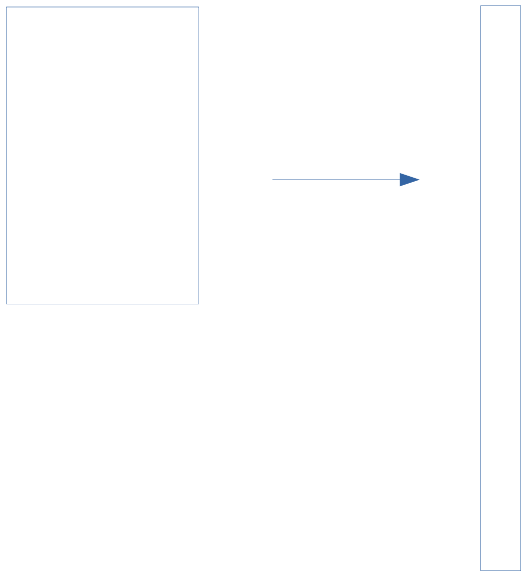
Programming Assignment 2

Machine Learning, Spring 2019

Age prediction from facial images

In this assignment 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!

Backgrounds on PCA



$n^2 \times 1$ vector, Γ

Problems arise when performing learning in a higher-dimensional space due to the phenomenon known as "Curse of the dimensionality" (https://en.wikipedia.org/wiki/Curse_of_dimensionality). Significant improvements can be achieved by first mapping the data into a lower-dimensional space. And you already have heard about principal component analysis, which is a fantastic method to do the same. In image learning paradigm, principal components obtained from the given images are affectionately called the "eigenfaces".

Suppose Γ is an $n^2 \times 1$ vector, corresponding to an $n \times n$ face image, I.
Now the steps to compute the eigenfaces (i.e., the principal components):
Step 1: Obtain the 2D face images, $I_{1,} I_{2,} \ldots, I_m$ (the training faces). All faces must be of the same resolution.
Step 2: Represent every image I_i as a vector Γ_i (as shown in the figure on the previous page)

Step 3: Compute the average face vector, Ψ, dimension of which is $n^2 \times 1$:

$$\Psi = \frac{1}{m} \sum_{i=1}^{m} \Gamma_i$$

Step 4: Subtract the mean face from the original faces. This step is very essential, and is called to centerize the data.

$\Phi_i = \Gamma_i - \Psi$, It is also a $n^2 \times 1$ dimensional vector.

Step 5: If the matrix, A is represented as:

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

, and it is certainly an $m \times n^2$ matrix, then compute the covariance matrix, C:

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

, which is an $m \times m$ matrix.

Step 6: Compute the eigenvectors u_i of $A^T A$. The dimension of each of the eigenvector will be n^2. For some reason, if you see that the dimension of $A^T A$ becomes very large, find the eigenvectors, v_i of AA^T instead. It can be proved that $A^T A$ and AA^T have 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)

Step 7: Keep only K eigenvectors, corresponding to the K largest eigenvalues. Now you have it! You have got K eigenfaces (a.k.a., principal components). Since each of the K eigenfaces are essentially n^2 dimensional vectors, out of curiosity if you do reshape the eigenfaces to $n \times n$ and display them as images, you will be astonished (or get scared!! haha) to see the eigenfaces. They may look like ghosts! Rumor has it: you will be able to recover a human from these ghosts. Just joking! But, I would like to make a note on a property of PCA that makes it one of the most beautiful (and extraordinary) algorithm. Each of the centered image, Φ_i in the training dataset can be represented as a linear combination of the best K (ghosts) eigenvectors (i.e., eigenfaces):

Illustration 1: An example of 4 eigenfaces

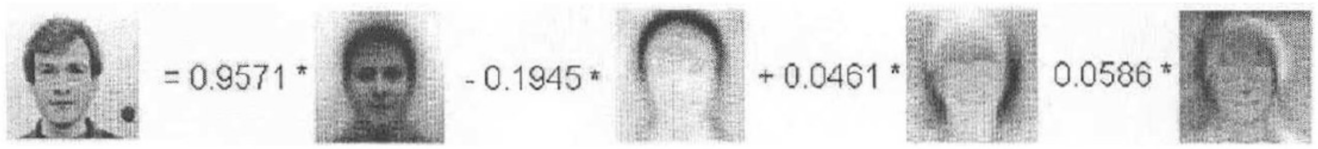


Illustration 2: Any image from the training dataset can be represented as a linear combination of the best K eigenfaces.

Step 8: So, let's project all the original faces (after centering) from the training dataset onto this eigenfaces direction:

$$\hat{\Phi}_i = \sum_{j=1}^{K} w_j^i u_j = \sum_{j=1}^{K} u_j^T \Phi_i u_j$$

Here, we are projecting our original faces onto a subset of K eigenfaces, thus reducing each image from n^2 dimensions down to a vector Ω of only K dimensions. Each of the normalized training face Φ_i is projected onto the eigenfaces by a vector, $\Omega^i = \begin{bmatrix} w_1^i \\ w_2^i \\ \vdots \\ w_K^i \end{bmatrix} = \begin{bmatrix} u_1^T \Phi_i \\ u_2^T \Phi_i \\ \vdots \\ u_K^T \Phi_i \end{bmatrix}$.

The images can then be reconstructed in n^2 dimensions from the K dimensional Ω encodings, with some loss in accuracy, using the formula above, or if you need more elaboration, here it is:

$$Reconstructured\,image = \hat{\Phi}_i = (\Omega_1^i u_1 + \Omega_2^i u_2 + \cdots + \Omega_K^i u_K)$$

More resources on this topic can be found here: https://mikedusenberry.com/on-eigenfaces

Please proceed to the next page to find your assignment! Cheers!!

Now your assignment: *Please show your works (codes+execution results) by leaving/saving the execution results in a submitted jupyter notebook.*

1. From Canvas download **wiki_labeled.zip** (MD5sum: 8af376e8a0f9898d8d3dd5653351070e) , and extract the contents. The wiki_labeled/ directory will contains 60327 facial images kept in 100 folders, naming 00-99. Dimension of each image is 100 pixels by 100 pixels. Also, download the **wiki_labeled.mat** (md5sum: d5b65bb4c4d414a230b60b8025ece276) file, containing meta information of each of the 60327 images:
   - ID: identification number of the subject (starting from 2002)
   - dob: the date of birth of the subject. (It is Matlab's datenum value calculated based on total number of days since January 0, 0000.)
   - dob_str: the DD-MMM-YYYY format dob value.
   - photo_taken: when the photo was taken (only the year value)
   - full_path: directory path, including filename of the image
   - gender: Gender of the subject (0: female, 1: male, NaN if unknown)
   - name: name of the subject
   - face_location: location of the face.
   - face_score: detector score (the higher the better). Inf implies that no face was found in the image, and the face_location then just returns the entire image.
   - second_face_score: detector score of the face with the second highest score. This is useful to ignore images with more than one face. second_face_score is NaN (not a number) if no second face was detected.
   - age: age of the person (in years), and was calculated based on the "dob" value and the "photo_taken" values.

   Hint: To read/extract information from the mat file above, please use the loadmat library from scipy.io in python . [from scipy.io import loadmat]
2. Randomly split the dataset into 80% training and 20% test sets.
3. Compute the principal components (i.e., eigenfaces) from the training dataset by following the steps to compute principal components described in the "**Backgrounds**" section. Please note that: you can not call a library function to directly compute the principal components. For example: the PCA library in sklearn. However, you can use library functions to calculate the eigenvalues and eigenvectors of a square matrix.
4. Draw a scree-plot to choose a best value for K that denotes how many principal components to retain.
5. Show the top 20 ghosts (i.e., eigenfaces) in a 10x10 grid.
6. Considering the chosen K value above, project the training and test images on to the eigenfaces to reduce the dimensionality.
7. Perform Stochastic gradient descent (SGD) based linear regression on the training dataset to learn "age". Please do a trial-and-error search to tune the hyper-parameters of the SGD based linear regression (e.g., number of epochs, learning rate), and make a note why you select a specific set of hyper-parameters. **Once again: Library function that does the linear regression for you will NOT be allowed!**

8. Predict the test dataset (from step 2) based on the learned model in step 7, and report Root Mean Square Error (RMSE).
9. From Canvas download the **wiki_judge_images.zip** (md5sum: 551a46ee18286824ad03d9110750da01) containing 2001 facial images (1.png, 2.png, …, 2001.png) of resolution 100x100. You can obtain another meta file, **wiki_judgeX.mat** (md5sum: b7382fd03b8fc05575bb5701c0eec25b) from Canvas containing some information about the images. However, the information does not include "age" field. Using your best regression model, predict age of each of the 2001 facial image, and prepare a **submission.csv** file with the following CSV format:

```
ID,age
1,38.8
2,25
etc.
```

For CSCI-5930 students (and extra-credit for CSCI-4930)

10. Repeat steps 2-8 four more times, and report average RMSE and standard deviation of the RMSE. Please make sure in step 2 you are actually randomly shuffling the dataset before splitting it every time.
11. Draw a plot (K vs RMSE) after experimenting with steps 2-8 by varying values of K. The maximum value K can take is 100x100 = 10000, so please draw the plot for the K values from the set {2, 10, 20, 40, 50, 60, 80, 100, 200}.
12. Submit **submission.csv** into Kaggle (https://www.kaggle.com/c/csci-ml-s19-pa2). During submission of your entry, please mention your model parameters, including K, learning rates, epochs, and ofcourse your student ID and/or your full name.

Extra-credit for both CSCI-4930 and 5930

13. Identify and discard outliers from the given dataset, and repeat steps 2-9.
14. Instead of stochastic gradient descent, implement **mini-batch gradient descent** with batch size of your choice for the tasks above. Please make a note on your batch size in the notebook.