24-678: Computer Vision for Engineers

Carnegie Mellon University

PS8

Due: **12/15/2024 (Sun) 5 PM @ Gradescope**

Issued: **11/20/2024 (Wed)**

Weight: **5% of total grade**

Note:

PS8 Finding Eigenfaces Using PCA

The idea of eigenfaces was first proposed by Sirovich and Kirby to solve the computer vision problem of human face recognition. The eigenfaces, the name of a set of eigenvectors, form a set of bases. A linear combination of these bases can be used to reconstruct the faces. By this eigenface transformation, we can reduce the dimensionality of the data. Principal Component Analysis (PCA) is a typical way to find these eigenfaces.

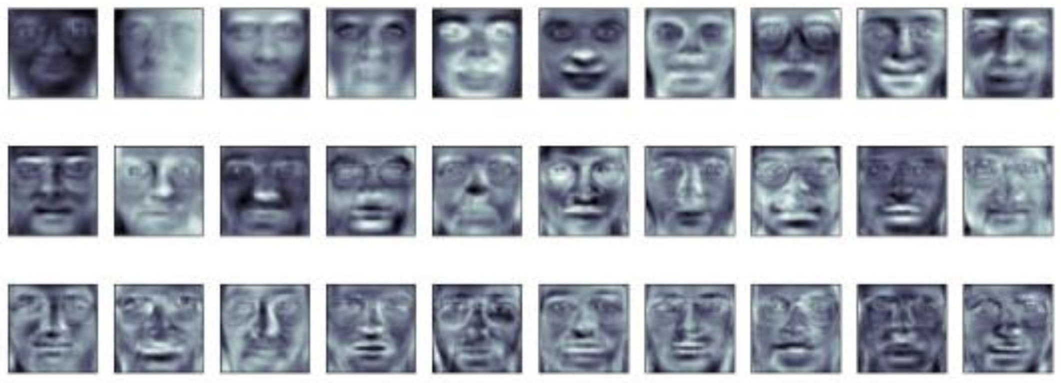

Figure 1: A set of eigenfaces derived from the Olivetti faces dataset using PCA

In this problem set, you work on a dataset of tens of thousands of pictures from thousands of celebrities, find eigenfaces using PCA and visualize your results.

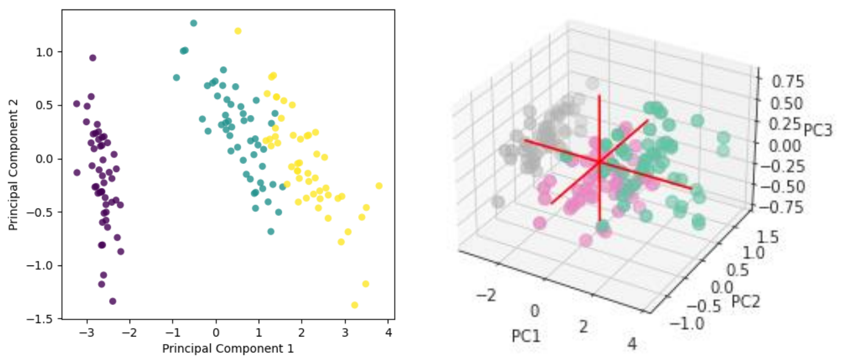

Figure 2: An example of visualizing PCA results in 2D and 3D scatterplots

To implement the described process, you will study the starter code (shown in Appendix A) and complete functions of loading data, perform PCA, and plot eigenfaces listed as TODO 0~4 in the starter code. Each task comes with a detailed explanation of what you need to do.

As the process involves producing multiple plots, you can do your work using a Jupyter Notebook. You can also change function parameters, return values, or define new functions. But please reformat it into a .py python file identical to the starter file upon submission.

Submission

To prepare for the submission of your work on Gradescope, create:

(1) a folder called "ps8," that contains the following files:

- source code file(s) (shared with ps8a-2)
- "readme.txt" file that includes:
    - Operating system
    - IDE you used to write and run your code
    - The number of hours you spent to finish this problem

(2) a PDF file with all the results asked in TODO 0~4 in the code, which includes:

- shape of the input image
- a histogram of the number of images per person, top 10 person only
- a 3-dimensional plot of the transformed training and nonface data, represented by the first 3 principal components
- a plot of the first 8 eigenfaces (principal components)

(Include, if any, the mathematical derivation and/or description of your method in the PDF file. Handwritten notes should be scanned and included in the PDF file.)

Submit your work on Gradescope

Submit two files on Gradescope – replace "andrewid" with your own Andrew ID:

(1) **andrewid-ps8-files.zip** – this ZIP file should contain one folder, "ps8", and all the files requested in PS8.

(2) **andrewid-ps8-report.pdf** – this PDF file serves as the report of your work, and it should contain the printouts and screenshots of all the files in the "ps8" folder. (Include, if any, the mathematical derivation and/or description of your method in the PDF file. Handwritten notes should be scanned and included in the PDF file.)

Please organize pages with section titles and captions to make the report easy to read.

Appendix A

```
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn import decomposition
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.utils.fixes import loguniform
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import ConfusionMatrixDisplay

import re
import random
from os import listdir


"""
TODO 0: Find out the image shape as a tuple and include it in your report.
"""
IMG_SHAPE = ()


def load_data(data_dir, top_n=10):
    """
    Load the data and return a list of images and their labels.

    :param data_dir: The directory where the data is located
    :param top_n: The number of people with the most images to use

    Suggested return values, feel free to change as you see fit
    :return data_top_n: A list of images of only people with top n number of images
    :return target_top_n: Corresponding labels of the images
    :return target_names: A list of all labels(names)
    :return target_count: A dictionary of the number of images per person
    """
    # read and randomize list of file names
    file_list = [fname for fname in listdir(data_dir) if fname.endswith('.pgm')]
    random.shuffle(file_list)
    name_list = [re.sub(r'_\d{4}.pgm', '', name).replace('_', ' ') for name in file_list]

    # get a list of all labels
    target_names = sorted(list(set(name_list)))

    # get labels for each image
    target = np.array([target_names.index(name) for name in name_list])

    # read in all images
    data = np.array([cv2.imread(data_dir + fname, 0) for fname in file_list])
    """
    TODO 1: Only preserve images of 10 people with the highest occurence, then plot
            a histogram of the number of images per person in the preserved dataset.
            Include the histogram in your report.
    """
    # YOUR CODE HERE
    # target_count is a dictionary of the number of images per person
    # where the key is an index to label ('target'), and the value is the number of images
    # Try to use sorted() to sort the dictionary by value, then only keep the first 10 items of the output list.
    target_count = {}

    # data_top_n is a list of labels of only people with top n number of images
    target_top_n = []
    data_top_n = []

    # You can plot the histogram using plt.bar()
    # autofmt_xdate() is also useful for rotating the x-axis labels

    return data_top_n, target_top_n, target_names, target_count


def load_data_nonface(data_dir):
    """
    Your can write your functin comments here.
    """

    """
    TODO 2: Load the nonface data and return a list of images.
    """
    # YOUR CODE HERE
    # Take a look at the load_data() function for reference
    file_list = []
    data = np.array([])

    return data


def perform_pca(data_train, data_test, data_noneface, n_components, plot_PCA=False):
    """
    Your can write your functin comments here.
    """

    """
    TODO 3: Perform PCA on the training data, then transform the training, testing,
            and nonface data. Return the transformed data. This includes:
            a) Flatten the images if you haven't done so already
            b) Standardize the data (0 mean, unit variance)
            c) Perform PCA on the standardized training data
            d) Transform the standardized training, testing, and nonface data
```

```
        e) Plot the transformed training and nonface data using the first three
          principal components if plot_PCA is True. Include the plots in your report.
        f) Return the principal components and transformed training, testing, and nonface data
    """
    # YOUR CODE HERE
    # You can use the StandardScaler() function to standardize the data
    data_train_centered = None
    data_test_centered = None
    data_noneface_centered = None

    # You can use the decomposition.PCA() and function to perform PCA
    # You can check the example code in the documentation using the links below
    # https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
    pca = None

    # You can use the pca.transform() function to transform the data
    data_train_pca = None
    data_test_pca = None
    data_noneface_pca = None

    # You can use the scatter3D() function to plot the transformed data
    # Please not that 3 principal components may not be enough to separate the data
    # So your plot of face and nonface data may not be clearly separated
    # if plot_PCA:

    return pca, data_train_pca, data_test_pca, data_noneface_pca


def plot_eigenfaces(pca):
    """
    TODO 4: Plot the first 8 eigenfaces. Include the plot in your report.
    """
    n_row = 2
    n_col = 4
    fig, axes = plt.subplots(n_row, n_col, figsize=(12, 6))
    for i in range(n_row * n_col):
        # YOUR CODE HERE
        # The eigenfaces are the principal components of the training data
        # Since we have flattened the images, you can use reshape() to reshape to the original image shape
        pass
    plt.show()


def train_classifier(data_train_pca, target_train):
    """
    TODO 5: OPTIONAL: Train a classifier on the training data.
        SVM is recommended, but feel free to use any classifier you want.
        Also try using the RandomizedSearchCV to find the best hyperparameters.
        Include the classifier you used as well as the parameters in your report.
        Feel free to look up sklearn documentation and examples on usage of classifiers.
    """
    # YOUR CODE HERE
    # You can read the documents from sklearn to learn about the classifiers provided by sklearn
    # https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
    # If you are using SVM, you can also check the example below
    # https://scikit-learn.org/stable/modules/svm.html
    # Also, you can use the RandomizedSearchCV to find the best hyperparameters
    clf = None
    return clf


if __name__ == '__main__':
    """
    Load the data
    Face Dataset from https://conradsanderson.id.au/lfwcrop/
    Modified from original dataset http://vis-www.cs.umass.edu/lfw/
    Noneface Dataset modified from http://image-net.org/download-images
    All modified datasets are available in the Box folder
    """
    data, target, target_names, target_count = load_data('lfw_crop/', top_n=10)
    data_train, data_test, target_train, target_test = train_test_split(data, target, test_size=0.25, random_state=42)
    data_noneface = load_data_nonface('imagenet_val1000_downsampled/')
    print("Total dataset size:", data.shape[0])
    print("Training dataset size:", data_train.shape[0])
    print("Test dataset size:", data_test.shape[0])
    print("Nonface dataset size:", data_noneface.shape[0])

    # Perform PCA, you can change the number of components as you wish
    pca, data_train_pca, data_test_pca, data_noneface_pca = perform_pca(
        data_train, data_test, data_noneface, n_components=3, plot_PCA=True
    )

    # Plot the first 8 eigenfaces. To do this, make sure n_components is at least 8
    plot_eigenfaces(pca)

    """
    Start of PS 8-2
    This part is optional. You will get extra credits if you complete this part.
    """

    # Train a classifier on the transformed training data
    classifier = train_classifier(data_train_pca, target_train)

    # Evaluate the classifier
    pred = classifier.predict(data_test_pca)
    # Use a simple percentage of correct predictions as the metric
    accuracy = np.count_nonzero(np.where(pred == target_test)) / pred.shape[0]
    print("Accuracy:", accuracy)
    """
    TODO 6: OPTIONAL: Plot the confusion matrix of the classifier.
```

```
        Include the plot and accuracy in your report.
        You can use the sklearn.metrics.ConfusionMatrixDisplay function.
"""
# YOUR CODE HERE

"""
TODO 7: OPTIONAL: Plot the accuracy with different number of principal components.
        This might take a while to run. Feel free to decrease training iterations if
        you want to speed up the process. We won't set a hard threshold on the accuracy.
        Include the plot in your report.
"""
n_components_list = [3, 5, 10, 20, 40, 60, 80, 100, 120, 130]
# YOUR CODE HERE
```