Álgebra Linear
Trabalho Final - Eigenfaces

Professor: Yuri Saporito

Grupo: Caio Lins e Juliane Martins

1 Introdução

Reconhecimento facial é, em geral, uma tarefa complexa, pois faces são objetos naturais que não são descritos por um conjunto bem definido de parâmetros. Enquanto seres humanos, por milhares de anos, foram condicionados a desenvolver essa habilidade devido ao processo evolutivo, uma máquina não possui por padrão essa funcionalidade, necessitando de instruções claras para realizar a mesma tarefa.

As primeiras tentativas de reconhecimento computacional de faces se baseavam em detectar características específicas do rosto que pessoas consideram marcantes, como olhos, nariz e uma boca. O método *Eigenfaces*, por outro lado, tem como objetivo utilizar uma abordagem pautada em Teoria da Informação para identificar os aspectos mais significativos de uma face, os quais podem, ou não, coincidir com as referências utilizados por humanos.

Essa técnica consiste em uma aplicação de *Principal Component Analysis* (PCA) na tarefa de reconhecimento facial. As imagens de rostos são transformadas em um pequeno conjunto de componentes principais, os quais representam as características faciais mais relevantes. Esses componentes são denominados *eigenfaces*. Uma imagem nova é projetada no subespaço gerado pelas eigenfaces e, então, classificada de acordo com a posição relativa de sua projeção às posições de indivíduos conhecidos.

2 Reconhecimento utilizando Eigenfaces [2]

Dado um conjunto de imagens, as quais podem ser interpretadas como elementos de um espaço vetorial, deseja-se encontrar um conjunto de vetores ortonormais que gere um espaço semelhante ao gerado pelas imagens e cujas direções sejam aquelas em que haja maior variabilidade entre elas. Esses vetores, por poderem ser mostrados graficamente como uma imagem parecida com uma face, são chamados de eigenfaces.

Se temos N imagens, podemos gerar até N eigenfaces. Cada imagem pode ser escrita como uma combinação linear desses vetores, mas também pode ser aproximada pelas $\mathsf{N}' < \mathsf{N}$ "melhores" eigenfaces (que representam a maior variância). Isso melhora o desempenho computacional do programa. Então, ao receber uma imagem nova, pode-se aproximá-la utilizando os N' melhores componentes e comparar a combinação linear obtida com as combinações que geram as faces já conhecidas.

Computacionalmente, uma imagem em preto e branco com resolução $\mathfrak{mn}$ pode ser descrita como uma matriz $\mathfrak{m} \times \mathfrak{n}$ cujos elementos são números reais que representam valores na escala de cinza, ou,

equivalentemente, um ponto em $\mathbb{R}^{mn}$. Sendo $\Gamma_i \in \mathbb{R}^{mn}$, $1 \leqslant i \leqslant N$, a i-ésima imagem, definimos a *face média*, denotada por Ψ, como:

$$\Psi = \frac{1}{N}\sum_{i=1}^{N}\Gamma_i. \tag{1}$$

Como queremos analisar a variabilidade dos dados, inicialmente devemos centralizá-los, eliminando características que são comuns a todas as imagens. Fazemos isso subtraindo Ψ de cada Γ_i. Logo, definimos os vetores Φ_i:

$$\Phi_i = \Gamma_i - \Psi. \tag{2}$$

Em seguida, definimos a matriz A, cujas colunas são os vetores Φ_i:

$$A = \begin{bmatrix}\Phi_1 & \cdots & \Phi_N\end{bmatrix}. \tag{3}$$

Desejamos encontrar os componentes principais da matriz A. Para tanto, utilizamos a decomposição SVD dessa matriz, ou seja, obtemos matrizes ortogonais $U_{mn\times N}$ e $V_{N\times N}$ bem como uma matriz diagonal $\Sigma_{N\times N}$, tais que:

$$U\Sigma V^T = A. \tag{4}$$

Com isso, as colunas de U são os autovetores da matriz de covariância $\frac{1}{N-1}AA^T$, os quais, portanto, compõem o conjunto de eigenfaces. Esses vetores podem ser ordenados de acordo com seus valores singulares correspondentes, de modo a identificar quais deles representam os desvios mais significativos nos dados. Dessa forma, sendo $q_1, \ldots, q_N$ as eigenfaces e $\sigma_1 < \cdots < \sigma_N$ os valores singulares associados, escolhemos as N' primeiras.

Tendo os componentes principais de A, devemos projetar os vetores Φ_i no subespaço $E \subset \mathbb{R}^{mn}$ gerado pelas N' eigenfaces, denominado *face space*. Como elas são ortonormais, temos:

$$\text{proj}_E\,\Phi = \widetilde{\Phi_i} = \omega_{i1}q_1 + \cdots + \omega_{iN'}q_{N'}, \tag{5}$$

onde $\omega_{ij} = q_j^T\Phi_i$. Definimos $\Omega_i^T = \begin{bmatrix}\omega_{i1}\cdots\omega_{iN'}\end{bmatrix}$.

Então, dada uma face desconhecida Γ, subtraímos dela a média Ψ, obtendo o vetor Φ. O vetor $\Omega^T = \begin{bmatrix}\omega_1\cdots\omega_{N'}\end{bmatrix}$ é obtido de maneira análoga. A projeção de Φ em E é dada por

$$\widetilde{\Phi} = \begin{bmatrix}q_1 & \cdots & q_{N'}\end{bmatrix}\Omega. \tag{6}$$

Para classificar a nova face, testamos dois algoritmos diferentes, os quais diferem em relação a como calcular o parâmetro utilizado para reconhecer a foto.

- Algoritmo 1: Classificação pelo erro médio por indivíduo.

  Primeiro, particionamos o conjunto de índices das imagens em conjuntos disjuntos $P_1, \ldots, P_N$, onde os vetores Φ_i, com $i \in P_k$, são fotos da k-ésima pessoa no banco de dados. Então, a foto ainda não identificada tem maior chance de pertencer ao indivíduo que minimizar a quantia

  $$\varepsilon_k = \frac{1}{|P_k|}\sum_{i\in P_k}\|\Omega - \Omega_i\|. \tag{7}$$

  Ou seja, se $\varepsilon_j = \min\varepsilon_k$, então a foto deve pertencer ao j-ésimo indivíduo.

- Algoritmo 2: Classificação pelo erro mínimo entre todas as imagens.

  Para a i-ésima foto no conjunto de treino, calculamos a quantia

  $$\varepsilon_i = \|\Omega - \Omega_i\|, \tag{8}$$

  onde $1 \leqslant i \leqslant N$. Suponhamos que tem-se $\varepsilon_j = \min \varepsilon_i$. Então, a foto ainda não identificada tem maior chance de pertencer ao indivíduo presente na j-ésima foto.

Utilizando qualquer um desses dois métodos de classificação, obtemos 3 possíveis resultados:

i. Caso o erro de projeção de Φ em E, dado por $\delta = \left\|\Phi - \widetilde{\Phi}\right\|$, seja maior ou igual que um limite estipulado θ_δ, então a nova imagem não é uma face, pois está muito distante do face space.

ii. Caso tenhamos $\delta < \theta_\delta$, a nova imagem é, de fato, uma face. Entretanto, se ε_j é maior que um limite estipulado θ_ε, então a imagem pertence a um indivíduo desconhecido.

iii. Por fim, se temos $\delta < \theta_\delta$ e $\varepsilon_j < \theta_\varepsilon$, então a imagem pertence a um indivíduo conhecido.

Os limites θ_δ e θ_ε são escolhidos empiricamente.

3 Aplicação em um conjunto de dados

3.1 Implementação da técnica

3.1.1 Características dos dados utilizados

Para aplicação da técnica, utilizamos o banco de imagens "AT&T Database of Faces" [1]. Esse conjunto de dados possui 10 fotos para cada um dentre 40 indivíduos distintos. Para alguns indivíduos, as fotos foram tiradas variando condições de luminosidade, expressões faciais e detalhes faciais, como o uso de óculos. Para outros, as fotos não apresentam tantas diferenças entre si. Todas imagens foram tiradas contra um fundo escuro homogêneo com os indivíduos eretos e em posição frontal, admitindo pequenos movimentos laterais. As fotos possuem resolução 92 por 112 e estão em uma escala de 256 tons de cinza.

Como base de dados para treinamento do programa, utilizamos 9 fotos de 39 dos indivíduos. A Figura 1 é a face média obtida.

Como utilizamos 351 imagens, obtivemos, no total, 351 eigenfaces. Algumas delas se encontram na Figura 3. Porém, como mencionado na seção anterior, utilizamos apenas um pequeno subconjunto delas, que apresentou os maiores valores singulares. A Figura 2 contém um gráfico que representa o decaimento dos valores singulares. Analisando-o, decidimos utilizar as 33 primeiras eigenfaces, pois os valores correspondentes após essa posição são muito próximos de 0.

Para realizar o reconhecimento facial, precisamos atribuir valores aos limites θ_δ e θ_ε. Realizamos testes para verificar quais seriam os erros médios obtidos ao se tentar reconhecer imagens pertencentes ao conjunto de treino, ou não, bem como imagens de pessoas que estão, ou não, no conjunto de treino. Chegamos à conclusão de que valores razoáveis seriam:

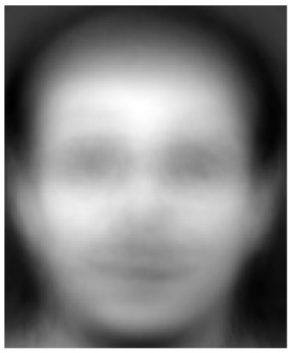

Figura 1: Face média.

- Para o Algoritmo 1: $\theta_\varepsilon = 10.00$ e $\theta_\delta = 2850.00$.
- Para o Algoritmo 2: $\theta_\varepsilon = 6.00$ e $\theta_\delta = 2850.00$.

Escolhemos θ_δ de modo que nenhuma das imagens fosse classificada como não sendo de uma face. Esses parâmetros podem ser adaptados para se adequarem à aplicação desejada.

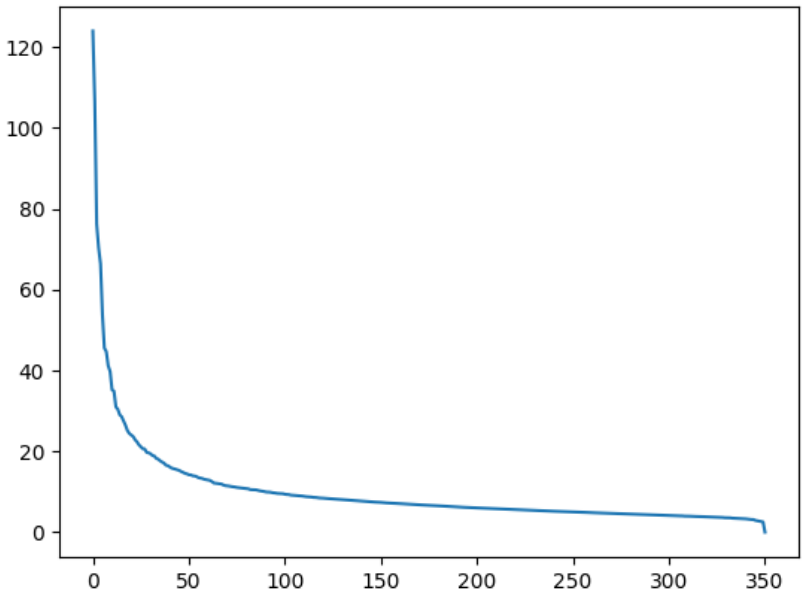

Figura 2: Decaimento dos valores singulares.

3.1.2 Ferramenta utilizada

A implementação do método foi feita utilizando a linguagem de programação Python. As bibliotecas utilizadas foram NumPy, PIL e MatPlotLib. A seguir estão os trechos de código que implementam as principais funcionalidades do programa:

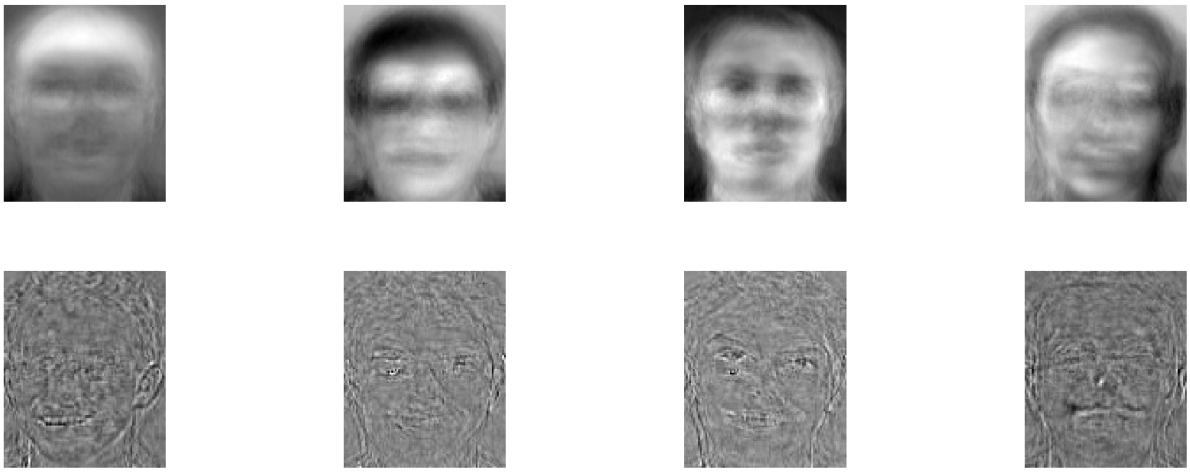

Figura 3: A primeira linha contém as 4 eigenfaces com maiores valores singulares correspondentes. A segunda linha contém 4 das eigenfaces com menores valores singulares associados.

- Após importar as bibliotecas, inicializamos os dados criando o NumPy array `face_matrix` cujas colunas são as imagens. Em seguida, o transformamos no array `Data` que é a matriz A definida na seção anterior. Com isso, conseguimos calcular a decomposição SVD de `data`.

```
### Prepare data to be used for facial recognition

# Compute the matrix whose columns are the images
# rearrenged into vectors.

face_matrix = np.empty((image_size, 0))
# It's important that the columns of the matrix
# are organized. Columns 0-8 belong to subject 1,
# 9-17 to subject 2 and so forth.
for i in range(1, 41):
    subject = "s" + str(i)
    # Expression used to select a subset of subjects
    selected_subjects = not "40" in subject
    if selected_subjects:
        for image in os.listdir(source_directory + subject):
            # Expression used to select a subset of faces for a given
   subject
            selected_faces = not "_2" in image
            if selected_faces:
                file_name = source_directory + subject + "/" + image
                with img.open(file_name) as face_image:
                    face_vector = np.array(face_image).reshape((image_size,
   1))
                    face_vector = face_vector/255 # work with entries from 0
    to 1
                    face_matrix = np.hstack((face_matrix, face_vector))

```

```
# Compute average face
average_face = np.mean(face_matrix, axis=1).reshape((image_size, 1))

# Subtract average face from matrix of faces
data = face_matrix - average_face
num_images = len(data[0, :])

# Compute the data's SVD (U and V are not square)
u, Sigma, vt = np.linalg.svd(data, full_matrices=False)
```

- Com isso, definimos o array `eigenfaces` cujas colunas são as 33 primeiras colunas do array `u`. Convertemos a imagem a ser reconhecida no array `unknown_face` e calculamos os coeficientes que geram a sua projeção no face space, por meio de produtos internos com as eigenfaces. Em seguida, calculamos o erro de projeção.

```
    # Project the unknown face onto the space generated by the
    # eigenfaces (the face space). Store the coefficients that
    # are used to generate this projection using the basis of
    # eigenvectors.
    unknown_face_coeff = []
    for i in range(len(eigenfaces[0, :])):
        coeff = np.dot(eigenfaces[:, i], unknown_face)
        unknown_face_coeff.append(coeff[0])
    unknown_face_coeff = np.array(unknown_face_coeff)

    # Compute, in the canonical base, the projection of the
    # uknown face onto the face space
    projection = eigenfaces @ unknown_face_coeff

    projection_error = np.linalg.norm(unknown_face - projection)
```

- Verificamos se o erro de projeção é maior que o θ_δ, representado pela variável `proj_err_limit`:

```
    if projection_error > proj_err_limit:
        print("Projection error:", projection_error)
        print("Not a face.")
```

- Caso não seja, projetamos as imagens do conjunto de dados no face space. No Algoritmo 1, calculamos a média das normas das diferenças entre o array `unknown_face_coeff` os arrays de coeficientes das faces no conjunto de treino, para cada indivíduo. Identificamos a menor dessa médias, ou seja, o valor $\varepsilon_j = \min \varepsilon_k$, e o armazenamos na variável `face_error`.

```
    else:
        # Compute coefficiets of the projection of the
        # known faces onto the face space
        known_faces_coeff = eigenfaces.T @ data

        # Compute the norm of the difference between the
```

```
        # coefficients that generate the unknown face and
        # the coefficients that generate each of the known
        # faces
        difference = []
        for i in range(len(data[0, :])):
            diff_norm = np.linalg.norm(known_faces_coeff[:, i] -
    unknown_face_coeff)
            difference.append(diff_norm)
        difference = np.array(difference)

        # Group differences by subject (one in each line)
        num_groups = len(difference)//imgs_per_subject
        difference = difference.reshape(num_groups, imgs_per_subject)

        # Compute mean of differences per subject
        difference_mean = difference.mean(axis=1)

        # Take the minimum of those differences
        face_error = difference_mean.min()
```

- No Algoritmo 2, calculamos as normas das diferenças entre o array `unknown_face_coeff` e os arrays de coeficientes das fotos no conjunto de treino, sem agrupar por indivíduo. Em seguida, identificamos a menor dessas diferenças, ou seja, o valor $\varepsilon_j = \min \varepsilon_i$. Também armazenamos esse valor na variável `face_error`.

```
    else:
        # Compute coefficiets of the projection of the
        # known faces onto the face space
        known_faces_coeff = eigenfaces.T @ data

        # Compute the norm of the difference between the
        # coefficients that generate the unknown face and
        # the coefficients that generate each of the known
        # faces
        difference = []
        for i in range(num_images):
            diff_norm = np.linalg.norm(known_faces_coeff[:, i] -
    unknown_face_coeff)
            difference.append(diff_norm)
        difference = np.array(difference)

        # Take the minimum of those differences
        face_error = difference.min()
```

Por fim, nos dois Algoritmos verificamos se esse erro é maior que θ_ε. Caso não seja, reconstruímos, a partir do array, uma imagem do indivíduo identificado para poder mostrá-lo na tela.

3.2 Desempenho do programa

Para os testes dos Algoritmos 1 e 2, o programa foi treinado com 9 das 10 fotos dos indivíduos 1 a 39 e nenhuma foto do indivíduo 40.

A Tabela 1 contém os resultados para as tentativas de reconhecimento facial, pelo Algoritmo 1, das imagens dos 39 primeiros indivíduos do conjunto de dados utilizado. A Tabela 2 contém os resultados das tentativas de reconhecimento, pelo Algoritmo 1, das fotos do indivíduo 40.

	Não é face	Face desconhecida	Reconheceu certo	Reconheceu errado
Foto no Dataset	0 %	25.36 %	73.79 %	0.85 %
Pessoa no Dataset	0 %	38.46 %	61.54 %	0 %

Tabela 1: Resultados do reconhecimento, pelo **Algoritmo 1**, de dois tipos de imagens: um em que a própria imagem foi usada no banco de dados (campo "Foto no Dataset", 351 imagens), e outro em que a pessoa representada na imagem está no Dataset, mas com outras fotos (campo "Pessoa no Dataset", 39 imagens).

Não é face	Face desconhecida	Reconheceu
0 %	100 %	0 %

Tabela 2: Resultados do reconhecimento, pelo **Algoritmo 1**, das fotos do indivíduo 40.

A Tabela 3 contém os resultados para as tentativas de reconhecimento facial, pelo Algoritmo 2, das imagens dos 39 primeiros indivíduos do conjunto de dados utilizado. A Tabela 4 contém os resultados das tentativas de reconhecimento, pelo Algoritmo 2, das fotos do indivíduo 40.

	Não é face	Face desconhecida	Reconheceu certo	Reconheceu errado
Foto no Dataset	0 %	0 %	100.00 %	0.00 %
Pessoa no Dataset	0 %	17.95 %	82.05 %	0 %

Tabela 3: Resultados do reconhecimento, pelo **Algoritmo 2**, de dois tipos de imagens: um em que a própria imagem foi usada no banco de dados (campo "Foto no Dataset", 351 imagens), e outro em que a pessoa representada na imagem está no Dataset, mas com outras fotos (campo "Pessoa no Dataset", 39 imagens).

Considerando as imagens que estavam no banco de dados, a proporção de reconhecimentos certos em relação ao total de imagens reconhecidas foi de 98.85 % para o Algoritmo 1 e de 100.00 % para o Algoritmo 2. Para as imagens que não estavam no banco de dados, porém que são de pessoas que possuem fotos nele, a proporção foi de 100.00 % para os dois Algoritmos.

Não é face	Face desconhecida	Reconheceu
0 %	100 %	0 %

Tabela 4: Resultados do reconhecimento, pelo **Algoritmo 2**, das fotos do indivíduo 40.

4 Discussão dos resultados e conclusão

Pela seção anterior, percebe-se uma evidente superioridade do Algoritmo 2 em relação ao Algoritmo 1. Isso contradiz os resultados esperados. Como o Algoritmo 2 classifica uma imagem pela menor diferença entre ela e as imagens do conjunto de treino, esperava-se que ele apresentasse um melhor desempenho ao reconhecer fotos que estavam no conjunto de treino.

Por outro lado, o Algoritmo 1 utiliza a média das diferenças entre a imagem desconhecida e as imagens de cada indivíduo. Logo, por levar em conta mais informação, esperava-se que ele tivesse um melhor desempenho ao reconhecer imagens de pessoas que estavam representadas no conjunto de treino, mas com outras fotos.

Entretanto, de maneira inesperada, o Algoritmo 2 se saiu melhor ao reconhecer ambos tipos de imagem. A principal diferença entre os dois Algoritmos foi a taxa de faces desconhecidas. Inicialmente, uma possível forma de diminuir essa diferença seria aumentar o parâmetro θ_ε para o Algoritmo 1. Entretanto, isso faria com que fotos de indivíduos desconhecidos, como o indivíduo 40, fossem reconhecidas.

Como o Algoritmo 1 utiliza uma média, os valores $\min \varepsilon_k$, obtidos ao reconhecer cada foto, ficam muito próximos entre si, de modo que o erro mínimo ao reconhecer o indivíduo 40 não seja o maior, impossibilitando a escolha de um θ_ε melhor.

Em contrapartida, o Algoritmo 2 se baseia no mínimo global entre todas as fotos, de modo que, para pessoas presentes no conjunto de treino, os valores $\min \varepsilon_i$ são pequenos (ou iguais a 0, se a própria foto estiver no conjunto de treino), enquanto para pessoas fora dele esse valor é mais elevado. Com isso, há espaço para escolher um θ_ε que possibilite uma maior taxa de reconhecimento.

O Algoritmo 2 reconheceu uma alta parcela (82.05 %) das fotos que não estavam no conjunto de treino. As imagens que não foram reconhecidas podem ser explicadas pelo fato de que o conjunto de imagens disponível para cada pessoa não é uniforme. Experimentalmente, verificamos que as variações que mais aumentam o erro $\|\Omega - \Omega_i\|$ para um dado indivíduo são diferenças no tamanho e na inclinação do rosto na foto.

A Figura 4 exemplifica essa limitação do modelo. A foto do indivíduo 1 que não estava no conjunto de treino foi uma das não reconhecidas pelo Algoritmo 2.

Portanto, o método Eigenfaces para reconhecimento facial se sobressai em aplicações nas quais é possível controlar as características das fotos que fazem parte do conjunto de treino do modelo, bem como as características da foto que deve ser reconhecida. Sob essas condições, é uma abordagem simples e rápida que apresenta elevada precisão.

Figura 4: Quatro imagens do indivíduo 1.

Referências

[1] AT&T Face Laboratories Cambridge. The database of faces, 2002. Dados obtidos de, `https://www.kaggle.com/kasikrit/att-database-of-faces`.

[2] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991.