PATTERN RECOGNITION AND MACHINE LEARNING
LAB REPORT
ASSIGNMENT-3

Lavangi Parihar
B22EE044

Question 1 Perceptron

Task 0 Generating Dataset

- Perceptron Function:
  - The perceptron function is defined to take an input vector `x` and generate binary labels (0 or 1) based on a randomly initialized set of weights.
  - The weights are drawn from a uniform distribution between -1 and 1.
  - The result is calculated using the dot product of weights and input, and a binary output is generated based on whether the result is greater than 0.

- Dataset Generation:
  - 5000 vectors of dimension 4 are generated using NumPy's random integer function.
  - The perceptron function is applied to each vector to obtain corresponding labels.

- Dataframe and txt Export:
  - The data is organized into a Pandas DataFrame with columns named 'x1', 'x2', 'x3', 'x4', and 'label'.
  - The DataFrame is then exported to a txt file named 'data.txt'.
  - The data is further split into training (80%) and testing (20%) sets, and both sets are saved as txt files named 'train_data.txt' and 'test_data.txt', respectively.

- Dataset Splitting:
  - The dataset is split into three different proportions for training and testing: 50% - 50%, 20% - 80%, and 70% - 30%.

Task 1 Training Function

- Weight Initialization:

  - The `initialize_weights` function was defined to generate random initial weights for the perceptron. The weights include an additional bias term.

- Threshold Activation Function:

  - The `threshold_function` was implemented to classify the output of the perceptron based on a threshold. If the net input is greater than 0, the output is 1; otherwise, it is 0.

- Prediction Function:
 - The `predict` function was implemented to calculate the net input, apply the threshold function, and produce the final prediction.

- Weight Update Function:
 - The `update_weights` function was defined to adjust the weights based on prediction errors using the perceptron learning rule.

- Perceptron Training:
  - The training process involves iterating through the dataset, making predictions, updating weights, and checking for convergence.
  - The maximum number of epochs (`max_epochs`) and the learning rate (`learning_rate`) are adjustable parameters.

- Weights generated:
```
Maximum number of epochs reached. Model may not have converged.
weights: [ 1.02494968  0.58220288 -0.3912704  -0.84272376
-0.12733051]
```

Task 2 Test function
- A `test` function was implemented to apply the trained weights to the test data and generate predicted labels.
- The function iterates through each test example, inserts a bias term, and uses the `predict` function to obtain the predicted label.

Task 3 Calculating Accuracy
- A function named `calculate_accuracy` was implemented to compute the accuracy of the perceptron's predictions.

- Accuracy was calculated for 20%, 50% and 70% train-test split of the training set.

- Results:The perceptron's accuracy on the test dataset was calculated, and the result is as follows:

Percentage Split	Accuracy
20%	90%
50%	91.4%
70%	91.6%

Question 2 Eigenfaces

Task 1 Data Pre-Processing

- Dataset Loading:
  - The `fetch_lfw_people` function from scikit-learn was utilized to load the LFW dataset.
  - The parameter `min_faces_per_person=70` was set to ensure that only individuals with at least 70 images are included in the dataset.
  - The images were resized to 40% of their original dimensions using the `resize=0.4` parameter.

- Dataset Information:
  - The shape of the images array was introspected to understand the dimensions of the images.
  - The data matrix `X` and target vector `y` were extracted for machine learning purposes.
  - Key information such as the total number of samples (`n_samples`), the number of features (`n_features`), the target names, and the number of classes (`n_classes`) were printed.

- Results:

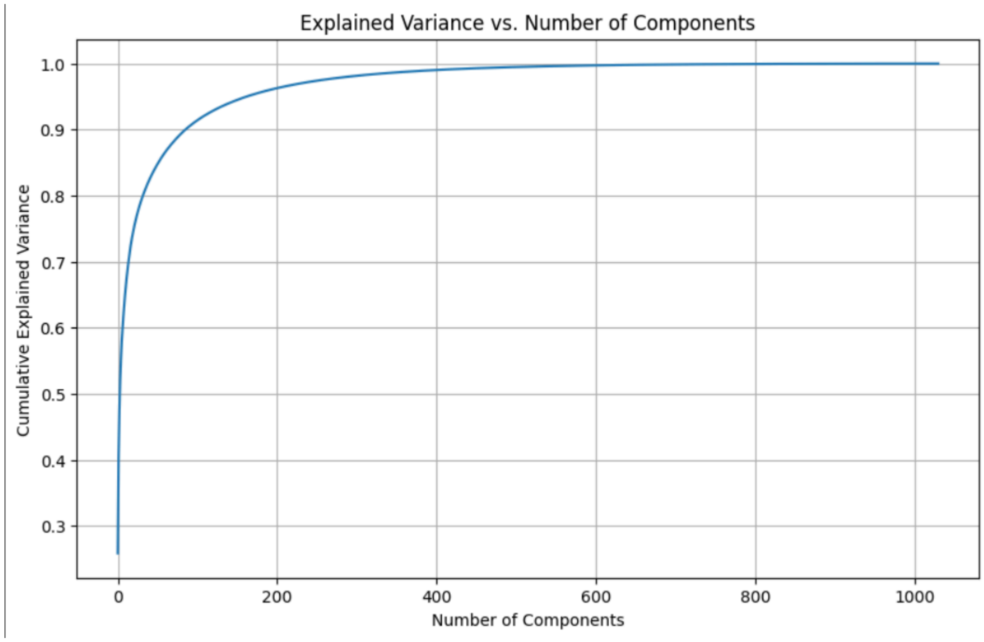
```
Total dataset size:
n_samples: 1288
n_features: 1850
n_classes: 7
```

Task 2 Eigenface Implementation

- PCA Fitting:
  - The `PCA` class from scikit-learn was utilized to fit the PCA transformation on the training data (`X_train`).
  - The explained variance ratio for each principal component was computed during the fitting process.

- Explained Variance Visualization:
  - A plot was created to visualize the cumulative explained variance against the number of components.
  - The x-axis represents the number of components, and the y-axis represents the cumulative explained variance.

- The plot assists in identifying an appropriate number of components to retain a desired amount of information.



- Choosing Number of Components:
  - An appropriate value for `n_components` was chosen based on the plot to retain a high percentage of the total variance. The threshold was set at 95%, but this value can be adjusted as needed.
  - Result : `n_components 163`

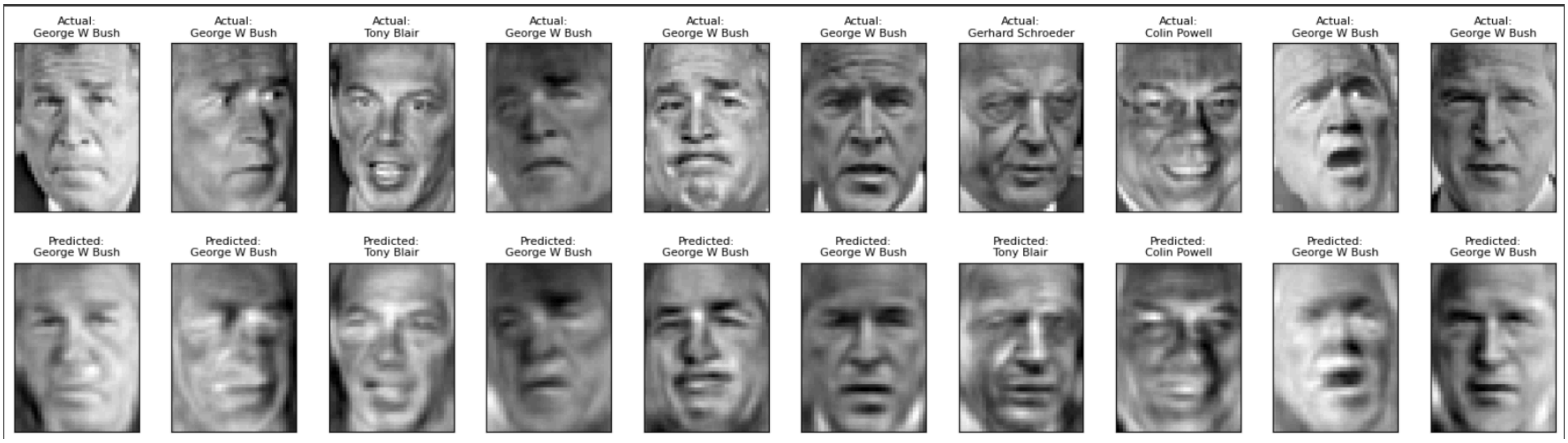
- Applying PCA Transformation:
  - PCA was applied to both the training and testing sets using the chosen `n_components`.
  - Transformed datasets (`X_train_pca` and `X_test_pca`) were obtained for further analysis.

Task 3 Model Training

- KNN Classifier Prediction:
  - The KNN classifier, trained on the reduced feature space (`X_train_pca`), was used to predict labels for the test set (`X_test_pca`).
  - The predicted labels were stored in the `y_pred` variable.

- Data Reconstruction for Visualization:
  - The PCA transformation was inverted using `pca.inverse_transform` to reconstruct the data in the original feature space.
  - The reconstructed images were reshaped for visualization (`X_test_reconstructed_images`).

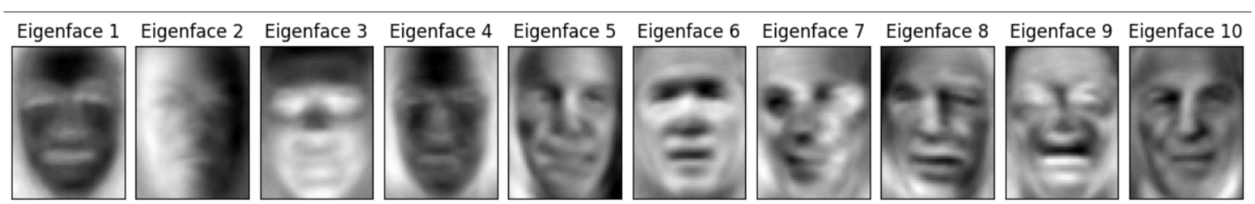
- Visualization:
  - Original and reconstructed faces were plotted side by side for a visual comparison.
  - Actual and predicted names were displayed for each face to assess the accuracy of the KNN classifier.



Task 4 Model Evaluation

- Accuracy Calculation:
  - The accuracy of the KNN classifier on the test set was calculated using the `accuracy_score` function from scikit-learn.
  - This model with the provided n_components value is 63% efficient

- Visualization of Eigenfaces:
  - A subset of the principal components, referred to as Eigenfaces, was visualized to gain insights into the features contributing to the variance in the dataset.



- Analysis of Misclassifications:
  - The indices of misclassified instances were identified, and the actual and predicted

Confusion Matrix:

```
[[  7   1   0   2   0   0   0]
 [  2  29   2   4   0   0   0]
 [  0   0  16   1   0   0   0]
 [  2   2   1 133   2   1   0]
 [  0   1   0   3  22   0   1]
```

[0 1 0 1 0 15 0]
[0 0 0 1 2 0 18]]

Misclassified Images:
Actual: Ariel Sharon, Predicted: Colin Powell
Actual: George W Bush, Predicted: Colin Powell
Actual: George W Bush, Predicted: Colin Powell
Actual: George W Bush, Predicted: Colin Powell
Actual: George W Bush, Predicted: Colin Powell

Task 5 Experimenting with different values of n_components

- A loop was implemented to iterate over a range of `n_components` values (10, 20, ..., 100).
- For each iteration, PCA was applied to both the training and testing sets with the current `n_components` value.
- A KNN classifier was then trained on the reduced feature space, and predictions were made on the test set.
- The accuracy of the classifier was calculated and printed for each iteration.

Results:

Accuracy with 10 components: 0.42
Accuracy with 20 components: 0.53
Accuracy with 30 components: 0.59
Accuracy with 40 components: 0.63
Accuracy with 50 components: 0.63
Accuracy with 60 components: 0.63
Accuracy with 70 components: 0.64
Accuracy with 80 components: 0.63
Accuracy with 90 components: 0.63
Accuracy with 100 components: 0.63