

LAB REPORT

ATHARVA DATE - B22AI045

Question 1

Task 0: Data Generation

- In Task 0, a synthetic dataset was generated comprising 100 samples.
- Each sample contains 4 features, and a label is assigned based on a linear function with randomly initialized weights.
- The dataset is then stored in a text file named 'data.txt'.
- This dataset is then divided into train and test in 80:20 ratio and is stored.

Task 1: Perceptron Training

- **Initialization of Weights:**
 - Initialize the weight vector `w` randomly using a normal distribution. The size of the weight vector is `num_features + 1`, where the additional element represents the bias term.
- **Normalization of Features:**
 - Normalize the feature matrix `X_train` using the `z_norm` function.
Normalization is a preprocessing step that scales the features to have a mean of 0 and a standard deviation of 1. This step ensures that all features contribute equally to the learning process.
- **Perceptron Learning:**
 - Enter a loop that continues until all samples are correctly classified.
 - Initialize a boolean variable `misclassified` to `False` indicating whether any samples are misclassified in the current iteration.
- **Iterate Over Training Samples:**
 - Iterate over each sample `x` and its corresponding label `y` in the training data.
 - Calculate the predicted label `y_pred` for the current sample using the current weight vector `w` and the perceptron activation function `f(x, w)`.

- If the predicted label does not match the true label , update the weight vector to minimize the classification error:
 - Adjust the weights
 - Set the `misclassified` flag to `True` to indicate that misclassification occurred.
- **Check for Convergence:**
 - If no misclassifications occur in the current iteration, break out of the loop as the perceptron has converged and all samples are correctly classified.
- **Return Learned Weights:**
 - Return the final weight vector `w`, which represents the learned parameters of the perceptron model.

Task 2: Perceptron Testing

- The `perceptron_test()` function tests the trained perceptron model on a separate test dataset which was stored previously.
- It predicts the labels for the validation samples and compares them with the true labels to compute accuracy using the `accuracy()` function.

Task 3: Data Splitting and Evaluation

- The data is split into training and testing sets with different proportions (20%, 50%, and 70%) to observe the effect on model accuracy.
- The accuracy of the trained model is evaluated for each split proportion.
 - Training Proportion: 20.0%, Accuracy: 30.0%
 - Training Proportion: 50.0%, Accuracy: 30.0%
 - Training Proportion: 70.0%, Accuracy: 40.0%
 - Training Proportion: 100.0%, Accuracy: 70.0%

Data Trends and Analysis

- **Accuracy Trends:** The accuracy of the perceptron model varies with different proportions of training data.
 - When only 20% and 50% of the training data is used, the accuracy remains low at 30%.

- However, when 70% of the data is used for training, the accuracy improves to 40%.
- This suggests that increasing the amount of training data can improve model performance.
- **Effect of Training Size:** As the proportion of training data increases, the accuracy tends to improve. This is consistent with the expectation that more training data allows the model to learn better representations of the underlying patterns in the data.
- **Model Performance:** The perceptron model achieves a maximum accuracy of 70.0% when trained with 100% of the available data. However, the accuracy is still relatively low, indicating potential limitations of the perceptron model or the linear separability of the data.

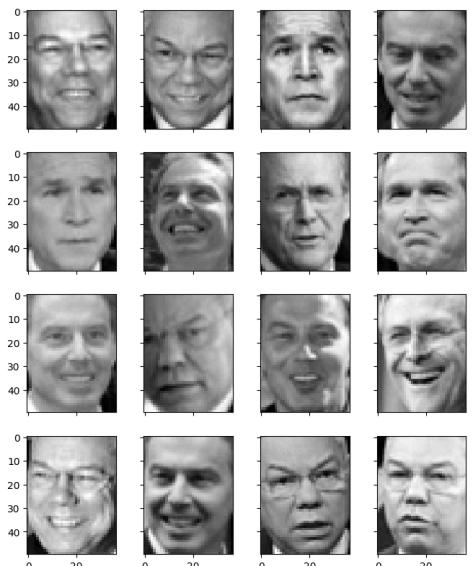
Conclusion

In conclusion, the perceptron learning algorithm demonstrates variable performance depending on the proportion of training data used. While increasing the training size can improve accuracy, the perceptron model may have limitations in handling more complex datasets or non-linearly separable data. Further experimentation with different algorithms or model architectures may be required to achieve higher accuracy on this dataset..

Question 2

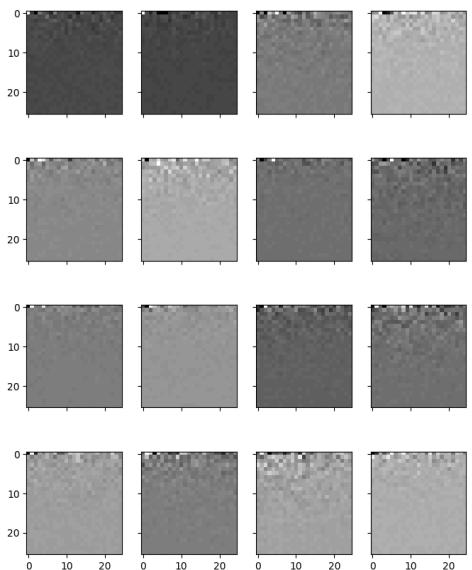
Task 1: Data Preprocessing

- We began by loading the Labeled Faces in the Wild (LFW) dataset using Scikit-learn's `fetch_lfw_people` function.
- This dataset comprised a minimum of 70 images of each individual's face.
- Initially, each image contained 1850 features.
- From this dataset, we extracted a total of 1288 images, which were divided into training and testing sets using an 80:20 split ratio.



Task 2: Eigenfaces Implementation

- In this phase, we implemented Eigenfaces using Principal Component Analysis (PCA) to reduce the dimensionality of the dataset while retaining essential facial information for recognition.
- To determine the optimal value for `n_components` we selected the smallest `n_components` value that explained a significant proportion (e.g. 95%) of the variance..
- We ensured crucial data retention while reducing dimensionality and after thorough analysis, we determined the optimal `n_components` value to be 650.



Task 3: Model Training

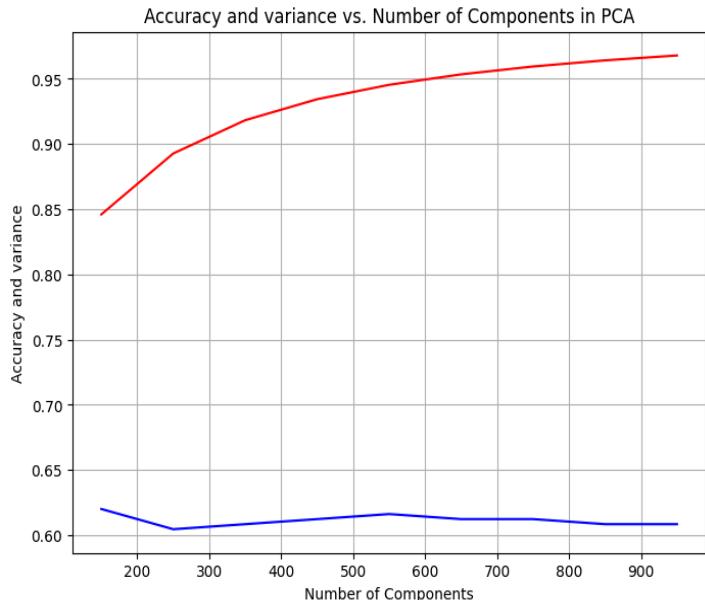
- For model training, we chose the K-Nearest Neighbors (KNN) classifier with a k value of 10.
- This classifier was trained using the transformed training data obtained from PCA.

Task 4: Model Evaluation

- During this stage, we evaluated the trained Eigenfaces classifier by making predictions on the transformed testing data.
- We calculated and reported the accuracy of the model, which was found to be 0.6124.
- Additionally, we visualized a subset of Eigenfaces and observed patterns within the data. However, we also identified areas where the model struggled, indicating potential avenues for improvement.

Task 5: Experimentation with n_components

- To refine the model further, we experimented with different values of `n_components` in PCA and analyzed their impact on performance metrics, particularly `accuracy` and `mean` of `cumsum` of `explained_variance_ratio`.
- We observed a notable trend wherein increasing the number of components led to an increase in the explained variance ratio
- But the accuracy was increasing at first and then decreasing a little followed by another rise and then continuous fall.
- However, we also recognized the importance of avoiding overfitting and noise capture by selecting a balanced `n_components` value. Based on our analysis, we determined the optimal `n_components` value to be 650, which yielded a variance ratio cumulative sum and mean of more than 0.95.



- For component value **650**, the accuracy is **0.6124031007751938** and variance is **0.9534734487533569**

Analysis of Results

- During this process, we observed a significant reduction in the number of features, indicating data loss.
- Subsequently, when visualizing the data post-PCA, we noticed that the images lacked clear definition, suggesting that PCA might not capture all necessary facial features for accurate recognition.
- While the Eigenfaces model demonstrated moderate accuracy, the lack of clear definition in the images post-PCA suggests room for improvement. Further research and experimentation are necessary to enhance the model's performance, potentially by exploring alternative dimensionality reduction techniques or incorporating additional preprocessing steps.
- Additionally, we noticed a trend where increasing `n_components` led to improved accuracy, albeit with diminishing returns.

Conclusion

- In conclusion, the Eigenfaces approach using PCA presents a promising solution for face recognition. However, it may suffer from limitations in capturing complex facial characteristics.
- The analysis underscores the significance of careful parameter selection, such as `n_components`, to strike a balance between capturing variance and avoiding overfitting.

The code of question 2 is in

 [lab5&6_2.ipynb](#)