Ana Carolina Erthal Fernandes
Tiago Barradas Figueiredo

Eigenfaces

Brasil

2021

Ana Carolina Erthal Fernandes
Tiago Barradas Figueiredo

Eigenfaces

Trabalho apresentado à disciplina de Álgebra Linear do curso de Ciência de Dados e Inteligência Artificial da Fundação Getúlio Vargas.

Fundação Getúlio Vargas – FGV

Escola de Matemática Aplicada

Orientador: Yuri Saporito

Brasil

2021

Sumário

Introdução

Não é difícil vislumbrar a importância e aplicações do reconhecimento facial em nosso dia a dia. A tarefa pode ser simples para o cérebro humano, uma vez que temos a capacidade de reconhecer faces a todo instante, e grande parte de nossas interações sociais dependem desse fato. Na verdade, reconhecer outros rostos é tão rotineiro para nós que projetamos esse traço de humanidade em objetos inanimados, e é comum nos depararmos reconhecendo olhos, bocas e etc. em artefatos do dia a dia. Por isso, sempre foi de grande interesse que nossas máquinas pudessem realizar o mesmo processo.

As aplicações práticas são inúmeras, desde facilitadores como reconhecimento facial para celulares, substituindo senhas, até usos para segurança, como acontece em processos de imigração de diversos países. A frequência da utilização desse tipo de algoritmo é tal que ele pode parecer um processo mais trivial do que de fato é, não revelando a princípio a complexidade da questão.

O método foi uma criação conjunta em duas etapas, inicialmente discutido por Sirovich and Kirby, na Brown University, em 1987. Nesse primeiro trabalho discutiam-se as *eigenpictures*, que consistiam em simplificar um conjunto de imagens originais, armazenando uma representação ótima da face média. Anos depois, em 1991, Matthew Turk and Alex Pentland, do MIT, publicaram o artigo *Eigenfaces for Recognition*, discutindo um algoritmo de reconhecimento rápido e eficiente de reconhecimento. O processo será discutido de forma mais profunda adiante.

Nosso objetivo, através desse relatório será de explicitar o processo por trás de algoritmo, dando destaque à teoria, que trabalha diretamente a Álgebra Linear, resumidamente tratar de nosso algoritmo de reconhecimento, construído em Python, e, por fim, discutirmos resultados, testes e questionamentos que julgamos interessantes de realizar.

1 O reconhecimento por Eigenfaces

1.1 Processo

Trataremos aqui do trabalho desenvolvido no MIT pelos professores Matthew Turk and Alex Pentland, que chegaram a conclusão que, diferentemente do que fora proposto anteriormente, as características que são relevantes para o reconhecimento facial não devem ser, necessariamente, as mesmas que têm importância para o ser humano. Antes dessa realização era utilizado , por exemplo, o método de *eigenpictures*, que focava fortemente em características específicas do rosto, como olhos, boca, nariz e etc.. A análise aqui desenvolvida será focada nesse método mais sofisticado, denominado *eigenfaces*.

Nesse processo, inicialmente, temos um *dataset* inicial composto por fotos de rostos que tenham aproximadamente a mesma centralização e luminosidade, além de resolução $m \times n$ pixels, que serão convertidas para vetores $1 \times mn$ em *grayscale*. Se tivermos N fotos, montaremos uma matrix $A_{N \times mn}$, onde cada linha corresponde a uma imagem. Os autovetores da matriz de correlação de A, $A^T A$, quando retornados ao formato original das imagens, representam as **eigenfaces**.

Falando em termos não muito técnicos, essas *eigenfaces* são como 'ingredientes' que, ao serem combinados em uma receita, conseguirão gerar qualquer imagem original do *dataset*. Cada uma dessas imagens originais têm uma receita diferente, com proporções específicas de cada um desses autovetores sendo somadas uns aos outros.

Na verdade, cada *eigenface* faz um mapeamento das variações das imagens inseridas no *dataset*, e cada uma delas tem maior ou menor participação de cada imagem original. Elas podem, ainda, ser ordenadas por relevância, que nesse caso representa a quantificação dessa variação entre as faces originais. Em termos técnicos, esses são os autovalores associados aos autovetores que encontramos: os Note que essa ordenação é importante, já que no algoritmo de reconhecimento é interessante utilizarmos apenas *eigenfaces* em que há maior variação, que são consideradas melhores.

A combinação linear dos componentes principais de cada uma delas (ou seja, a 'receita' de cada imagem) é o que importa para determinarmos se duas fotos representam ou não a mesma pessoa. Note que, se utilizassemos todos os autovetores obtidos, inserindo uma imagem que fazia parte do dataset original, essa combinação nos daria exatamente a imagem anterior. Entretanto, não há ganho nenhum em fazer isso, já que muitos desses autovetores teriam pouquíssima contribuição e perderíamos eficiência. Na prática, determinaremos um conjunto dos M melhores autovetores, tomando eles como o span do chamado *face space*.

Além da aplicação em reconhecimento facial, esse *face space* nos traz uma outra possibilidade muito interessante: através de combinações lineares das *eigenfaces* presentes nele, é possível obter reconstruções aproximadas de qualquer imagem, inclusive de uma que não esteja presente no *dataset* original.

1.2 O Cálculo

Agora que a ideia já está mais clara, falaremos diretamente sobre a Álgebra Linear utilizada nesse processo, justificando as passagens que nos levam às *eigenfaces*.

Conforme já mencionado, começamos a desenvolver a situação partindo de um *dataset* composto por imagens com $m \times n$ pixels, em *grayscale*. Os valores originais do *grayscale* vão de 0 a 255, mas é interessante utilizarmos valores normalizados, então dividiremos os valores por 255. Em seguida, faremos a conversão de cada uma dessas matrizes $m \times n$ em vetores linha, de formato $1 \times mn$, que serão chamados de $f_1, f_2, ..., f_k$.

A partir desses vetores criamos a matriz F, em que cada linha será um desses vetores. Na verdade, matriz F , de formato $k \times mn$, realiza a função de armazenar todos os dados do *dataset* analisado:

$$F = \begin{bmatrix} — & f_1 & — \\ — & f_2 & — \\ & \vdots & \\ — & f_k & — \end{bmatrix}$$

Agora, a fim de reduzir a dimensionalidade e redundância de informação presente nas operações que iremos realizar em cima da matriz F e seus autovalores e autovetores, faremos um processo de análise de componentes principais, ou PCA. O primeiro passo desse processo será retirar de cada linha de F a chamada *face média*, obtendo linhas centralizadas.

$$f_{media} = \frac{1}{k} \sum_{i=1}^{k} f_i$$

Fazemos isso para termos linhas que não contém essa informação redundante, armazenando somente as variações de cada imagem em relação à média, que é a parte que nos interessa:

$$F_c = \begin{bmatrix} — & f_1 - f_{media} & — \\ — & f_2 - f_{media} & — \\ & \vdots & \\ — & f_k - f_{media} & — \end{bmatrix}$$

O próximo passo será realizar a decomposição em valores singulares, o SVD, da matriz F_c:

$$F_c = U\Sigma V^T$$

Nessa decomposição, obtemos $U_{k\times k}$, $\Sigma_{k\times k}$ e $V^T_{k\times mn}$ tais que as linhas de V^T representam os autovetores que buscamos. Retornando-os ao seu formato original obtemos as *eigenfaces*. Esse procedimento, na verdade, equivale à obtenção dos autovetores da matriz de covariância de F_c, dada por $F_cF_c^T$, e multiplicação destes pelo próprio F_c

A ordenação destes ocorre, em ordem decrescente, de acordo com os valores singulares associados a eles. Assim, escolhemos os M primeiros que tiverem significância, conforme o discutido na seção anterior. Esses M autovetores, então, se tornarão as linhas da matriz de *eigenfaces* E, determinando o chamado *face space*. O método utilizado para determinar essa quantidade M será melhor descrita mais adiante, conforme desenvolvermos o algoritmo, mas vale notar que ele é diretamente relacionado à queda brusca da relevância atribuída a eles.

Agora que já obtivemos as *eigenfaces*, vamos tratar da utilização destas para reconhecimento facial. Primeiramente, é necessário realizar a projeção de cada imagem f_c para o *face space*. Isso nos dará o coeficiente que representa a importância de cada uma dessas *eigenfaces* na variância dessa imagem. Como a base do *face space* é ortonormal, basta realizar uma multiplicação de matrizes para conseguir esses valores:

$$W = F_cE^T$$

Cada linha w_i da matriz W corresponde aos coeficientes ligados à imagem i da matriz F_c.

A partir de uma nova imagem, que deverá seguir as mesmas condições das imagens utilizadas anteriormente (*grayscale*, luminosidade e posicionamento), podemos realizar a vetorização e transformá-la em um vetor linha f_{input}. Podemos, então, de forma semelhante à realizada acima para as imagens já computadas, encontrar os pesos de f_{input} em relação aos *eigenfaces*:

$$w_{input} = (f_{input} - f_{media}) \cdot E^T$$

Onde w_{input} é o vetor linha $1 \times M$ formado pelos 'pesos' que definem a contribuição de cada *eigenface* na formação da imagem f_{input}. Em seguida, faremos a própria projeção da nova imagem no *face space*, utilizando *eigenfaces* e seus pesos:

$$p = E^T \cdot w_{input}$$

O objetivo desse processo é podermos, inicialmente, analisar a distância d entre a projeção da imagem no *face space* e a imagem em si (na verdade, sua versão mais relevante, reduzida da imagem média, $f_m = f_{input} - f_{media}$), então faremos a seguinte norma:

$$d = ||f_m - p||^2$$

Note que a interpretação dessa distância, que representa a distância do vetor da imagem à sua projeção no *face space*, é tal que, se d for maior que um limite determinado comparativamente em testes, simplesmente nos indica que a imagem não deve representar uma face. Essa informação é interessante, mas temos interesse em ir além e reconhecer, quando a imagem de fato é um rosto, a quem ele pertence.

Para tal, utilizaremos os pesos que já encontramos, relativos à imagem input e às imagens do *dataset*. Faremos uma operação análoga de norma, calculando uma distância d' que compara distâncias no próprio *face space*:

$$d' = ||w_{input} - w_i||^2$$

Essa distância é analisada para cada w_i, ou seja, cada linha de W, comparando os pesos da imagem inserida com todas as imagens analisadas. A menor dessas distâncias será indicada como o reconhecimento facial que buscamos: salvo a uma distância muito grande, que também deve obedecer a um limite comparativo, o resultado nos dirá a pessoa dentre as imagens originais com quem essa imagem nova se parece.

A tabela a seguir pode clarificar a situação:

	$d < lim_1$	$d > lim_1$
$d' < lim_2$	Face humana reconhecida	Não é uma face humana
$d' > lim_2$	Face humana, mas não reconhecida	Não é uma face humana

Vale comentar, ainda, que o cálculo das *eigenvalues* nos permite realizar outras experiências, como a de reconstrução de um rosto. Nesse processo, recebemos uma imagem input e temos interesse em produzir uma reprodução muito semelhante com as imagens de nosso *dataset*. Para isso, basta retomarmos f_m, ou seja, a face média subtraída do input vetorizado e as eigenfaces que descobrimos:

$$w_{input} = f_m \cdot E^T$$

Em seguida faremos a projeção dessa imagem no *face space*, utilizando os pesos encontrados e as próprias *eigenfaces* e adicionamos a imagem média novamente, obtendo a imagem f_r, muito próxima da original, utilizando apenas as imagens originais do *dataset*.

$$f_r = (w_{input} \cdot E) + f_{media}$$

Estamos prontos, então, para partir para as aplicações destes fatos, onde explicitaremos o algoritmo que elaboramos e os resultados deste.

2 Aplicação

2.1 Base de dados

Para o desenvolvimento desse trabalho, tínhamos interesse em realizar análises em um dataset composto por imagens de personalidades da mídia, famosos em geral. A maior parte dos algoritmos de reconhecimento através de *eigenfaces* é feita em imagens capturadas para esse próposito. Esse fator garante, em geral, boas condições de luminosidade, centralização e qualidade.

Acabamos nos deparando com uma base de dados chamada "Labeled Faces in the Wild"[Huang et al. 2007], desenvolvida pela Universidade de Massachusetts e apelidada de LFW. O próprio site da base declara alguns problemas, como a predominância de alguns grupos: há pouquíssimas crianças, proporcionalmente poucas mulheres e variações étnicas inconsistentes.

Buscando padronização, encontramos uma variação interessante da base original, "LFWcrop"[LFWcrop Face Dataset 2019], que já nos fornecia as imagens mais aproximadas, com posicionamento melhor alinhado, e como bônus essas imagens já estavam em *grayscale*. Para podermos aplicar o método desejado, mantivemos apenas as personalidades com no mínimo 8 fotos, mantendo exatamente 7 destas como conjunto para construção das *eigenfaces* e uma para testes, e optamos por filtrar, destes, os famosos de fato mais conhecidos, terminando com 62 personalidades para análise.

Figura 1 – Sample do dataset

Cabe mencionar que, conforme declara o nome da base, as imagens são de fato tiradas "in the wild", ou seja, em situações aleatórias, com diferentes luminosidades e etc.. Trataremos dos descobramentos desses fatores mais adiante.

2.2 Código

Agora, iremos mostrar o código do algoritmo e seu funcionamento. Primeiramente, vamos importar os pacotes necessários e definir uma função de auxílio na leitura de imagens:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
from PIL import Image

def read_image(path):
    # Reads an image from the determined path
    # The image has to have a bit-depth of 8 (each pixel's value is in the 0-255 range)
    img_to_recognize = img.imread(path)
    img_to_recognize = Image.fromarray(np.uint8(img_to_recognize))
    img_to_recognize = img_to_recognize.resize((150, 150)) # Resizes the image to the appropriate size
    img_to_recognize = img_to_recognize.convert('L') # Converts it to grayscale
    img_to_recognize = np.array(img_to_recognize) # Converts it into an array
    return img_to_recognize.flatten()/255 # Flattens and normalizes the data
```

Também iremos criar uma função para auxiliar no processo de display de imagens a partir de uma array achatada:

```
def display_image(array):
    # Displays the image represented by a flattened array
    resized = np.resize(array, (150, 150))
    fig = plt.imshow(resized, cmap="gray")
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
    fig
```

Com essas funções disponivel, vamos ler todas as 434 imagens no diretório 'faces' e montar a matriz F:

```
images = [] # List that will house all individual arrays of images, it will become a 2D array later
```

```
2
3 for file in os.listdir('faces'):
4     images.append(read_image(os.path.join('faces', file)))
5
6 image_matrix = np.row_stack(tuple(images))
```

Agora criaremos uma função para auxiliar no processo de PCA:

```
1 def pca(X):
2     # X is the data matrix
3     mean = np.mean(X, axis=0)
4     centered_data = X-mean
5     U, S, Vh = np.linalg.svd(centered_data, full_matrices=False)
6
7     return Vh, mean, centered_data, S**2
8
9
10 Vh, average_matrix, subtracted, eigenvalues = pca(image_matrix)
```

Essa função nos retorna quatro arrays:

- O array *Vh*, que representa os autovetores da matriz de covariância;
- O array *mean*, que representa a face média;
- O array *centered_data*, que representa a matriz F_c (Matriz de imagens subtraídas pela média);
- O array *S**2*, que representa os autovalores da matriz de covariância

Podemos visualizar, por exemplo, a face média:

Figura 2 – Face média proveniente do dataset

Também podemos, agora, mostrar a discrepância entre as *eigenfaces* de maior e menor valores:

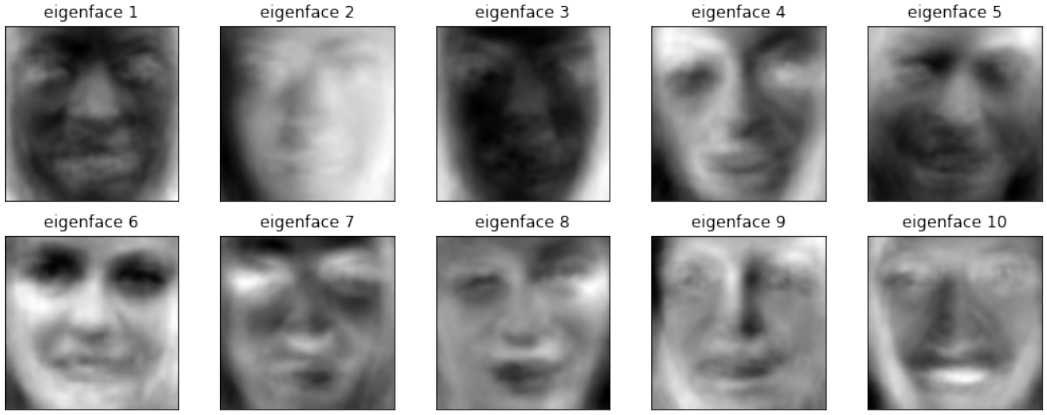

Figura 3 – Eigenfaces com maiores valores

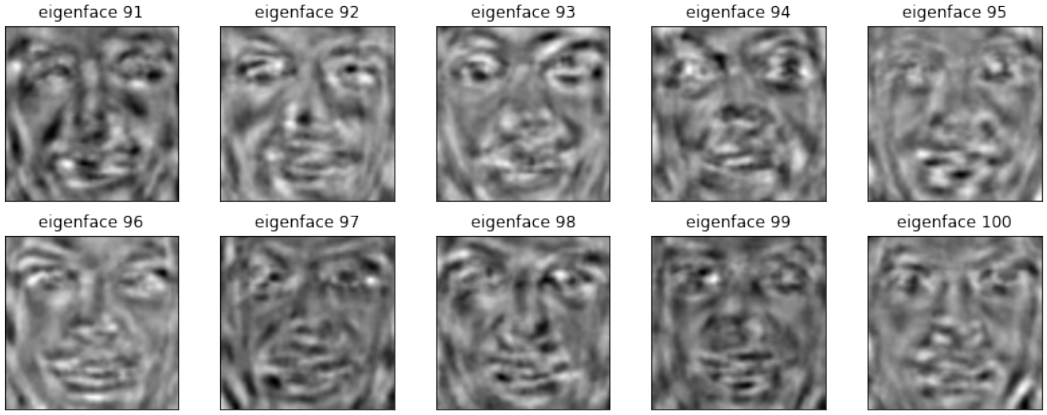

Figura 4 – Eigenfaces com valores médios

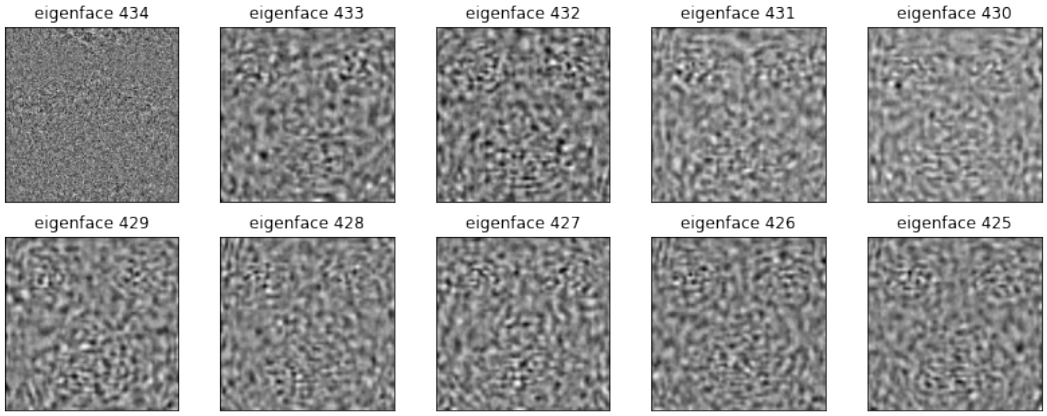

Figura 5 – Eigenfaces com menores valores

Agora que temos esses autovalores, precisamos descobrir qual é o número M que define os melhores para serem usados no *face space*. Para isso, basta realizar a seguinte operação:

```
percent_eigenvalues = [eigenvalue/np.sum(eigenvalues) for eigenvalue in
    eigenvalues]
count = 0
total_var = 0

for eigenvalue in percent_eigenvalues:
    total_var += eigenvalue
    count += 1
    if total_var > 0.95:
        break

print("Count:", count, "\nTotal Variance:", total_var)
>>Count: 106
>>Total Variance: 0.9503120935939878
```

Assim, conseguimos ver que, apesar de termos 434 autovetores, os primeiros 106 contém as informações necessárias para explicar 95% da variação presente no conjunto de imagens! Isso nos permite reduzir a dimensionalidade do *face space* em aproximadamente 75%, com uma perda bem pequena de informação.

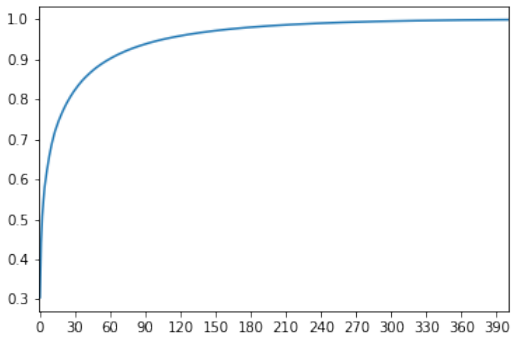

Figura 6 – Gráfico que mostra a progressão do acúmulo de variância

Podemos mostrar esse efeito em ação ao tentar reconstruir imagens do dataset usando quantias diferentes de *eigenfaces*:

Figura 7 – Diferentes reconstruções

(a) Imagem original

(b) Reconstruida com os primeiros 106 autovetores

(c) Reconstruida com os primeiros 300 autovetores

Essas imagens foram reconstruidas através da seguinte função:

```
def reconstruction(centered_data, eigenfaces, average, h, w, image_index):
    weights = np.dot(centered_data, eigenfaces.T) # Gets the weight significance of each eigenface
    weighted_vectors=np.dot(weights[image_index, :], eigenfaces) # Multiplies each eigenface by its weight
    recovered_image=(average + weighted_vectors).reshape(h, w) # Adds each weighted eigenface to the average face
    return recovered_image
```

Tambem criaremos uma função de reconstrução que aceita imagens de fora do dataset original, para usarmos posteriormente:

```
def reconstruction_outsider(eigenfaces, average, h, w, path):
    outsider = read_image(path) # Reads the image
    outsider = outsider - average # Centralizes the data
    weights = np.dot(outsider, eigenfaces.T) # Gets the weight significance of each eigenface
    weighted_vectors = np.dot(weights, eigenfaces) # Multiplies each eigenface by its weight
    recovered_image = (average + weighted_vectors).reshape(h, w) # Adds each weighted eigenface to the average face
    return recovered_image
```

Agora, basta implementar a função de reconhecimento:

```
def recognize(path, eig_num, face_limit, person_limit):
    # Path is the file path, eig_num is the amount of wanted eigenfaces on the facespace
    img_to_recognize = read_image(path)
    subtracted_matrix_rec = img_to_recognize - average_matrix # Centralizes the inputted image
    subtracted_matrix_rec = subtracted_matrix_rec.flatten()
```

```
6
7     eigenfaces_matrix = Vh[:eig_num,:] # Gets the requested amount of
    eigenfaces
8
9     weight = subtracted_matrix_rec @ eigenfaces_matrix.T # Gets the
    eigenface weights
10
11    projection = eigenfaces_matrix.T @ weight # Gets the projection of
    the image on the facespace
12
13    proj_error = np.linalg.norm(subtracted_matrix_rec - projection)*255
    # Gets the projection error
14
15
16    original_faces_weights = eigenfaces_matrix @ subtracted.T # Gets the
     eigenface weights of each original face
17
18
19    dist_in_space = []
20    # Checks the distance between the weights
21    # of the unknown face and every original face
22    for i in range(len(subtracted[:,0])):
23        dist = np.linalg.norm(original_faces_weights[:,i] - weight)
24        dist_in_space.append(dist)
25
26    dist_in_space = np.array(dist_in_space)
27
28    face_error = dist_in_space.min() # Gets the lowest distance
29
30    guess_index = np.argmin(dist_in_space) # Gets the image tied to that
     lowest distance
31
32    if proj_error > face_limit: # Checks if the error is higher than the
     set limit
33        guess = "Not a face"
34    elif face_error > person_limit:
35        guess = "Unknown face"
36    else:
37        celebrity_photos = os.listdir('faces')
38        celebrity_names = [name[:name.find('0')-1].replace("_", " ") for
     name in celebrity_photos] # Gets the names of all images
39        guess = celebrity_names[guess_index] # Gets the predicted name
40
41    display_image(image_matrix[guess_index]) # Displays the closest
    image
42    return proj_error, face_error, guess
```

Com toda essa estrutura pronta, basta começar a testagem e relatar os resultados e problemas obtidos, tópico que será comentado na próxima seção. O código inteiro, inclusive das testagens, está disponível em nosso GitHub.

2.3 Resultados e problemas

Após o desenvolvimento do algoritmo, iniciamos nossa fase de testes, em que o objetivo inicial era testar diferentes situações, forçando seus limites.

A princípio tivemos interesse em testar nosso algoritmo quanto à distinção entre faces e não-faces. Testamos objetos e itens que achamos interessantes, e avaliamos resultados muito bons nesse quesito. Através da comparação entre resultados, definimos $lim_1 = 3500$, ou seja, sempre que encontrarmos $d \leq 3500$, consideramos que encontramos, de fato, uma face. Veja a seguir alguns casos:

Figura 8 – Respectivos reconhecimentos, resultados e distância calculada pelo algoritmo

(a) Não é uma face
Correto
$d \approx 8462, d' \approx 36$

(b) Não é uma face
Correto
$d \approx 5709, d' \approx 35$

(c) É uma face
Incorreto
$d \approx 2150, d' \approx 14$

De fato, objetos e fotos muito distantes de faces são consideradas não-faces, e escolhemos testar a imagem de uma boneca "Barbie" nas proporções padrão das faces de famosos do *dataset*. O algoritmo considerou a imagem uma face, e acreditamos que, levando em conta o teste, esse resultado foi satisfatório!

Dessa forma, passamos a realizar testes de imagens que pertenciam ao conjunto de imagens utilizadas para construir as eigenfaces, buscando avaliar os resultados a partir de d' e tivemos resultados muito satisfatórios. Utilizamos, conforme discutido, 106 *eigenfaces*, definimos o $lim_2 = 22.2$, e obtivemos resultados conforme os exemplos:

Figura 9 – Respectivos reconhecimentos, resultados e distâncias calculados pelo algoritmo

(a) Angelina Jolie
Correto
$d \approx 2149, d' \approx 0$

(b) Leonardo diCaprio
Correto
$d \approx 2454, d' \approx 0$

(c) Rubens Barrichello
Correto
$d \approx 1306, d' \approx 0$

Todos os resultados, sob essas condições, foram positivos. Das 434 imagens que pertencem ao *subdataset* que utilizamos, 434 foram reconhecidas de maneira acertada, então atingimos uma taxa de 100% de sucesso.

Realizar os testes em imagens que não foram utilizadas para construir as *eigenfaces*, no entanto, se mostrou uma situação mais delicada que o esperado por nós. As condições naturais e diferenciadas em que as imagens foram tiradas influenciaram muito fortemente nos resultados dessa tentativa de reconhecimento facial, gerando situações que chegaram a pontos cômicos:

Figura 10 – Respectivos reconhecimentos, resultados e distâncias calculados pelo algoritmo

(a) Julianne Moore
Incorreto
$d \approx 2149, d' \approx 0$

(b) Luiz Inacio Lula da Silva - Incorreto
$d \approx 2234, d' \approx 14$

(c) Rubens Barrichello
Correto
$d \approx 2250, d' \approx 13$

Os resultados não foram satisfatórios, conseguimos apenas 25,8% de sucesso em reconhecer indivíduos e 1,6% foram consideradas faces desconhecidas, já que o d' ultrapassou o lim_2. A taxa de erro foi bem grande, e nesse ponto nos questionamos se haveria algo de errado com nosso algoritmo, então buscamos uma base de dados que tínhamos certeza que seria propícia para esta análise. Utilizamos uma base padrão, chamada "AT&T Database of Faces"[AT&T 1992], em que temos imagens produzidas em ambiente controlado, sob a

mesma luminosidade. Cada indivíduo presente no conjunto conta com uma série de fotos tiradas em sequência em que há leves mudanças de ângulo.

A partir dessa análise realizada com o mesmo algoritmo tivemos uma taxa de sucesso de 92,5% utilizando imagens que não foram utilizadas para a construção das *eigenfaces*, que certamente é bastante satisfatório. Veja a seguir exemplos desse reconhecimento (vale mencionar que os limites de d e d' não serão os mesmos da base anterior):

Figura 11 – Respectivos reconhecimentos, resultados e distâncias calculados pelo algoritmo

(a) Indivíduo 1
Correto
$d \approx 3220, d' \approx 15$

(b) Indivíduo 3
Correto
$d \approx 2274, d' \approx 5$

(c) Indivíduo 6
Correto
$d \approx 2386, d' \approx 3$

Para termos mais segurança, analisamos as taxas de sucesso de cada:

	LFW	AT&T
Imagens do dataset	100%	100%
Outras imagens	25.8%	92.5%

Foi possível verificar, então, que o problema residia na base LFW. O método de *eigenfaces*, como mencionado anteriormente, não é robusto a ponto de sustentar conjuntos de imagens deficitárias, e a falta de condições propícias como luminosidade e posicionamento mais padronizado prejudicou muito estes resultados.

Com as análises do algoritmo de reconhecimento facial já realizadas, podemos falar brevemente do resultado da reconstrução de imagens.Como aqui tínhamos bastante interesse em que elas ficassem o mais próximas possíveis do original, utilizamos todas as 434 *eigenfaces*. As imagens que já pertenciam ao *dataset* terão, é claro, resultados mais satisfatórios. Veja exemplos a seguir:

Figura 12 – Reconstrução do Bill Gates

(a) Foto original

(b) Reconstrução

A reconstrução teve um resultado extremamente próximo da imagem original, uma vez que a imagem fazia parte das utilizadas para criar as *eigenfaces*.

Figura 13 – Reconstrução do Eduardo Adame

(a) Foto original

(b) Reconstrução

Utilizamos novamente a função de reconstrução com a imagem de um humano que não pertencia ao *dataset* original. Ainda que claramente menos ideal, ainda obtivemos um resultado satisfatório, já que ainda a nova imagem representa um rosto humano.

Figura 14 – Reconstrução da gata Pandora

(a) Foto original

(b) Reconstrução

Em nosso último teste utilizamos uma foto de um gato, e o resultado imperfeito ainda nos surpreendeu positivamente. As *eigenfaces*, todas criadas a partir de fotos de humanos, foram suficientes para recriar uma imagem que se assemelha bastante a um gato. De fato, as *eigenfaces* são de um poder notável, e nos dão a esperança de que temos todos um pouquinho de rostos famosos também.

Conclusão

Esse trabalho certamente clarifica a importância e o poder da Álgebra Linear quando posta em uma situação prática e real do dia a dia. Apesar das *eigenfaces* não se compararem aos sistemas de reconhecimento facial presentes no mercado atualmente, ainda é chocante observar a forma como essas simples operações algébricas podem ser aplicadas para reconhecimento e reconstrução de imagens.

Estudar e replicar aplicações como a das *eigenfaces* tornam as diversas fórmulas vistas em aula, que normalmente são difíceis de interpretar por conta da pesada notação matemática, mais simples de entender, colocando a mostra resultados visíveis e intuitivos. Por conta disso e pela performance exemplar no *dataset* alternativo em que testamos o algoritmo, consideramos esse projeto um sucesso, mesmo que os resultados não tenham sido muito desejáveis com as imagens escolhidas originalmente.

Referências

[Acar 2018]ACAR, N. *Towards Data Science, Eigenfaces: Recovering Humans from Ghosts*. 2018. Disponível em: <https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184>. Nenhuma citação no texto.

[AT&T 1992]AT&T. *AT&T Database of Faces*. [S.l.], 1992. Citado na página 17.

[Huang et al. 2007]HUANG, G. B. et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. [S.l.], 2007. Citado na página 9.

[Jaadi 2021]JAADI, Z. *Built In: A Step-by-Step Explanation of Principal Component Analysis (PCA)*. 2021. Disponível em: <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>. Nenhuma citação no texto.

[Kutz 2015]KUTZ, N. *Beginning Scientific Computing: Singular Value Decomposition*. 2015. Disponível em: <https://faculty.washington.edu/kutz/am301/page1/page15/>. Nenhuma citação no texto.

[LFWcrop Face Dataset 2019]LFWCROP Face Dataset. 2019. Disponível em: <https://conradsanderson.id.au/lfwcrop/>. Citado na página 9.

[Navarrete 2020]NAVARRETE, W. *Towards Data Science: Principal Component Analysis with NumPy*. 2020. Disponível em: <https://towardsdatascience.com/pca-with-numpy-58917c1d0391>. Nenhuma citação no texto.

[Sandipanweb: EigenFaces and A Simple Face Detector with PCA/SVD in Python 2018] SANDIPANWEB: EigenFaces and A Simple Face Detector with PCA/SVD in Python. 2018. Disponível em: <https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/>. Nenhuma citação no texto.

[Turk 1991]TURK, A. P. M. *Eigenfaces for Recognition*. [S.l.], 1991. Nenhuma citação no texto.

[Wikipedia 2021]WIKIPEDIA. 2021. Disponível em: <https://en.wikipedia.org/wiki/Eigenface>. Nenhuma citação no texto.