Escola de Ciências da Universidade do Minho
Departamento de Informática
Mestrado em Matemática e Computação
Mestrado Integrado em Engenharia Informática

Eigenfaces: reconhecimento facial através da Análise de Componentes Principais

Cecília Martins, PG37016
Fernando Arraz, PG37014
João Alves, PG22591
Joel Morais, A70841
Luísa Caldas, PG35974
Tiago Fraga, A74092
Valéria Romanciuc, PG33724

15 de janeiro de 2019

Conteúdo

1 Introdução

A maior parte da captação de informação é feita através da visão, sendo a associação correcta a objetos e feições fulcral no desenrolar natural do nosso quotidiano. Porém, o reconhecimento facial é um processo extremamente complexo, sendo a sua dificuldade derivada da riqueza das expressões faciais humanas e das diferentes nuances subjacentes. Não há dúvida de que os seres humanos possuem a ferramenta mais potente no que toca a reconhecimento facial, resistente a mudanças físicas associadas ao envelhecimento ou certos adornos (óculos, maquilhagem, etc). É isto que nos permite, por exemplo, conseguir reconhecer uma pessoa que não tenhamos visto há muito tempo, mesmo que o seu aspeto físico se tenha alterado [1].

Isto torna evidente o interesse de desenvolver ferramentas computacionais capazes de reconhecer feições eficazmente. A investigação relacionada com este tipo de tecnologia tem sido desenvolvida há várias décadas, mas ganhou especial relevância mais recentemente, com a evolução das áreas de segurança e dos dispositivos móveis, onde tem grande aplicabilidade.

Em 1991, Matthew Turk e Alex Pentland usaram a Análise das Componentes Principais (PCA) para criar um método para o reconhecimento de faces, utilizando, para isso, os vectores próprios associados aos maiores valores próprios da matriz de covariância de um certo *dataset*.

Neste trabalho, vamos basear-nos no artigo resultante dessa investigação para aplicar o PCA no contexto do reconhecimento de faces. Para isso, vamos começar por descrever o PCA e como é aplicado no contexto do nosso trabalho. De seguida, iremos implementar o modelo descrito em Python e testá-lo com os *datasets* escolhidos. Por fim, iremos analisar os resultados obtidos.

2 PCA no reconhecimento de faces

A ideia apresentada em [1] baseia-se numa abordagem que pretende extrair informação relevante de uma imagem e codificá-la eficientemente para, de seguida, comparar essa codificação com uma base de dados de imagens de faces codificadas de uma forma similar. Essa extração de informação pode dar-se de diferentes formas, sendo uma possibilidade a caracterização da variância que ocorre dentro de um conjunto de imagens, independente das características que consideramos "intuitivas"na descrição de uma face, tais como os olhos, a boca ou o nariz. Matematicamente, esta abordagem traduz-se na determinação das componentes principais da matriz de covariância do *dataset* a ser estudado. Apresentamos, de seguida, a sua formulação.

Comecemos por considerar uma imagem do *dataset*. Esta imagem pode ser vista como uma matriz de dimensão $N \times N$ ou como um vetor de comprimento N^2, em que cada coluna da imagem está "concatenada"com a coluna seguinte, formando um único vector. Nesse caso, o vector representa um elemento de um espaço de dimensão N^2. No entanto, à partida, não será necessário conside-

rarmos todo este espaço para fazer o reconhecimento, uma vez que as nossas feições partilham de várias semelhanças entre elas. Assim, seguirão uma certa distribuição, em vez de se distribuírem aleatoriamente no espaço.

Desta forma, temos como objectivo determinar um subespaço do espaço inicial (mais precisamente, um espaço que é isomorfo a um subespaço próprio do primeiro) gerado pelos vectores próprios associados aos k maiores valores próprios da matriz de covariância do *dataset*. Estes vectores são designados por componentes principais do conjunto de imagens inicial. Cada um destes vectores representa um diferente nível de variação entre as várias imagens, e são eles que "contêm" a maioria da informação associada ao *dataset*, gerando o "espaço das faces". Uma vez que são vectores próprios e visualmente assemelham-se a faces, os autores decidiram designa-los por "eigenfaces"(faces próprias).

A matriz de covariância é dada por

$$S = \frac{1}{N}\sum_{i=1}^{N}\phi_i\phi_i^T = \frac{1}{N}BB^T$$

onde Φ_i é a imagem i centrada na média e $B = \phi_1, \ldots, \phi_N$.

Podemos utilizar diferentes métodos para calcular os vectores próprios de S, sendo que no presente trabalho escolhemos utilizar a decomposição em valores singulares. Este método dá-nos os vectores próprios de S normalizados, ou seja, obtemos um conjunto ortonormado de vectores. Daqui, podemos, então, ordenar os valores próprios e os vectores próprios associados de forma a considerar as componentes principais: $V_k = \{v_1, v_2, \ldots, v_k\}$.

Relativamente ao reconhecimento, queremos determinar se um certo *input* Φ é identificado com algum elemento conhecido do *dataset*. Para isso, começamos por centrá-lo na média e, de seguida, calculamos a sua projecção no espaço gerado pelas componentes principais, que denotamos por $CS(V_k) =< v_1, v_2, \ldots, v_k >$. A projecção de um vector v ao longo de outro vector w é dada por

$$\text{proj}_w v = \frac{< v, w >}{||w||}w.$$

Uma vez que estes vectores são ortogonais dois a dois e de norma 1 (graças ao SVD), temos que

$$\text{proj}_{<v_1,\ldots,v_k>}\Phi = \sum_{i=1}^{k}\text{proj}_{v_i}\Phi = \sum_{i=1}^{k} < \Phi, v_i > v_i = \sum_{i=1}^{k}(\Phi^T v_i)v_i.$$

Assim, basta procurar entre o conjunto de dados qual o indivíduo cuja projecção minimiza a distância para a projecção de Φ. Se esse mínimo estiver dentro de um certo limite, consideramos que o novo *input* foi reconhecido como sendo um elemento do *dataset*. Caso contrário, consideramos que houve uma falha no reconhecimento.

De forma a determinar o mínimo, comecemos por calcular as projecções em $CS(V_k)$ das faces (centradas) do *dataset*. Seja, então, $x_i \in D$, com $i = 1, \ldots, N$. Temos

$$\text{proj}_{CS(V_k)} x_i = < x_i, v_1 > v_1 + < x_i, v_2 > v_2 + \ldots < x_i, v_k > v_k.$$

Consideremos, então, os vectores dos coeficientes das projecções de x_i, $i = 1, \ldots, N$, e de Φ, respectivamente:

$$C_i = \begin{bmatrix} < x_i, v_1 > \\ \vdots \\ < x_i, v_k > \end{bmatrix} = \begin{bmatrix} x_i^T v_1 \\ \vdots \\ x_i^T v_k \end{bmatrix}, \qquad \text{proj}_{CS(V_k)} \Phi = C = \begin{bmatrix} \Phi^T v_1 > \\ \vdots \\ \Phi^T v_k > \end{bmatrix}.$$

Temos que Φ é identificado com x_j se satisfizer duas condições:

$$C_j = \text{argmin}\, d(C_i, C) \qquad \text{e} \qquad d(C_j, C) < L$$

onde L é uma curta tolerância.

Para efectuar estes cálculos considerámos duas distâncias: a distância Euclidiana e a distância de Mahalanobis, que se define por

$$d_M \left(\begin{bmatrix} C_1 \\ \vdots \\ C_k \end{bmatrix}, \begin{bmatrix} C_1^i \\ \vdots \\ C_k^i \end{bmatrix} \right) = \sum_{j=1}^{k} \frac{1}{\lambda_j} (C_j - C_j^i)^2$$

onde λ_j são os valores próprios associados às k componentes principais, $j = 1, \ldots, k$.

Esta distância permite-nos ter em conta a relevância dos valores próprios, atribuindo um peso maior aos maiores valores próprios. Por sua vez, a distância Euclidiana atribui o mesmo peso a todas as componentes. Pareceu-nos relevante averiguar a diferença entre as duas no contexto do nosso problema, uma vez que as nossas componentes não têm o mesmo "peso". Assim, iremos testar ambas as distâncias e comparar os resultados obtidos.

3 Implementação do modelo

O modelo foi implementado em Python, sendo que os testes foram realizados utilizando o Jupyter Notebook. De seguida, vamos apresentar as funções implementadas. O código respectivo pode ser visto no Anexo A.

- **Função `readImages`:** Esta função tem o objetivo de processar as imagens de um *dataset*, carregando para uma matriz as informações de cada fotografia. Retorna uma matriz com as informações de todas as fotografias e a quantidade de fotografias do *dataset*.

- **Função `pca`:** Recebe a matriz com as informações do *dataset* e calcula, através de um parâmetro de confiança, o número de componentes principais em relação ao espaço original. Esta função centra os dados na média e obtém os valores e vectores próprios através do SVD. Além disso, a função armazena e ordena os valores próprios por ordem decrescente e os seus vectores próprios associados, permitindo obter as componentes principais. São retornadas: uma lista ordenada dos k vectores próprios associados aos maiores valores próprios, uma lista ordenada dos k maiores valores próprios, os valores centrados na média e a média.

- **Função `coefProj`:** Dada a matriz com os dados centrados, é realizada a projecção dos elementos desta matriz no espaço gerado pelas componentes principais, cujos coeficientes são retornados em forma de lista.

- **Função `testar`:** Esta função tem como objectivo testar se o modelo reconhece um novo *input*. Dada uma nova imagem, a função centra a mesma na média e projecta este resultado nas *eigenfaces*, testando, de seguida, se este novo elemento pertence ou não ao conjunto de dados através de duas distâncias diferentes: a Euclidiana e a de Mahalanobis.

- **Função `euclidiana`:** Recebe dois elementos x e y e calcula a distância entre esses elementos através da distância Euclidiana.

- **Função `mahalanobis`:** Calcula a distância de Mahalanobis entre dois elementos x e y, utilizando os k maiores valores próprios.

4 Análise de resultados

4.1 *Datasets* estudados

Para testar o modelo implementado, foram utilizados dois *datasets*. O primeiro, chamado "Yale Face Database B" publicado pela UCSD Computer Vision, é composto por 11 fotografias a preto e branco (em formato GIF de dimensões 320x243) por cada um de 15 indivíduos diferentes, correspondendo a 11 expressões faciais ou configurações distintas (feliz, normal, triste, sonolento, surpreendido, a piscar um olho, com óculos, sem óculos, luz frontal, luz à esquerda, luz à direita), perfazendo um total de 165 fotografias [2]. O segundo *dataset* é constituído por 28 fotos dos elementos do grupo (4 por cada elemento), tiradas sob um fundo branco, uniforme e com iluminação estável. Cada pessoa está representada em diversas fotos de forma aproximadamente centrada, cada uma com uma expressão facial diferente de modo a criar uma certa variabilidade na sua representação (a sorrir, normal, a piscar um olho, surpreendido). Cada foto tem dimensões de 255x255 pixeis e contém 3 canais de cor (RGB) por pixel.

4.2 Resultados obtidos

4.2.1 Número de Componentes Principais

Relativamente à determinação do número de componentes k, foi utilizado o quociente

$$\frac{\lambda_1 + \lambda_2 + \ldots + \lambda_k}{\lambda_1 + \lambda_2 + \ldots + \lambda_N} < L$$

onde L é um certo nível de confiança que pretendemos atingir.

Para analisar a relação entre a confiança e o número de componentes, testámos o modelo com o *dataset* YaleFaces, utilizando 120 fotos para treino e 45 para teste. Usando a distância de Mahalanobis, podemos observar a partir da tabela 1 que à medida que a confiança diminui temos, de forma esperada, uma redução das componentes principais que representam o novo espaço reduzido. É de notar que, por exemplo, para uma confiança de 85% e 70%, são usados 15 e 6 vectores próprios, respetivamente. Por outro lado, usando o mesmo nível de confiança para a detecção de faces, foi observado que a precisão (calculada com a fórmula $\frac{TP}{TP+FP}$) é de 82.2% e 68.9% respetivamente. Ora, apesar de se ter reduzido o número de vectores próprios a menos de metade (de 15 para 6), verifica-se que a precisão não sofreu uma redução na mesma proporção. Este é um facto a ser tido em conta em possíveis aplicações do modelo.

Confiança	k	TP	FP	TN	FN	Precisão
0.95	42	29	1	3	12	97.8%
0.9	23	39	5	1	0	88.9%
0.85	15	37	8	0	0	82.2%
0.8	10	37	8	0	0	82.2%
0.75	8	31	14	0	0	68.9%
0.7	6	31	14	0	0	68.9%

Tabela 1: Resultados obtidos para o *dataset* de teste, com a distância de Mahalanobis. TP - True Positive, FP - False Positive, TN - True Negative, FN - False Negative

Fazendo a mesma análise para a distância Euclidiana, pudemos concluir que, tal como no caso anterior, à medida que a confiança e, consequentemente, o número de componentes principais diminui, a precisão também vai diminuindo, como podemos observar pela tabela 2.

No entanto, se compararmos as duas tabelas, vemos que à medida que a confiança diminui, a precisão tem uma diminuição menos acentuada na distância Euclidiana do que na distância de Mahalanobis. Um factor que poderá ter contribuído para esta diferença é o limite L (curta

tolerância) na distância de Mahalanobis, para o qual era necessário uma análise mais aprofundada para obter melhores resultados.

Confiança	k	TP	FP	TN	FN	Precisão
0.95	42	30	0	7	8	100%
0.9	23	32	0	6	7	100%
0.85	15	33	4	5	3	89.2%
0.8	10	31	6	5	3	83.8%
0.75	8	32	7	5	1	82.1%
0.7	6	32	9	3	1	78,0%

Tabela 2: Resultados obtidos para o *dataset* de teste, com a distância Euclidiana.

Em alternativa a nos basearmos na confiança para obter o número de componentes, podíamos obter o k através do "elbow method". Podemos visualizar este método na figura 1, através da qual concluímos que um número adequado para k seria igual a 10, aproximadamente.

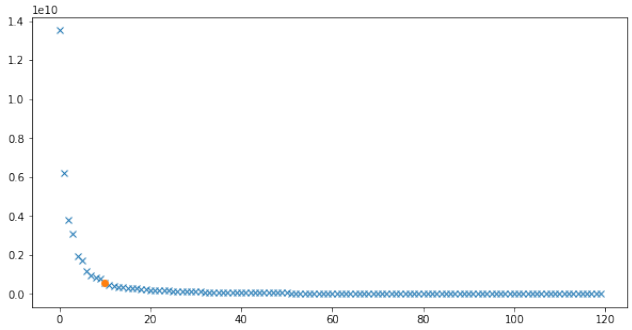

Figura 1: "Elbow method"para determinar o número de componentes principais.

Podemos observar que a aplicação do PCA irá depender do grau de confiança desejado em função dos recursos computacionais disponíveis, sendo, portanto, uma importante ferramenta de decisão para as mais diversas aplicações. Por exemplo, caso tenhamos alta disponibilidade computacional/recursos, podemos adotar um modelo com alto nível de precisão. É o caso de uma instituição bancária, em que se pretenda dar acesso apenas às pessoas autorizadas, e, em caso de dúvidas, o elemento em questão é barrado para uma verificação. Por outro lado, dada uma "blacklist" num hotel, poderá pretender-se adotar um sistema em que, caso haja dúvida para detectar se o elemento pertence ou não à lista, é preferível que seja dado o acesso, a fim de evitar constrangimento

	Dist. Euclidiana		Dist. de Mahalanobis	
Fotografia	Minímo	Máximo	Mínimo	Máximo
cecilia_smile	709.74	20124.75	0.0022	0.9264
fernando_wink	1796.26	18316.52	0.0185	0.6745
joao_normal	1074.92	20239.65	0.0055	0.8297
joel_normal	612.59	17238.84	0.0025	0.7422
luisa_surprised	793.24	16022.53	0.0052	0.7386
tiago_wink	759.54	18379.89	0.0050	0.6664
valeria_normal	1401.51	20428.22	0.0179	0.7243

Tabela 3: Resultados obtidos para o *dataset* de teste das fotos do grupo.

a clientes que não estejam na referida lista.

No nosso caso, se quisermos que o nosso sistema seja mais fidedigno no reconhecimento de faces que já estão no *dataset*, se calhar é preferível que ele tenha menos falsos positivos do que falsos negativos, isto porque desta forma conseguimos uma maior segurança no que diz respeito ao sistema reconhecer apenas as caras do *dataset* e não reconhecer imagens diferentes.

4.2.2 Curta tolerância

Para a determinação da curta tolerância, fizemos testes com cada uma das métricas de forma a percebermos o intervalo de variação das distâncias e podermos estabelecer um limite. Estes são os dados obtidos para o nosso *dataset*, sendo que para o fazer foi retirada 1 foto de cada pessoa para servir de teste (perfazendo 7), e as restantes foram usadas para treino (21 imagens).

Tal como podemos observar pela tabela 3, usando a distância Euclidiana, o maior valor dos mínimos é de 1401,51. Posto isto, decidimos que o limite para a distância euclidiana neste *dataset* seria de 1900, isto é, para distâncias superiores a 1900 o nosso modelo não reconhece as imagens. Para a distância de Mahalanobis decidimos seguir a mesma ideia e definimos o limite a 0,035.

Para o *dataset* das YaleFaces não definimos uma tabela de resultados, pelo facto de que o nosso conjunto de teste ser constítuido por 45 imagens. Por observação de alguns testes, definimos que o limite para a distância Euclidiana seria de 7600 e para a distância de Mahalanobis de 0,05.

5 Considerações finais

Após a análise dos resultados, podemos afirmar que a elaboração de um sistema de reconhecimento facial suportado pelo algoritmo de Análise de Componentes Principais é bastante limitador. A primeira dificuldade encontrada no momento da escolha do *dataset* residiu no facto de que as fotos não podem conter qualquer tipo de sombra, caso contrário o reconhecimento é seriamente afectado. Outra dificuldade detetada prende-se no facto de que todas as fotografias em estudo têm de estar sempre centradas e na mesma posição, isto é, se o mesmo indivíduo numa fotografia estiver deslocado para a direita e noutra fotografia estiver deslocado para a esquerda, a aplicação do PCA neste cenário irá apresentar resultados muito fracos na deteção da face deste indivíduo.

Após o estudo de cada uma das distâncias utilizadas no projeto, verificámos, ainda, que não existe um método directo que justifique os limites estabelecidos. Como explicado no ponto 4.2.2, a escolha dos mesmos foi efetuada depois de analisar o valor mínimo das distâncias. Reconhecemos, portanto, que a atribuição dos valores dos limites carecem de um estudo mais aprofundado, mas que estava fora do âmbito do trabalho.

Referências

[1] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, January 1991.

[2] Yale faces database. `http://vision.ucsd.edu/content/yale-face-database`. Consultado em 12-01-2019.

A Anexo

Apresentamos o código das funções implementadas.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import glob
import math


# Funcao de processamento do dataset
def readImages (path, extension):
    # Leitura das imagens
    imgs = glob.glob(path+'/*.'+extension)
    base = [Image.open(i).convert('L') for i in imgs]

    # Tamanho do dataset
    size = len(base)

    # Passar as imagens para um array
    X = np.array([base[i].getdata() for i in range(size)])
    return X, base, size
```

```
# Implementacao do PCA
def pca(X, confidence=0.8):
    # Media do dataset
    mean = np.mean(X,0)

    # Centrar os dados
    phi = X - mean

    # Calcular os vetores e valores proprios atraves do SVD
    eigenvectors, sigma, variance = np.linalg.svd(phi.transpose(),
        full_matrices=False)
    eigenvalues = sigma*sigma

    # Ordenacao dos valores pp
    idx = np.argsort(-eigenvalues)
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:,idx]

    # Determinar o n. de vectores pp a usar
    k = 0
```

```
20      traco = np.sum(eigenvalues)
21      while(np.sum(eigenvalues[:k])/traco < confidence):
22          k = k+1
23
24      # Escolher os vetores pp associados
25      eigenvectors = eigenvectors[:,0:k]
26      return eigenvalues, eigenvectors, phi, mean
```

```
1  # Calculo dos coeficientes da projeccao
2  def coefProj(phi, eigenvectors, size):
3      coef_proj = [np.dot(phi[i], eigenvectors) for i in range(size)]
4      return coef_proj
```

```
1  # Verificar se identifica ou nao o input
2  def testar(input_img, mean, eigenvectors, eigenvalues, size, coef_proj,
       distance="mahalanobis"):
3      # Centrar o input
4      gamma = np.array(input_img.getdata())
5      test_phi = gamma - mean
6
7      # Calcular os coeficientes da projeccao do input
8      test_coef_proj = np.dot(test_phi, eigenvectors)
9
10     if distance == "euclidian":
11         dist = [np.linalg.norm(coef_proj[i] - test_coef_proj) for i in range
               (size)]
12         limit = 7600
13     elif distance == "mahalanobis":
14         dist = mahalanobis(coef_proj, test_coef_proj, eigenvalues,
               eigenvectors.shape[1])
15         limit = 0.02
16     else:
17         print("Distancia invalida.")
18         return (-1)
19
20     d_min = np.min(dist)
21
22     if d_min < limit:
23         print('Imagem nr.: '+str(np.argmin(dist))+'\n'+'Distancia minima: '+
               str(d_min)+'\n')
24     else:
25         print('Falhou no reconhecimento.')
```

```
1  # Distancia euclidiana
```

```
def euclidian(x, y):
    if x.size != y.size:
        return (-1)
    z = y - x
    distance = math.sqrt(sum(z**2))
    return round(distance, 2)
```

```
# Distancia de Mahalanobis
def mahalanobis(x, y, eigenvalues, k):
    if len(x[0]) != len(y):
        return (-1)
    N = len(x)
    distance=[]
    for i in range(N):
        distance.append(np.sum(np.divide((x[i]-y)**2, eigenvalues[:k])))
    return distance
```