**PCA**

**CSE 303: Machine Learning**

Submitted by

Name: Roda Chinthapalli

Roll No: AP22110011496

Section:M

Lab Date: 15/10/24

Submission Date: 21/10/24

**A picture containing text

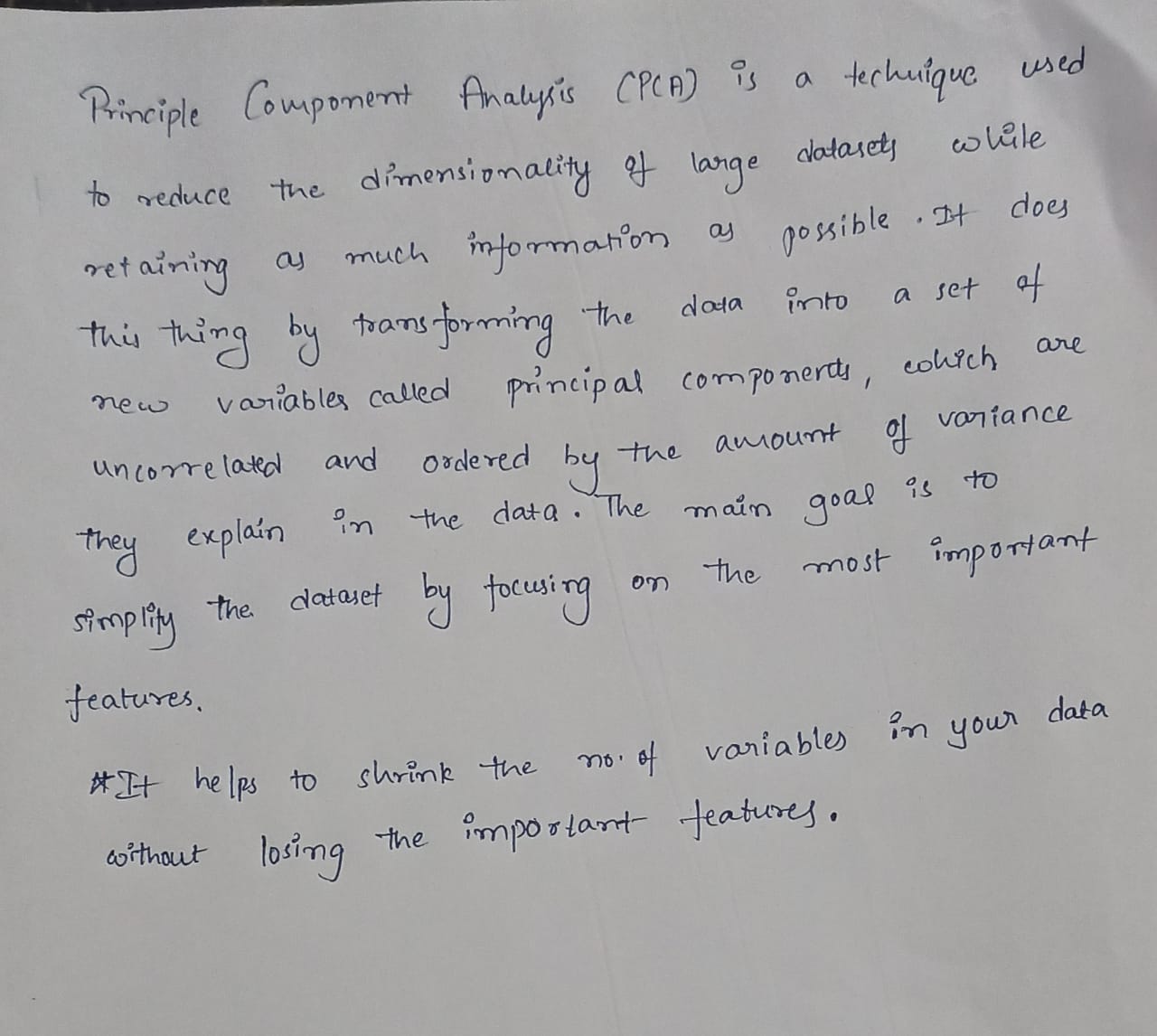
Description automatically generated**

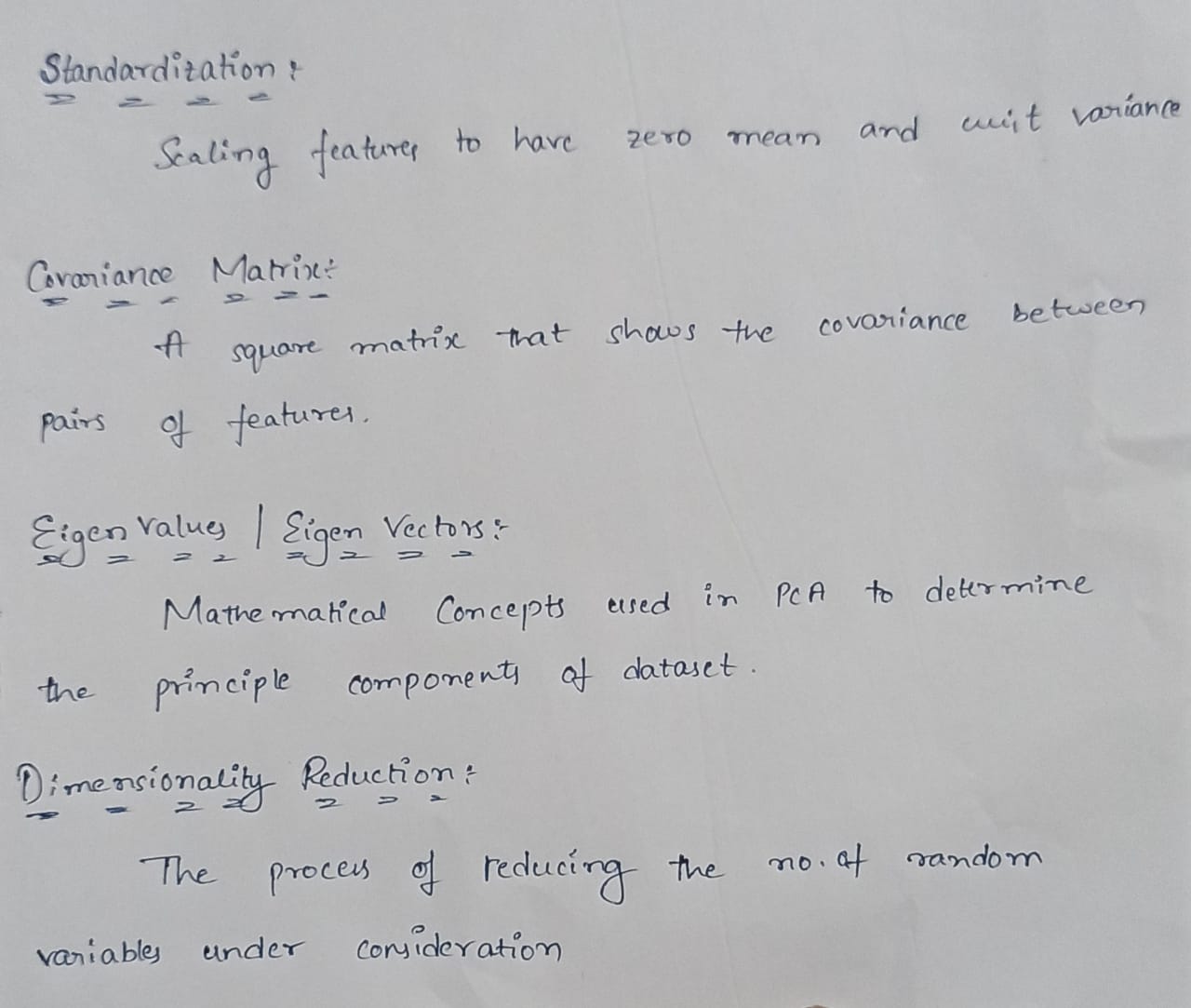
**Department Computer Science and Engineering**

**School of Engineering and Sciences**

**SRM University–AP**

**Amaravati, Andhra Pradesh – 522 240, India**

****

****

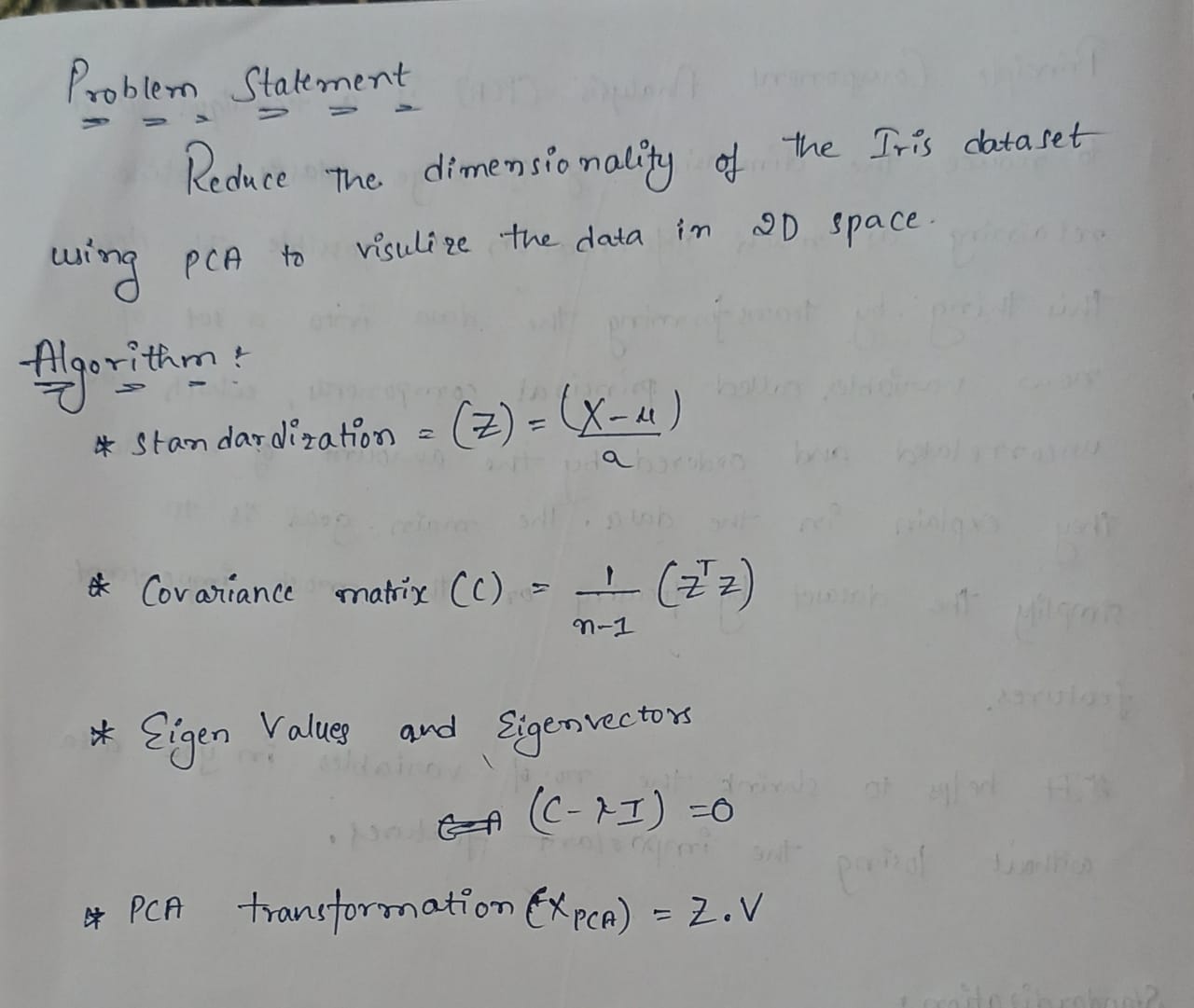
**Lab Exercise 1: Understanding PCA with a Simple Dataset**

**Objective:** To understand the basic principles of PCA by reducing the dimensionality of a simple dataset.

**Instructions:**

1. **Load the Dataset:**
   * Use the Iris dataset (available in sklearn or seaborn) which contains 4 features and 3 classes.
   * Perform a quick exploratory data analysis (EDA) to visualize the data and identify potential patterns.
2. **Standardization:**
   * Standardize the features so that they have a mean of 0 and a standard deviation of 1.
3. **Compute the Covariance Matrix:**
   * Calculate the covariance matrix of the standardized data.
4. **Eigenvalues and Eigenvectors:**
   * Calculate the eigenvalues and eigenvectors of the covariance matrix.
5. **PCA Transformation:**
   * Sort the eigenvectors by the magnitude of their corresponding eigenvalues.
   * Project the data onto the first two principal components.
6. **Visualization:**
   * Create a scatter plot of the data in the new 2D space defined by the first two principal components. Use different colors for each class in the Iris dataset.

**Problem Statement & Algorithm:**

****

**Solution:**

**Dataset: Iris**

**# Step 1: Import necessary libraries**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.preprocessing import StandardScaler**

**# Step 2: Load the dataset from the given path**

**data = pd.read\_csv('/content/iris.csv')**

**# Step 3: Exploratory Data Analysis (EDA)**

**# Look at the first few rows of the dataset**

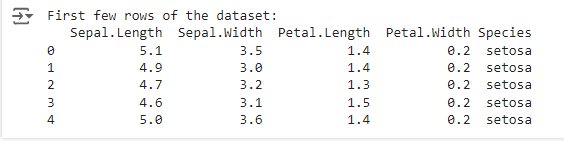
**print("First few rows of the dataset:")**

**print(data.head())**

**# Extract features and labels**

**X = data.iloc[:, :-1].values # Features (first 4 columns)**

**y = data.iloc[:, -1].values # Labels (last column)**

****

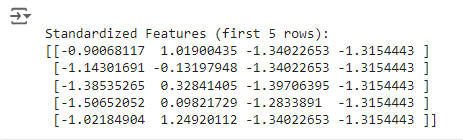
**# Step 4: Standardize the dataset**

**scaler = StandardScaler()**

**X\_std = scaler.fit\_transform(X)**

**print("\nStandardized Features (first 5 rows):")**

**print(X\_std[:5])**

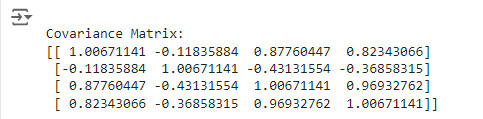
****

**# Step 5: Compute the covariance matrix**

**cov\_matrix = np.cov(X\_std.T)**

**print("\nCovariance Matrix:")**

**print(cov\_matrix)**

****

**# Step 6: Compute eigenvalues and eigenvectors**

**eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)**

**# Step 7: Sort the eigenvalues and corresponding eigenvectors**

**sorted\_indices = np.argsort(eigenvalues)[::-1]**

**eigenvalues = eigenvalues[sorted\_indices]**

**eigenvectors = eigenvectors[:, sorted\_indices]**

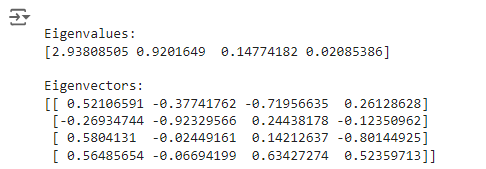
**# Print eigenvalues and eigenvectors**

**print("\nEigenvalues:")**

**print(eigenvalues)**

**print("\nEigenvectors:")**

**print(eigenvectors)**

****

**# Step 8: Project the data onto the first two principal components**

**X\_pca = X\_std.dot(eigenvectors[:, :2])**

**# Step 9: Visualization**

**# Create a scatter plot with different colors for each class**

**plt.figure(figsize=(8, 6))**

**for label, color in zip(np.unique(y), ['r', 'g', 'b']):**

**plt.scatter(X\_pca[y == label, 0], X\_pca[y == label, 1], label=label, color=color)**

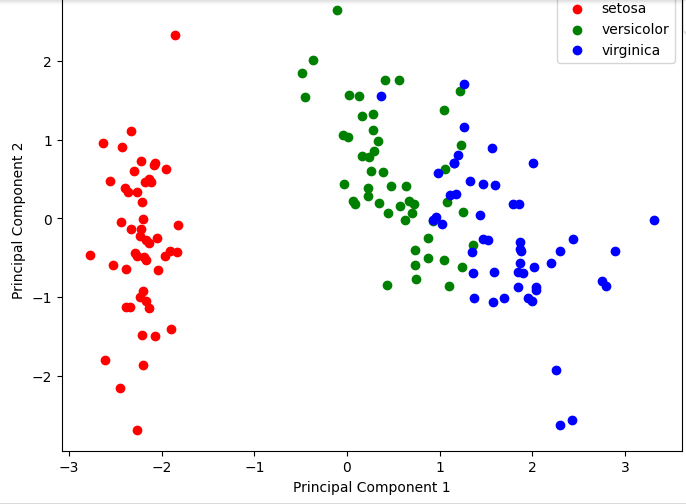
**plt.title('PCA of Iris Dataset')**

**plt.xlabel('Principal Component 1')**

**plt.ylabel('Principal Component 2')**

**plt.legend()**

**plt.show()**

****

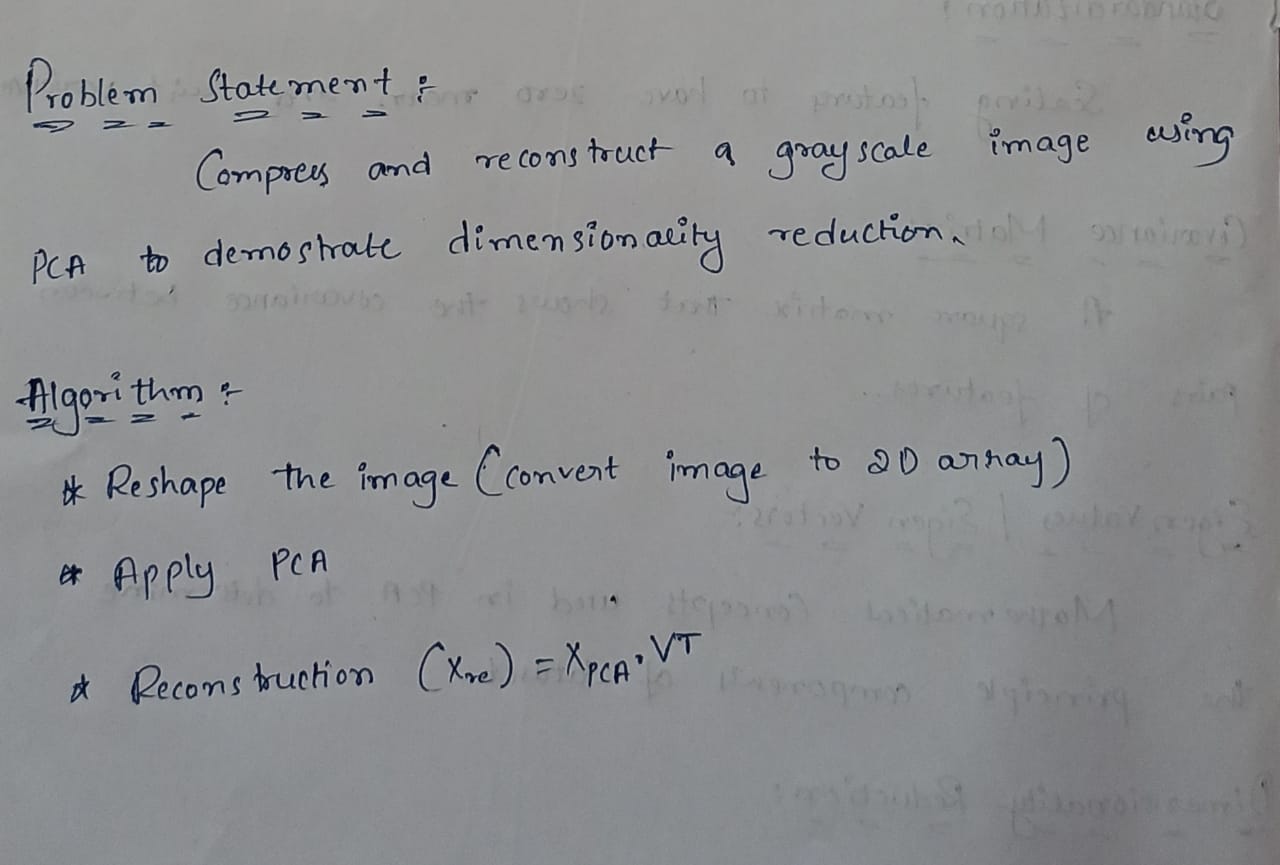
**Lab Exercise 2: PCA for Image Compression**

**Objective:** Use PCA to compress and then reconstruct an image, demonstrating the power of dimensionality reduction in data compression.

**Instructions:**

1. **Load an Image:**
   * Load a grayscale image (e.g., a 256x256 image of a face or any simple object).
2. **Reshape the Image:**
   * Treat the image as a matrix and flatten it to a 2D matrix where each row is a pixel and each column is a feature (intensity values of pixels).
3. **Apply PCA:**
   * Perform PCA on the image data, reducing the number of principal components used for reconstruction.
4. **Reconstruction:**
   * Reconstruct the image using a different number of principal components (e.g., 5, 20, 50, 100).
5. **Visualize Results:**
   * Display the original image and the reconstructed images at various levels of dimensionality reduction.

**Problem Statement & Algorithm:**

****

**Solution :**

**Image.jpeg**

**from google.colab import files**

**from skimage import io, color**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.decomposition import PCA**

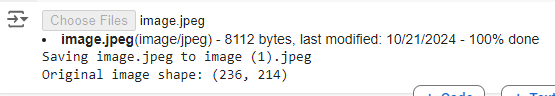
**# Step 1: Upload the image**

**uploaded = files.upload()**

**# Step 2: Load the grayscale image**

**image = io.imread('/content/image.jpeg', as\_gray=True)**

**print("Original image shape:", image.shape)**

****

**# Step 3: Reshape the image to a 2D array (flatten it)**

**X = image.reshape(-1, image.shape[1])**

**print("Flattened image shape:", X.shape)**

****

**# Step 4: Apply PCA and reduce dimensionality**

**def pca\_compression(X, n\_components):**

**pca = PCA(n\_components=n\_components)**

**X\_pca = pca.fit\_transform(X)**

**X\_reconstructed = pca.inverse\_transform(X\_pca)**

**return X\_reconstructed**

**# Step 5: Reconstruct the image with different components**

**components = [5, 20, 50, 100]**

**fig, ax = plt.subplots(1, len(components)+1, figsize=(20, 5))**

**# Show the original image**

**ax[0].imshow(image, cmap='gray')**

**ax[0].set\_title('Original Image')**

**# Reconstruct and display images with different components**

**for i, n\_comp in enumerate(components):**

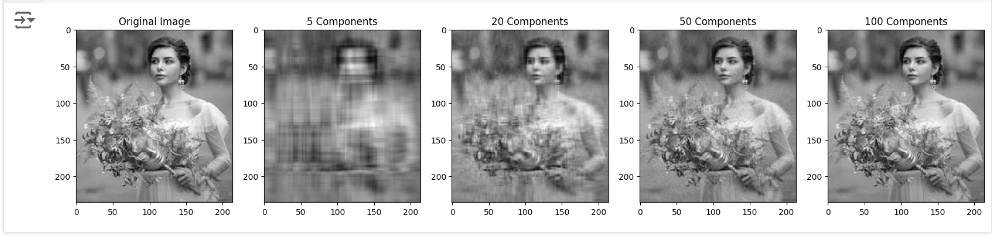
**X\_reconstructed = pca\_compression(X, n\_comp)**

**reconstructed\_image = X\_reconstructed.reshape(image.shape)**

**ax[i+1].imshow(reconstructed\_image, cmap='gray')**

**ax[i+1].set\_title(f'{n\_comp} Components')**

**plt.show()**

****

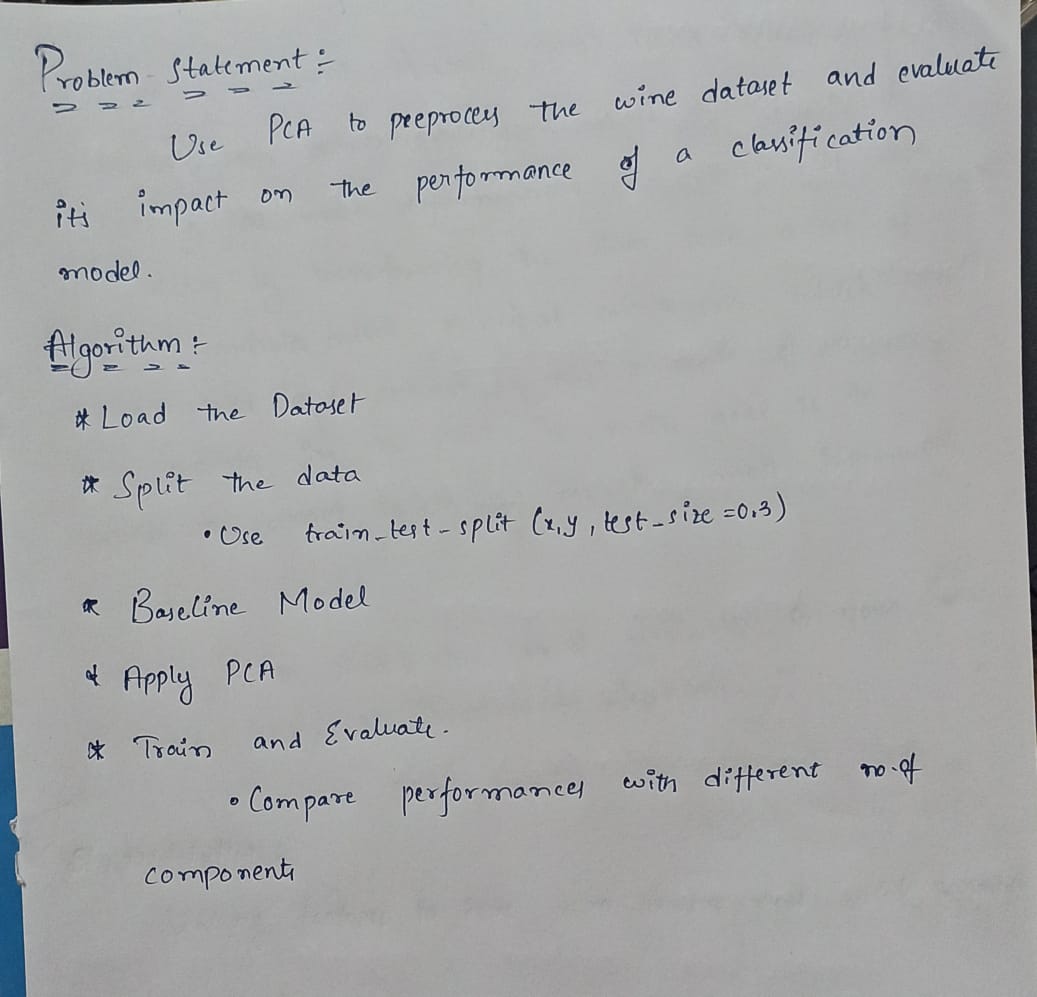
**Lab Exercise 3: PCA for Feature Reduction in a Classification Task**

**Objective:** Apply PCA as a preprocessing step to reduce the feature space for a classification problem and compare its effect on model performance.

**Instructions:**

1. **Load a Dataset:**
   * Use a dataset with many features, such as the Wine dataset from sklearn, which has 13 features.
2. **Split the Data:**
   * Split the data into training and testing sets (e.g., 70% training, 30% testing).
3. **Baseline Model (No PCA):**
   * Train a classification model (e.g., Logistic Regression or SVM) on the raw dataset without applying PCA. Evaluate its performance using accuracy, precision, and recall.
4. **Apply PCA:**
   * Apply PCA to the training data, retaining different numbers of components (e.g., 2, 5, 10).
   * Project the test data onto the same principal components.
5. **Train and Evaluate:**
   * Train the same classification model on the reduced dataset.
   * Compare the performance of the models trained with different numbers of principal components.
6. **Visualization:**
   * Plot a graph showing how accuracy changes as the number of components increases.

**Problem Statement & ALgorithm:**

****

**Solution:**

**Dataset: Wine Dataset**

**# Step 1: Load libraries**

**import numpy as np**

**import pandas as pd**

**import zipfile**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.decomposition import PCA**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, confusion\_matrix**

**import matplotlib.pyplot as plt**

**# Step 2: Unzip and load the dataset**

**with zipfile.ZipFile("/content/archive (1).zip", 'r') as zip\_ref:**

**zip\_ref.extractall('/content/')**

**# Assuming the CSV file is named 'winequality.csv' after extraction**

**wine\_data = pd.read\_csv('/content/winequality-red.csv')**

**# Step 3: Separate features and target (based on the 'quality' column)**

**X = wine\_data.drop('quality', axis=1) # Features**

**y = wine\_data['quality'] # Target**

**# Step 4: Split the dataset (70% training, 30% testing)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

**# Step 5: Baseline Model without PCA (Logistic Regression with class balancing)**

**scaler = StandardScaler()**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**# Train Logistic Regression without PCA, with class balancing**

**log\_reg = LogisticRegression(max\_iter=10000, class\_weight='balanced')**

**log\_reg.fit(X\_train\_scaled, y\_train)**

**# Evaluate the baseline model**

**y\_pred\_baseline = log\_reg.predict(X\_test\_scaled)**

**baseline\_accuracy = accuracy\_score(y\_test, y\_pred\_baseline)**

**baseline\_precision = precision\_score(y\_test, y\_pred\_baseline, average='weighted', zero\_division=1)**

**baseline\_recall = recall\_score(y\_test, y\_pred\_baseline, average='weighted')**

**print(f"Baseline Model - Accuracy: {baseline\_accuracy:.4f}, Precision: {baseline\_precision:.4f}, Recall: {baseline\_recall:.4f}")**

****

**# Step 6: Apply PCA and evaluate model performance with different components**

**components = [2, 5, 10]**

**accuracies = []**

**precisions = []**

**recalls = []**

**for n\_comp in components:**

**# Apply PCA**

**pca = PCA(n\_components=n\_comp)**

**X\_train\_pca = pca.fit\_transform(X\_train\_scaled)**

**X\_test\_pca = pca.transform(X\_test\_scaled)**

**# Train Logistic Regression on reduced features with class balancing**

**log\_reg\_pca = LogisticRegression(max\_iter=10000, class\_weight='balanced')**

**log\_reg\_pca.fit(X\_train\_pca, y\_train)**

**# Evaluate model performance**

**y\_pred\_pca = log\_reg\_pca.predict(X\_test\_pca)**

**accuracy = accuracy\_score(y\_test, y\_pred\_pca)**

**precision = precision\_score(y\_test, y\_pred\_pca, average='weighted', zero\_division=1)**

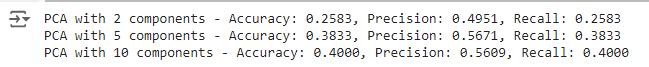
**recall = recall\_score(y\_test, y\_pred\_pca, average='weighted')**

**accuracies.append(accuracy)**

**precisions.append(precision)**

**recalls.append(recall)**

**print(f"PCA with {n\_comp} components - Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}")**

****

**# Step 7: Visualization - Accuracy vs. Number of Components**

**plt.figure(figsize=(10, 6))**

**plt.plot(components, accuracies, marker='o', label='Accuracy')**

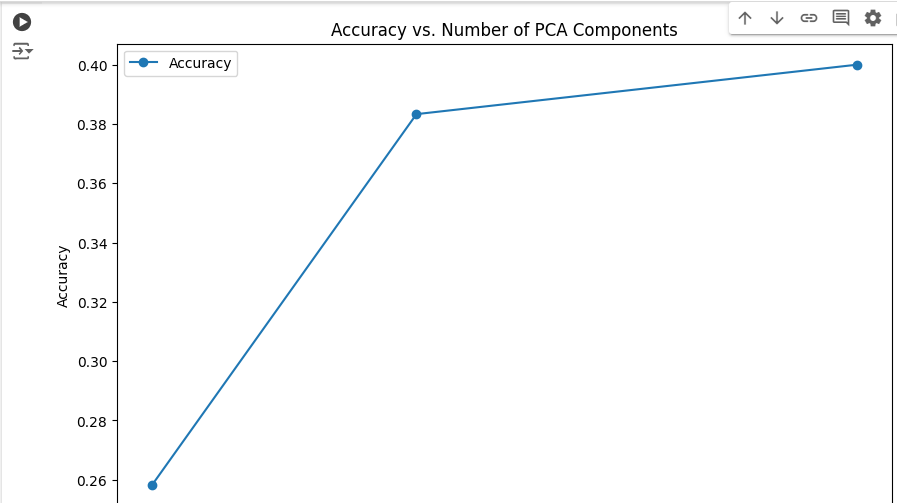
**plt.xlabel('Number of PCA Components')**

**plt.ylabel('Accuracy')**

**plt.title('Accuracy vs. Number of PCA Components')**

**plt.legend()**

**plt.show()**

****

**CODE REPOSITORY:**

**https://github.com/Roda1458/PCA**