CONCLUSION

The application of PCA on random data provided some very useful insights on how PCA works and what kind of data is applicable for PCA process. The data in which features are highly correlated was suitable for applying PCA. The iris dataset could also be represented in only 1 or 2 components and still capture meaningful information. Similarly the very high dimensional images were also represented by relatively few dimensions . Different validation mechanisms also showed that the dimensionality reduction was indeed effective.

REFERENCES

[1] J. Grimmer. *"We Are All Social Scientists Now: How Big Data, Machine Learning, and Causal Inference Work Together."* Stanford University.
[2] Jolliffe, I. T., & Cadima, J. (2016). *"Principal component analysis: a review and recent developments."* Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences.
[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *"Deep Learning."* MIT Press.
[4] Jitrawadee, R., Praisan P., &Kittichai . "Logistic Principle Component Analysis (L-PCA) for Feature Selection in Classification." In *"2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery"*, pages 745. IEEE, 2018.
[5] R. A. Fisher (1936). *"The use of multiple measurements in taxonomic problems"*. Annals of Eugenics. 7 (2): 179–188.
[6] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," arXiv:1708.07747 [cs.LG], 2017.

PILOT KHADKA is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. During his time at Thapathali Campus, Pilot has actively engaged in various academic and extracurricular activities. He has participated in coding competitions, collaborated on software development projects, and demonstrated a keen interest in exploring new technologies.

KSHITIZ POUDEL is a student at Institute of Engineering, Thapathali Campus. He is expected to graduate in Bachelor of Computer Engineering in 2024. His relentless pursuit of knowledge and dedication to uplifting and empowering others make him an exceptional contributor and an invaluable asset to the academic community.

CODE

A. CIRCLE TRANSFORMATION

```
import numpy as np
import matplotlib.pyplot as plt


def plot_transformed_circle(mat):
    theta = np.linspace(0, 2 * np.pi, 100)
    eigenvalues, eigenvectors = np.linalg.eig(mat)

    # Generate the x and y coordinates of the unit circle
    x_circle = np.cos(theta)
    y_circle = np.sin(theta)

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(5, 10))

    ax1.plot(x_circle, y_circle, label="Original Circle")
    ax1.set_title("Before transformation")
    ax1.set_xlabel("X-axis")
    ax1.set_ylabel("Y-axis")
    ax1.set_xlim(-2, 2)
    ax1.set_ylim(-2, 2)
    ax1.axhline(0, color="black")
    ax1.axvline(0, color="black")
    ax1.text(
        0.9,
        0.1,
        "Roll: 18, 25",
        ha="right",
        va="bottom",
        transform=plt.gca().transAxes,
        fontsize=14,
    )
    ax1.grid()

    for eigenvector in eigenvectors.T:
        scaled_eigenvector = np.dot(mat, eigenvector)
        ax1.arrow(
            0,
            0,
            scaled_eigenvector[0],
            scaled_eigenvector[1],
            head_width=0.1,
            head_length=0.1,
            fc="red",
            ec="red",
        )

    # Matrix multiplation of circle with matrix
    x_transformed = mat[0, 0] * x_circle + mat[0, 1] * y_circle
    y_transformed = mat[1, 0] * x_circle + mat[1, 1] * y_circle

    ax2.plot(x_transformed, y_transformed, label="Transformed Circle")
    ax2.set_title("After transformation")
    ax2.axhline(0, color="black")
    ax2.axvline(0, color="black")
    ax2.set_xlabel("X-axis")
    ax2.set_ylabel("Y-axis")
    ax2.set_xlim(-2, 2)
    ax2.set_ylim(-2, 2)
    ax2.grid()

    for eigenvector in eigenvectors.T:
        scaled_eigenvector = np.dot(mat, eigenvector)
        ax2.arrow(
            0,
            0,
            scaled_eigenvector[0],
            scaled_eigenvector[1],
            head_width=0.1,
            head_length=0.1,
            fc="red",
            ec="red",
        )
    ax2.text(
        0.9,
        0.1,
        "Roll: 18, 25",
        ha="right",
        va="bottom",
        transform=plt.gca().transAxes,
        fontsize=14,
    )
    plt.show()


mat1 = np.random.rand(2, 2)
plot_transformed_circle(mat1)
```

B. 20 RANDOM POINTS

```
import src
import numpy as np

x = np.random.randn(20, 2)
mat = np.random.rand(2, 2)
result_mat = np.dot(x, mat)

"""
plot_scatter(x,y,title,label)
"""

src.plot_scatter(x[:, 0], x[:, 1], "Random 20 points", "Original")
src.plot_scatter(
    result_mat[:, 0], result_mat[:, 1], "Transformed 20 points", "Transformed"
)

"""
contour_plot (x,axis_min,axis_max)
  -- gaussian_kde to estimate pdf of x
  -- np.mpgrid for grid generation within min and max
"""
src.contour_plot(x, -3, 3)
src.contour_plot(result_mat, -3, 3)


var = np.cov(np.transpose(result_mat))
eigen_values, eigen_vectors = np.linalg.eig(var)
best_eigen_vector = np.transpose(eigen_vectors[:, 0])

new_data = np.dot(result_mat, eigen_vectors)
src.plot_scatter(new_data[:, 0], new_data[:, 1], "Transformed Data", "Transformed")

best_eigen_data = np.dot(result_mat, best_eigen_vector)
src.plot_scatter(
    best_eigen_data,
    np.zeros(len(best_eigen_data)),
    "Lower dim representation",
    "Best eigen",
)
```

C. SOURCE FILE FOR 20 POINTS

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.stats import gaussian_kde


def plot_scatter(x, y, title, label):
    colors = np.random.rand(len(x))
    plt.scatter(x, y, c=colors, label=label)
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)
    add_labels(title, label)
    plt.show()


def add_labels(title, label):
    plt.title(title)
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.axhline(0, color="black")
    plt.axvline(0, color="black")
    plt.grid()
    plt.legend(loc="upper right", markerscale=0.7)
    plt.text(
        0.9,
        0.1,
        "Roll: 18, 25",
        ha="right",
        va="bottom",
        transform=plt.gca().transAxes,
        fontsize=14,
    )


def contour_plot(x, axis_min, axis_max):
    k = gaussian_kde([x[:, 0], x[:, 1]])
    xi, yi = np.mgrid[axis_min:axis_max:100j,
    axis_min:axis_max:100j]
    zi = k(np.vstack([xi.flatten(), yi.flatten()])
    )

    contour = plt.contourf(xi, yi, zi.reshape(xi.
    shape), cmap="viridis")

    # Retrieve the contour lines
    contour_lines = contour.collections[0]
    plt.colorbar()
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)
    add_labels("2D Contour Plot", "")
```

D. IRIS DATASET CODE

```
import random
import sklearn
import numpy as np
import pandas as pd
from numpy import random
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import mean_squared_error

# Load the Iris dataset
iris = load_iris()

# Access the features (X) and target variable (y)
X = iris.data  # Features
y = iris.target  # Target variable
df = pd.DataFrame(data=iris.data, columns=iris.
    feature_names)
df["target"] = iris.target
x_train = iris.data
data_matrix = np.array(x_train)
mean = np.mean(x_train, axis=0)
x_train = x_train - mean

"""
Size of data matrix = (150, 4)

So covariance matrix size must be 4 x 4
"""

cov_matrix = np.cov(x_train, rowvar=False)

# Visualize covariance matrix
plt.imshow(cov_matrix, cmap="Blues", interpolation
    ="nearest")
extra_legend = "ROLL 18, 25"
plt.text(
    0.9,
    1.03,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)
plt.colorbar()
plt.title("Covariance Matrix")
plt.show()


# Calculate the eigenvectors and eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(
    cov_matrix)

# Sort the eigenvalues and corresponding
    eigenvectors in descending order
idx = np.argsort(eigenvalues)[::-1]  # Reverse the
     order to sort in descending order
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Proportion of variance of each eigenvalue
pov_list = []
for i in range(0, 4):
    pov = eigenvalues[i] / sum(eigenvalues)
    pov_list.append(pov)


# Print incremental proportion of variance
# Calculating the proporiton of
proportion_of_variance_list = []
for k in range(1, 5):
    selected_eigenvalues = eigenvalues[:k]
    proportion_of_variance = sum(
    selected_eigenvalues) / sum(eigenvalues)
    proportion_of_variance_list.append(
    proportion_of_variance)

x = range(1, 5)
plt.plot(x, pov_list, color="blue", label="
    explained variance")
plt.plot(
    x, proportion_of_variance_list, color="red",
    label="cumulative explained variance"
)
plt.grid()
# Set the axis labels
plt.xlabel("eigenvalues")
plt.ylabel("Proportion of variance")
extra_legend = "ROLL 18, 25"
plt.text(
    0.8,
    0.6,
    extra_legend,
    ha="left",
    va="center",
```

```
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red", boxstyle="round"),
)
plt.title("proportion of variance vs. eigenvalues")
plt.legend()
plt.show()

final_data = {}
for j in range(1, 5):
    print("If   %s  principal components(eigenvectors) is/are used" % (j))
    selected_eigenvectors = eigenvectors[:, :j]
    row_zero_mean_data = np.transpose(x_train)
    print(
        "original dataset size = ", row_zero_mean_data.shape
    )  # Original data= (150 x 4), transposed= (4 X 150)

    row_feature_vector = np.transpose(selected_eigenvectors)
    print(
        "shape of eigenvector matrix row-wise =", row_feature_vector.shape
    )  # Eigen vectors rowise

    final_data[j] = np.transpose(row_feature_vector @ row_zero_mean_data)
    print(
        "final data shape =", final_data[j].shape
    )  # Final data plotted only in the top j principal components discarding other dimensions
    print()


# Plotting covariance matrix of new data
cov_matrix_new = {}
for data in range(1, 5):
    new_matrix = np.cov(final_data[data], rowvar=False)
    cov_matrix_new[data] = new_matrix

plt.imshow(cov_matrix_new[2], cmap="Blues", interpolation="nearest")
plt.colorbar()
plt.text(
    0.9,
    1.03,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red", boxstyle="round"),
)
plt.title("New Covariance Matrix( 2 pcs )")
plt.show()

plt.imshow(cov_matrix_new[3], cmap="Blues", interpolation="nearest")
plt.colorbar()
plt.text(
    0.9,
    1.03,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red", boxstyle="round"),
)
plt.title("New Covariance Matrix (3 pcs)")
plt.show()

plt.imshow(cov_matrix_new[4], cmap="Blues", interpolation="nearest")
plt.colorbar()
plt.text(
    0.9,
    1.03,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red", boxstyle="round"),
)
plt.title("New Covariance Matrix(4 pcs)")
plt.show()

np.array(final_data[3])

"""Reconstruction of original data from transformed data"""


# Perform inverse transform and reconstruct the data
def inverse_transform(
    transformed_data, eigenvectors, mean
):  # mean of  original data column-wise
    # Step 1: Multiply the transformed data by the transpose of the eigenvectors
    # Transformed data shape = (150,k) where k=1,2,3,4

    inverse_transformed_data = np.dot(transformed_data, eigenvectors.T)

    # Step 2: Add the mean vector to the result
    reconstructed_data = inverse_transformed_data + mean

    return reconstructed_data


reconstructed_data = {}
for no_of_pcs in range(1, 5):
    print(
        "WHEN RECONSTRUCTED FROM %s  PRINCIPAL COMPONENTS(EIGENVECTORS) " % (no_of_pcs)
    )
    reconstructed_data[no_of_pcs] = inverse_transform(
        np.array(final_data[no_of_pcs]), eigenvectors[:, :no_of_pcs], mean
    )
    print(
        inverse_transform(
            np.array(final_data[no_of_pcs]), eigenvectors[:, :no_of_pcs], mean
        )
    )

"""Plot reconstruction mean squared loss for different number of components used"""

# calculating the reconstruction loss
mse_values = []
for no_of_pcs_used in range(1, 5):
    mse = mean_squared_error(X, reconstructed_data
```

```
    [no_of_pcs_used])
    mse_values.append(mse)

no_of_components = [1, 2, 3, 4]
plt.plot(no_of_components, mse_values, marker="o")
plt.xlabel("Number of Principal Components")
plt.ylabel("Mean Squared Error (MSE)")
plt.text(
    0.8,
    0.95,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)

plt.title("Reconstruction Quality: MSE vs. Number
    of Principal Components")
plt.xticks(no_of_components)
plt.grid(True)
plt.show()

final_data[2][:, 1].shape

final_data[2][:, 0]

plt.text(
    0.8,
    1.05,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)
plt.xlabel("PC1")
plt.ylabel("PC2")

legend_labels = ["class1", "class2", "class3"]
legend_colors = ["purple", "cyan", "yellow"]

# Create the legend
legend_elements = [
    plt.Line2D([0], [0], marker="o", color="w",
    markerfacecolor=color, markersize=10)
    for color in legend_colors
]
plt.legend(legend_elements, legend_labels)


plt.scatter(final_data[2][:, 0], final_data[2][:,
    1], c=y)


def get_data(selected_vectors):
    row_feature_vector = np.transpose(
    selected_vectors)
    return np.transpose(row_feature_vector @
    row_zero_mean_data)


# But if we do not select the top 2  PCs and
    instead use other combination like bottom 2
    PCA then,

num_eigenvectors = eigenvectors.shape[1]

# Select the bottom 2 eigenvectors
selected_eigenvectors_buttom = eigenvectors[:,
    num_eigenvectors - 2 : num_eigenvectors]
row_vector = np.transpose(
    selected_eigenvectors_buttom)
data = np.transpose(row_vector @
    row_zero_mean_data)
plt.text(
    0.8,
    1.05,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)
legend_labels = ["class1", "class2", "class3"]
legend_colors = ["purple", "cyan", "yellow"]

# Create the legend
legend_elements = [
    plt.Line2D([0], [0], marker="o", color="w",
    markerfacecolor=color, markersize=10)
    for color in legend_colors
]
plt.legend(legend_elements, legend_labels)

plt.xlabel("PC3")
plt.ylabel("PC4")
plt.scatter(data[:, 0], data[:, 1], c=y)


# Create a 3D scatter plot
fig = plt.figure()
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(
    final_data[3][:, 0], final_data[3][:, 1],
    final_data[3][:, 2], c=y, marker="o"
)
ax.text(
    50,
    10,
    1,
    "Roll 18,25",
    ha="left",
    va="center",
    color="red",
    transform=ax.transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)

legend_labels = ["class1", "class2", "class3"]
legend_colors = ["purple", "cyan", "yellow"]

# Create the legend
legend_elements = [
    plt.Line2D([0], [0], marker="o", color="w",
    markerfacecolor=color, markersize=10)
    for color in legend_colors
]
plt.legend(legend_elements, legend_labels)


ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
ax.set_title("3D Scatter Plot")
plt.show()

# but if we do not select the top 3 PCs and
    instead use other combination like bottom 3
    PCA then,
```

```

# Select the bottom 3 eigenvectors
selected_eigenvectors_buttom = eigenvectors[:,
    num_eigenvectors - 3 : num_eigenvectors]
row_vector = np.transpose(
    selected_eigenvectors_buttom)
data = np.transpose(row_vector @
    row_zero_mean_data)

# Create a 3D scatter plot
fig = plt.figure()
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(data[:, 0], data[:, 1], data[:, 2], c=y
    , marker="o")

legend_labels = ["class1", "class2", "class3"]
legend_colors = ["purple", "cyan", "yellow"]

# Create the legend
legend_elements = [
    plt.Line2D([0], [0], marker="o", color="w",
    markerfacecolor=color, markersize=10)
    for color in legend_colors
]
plt.legend(legend_elements, legend_labels)


ax.text(
    20,
    10,
    1,
    "Roll 18,25",
    ha="left",
    va="center",
    color="red",
    transform=ax.transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)

ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
ax.set_title("3D Scatter Plot")
plt.show()

y = iris.target

np.zeros(len(final_data[1]))

# 1D scatter plot

y1 = np.zeros(len(final_data[1]))  # x-coordinates
     for the scatterplot
x1 = final_data[1]  # y-coordinates for the
    scatterplot

plt.title("1 dimensional plot")
plt.xlabel("PC1")
plt.scatter(x1, y1, c=y, marker="o", alpha=0.5)
plt.text(
    0.8,
    0.1,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)
# Define the legend labels and colors
legend_labels = ["class1", "class2", "class3"]
legend_colors = ["purple", "cyan", "yellow"]

# Create the legend
legend_elements = [
    plt.Line2D([0], [0], marker="o", color="w",
    markerfacecolor=color, markersize=10)
    for color in legend_colors
]
plt.legend(legend_elements, legend_labels)
plt.show()
```

E. FASHION MNIST CODE

```
import keras
import numpy as np
import pandas as pd
import sklearn as sk
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers
from sklearn.metrics import classification_report


# Load the Fashion MNIST dataset
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images,
    test_labels) = fashion_mnist.load_data()

all_labels = list(train_labels)
unique_labels = np.unique(all_labels)

class_names = [
    "T-shirt",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]
unique_labels_strings = [class_names[label] for
    label in unique_labels]
unique_labels_strings


def plot_image(image, index):
    selected_image = image

    # Plot the image
    plt.imshow(selected_image, cmap="gray")
    plt.colorbar()
    plt.text(27, -1.2, "Roll 18,25", color="red",
    backgroundcolor="gray")

    # Display the plot
    plt.show()


plot_image(train_images[9], 9)

# Create a subplot grid
fig, axs = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle("Example Images from Each Class")

# Iterate over each class
for i, ax in enumerate(axs.flat):
    # Find an example image for the current class
    idx = np.where(train_labels == i)[0][0]
    image = train_images[idx]

    # Plot the image and set the title
```

```
    ax.imshow(image, cmap="gray")
    ax.set_title(class_names[i])
    ax.axis("off")

extra_legend = "ROLL 18, 25"
fig.text(0.5, 0.05, extra_legend, ha="center", color="red")
plt.tight_layout()
plt.show()

# Normalize pixel values
x_train = train_images / 255.0
x_test = test_images / 255.0

plot_image((train_images[5].reshape(28, 28) / 255), 5)

# reshape the x_train from (60000, 28, 28) to (60000,784) data matrix

x_train = x_train.reshape(60000, 784)

"""PCA

Step 1:Subtract mean values of each dimensions (columns)**
"""

mean = np.mean(x_train, axis=0)
x_train = x_train - mean
x_train[4]

# same image plotted after subtracting mean values
plot_image(x_train[5].reshape(28, 28), 5)

"""**step 2: Calculate covariance matrix of data matrix**"""

# size of data matrix=(60000, 784)
# so covariance matrix size must be 784*784

# Calculate the covariance matrix
cov_matrix = np.cov(x_train, rowvar=False)

print("Covariance matrix shape:", cov_matrix.shape)

# visualizing the covariance matrix

plt.imshow(cov_matrix, cmap="Blues", interpolation="nearest")
plt.colorbar()
plt.title("Covariance Matrix")
plt.show()

"""**Step 3: calculate eigenvectors and eigenvalues of covariance matrix**"""

##calculated using numpy library

# Calculate the eigenvectors and eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Sort the eigenvalues and corresponding eigenvectors in descending order
idx = np.argsort(eigenvalues)[::-1]  # Reverse the order to sort in descending order
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

eigenvectors.shape

"""**Step 4: select top k eigenvectors as basis vectors**"""

# Select the top k eigenvectors as basis

k = 50  # Replace with the desired number of eigenvectors
selected_eigenvectors = eigenvectors[:, :k]

print(selected_eigenvectors.shape)

"""Step 4.1: plotting the eigenimages(top eigenvectors)

"""

# Select the top 10 eigen vectors
top_10_eigen_vectors = eigenvectors[:, :10]
top_10_transposed = np.transpose(top_10_eigen_vectors)

# Create a subplot grid for displaying the eigen images
fig, axs = plt.subplots(2, 5, figsize=(12, 6))
fig.suptitle("Top 10 Eigen Images")

# Iterate over the top 5 eigen vectors
for i, ax in enumerate(axs.flat):
    eigen_image = np.real(top_10_transposed[i].reshape((28, 28)))
    ax.imshow(eigen_image, cmap="gray")
    ax.set_title(f"Eigen #{i+1}")
    ax.axis("off")

extra_legend = "ROLL 18, 25"
fig.text(0.5, 0.05, extra_legend, ha="center", color="red")
# Adjust the spacing between subplots
plt.tight_layout()
plt.show()

"""Step 4.2: calculate proportion of variance"""

# proportion of variance of each eigenvalue
pov_list = []
for i in range(0, 784):
    pov = eigenvalues[i] / sum(eigenvalues)
    pov_list.append(pov)

# print incremental proportion of variance
# calculating the proporiton of
proportion_of_variance_list = []
for k in range(0, 784):
    selected_eigenvalues = eigenvalues[:k]
    proportion_of_variance = sum(selected_eigenvalues) / sum(eigenvalues)
    proportion_of_variance_list.append(proportion_of_variance)

import matplotlib.pyplot as plt

x = range(1, 785)
plt.plot(x, pov_list, color="blue", label="explained variance", alpha=1)
plt.plot(
    x,
    proportion_of_variance_list,
    color="green",
    label="cumulative explained variance",
    alpha=0.65,
)
plt.grid()

# Set the axis labels
plt.xlabel(" eigenvalues")
```

```
plt.ylabel("explained variance")
plt.title("Proportion of Variance vs. Eigenvalues")

# Add extra legend for roll numbers
extra_legend = "ROLL 18, 25"
plt.text(
    0.8,
    0.1,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red", boxstyle="round"),
)

plt.legend()
plt.show()

"""**Step 5: Deriving the new dataset(data_matrix)**

FinalData=RowFeatureVector * RowZeroMeanData
"""
# for calculating final data for each number of selected input vectors, loop
final_data = {}
for j in range(1, 51):
    print("If   %s  principal components(eigenvectors) is/are used" % (j))
    selected_eigenvectors = eigenvectors[:, :j]
    row_zero_mean_data = np.transpose(x_train)
    print(
        "original dataset size = ", row_zero_mean_data.shape
    )  # original data= (60000 x 784), transposed=(784 X 60000)

    row_feature_vector = np.transpose(selected_eigenvectors)
    print(
        "shape of eigenvector matrix row-wise =", row_feature_vector.shape
    )  # eigen vectors rowwise

    final_data[j] = np.transpose(row_feature_vector @ row_zero_mean_data)
    print(
        "final data shape =", final_data[j].shape
    )  # final data plotted only in the top j principal components discarding other dimensions
    print()

"""Plotting covariance matrix of new data"""

cov_matrix_new = {}
for data in range(1, 51):
    new_matrix = np.cov(final_data[data], rowvar=False)
    cov_matrix_new[data] = new_matrix
    # Print the shape of the covariance matrix
    print("Covariance matrix shape:", new_matrix.shape)


# Create a figure and subplots
fig, axs = plt.subplots(5, 10, figsize=(12, 8))
# Iterate over the covariance matrices and plot them in subplots
for i in range(2, 51):
    if cov_matrix_new[i].shape == ():
        continue  # Skip the 1x1 covariance matrix
    ax = plt.subplot(5, 10, i)
    plt.imshow(cov_matrix_new[i])

# Adjust the spacing between subplots
fig.tight_layout()
plt.show()

"""##Reconstruction of original data from transformed data"""

print(mean)
print(mean.shape)


# Perform inverse transform and reconstruct the data
def inverse_transform(
    transformed_data, eigenvectors, mean
):  # mean of  original data column-wise
    # Step 1: Multiply the transformed data by the transpose of the eigenvectors
    # transformed data shape = (60000,k) where k=1,2,3,4....784

    inverse_transformed_data = np.dot(transformed_data, eigenvectors.T)

    # Step 2: Add the mean vector to the result
    reconstructed_data = inverse_transformed_data + mean

    return reconstructed_data


reconstructed_data = {}
for no_of_pcs in range(6, 52, 5):
    print(
        "WHEN RECONSTRUCTED FROM %s  PRINCIPAL COMPONENTS(EIGENVECTORS) "
        % (no_of_pcs - 1)
    )
    print()
    reconstructed_data[no_of_pcs - 1] = inverse_transform(
        np.array(final_data[no_of_pcs - 1]), eigenvectors[:, : no_of_pcs - 1], mean
    )
    print(
        inverse_transform(
            np.array(final_data[no_of_pcs - 1]), eigenvectors[:, : no_of_pcs - 1], mean
        )
    )
    print()

reconstructed_data[1] = inverse_transform(
    np.array(final_data[1]), eigenvectors[:, :1], mean
)


print(
    reconstructed_data[1].shape,
    reconstructed_data[5].shape,
    reconstructed_data[15].shape,
    reconstructed_data[20].shape,
    reconstructed_data[50].shape,
)

"""##comparing original data and reconstructed data"""

# original image
```

```python
plot_image((train_images[5] / 255).reshape(28, 28)
    , 5)

plt.title("reconstructed from 1 component")
plot_image(reconstructed_data[1][5].reshape(28,
    28), 5)

for b in range(5, 51, 5):
    plt.title(f"Reconstructed from {b} components"
    )
    plot_image(reconstructed_data[b][5].reshape
    (28, 28), 5)

# calculating the reconstruction loss
from sklearn.metrics import mean_squared_error

mse_values = []

for no_of_pcs_used in range(0, 51, 5):
    if no_of_pcs_used == 0:
        mse = mean_squared_error(x_train,
    reconstructed_data[1])
        mse_values.append(mse)
    else:
        mse = mean_squared_error(x_train,
    reconstructed_data[no_of_pcs_used])
        mse_values.append(mse)

print(mse_values)

no_of_components = [1, 5, 10, 15, 20, 25, 30, 35,
    40, 45, 50]
plt.plot(no_of_components, mse_values, marker="o")
plt.xlabel("Number of Principal Components")
plt.ylabel("Mean Squared Error (MSE)")
plt.title("Reconstruction Quality: MSE vs. Number
    of Principal Components")

extra_legend = "ROLL 18, 25"
plt.text(
    0.8,
    0.9,
    extra_legend,
    ha="left",
    va="center",
    color="red",
    transform=plt.gca().transAxes,
    bbox=dict(facecolor="white", edgecolor="red",
    boxstyle="round"),
)

plt.xticks(no_of_components)
plt.grid(True)
plt.show()

"""##Now lets train the new dataset with simple
    feed forward network"""

data_tensor = tf.convert_to_tensor(
    final_data[25], dtype=tf.float32
)  # 50 can be replaced with any pcs

# Print the shape of the tensor
print(data_tensor.shape)


model = keras.Sequential(
    [
        keras.layers.Dense(256, activation="relu",
     input_shape=(25,)),
        keras.layers.Dense(128, activation="relu")
    ,
        keras.layers.Dense(10, activation="softmax
    "),
    ]
)
model.compile(
    optimizer="adam", loss="
    sparse_categorical_crossentropy", metrics=["
    accuracy"]
)

model.fit(data_tensor, train_labels, epochs=5,
    batch_size=32)

## Now convert the test dataset through the same
    pipeline to convert it into 50 dimensions
    instead of 784
x_test = test_images.reshape(10000, 784)
mean_test = np.mean(x_test, axis=0)
x_test = x_test - mean_test
row_zero_mean_test_data = np.transpose(x_test)

selected_eigenvectors_test = eigenvectors[:, :25]
row_feature_vector_test = np.transpose(
    selected_eigenvectors_test)

final_test_data = row_feature_vector_test @
    row_zero_mean_test_data
final_test_data = np.transpose(final_test_data)

test_data_tensor = tf.convert_to_tensor(
    final_test_data, dtype=tf.float32)

# evaluate the test data
test_loss, test_acc = model.evaluate(
    test_data_tensor, test_labels)
print("Test accuracy:", test_acc)
print("Test Loss:", test_loss)

predicted_probabilities = model.predict(
    test_data_tensor)

# Convert probabilities to class labels
predicted_labels = np.argmax(
    predicted_probabilities, axis=1)

print(predicted_labels)

print(test_labels)

from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
cm = confusion_matrix(test_labels,
    predicted_labels)

print("Confusion Matrix:")
print(cm)

class_labels = [
    "T-shirt",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]

plt.figure(figsize=(10, 8))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
```

```
    cmap="Blues",
    xticklabels=class_labels,
    yticklabels=class_labels,
)
plt.title("Confusion Matrix")
plt.text(8.7, -0.2, "Roll 18,25", color="red",
     backgroundcolor="gray")
plt.xlabel("Predicted Labels", color="blue")
plt.ylabel("True Labels", color="blue")
plt.show()

plt.show()


# Generate the classification report
report = classification_report(test_labels,
     predicted_labels)

print(report)

""" **To compare it with similar model trained on
     full dataset**"""

model_full = keras.Sequential(
    [
        keras.layers.Dense(512, activation="relu",
     input_shape=(784,)),
        keras.layers.Dense(256, activation="relu")
    ,
        keras.layers.Dense(256, activation="relu")
    ,
        keras.layers.Dense(10, activation="softmax
    "),
    ]
)
model_full.compile(
    optimizer="adam", loss="
    sparse_categorical_crossentropy", metrics=["
    accuracy"]
)

model_full.fit(train_images.reshape(60000, 784),
     train_labels, epochs=5, batch_size=32)

# evaluate the test data
test_loss1, test_acc1 = model_full.evaluate(
    test_images.reshape(10000, 784), test_labels
)
print("Test accuracy:", test_acc1)
print("Test Loss:", test_loss1)

predicted_probability1 = model_full.predict(
     test_images.reshape(10000, 784))
predicted_labels1 = np.argmax(
     predicted_probability1, axis=1)
report1 = classification_report(test_labels,
     predicted_labels1)
print(report1)

"""The same model is performing even poorer with
     full dimensions. Maybe this is  due to more
     noise in original data?

performance of cnn on full dataset.
"""


model_cnn = tf.keras.Sequential(
    [
        layers.Reshape((28, 28, 1), input_shape
    =(784,)),
        layers.Conv2D(32, (3, 3), activation="relu
    "),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(10, activation="softmax"),
    ]
)

model_cnn.compile(
    optimizer="adam", loss="
    sparse_categorical_crossentropy", metrics=["
    accuracy"]
)

model_cnn.fit(train_images.reshape(60000, 784),
     train_labels, epochs=5, batch_size=32)

predicted_probability2 = model_cnn.predict(
     test_images.reshape(10000, 784))
predicted_labels2 = np.argmax(
     predicted_probability2, axis=1)
report2 = classification_report(test_labels,
     predicted_labels2)
print(report2)
```