

# Analisi tempo-frequenza e multiscala

Giulio Nenna - 292399

## Homework 1 - Riconoscimento facciale attraverso "Eigenfaces"

Lo scopo dell'Homework è quello di implementare e testare l'algoritmo "*Eigenfaces*" per il riconoscimento facciale attraverso uno script Python. Nell'elaborato verranno presentati gli strumenti teorici a supporto dell'algoritmo, frammenti di codice salienti utilizzati e risultati computazionali ottenuti. I frammenti di codice non sono rappresentativi dello script python completo e funzionante utilizzato nell'esperimento pertanto sono stati utilizzati nomi esplicativi delle variabili. La versione completa del codice è presente al [link](#).

Il dataset presenta 10 ritratti facciali per ciascuno dei 40 soggetti presenti al suo interno. Vengono utilizzate immagini in scala di grigio di dimensione  $m \times n$  pixels con  $m = 112$  e  $n = 92$ . Alcuni esempi di immagini utilizzate sono mostrati in Figura 1



Figure 1: Alcuni esempi di immagini presenti nel dataset

L'algoritmo si distingue per la fase di **Training** e quella di **Testing**.

## 1 Training phase

Le immagini vengono inizialmente importate e vengono generati i dataset di *Training* e *testing*. In particolare nel dataset di training sono presenti le prime 6 immagini per ciascun soggetto mentre in quello di testing le restanti 4. Le immagini sono importate sotto forma di vettori  $\{f_1, f_2, \dots, f_N\}$  in  $\mathbb{R}^{mn}$ . Il dataset di training contiene  $L = Np$  immagini, dove  $p$  è il rapporto tra dimensione del training set e dimensione del test set, nel nostro caso  $p = 0.6$ .

```
1 faces_all = np.zeros([num_subjects*num_faces_per_subject, size])
2 faces_train = np.zeros([train_set_size, size])
3 faces_test = np.zeros([test_set_size, size])
4 print('Importing data...')
5 for i in range(num_subjects): #for each subject
6     subject_faces = np.zeros([num_faces_per_subject, size]) #array containing all
7     faces of the subject
8     for j in range(num_faces_per_subject): #for each face
9         f=path+'s'+str(i+1)+'/'+str(j+1)+'.pgm' #compose filepath
10        img = pgm.read_pgm(f) #read the file (2D array of shape m by n)
11        img = img.reshape(size) #reshape the img into a 1D array
        subject_faces[j,:] = img
```

```

12     faces_all[i*num_faces_per_subject:(i+1)*num_faces_per_subject,:] =
13     subject_faces
14     faces_train[i*int(num_faces_per_subject*train_test_ratio):(i+1)*int(
15     num_faces_per_subject*train_test_ratio),:] = \
16     subject_faces[0:int(num_faces_per_subject*train_test_ratio),:]
17     faces_test[i*int(num_faces_per_subject*(1-train_test_ratio)):(i+1)*int(
18     num_faces_per_subject*(1-train_test_ratio)),:] = \
19     subject_faces[int(num_faces_per_subject*train_test_ratio):
20     num_faces_per_subject,:]

```

Listing 1: Data import

Viene adesso calcolata la faccia media  $\tilde{f} = \frac{1}{L} \sum_{l=1}^L f_l$  e il dataset di training viene centrato rispetto alla faccia media, ottenendo nuovi vettori  $\phi_l = f_l - \tilde{f}$  con  $l \in \{1, \dots, L\}$ .

```

1 L = faces_train.shape[0]
2 mean_face = faces_train.mean(axis = 0) #compute mean face
3 faces_train_center = faces_train - mean_face #centering the dataset (is the matrix
4 Phi^T)

```

Listing 2: Centering data

A questo punto l'idea è quella di trovare una particolare base ortonormale  $\{u_1, \dots, u_K\}$  con  $u_i \in \mathbb{R}^{mn}, i = 1, \dots, K$  tale per cui:

$$\phi_l = \tilde{f} + \sum_{n \geq 1} \alpha_n u_n$$

$$|\alpha_1| \geq |\alpha_2| \geq \dots \geq |\alpha_K|$$

La seconda condizione afferma che la base ortonormale che si sta cercando è tale per cui le componenti sono ordinate per "importanza". Questo concetto è fondamentale per la pratica della riduzione dimensionale: se si riesce a trovare una base di rappresentazione delle immagini per cui le prime componenti sono più importanti delle ultime, allora sarà possibile rappresentare le immagini utilizzando il sottoinsieme delle prime componenti perdendo la minor quantità di informazioni possibile.

Sia  $\lambda_1$  il coefficiente rispetto alla prima componente della base  $u_1$  per la generica immagine  $\phi_l$ .

$$\lambda_1 = \langle \phi_l, u_1 \rangle$$

Sia  $\Phi^T$  la matrice contenente per ciascuna riga le immagini di training:

$$\Phi^T = \begin{bmatrix} \phi_1^T \\ \phi_2^T \\ \vdots \\ \phi_L^T \end{bmatrix}$$

Allora il problema della ricerca della prima componente della base  $u_1$  che massimizza in media il modulo del relativo coefficiente per ogni immagine di training si formalizza come:

$$\max_{\|x\|=1} \frac{1}{L} \sum_{l=1}^L |\langle \phi_l, x \rangle|^2 = \max_{\|x\|=1} \frac{1}{L} \|\Phi^T x\|_2^2 = \max_{\|x\|=1} \left\langle \frac{1}{L} \Phi \Phi^T x, x \right\rangle = \lambda_1. \quad (1.1)$$

In particolare  $\frac{1}{L} \Phi \Phi^T$  è la matrice di varianza-covarianza campionaria di  $\{\phi_1, \dots, \phi_L\}$  e  $\lambda_1$  è il suo più grande autovalore in modulo.  $u_1$  sarà pertanto l'autovettore relativo a  $\lambda_1$ . Per la ricerca della seconda componente il procedimento è lo stesso, imponendo l'ortogonalità rispetto alla prima componente:

$$\max_{\|x\|=1, x \perp u_1} \left\langle \frac{1}{L} \Phi \Phi^T x, x \right\rangle = \lambda_2.$$

La ricerca della particolare base che stiamo cercando si riduce pertanto al problema del calcolo di autovalori e autovettori della matrice  $\frac{1}{L}\Phi\Phi^T$ .

Un'importante dettaglio implementativo è dato dalla dimensione della matrice  $\Phi\Phi^T \in \mathbb{R}^{mn \times mn}$ . La quantità  $mn$  è potenzialmente molto grande (nel nostro caso  $mn \sim 10^3$ ), rendendo il calcolo degli autovalori e autovettori un'operazione computazionalmente molto onerosa. Questo problema può tuttavia essere aggirato se si considera che  $\text{Rank}(\Phi\Phi^T) \leq L$  e che gli autovalori della matrice  $\Phi^T\Phi$  sono gli stessi della matrice  $\Phi\Phi^T$ . Inoltre:

$$\Phi\Phi^T x = \lambda x \iff \Phi^T\Phi\Phi^T x = \lambda\Phi^T x$$

Pertanto se  $y = \Phi^T x$  è un autovettore di  $\Phi^T\Phi$ , allora  $x = \Phi y$  sarà il corrispondente autovettore di  $\Phi\Phi^T$ . Il calcolo degli autovettori può pertanto essere eseguito sulla matrice  $\Phi^T\Phi \in \mathbb{R}^{L \times L}$  con  $L \ll mn$  alleggerendo di molto i costi computazionali.

La particolare base che stiamo cercando sarà quindi data da  $V = \{v_1, \dots, v_L\}$  dove

$$v_i = \Phi u_i \quad i = \{1, \dots, L\},$$

$u_i$  autovettore di  $\Phi^T\Phi$  relativo all'autovalore  $\lambda_i$ ,

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_L.$$

Una volta ottenuta la base  $V$  è possibile scegliere le sue prime  $L' < L$  componenti ottenendo una base  $V' = \{v_1, \dots, v_{L'}\}$  che approssima  $V$ . È possibile quantificare la perdita di informazioni se si utilizza la base  $V'$  piuttosto che  $V$  per rappresentare le immagini. In particolare possiamo decidere il numero di componenti trattenute  $L'$  calcolando la varianza trattenuta dalla base:

$$\frac{\sum_{l=1}^{L'} \lambda_l}{\sum_{l=1}^L \lambda_l} > \alpha \quad (1.2)$$

dove  $\alpha \in [0, 1]$  è una soglia decisa a priori. Al crescere di  $\alpha$  crescerà il numero di componenti necessarie a trattenere la varianza desiderata. Quanto spiegato fin ora può essere eseguito attraverso i seguenti comandi:

```
1 #computing modified covariance matrix (Phi^T * Phi)
2 cov_mat_mod = 1/L * (faces_train_center @ faces_train_center.transpose())
3 #computing eigenvalues and eigenvectors
4 eig_val, eig_vec = np.linalg.eig(cov_mat_mod)
5 #sorting eigenvalues and eigenvectors in decreasing order
6 sort_mask = np.argsort(eig_val)[::-1]
7 eig_val = eig_val[sort_mask]
8 eig_vec = eig_vec[:, sort_mask]
9 #computing the cumulative explained variance
10 explained_variance_cum = (eig_val/eig_val.sum()).cumsum()
11
12 threshold_mask = explained_variance_cum < variance_threshold
13 num_pcs_kept = (threshold_mask).sum()
14 #computing eigenvalues of the original covariance matrix
15 eigenfaces = faces_train_center.transpose() @ eig_vec[:, threshold_mask]
16 #normalize by columns the eigenvectors (eigenfaces is now an orthonormal matrix)
17 eigenfaces = normalize(eigenfaces, axis=0, norm='l2') # is the matrix V'
18 #project train faces onto the eigenfaces space
19 faces_train_projected = faces_train_center @ eigenfaces
```

Un dettaglio da non sottovalutare è dato dal fatto che la matrice  $V = [v_1, v_2, \dots, v_L]$  è stata ottenuta dal calcolo degli autovettori  $\{u_1, u_2, \dots, u_L\}$  ciascuno moltiplicato per  $\Phi$ . Se  $\{u_1, u_2, \dots, u_L\}$  è un insieme di vettori a norma 1, l'insieme  $\{v_1, v_2, \dots, v_L\}$  non lo è, pertanto la matrice  $V$  va normalizzata per colonne.

Una volta calcolata la matrice  $V'$  relativa alla base  $V'$  è possibile proiettare le facce dallo spazio originale a  $V'$ :

$$\phi'_l = V'^T \phi_l \quad l = 1 \dots L$$

In notazione matriciale:

$$\Phi'^T = \begin{bmatrix} \phi_1'^T \\ \phi_2'^T \\ \vdots \\ \phi_L'^T \end{bmatrix} \quad \Phi'^T = \Phi^T V' \quad \Phi'^T \in \mathbb{R}^{L \times L'}$$

```
1 #project train faces onto the eigenfaces space
2 faces_train_projected = faces_train_center @ eigenfaces # Phi'^T = Phi^T * V'
```

## 2 Test Phase

Siano  $\{g_1, g_2, \dots, g_K\}$  le facce da utilizzare per il test set. Sono anch'esse vettori  $g_i \in \mathbb{R}^{mn}$  per  $i = 1, \dots, K$ . Sulla falsa riga di quanto fatto per il training, possiamo centrare ciascuna faccia rispetto alla faccia media e definire la matrice delle facce di test:

$$\psi_i = g_i - \bar{f} \quad i = 1, \dots, K$$

$$\Psi^T = \begin{bmatrix} \psi_1^T \\ \psi_2^T \\ \vdots \\ \psi_K^T \end{bmatrix} \in \mathbb{R}^{K \times mn}$$

Grazie alla matrice  $V'$  calcolata in fase di training possiamo proiettare ciascuna faccia di test sullo spazio generato da  $V'$ :

$$\psi'_i = V'^T \psi_i \quad i = 1, \dots, K$$

$$\Psi'^T = \Psi^T V'$$

Dove  $\psi'_i \in \mathbb{R}^{L'}$  e  $\Psi'^T \in \mathbb{R}^{K \times L'}$ .

Una volta calcolate le proiezioni di ciascuna faccia sullo spazio generato dalle colonne di  $V'$ , è possibile calcolarne la rappresentazione rispetto alla base canonica di  $\mathbb{R}^{mn}$  utilizzando:

$$\psi''_i = V' \psi'_i \quad i = 1, \dots, K$$

$$\Psi''^T = \Psi'^T V'^T$$

Dove  $\psi''_i \in \mathbb{R}^{mn}$  e  $\Psi''^T \in \mathbb{R}^{K \times mn}$ .

A questo punto è possibile calcolare la distanza  $\epsilon$  tra ciascuna faccia di test e lo spazio generato da  $V'$ , infatti:

$$\epsilon_i = \|\psi_i - \psi''_i\| \quad i = 1, \dots, K$$

```
1 faces_test_centered = faces_test - mean_face
2 #project test faces onto eigenspace
3 faces_test_projected = faces_test_centered @ eigenfaces
4 #project back onto face space
5 faces_test_projected_back = faces_test_projected @ eigenfaces.transpose()
6 #compute distance from eigenspace for each face
7 distance_from_face_space = np.linalg.norm(faces_test_centered -
      faces_test_projected_back, axis=1)
```

Dato quindi un threshold di accettazione  $\Theta$ , è possibile verificare se una faccia appartiene al database utilizzato per generare l'autospazio controllando se  $\epsilon_i < \Theta$ .

Un possibile metodo per calcolare il threshold  $\Theta$  potrebbe essere il seguente: si esclude un soggetto dal database sia di training che di test, si calcola la distanza della sua faccia dall'autospazio e si utilizza tale distanza come threshold  $\Theta$ . Variazioni più accurate potrebbero coinvolgere l'utilizzo di più facce e più soggetti non presenti nel database per il calcolo di  $\Theta$ . Il [codice completo](#) utilizzato implementa un calcolo approssimato di  $\Theta$  escludendo le 10 facce dell'ultimo soggetto dal dataset.

Verificato che una faccia sia presente nel dataset sarà quindi possibile predire a quale soggetto corrisponde la faccia  $\psi'$  individuando la faccia  $\phi^*$  più vicina nell'autospazio. In particolare se  $\phi^*$  è la faccia appartenente al soggetto  $s$  più vicina a  $\psi'$  nell'autospazio, allora l'algoritmo assegnerà alla faccia  $\psi'$  il soggetto  $s$  come predizione.

```

1 for i in range(test_set_size):#for each face in the test set
2     face = faces_test_projected[i]
3     #if the face is too far from eigenspace
4     if distance_from_face_space[i]>acceptance_threshold:
5         continue #skip face
6     #compute distance of the face with every other face in the eigenspace
7     dist = np.linalg.norm(faces_train_projected - face, axis=1)
8     #find the nearest face
9     idx = np.argmin(dist)
10    #predict using idx of the nearest face
11    predicted[i] = idx // int(num_faces_per_subject*train_test_ratio)

```

### 3 Risultati Computazionali

Uno dei parametri più importanti che riguardano l'esito dell'analisi è il numero di componenti della base  $V'$  che vengono mantenute, le cosiddette "*Componenti Principali*". Come spiegato in [1.2](#), è possibile calcolare la varianza trattenuta in funzione del numero di componenti della base  $V'$ :

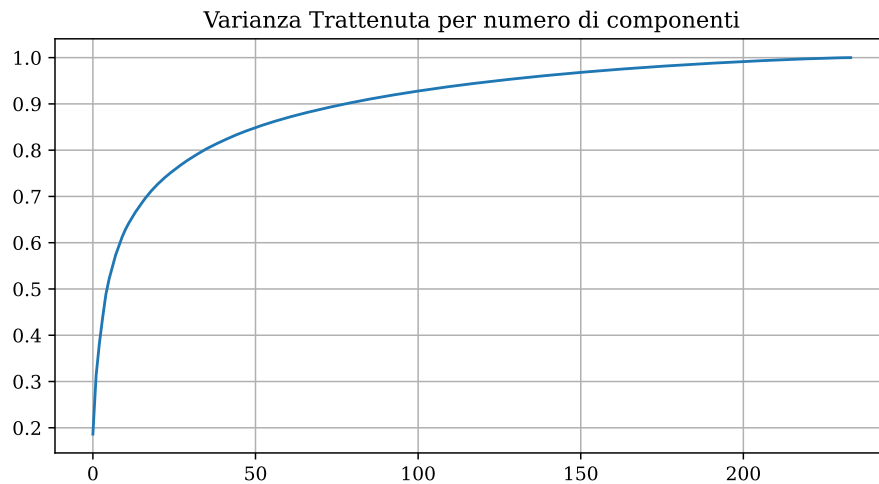


Figure 2: Varianza cumulativa in funzione delle componenti principali

Come si evince dalla Figura 2, per trattenere il 70% della varianza sono necessarie 17 componenti mentre per trattenerne il 90% ne sono necessarie ben 78. Sarà quindi interessante valutare l'accuratezza delle predizioni in funzione del numero di componenti principali utilizzate: ci si aspetta che un minor numero di componenti, sebbene trattengano meno varianza, rendano più snello il calcolo delle distanze a discapito dell'accuratezza. Al contrario, utilizzando un numero più elevato di componenti, le approssimazioni delle immagini risulteranno più accurate e ci si aspetta quindi un'accuratezza delle predizioni più elevata a discapito del tempo computazionale.

Sono stati testati pertanto vari livelli di varianza trattenuta. Per ciascun livello risulta un numero di componenti principali trattenute diverso. Per ogni livello di varianza trattenuta è stata calcolata l'accuratezza delle predizioni. In Figura 3 un riassunto dell'esperimento.

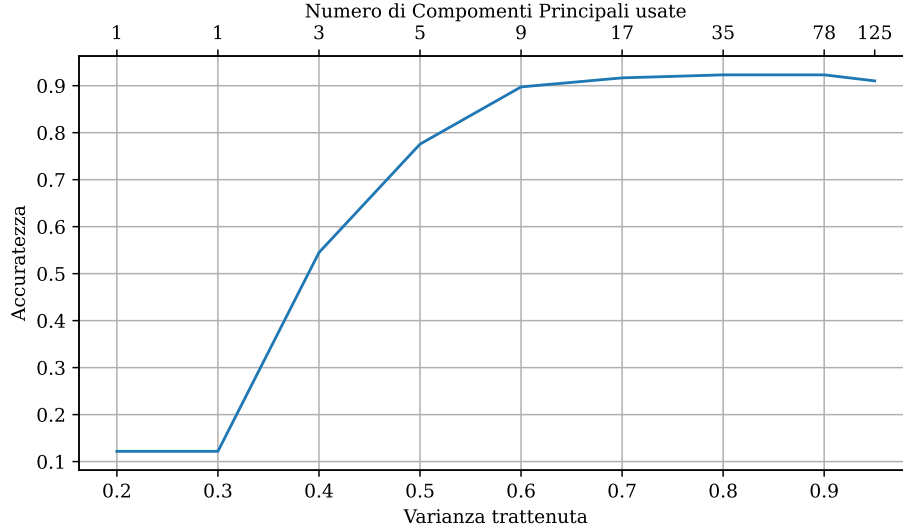


Figure 3: Accuratezza in funzione della varianza trattenuta e del numero di componenti principali usate

È interessante notare come utilizzando 125 componenti principali (corrispondenti al 95% di varianza trattenuta) l'accuratezza scenda leggermente rispetto all'esperimento con 78 componenti principali. Questo fenomeno è probabilmente dovuto alla cosiddetta "*curse of dimensionality*" per cui il concetto di "distanza" perde di significato in spazi di grandi dimensioni.

Tutti gli esperimenti sono stati eseguiti considerando una soglia di accettazione  $\Theta$  calcolata nel modo seguente:

- Si esclude un soggetto  $s_N$  dal dataset (l'ultimo soggetto del dataset)
- si calcola la distanza di ciascuna immagine del soggetto dall'autospazio
- si calcola la media  $\theta$  sulle immagini di ciascuna distanza
- si assegna  $\Theta = \gamma\theta$  dove  $\gamma$  è un valore di tolleranza settato su  $\gamma = 1.3$

## Visualizzazione dell'autospazio

Un'interessante grafico è quello che ci permette di visualizzare l'autospazio. Utilizzando infatti 2 componenti principali ciascuna immagine può essere collocata su un piano cartesiano:

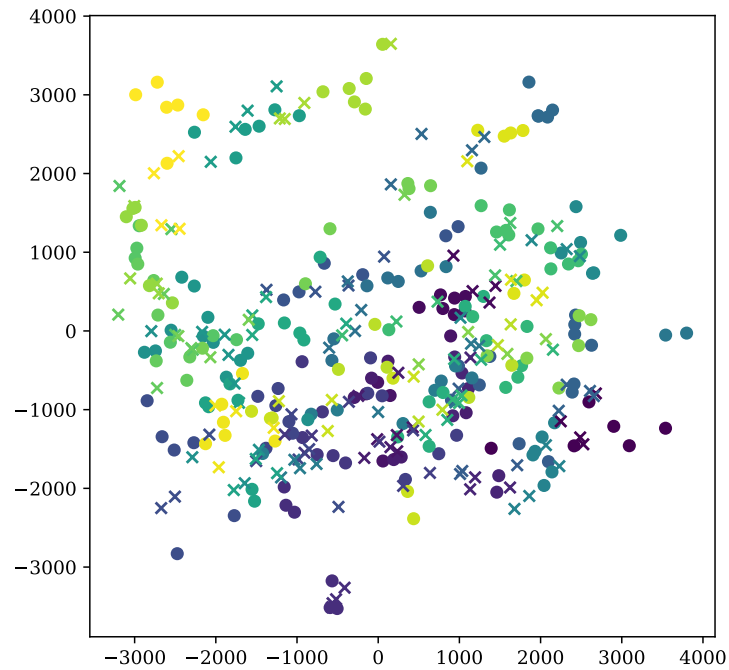


Figure 4: Rappresentazione 2D delle immagini. Le immagini rappresentate con marker "x" sono quelle utilizzate per il test mentre quelle con marker "o" sono le immagini di training. A colori diversi corrispondono soggetti diversi.