Week-12 activities

The task (short version)

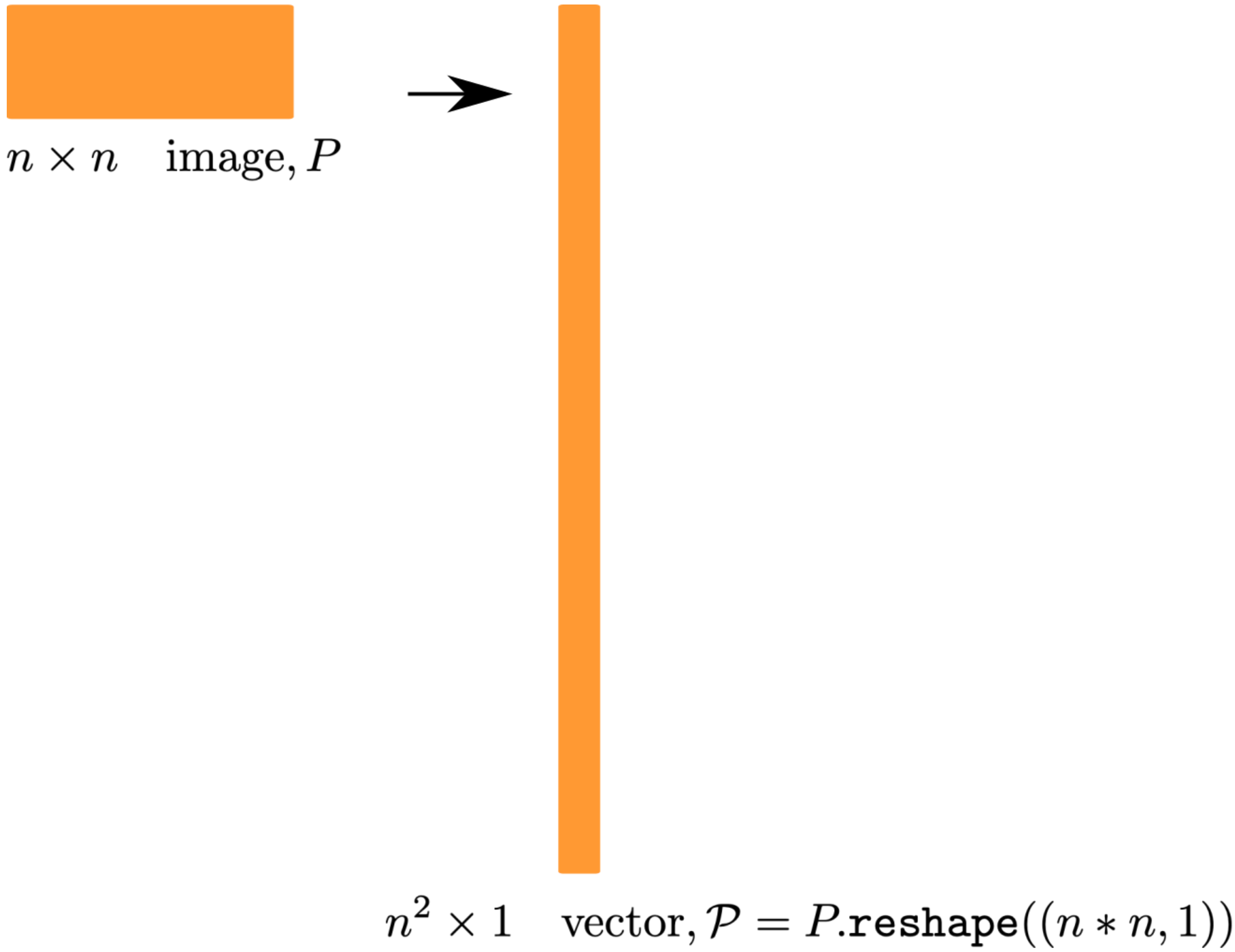
- Age prediction from facial images

> In this activity, you are going to develop a program (in python) that would be able to guess age of a person given his/her photo. Roughly speaking, it is expected that you will first do the principal component analysis (PCA) to perform dimensionality reduction of the given dataset. Then, train a linear regression model on the reduced-dimension dataset to learn their age. Done!

Principal Component Analysis (PCA)

Problems arise when we try to perform learning tasks involving data samples in higher-dimensional space; no thanks to the notorious phenomenon called `curse of dimensionality` (https://en.wikipedia.org/wiki/Curse_of_dimensionality). Significant improvements can be achieved by first mapping the data samples into a lower-dimensioal space. Principal Component Analysis (PCA) is a great method to do it, i.e., `extracting` lower dimensional features represented by the selected few principal components, or in the computer vision lingo we call them `eigenfaces`.

Suppose P is an $\mathrm{n}^2 \times 1$ vector corresponding to an input $\mathrm{n} \times \mathrm{n}$ face image, P.



{ width=250px }

Now, the following steps to compute the eigenfaces (i.e., the principal components) are:

1. Obtain the 2D face images, $P_1, P_2, \cdots, P_m$ which are the m training faces. All faces must have the same resolution.

2. Represent each image P_i as a vector $\mathcal{P}_i$ as show in the figure.

3. Compute the average face vector (i.e., the `mean` face (**pun intended**)) of shape $n^2 \times 1$: $\Psi = \frac{1}{m} \sum_{i=1}^{m} \mathcal{P}_i$

4. Subtract the `mean` face from all the original faces. The process is known as `centerizing data samples`: $\Phi_i = \mathcal{P}_i - \Psi$. This too is a $n^2 \times 1$ vector.

5. Now, assemble all centered faces into a matrix, A as a $m \times n^2$ matrix:

$$A = \begin{bmatrix} \Phi_1^T & \Phi_2^T & \vdots & \Phi_m^T \end{bmatrix}$$

6. Now, compute the covariance matrix, C as a $n \times n$ matrix:

$$C = \frac{1}{m-1} \sum_{i=1}^{m} \Phi_i^T \Phi_i = \frac{1}{m-1} A^T A$$

7. Then, compute the eigenvectors, u_i of $A^T A$. The dimension of each of the eigenvectors will be n^2.

1. **Gotcha**: You may want to check the dimension of $A^T A$ which can be huge! Your program might get crashed due to memory allocation issue. In that case, find the eigenvectors, v_i of AA^T instead. It can be proved that both $A^T A$ and AA^T has the same eigenvalues and their eigenvectors are related through this formula: $u_i = Av_i$. The proof can be found here: http://www.vision.jhu.edu/teaching/vision08/Handouts/case_study_pca1.pdf.

8. Keep only K eigenvectors with $K << n^2$, corresponding to the K largest eigenvalues. That's it! Since, K is chosen much less than n^2 (higher dimension).

   1. You now have the K eigenfaces (i.e., the K eigenvectors, or principal components).
   2. Since, each of the K eigenfaces are n^2 dimensional vectors, out of curiosity you may want to reshape all the eigenfaces to $n \times n$ and show them as images. You may be surprised. They may look like ghosts.
   3. This K eigenfaces will act as a projection matrix to project any high dimensional (i.e., $n^2 = n \times n$ dimensional) image into K dimensional image (!). You may not want to treat this K dimensional image as image, as they may bear no pixel by pixel values. But, don't get disappointed yet: this K numbers can be sufficient to retrieve/reconstruct the original images. After reconstruction you'd appreciate with this concept of feature extraction and be more motivated to learn that you do not need all pixel-by-pixel information to call an image `image`. This lower-dimensional entities can now be used in other learning algorithms whether your fancy algorithm was strugglying with higher dimensionality of the samples.

Transform higher-dimensional images into lower-dimensional entity

The set of K eigenfaces (i.e., eigenvectors) you got will now act as the projection matrix, $Z \in \mathbb{R}^{n^2 \times K}$. How to do that?

- Given an image, X of size $n \times n$, reshape it into vector x of size $n^2 \times 1$
- Centerize x by subtracting the `mean face` from it. $c = x - \Psi$
- Projected image (entity), $y = Z^T c$, where the size of y would be $K \times 1$.

Reconstruction of original images from the lower-dimensional entity

Reconstruction of the original image from the projected lower dimensional image (entity) is the reverse of the project:

- Project y onto the projection direction again: $c = Zy$, where size of c would be $n^2 \times 1$.
- Add the `mean face`: $x = c + \Psi$
- Reshape x into $n \times n$.

Task 1 : First solve at Kaggle

- Participate at this Kaggle competition by making submssions and trying to improve the model performance. Here is the invitation to the competition: https://www.kaggle.com/competitions/fall-24-age-prediction-from-images-week-12

Task 2 : Try to follow the following steps in preparing your solution

1. Make sure you create a separate jupyter notebook for this task.

2. Given the dataset as described in Task 1: `Please check Kaggle :: Data` section for obtaining the dataset,
    1. Shuffle the dataset randomly and then perform a 80/20 split for training and test.
3. Compute principal components (i.e., eigenfaces) from the training set by following the algorithm outlined at the beginning of this document. **Please note**: You are not allowed to call a library function that directly computes principal components. But, you are allowed to call library functions to compute the eigenvalues and eigenvectors of a square matrix.
4. Draw a `scree plot` to choose the best value for K that denotes number of principal components to keep.
5. Show the top 20 eigenfaces in a 10×10 grid.
6. Considering the chosen K value above, project the training and test images on to the eigenfaces to reduce dimensionality.
7. Perform stochastic gradient descent (SGD) based linear regression, which was outlined in class, to predict `age`.
8. Please do a trial-and-error search to tune the hyper-parameters of the SGD (e.g., number of epochs, learning rate, etc), and make a note on this parameter tuning procedure via plots.
9. Apply the model to evaluate on the test set. Please make a note on the RMSE value.
10. Repeat steps 3--9 four more times, and finally report mean and standard deviation of RMSE.