Eigenfaces: Uma aplicação para o reconhecimento facial

Kauan Mariani Ferreira e Pedro Henrique Coterli

Novembro de 2023

1 Introdução

Com o advento da câmera fotográfica, a humanidade criou um novo problema para si: lidar com imagens. A princípio, essas fotografias eram usadas apenas para registrar momentos, entretanto, com a globalização e a digitalização do mundo, fotos viraram um importante meio para se comunicar. Além disso, com os avanços tecnológicos na área de monitoramento, a análise de imagens tornou-se um dos principais pilares dos campos investigativo e de segurança.

Apesar de toda essa problemática, felizmente, imagens representadas por câmeras digitais nada mais são do que conjuntos de pixels, os quais podem ser representados por uma simples matriz de números. Assim, é possível utilizar ferramentas matemáticas para lidar com essas figuras, tornando possível também a implementação de tecnicas de computação em soluções para problemas envolvendo análises de imagens.

Com isso em mente, em 1991, Matthew Turk e Alex Pentland[1] publicaram um artigo a respeito de um processo relativamente simples para criar um modelo de reconhecimento facial quase em tempo real, ao qual deram o nome de **eigenfaces**. Baseado na vetorização de imagens, esse procedimento utiliza simplesmente de ferramentas da álgebra linear para identificar faces em imagens e até mesmo relacioná-las a uma pessoa específica.

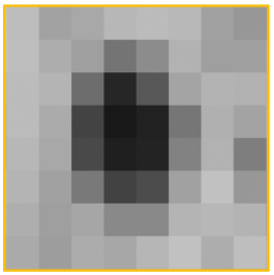
No presente trabalho, iniciaremos comentando a respeito da transformação das imagens em vetores, o que possibilita seu tratamento matemático e computacional. Em seguida, explicaremos o processo de redução de dimensionalidade denominado Análise de Componentes Principais (PCA, na sigla em inglês) e como ele é utilizado nesse tópico. Por fim, realizaremos a aplicação computacional das eigenfaces com o objetivo de, dada uma base de retratos, descobrir semelhanças entre um indíviduo exterior a esses dados e os presentes neles e até, eventualmente, identificá-lo.

2 Aproximação matemática

2.1 Transformando as imagens em vetores

Em nossa análise, precisaremos tratar as imagens como matrizes e, consequentemente, precisaremos de alguma forma de transformar nossas imagens em números e, em seguida, em vetores. De modo a não complicar muito o entendimento dos dados e não tornar a base muito grande, vamos trabalhar apenas com imagens na escala de cinza e de dimensão 231 x 195.

Para realizar esse procedimento, é importante ressaltar que cada imagem é formada por pixels e, como estamos trabalhando apenas com uma escala de cinza, cada pixel virá com um número entre 0 e 255, em que 0 representa o preto e 255 o branco, bem como mostrado na figura 1. A ideia para explorarmos esses dados será transfomar cada imagem 231 x 195 em um vetor linha, cujo tamanho será 1 x 45045, e adicioná-lo a uma matriz. Com isso, conseguimos transformar todas as nossas imagens em uma matriz M x 45045, com M representando o número de imagens utilizadas.



=

190	163	169	184	188	185	162	152
191	174	160	117	141	179	160	161
188	181	110	39	88	165	180	178
188	172	68	25	34	119	177	164
184	169	74	32	36	131	179	127
181	162	122	67	77	162	194	152
175	160	167	138	138	181	184	180
171	158	173	168	183	194	174	191

Figura 1: Transformação de uma imagem em uma matriz[2]

2.2 Análise de Componentes Principais (PCA)

A Análise de Componentes Principais é um método de redução de dimensionalidade frequentemente utilizado em conjuntos de dados com um número muito grande de variáveis. Com a aplicação desse processo, a quantidade de variáveis a serem analisadas é reduzida, de forma que seu processamento se torna bem mais simples. Para isso, a PCA é capaz de identificar as variáveis que melhor representam o dado como um todo e excluir apenas variáveis não tão significativas, fazendo com que, apesar da eliminação de valores, a perda de informação seja relativamente baixa em comparação à simplificação obtida.

2.2.1 Padronização

A primeira fase do processo de Análise de Componentes Principais refere-se à padronização dos dados, cujo objetivo é amenizar a diferença entre os registros e colocá-los na mesma escala. Dessa forma, cada variável contribui de forma relativamente equivalente para a análise.

Para isso, é subtraída de cada valor a média dos valores daquela variável e esse resultado é dividido pelo desvio-padrão desses registros, como mostrado abaixo:

$$valor_{novo} = \frac{(valor_{antigo} - média)}{desvio\ padrão}$$

Dessa forma, a distribuição dos dados é preservada, mas sua escala é alterada de modo a igualar sua média a 0 e a normalizar sua escala. Isso pode ser observado na figura 2.

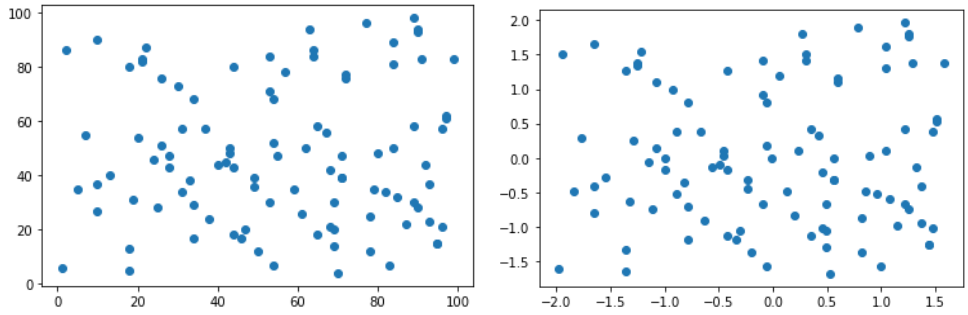


Figura 2: Padronização dos dados

2.2.2 Cálculo da matriz de covariância

O objetivo da matriz de covariância é estabelecer relações entre as variáveis da base de dados, de forma a identificar aquelas que são direta ou inversamente correlacionadas e aquelas que não têm relação umas com a outras.

Se uma covariância é positiva, então as variáveis são diretamente correlacionadas, ou seja, crescem ou decrescem juntas. Se for negativa, elas são inversamente correlacionadas, isto é, enquanto uma cresce, a outra decresce. Por fim, valores próximos de zero indicam ausência de correlação entre as variáveis.

Agora, suponhamos uma matriz de dados A com m registros e n variáveis, como indicado a seguir.

$$A = \begin{array}{c} \\ r_1 \\ r_2 \\ \vdots \\ r_m \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & \cdots & v_n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \end{array}$$

Para calcularmos as covariâncias entre as variáveis dessa matriz, precisamos fazer o produto interno entre cada par de variáveis, ou seja, entre todas as colunas dessa matriz. Para isso, basta fazermos $A^T A$ e obteremos a seguinte matriz de covariâncias:

$$\begin{bmatrix} Cov(v_1, v_1) & Cov(v_1, v_2) & \cdots & Cov(v_1, v_n) \\ Cov(v_2, v_1) & Cov(v_2, v_2) & \cdots & Cov(v_2, v_n) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(v_n, v_1) & Cov(v_n, v_2) & \cdots & Cov(v_n, v_n) \end{bmatrix}$$

Dois pontos importantes sobre essa matriz:

- A diagonal é composta pelas covariâncias de uma variável com ela mesmo. Por definição, esses valores representam a **variância** de cada variável.
- Devido ao fato de o produto interno ser comutativo ($u^T v = v^T u$), temos que $Cov(v_i, v_j) = Cov(v_j, v_i)$. Portanto, essa matriz é **simétrica**.

2.2.3 Autovalores e autovetores da matriz de covariância

Agora, vamos analisar essa matriz de covariância para ver que informações ela contém.

Os autovetores dessa matriz indicam as direções de maior variância do dado, enquanto que seus respectivos autovalores representam a variância na direção de seu autovetor. Com isso, podemos rankear os autovetores de acordo com seus autovalores de forma a descobrir as direções em que o dado mais varia e, portanto, mais significativas. Essas direções são as **componentes principais**.

Essas componentes são combinações lineares das variáveis originais, não tendo nenhum significado real para o dado. Apesar disso, podemos usá-las para simplificar nossa base, já que podemos desconsiderar as componentes principais de menores autovalores (ou seja, que compõem uma parcela pequena da variância do dado) e, assim, representar de forma mais simples a distribuição do dado. Além disso, devido ao fato de a matriz de covariância ser simétrica, esses autovetores são **ortogonais**, fazendo com que possamos formar com eles uma nova base ortogonal para os dados.

2.2.4 Exemplo de uso

Vamos aplicar esse procedimento a uma base de modelo para demonstrar suas etapas e seu funcionamento. Comecemos com o seguinte dado já normalizado:

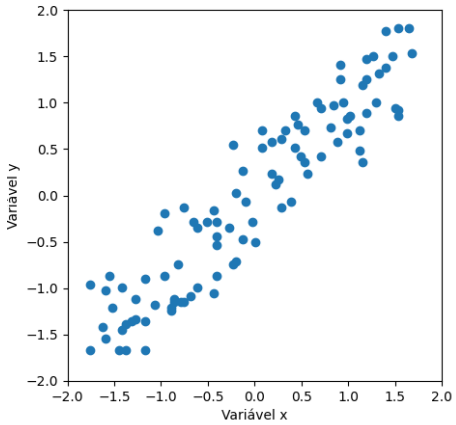


Figura 3: Dado original

Inicialmente, calculamos sua matriz de covariância e obtemos o seguinte resultado:

$$\begin{bmatrix} 100 & 92.37568063 \\ 92.37568063 & 100 \end{bmatrix}$$

A seguir, calculamos seus autovalores e autovetores:

$$\lambda_1 = 192,376 \qquad x_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\lambda_2 = 7,624 \qquad x_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Notemos que o autovalor do primeiro autovetor é bem mais significativo que o do primeiro. Assim, concluímos que a direção de x_1 é a de maior variância de todo o dado, enquanto que a de x_2 é a segunda maior. Além disso, é perceptível que esses dois autovetores são ortogonais, o que é lógico dado o fato de que a matriz de covariância é simétrica.

Dessa forma, conseguimos "separar"a variância total do dado entre as duas componentes principais obtidas por meio da comparação entre seus autovalores, ou seja, a proporção de cada um deles no somatório de todos os autovalores. Com isso, podemos gerar o seguinte gráfico:

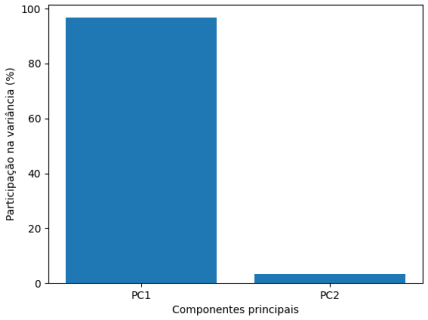


Figura 4: Proporção de cada componente principal (PC) na variância do dado em porcentagem

Podemos concluir que a direção do autovetor x_1 é responsável por quase toda a variância do dado. Se tivéssemos mais variáveis, obteríamos mais componentes principais, e poderíamos simplesmente desconsiderar as menos relevantes de modo a simplificar a análise dos dados. No entanto, nesse exemplo,

esse não é o objetivo.

Agora, plotemos no gráfico nossos autovetores para entendê-los melhor.

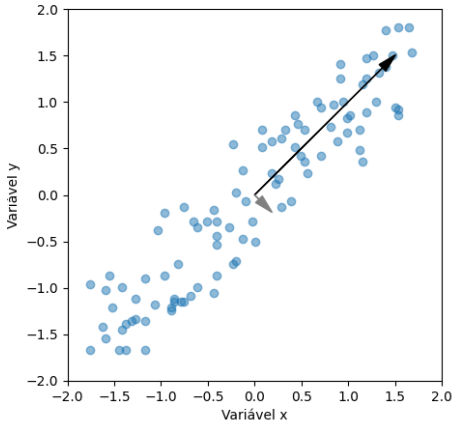


Figura 5: Dado original e suas direções de variância

Nessa figura, x_1 está representado em preto e x_2 em cinza. É visível o fato de a direção do vetor preto ser a direção de maior variância do dado.

Por fim, podemos agora projetar esses dados no espaço gerado pelas componentes principais para obter sua nova distribuição. Para isso, chamando a matriz de autovetores da matriz de covariância de $autovetores_{cov}$, basta fazermos:

$$dado_{novo} = (autovetores_{cov})^T dado_{original}$$

Fazendo isso com nossa base, obtemos, finalmente, a seguinte nova distribuição:

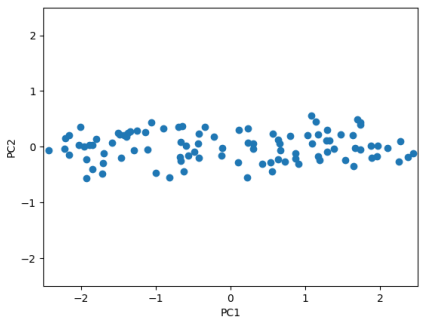


Figura 6: Novo dado

Com isso, conseguimos uma simplificação do dado fazendo com que seu principal eixo de variação seja o eixo horizontal e não mais uma direção genérica.

Vale ressaltar que o principal objetivo do método do PCA é a redução de dimensionalidade do dado. Nesse exemplo, apesar de aproximarmos nosso dado de algo unidimensional, não o fizemos realmente. A ideia geral desse processo é transformar uma base de dados de 50 variáveis, por exemplo, em uma base de 10, com essas novas 10 variáveis sendo combinações das variáveis originais que representem a maior parte da variação do dado. No entanto, isso não era muito prático de se mostrar aqui de uma forma simples (infelizmente, ainda não descobrimos como plotar 50 variáveis).

2.3 Eigenfaces

Seguindo a ideia da análise de componentes principais, podemos seguir com o cálculo das eigenfaces, que nada mais é do que uma base ortogonal do espaço das "faces". Cada imagem retratada por um vetor é um ponto no espaço $R^{N_1 \times N_2}$ no qual a dimensão da imagem é $N_1 \times N_2$. Como $N_1 \times N_2$ acaba sendo um número muito grande, procuraremos os principais componentes desse espaço, ou seja, os vetores que mais explicam a variação do dado.

Supomos que temos uma sequência de M fotos de $N_1 \times N_2$. Por consequência, temos uma matriz $N_1 \times N_2$ por M, na qual cada imagem é representado pela coluna Γ_i. Para seguir com a análise da PCA. calcularemos a imagem média Ψ do dataset:

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

E, para achar a base de dados centralizados, diminuiremos cada vetor do vetor médio e colocaremos todos numa matriz:

$$\Phi_i = \Gamma_i - \Psi$$
$$A = [\Phi_1, \Phi_2, ..., \Phi_M]$$

Agora buscaremos um conjunto de vetores ortonormais de forma que

$$\lambda_k = \frac{1}{M} \sum_{i=1}^{M} (u_k^T \Phi_i)^2,$$

no qual temos a restrição de u_k ser ortonormal. Os vetores u_k são os autovetores e os elementos λ_k são os autovalores da matriz de covariância, definida como:

$$C = \frac{1}{M} \sum_{i=1}^{M} (\Phi_i \Phi_i^T)$$
$$C = AA^T,$$

na qual A é a matriz com autovetores. No entanto, temos um problema com a quantidade de termos necessários para se calcular. A matriz C é de tamanho $(N_1 \times N_2)^2 \times (N_1 \times N_2)^2$, e como estamos tratando de vetores com um N muito grande, calcular uma matriz do tamanho de C seria algo muito custoso. No entanto, podemos utilizar o fato de que os autovalores de AA^T são os mesmos de $A^T A$, como mostrado abaixo:

$$A^T A v_i = \lambda v_i$$
$$AA^T A v_i = \lambda A v_i$$
$$(AA^T) A v_i = (\lambda) A v_i$$

Como consequência, ficamos com a matriz $A^T A$ muito mais simples de se calcular, pois no fim teremos uma matriz $M \times M$, com muito menos entradas. Reduzimos, assim, a quantidade de cálculos necessários de um número muito grande para M. Ficamos então com a matriz

$$L = A^T A$$
$$\text{onde} \quad L_{mn} = \Phi_i^T \Phi_i.$$

Chame os autovetores dessa matriz de v_i. Esses vetores formam as combinações lineares das N imagens escolhidas e, portanto, podemos voltar para encontrar os vetores u previamente especificados de acordo com:

$$u_l = \sum_{i=1}^{N} v_{lk} \Phi_k \qquad l = 1, 2..., M.$$

3 Aplicando eigenfaces para reconhecimento facial

Feita a explicação teórica, vamos à aplicação prática. Nesse projeto, temos 20 imagens do artista Will Smith e outras 20 da cantora Taylor Swift. Nosso objetivo é verificar se o método de eigenfaces é capaz de distingui-los e indentificá-los.

Antes de começarmos, um adendo: infelizmente, nessa aplicação computacional, não conseguimos utilizar a matriz dos dados com os registros nas colunas como explicado anteriormente. Em vez disso, tivemos que fazer das imagens vetores-linha. Tirando essa alteração, todo o restante do processo é semelhante ao descrito na parte teórica.

Inicialmente, importamos as bibliotecas necessárias.

[15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import os
```

Para que o método funcione, é necessário que todas as imagens possuam as mesmas dimensões. Dessa forma, optamos por figuras de tamanho 195x231.

[16]:
```
altura_imagem = 231
comprimento = 195
```

A seguir, fizemos a leituras das imagens e as convertemos em uma matriz tridimensional (cada imagem é uma matriz bidimensional e sua composição adiciona uma terceira dimensão).

[17]:
```
# Criando o path para onde as imagens estão
famousim = "imagens_artistas/"
# Criando uma lista para adicionar as imagens
famous_images = []

# Fazendo um loop for para iterar sobre todas as imagens
for folder in (os.listdir(famousim)):
    folder_path = os.path.join(famousim, folder)
    for image in (os.listdir(folder_path)):
        # Adicionando o path com cada imagem para termos o arquivo das imagens
        path = os.path.join(famousim, folder ,image)
        # Lendo a imagem
        img = cv2.imread(path)
        # Convertendo a imagem para preto e branco
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # Adicionando a imagem na lista
```

```
        famous_images.append(img)

# Transformando a lista num array
famous_images_np = np.array(famous_images)
```

Vamos dar uma olhada nas nossas imagens.

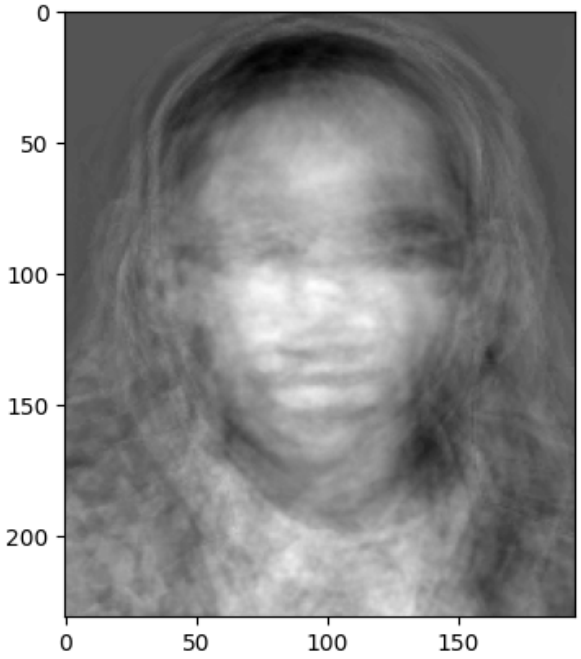
[18]:
```
# Plotando as imagens
count = 0
fig, ax = plt.subplots(5, 8, figsize=(10, 10))
for linha in range(5):
    for coluna in range(8):
        ax[linha,coluna].imshow(famous_images_np[count], cmap='gray')
        count += 1
        ax[linha, coluna].set_xticks([])
        ax[linha, coluna].set_yticks([])
        ax[linha, coluna].set_xticklabels([])
        ax[linha, coluna].set_yticklabels([])
        ax[linha, coluna].set_xlabel('')
        ax[linha, coluna].set_ylabel('')
```

Agora, transformamos essa matriz tridimensional em uma matriz bidimensional convertendo as matrizes de cada imagem para um único vetor-linha extremamente comprido. Após isso, calculamos a face média.

[19]:
```
# Cada imagem está armazenada em 3 dimensões, cada foto está numa lista, e cada␣
↪lista é uma lista de listas.
```

```
# Vamos transformar a imagem num vetor coluna muito grande.
train_famous_np_matrix = famous_images_np.reshape(famous_images_np.shape[0],␣
↪famous_images_np.shape[1]*famous_images_np.shape[2])
# Calculamos a face média
mean_train_face = np.mean(train_famous_np_matrix, axis=0)
# Criamos uma matriz subtraindo a face média
centered_train_face = train_famous_np_matrix - mean_train_face

# Mostrando a face média
plt.figure()
plt.imshow(mean_train_face.reshape(altura_imagem,comprimento), cmap='gray')
```

[19]:
```
<matplotlib.image.AxesImage at 0x15621f964a0>
```



A seguir, calculamos a decomposição SVD dessa matriz.

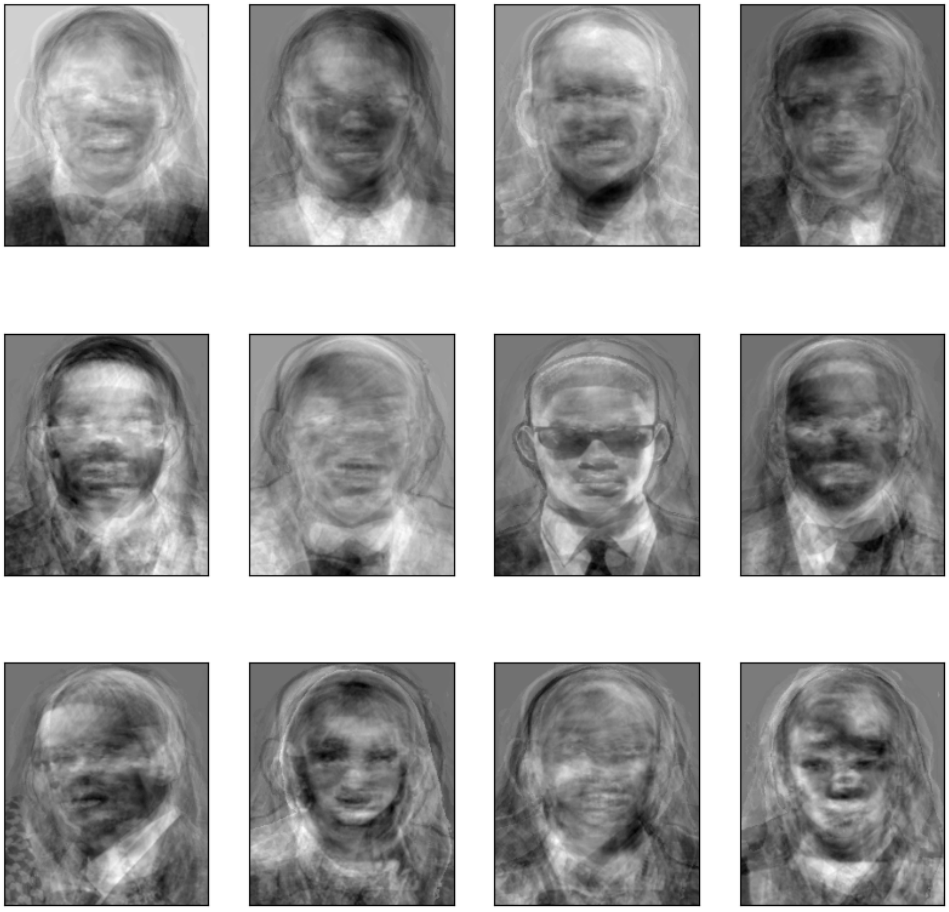
[20]:
```
# Decompomdo a matriz em SVD
U_Face_Famous, D_Face_Famous, V_Face_Famous = np.linalg.svd(centered_train_face,␣
↪full_matrices=False)
```

Vamos selecionar as 12 primeiras componentes principais e plotar suas eigenfaces.

[21]:
```
# Vamos obter a quantidade de componentes que queremos de nossa matriz.
C_Faces_TRAIN = V_Face_Famous[:12]
eigen_faces_train = C_Faces_TRAIN.reshape((-1, altura_imagem, comprimento))
```

[22]:
```
count = 0
fig, ax = plt.subplots(3, 4, figsize=(10, 10))
for linha in range(3):
    for coluna in range(4):
        ax[linha,coluna].imshow(eigen_faces_train[count], cmap='gray')
        count += 1
        ax[linha, coluna].set_xticks([])
        ax[linha, coluna].set_yticks([])
```

```
        ax[linha, coluna].set_xticklabels([])
        ax[linha, coluna].set_yticklabels([])
        ax[linha, coluna].set_xlabel('')
        ax[linha, coluna].set_ylabel('')
```



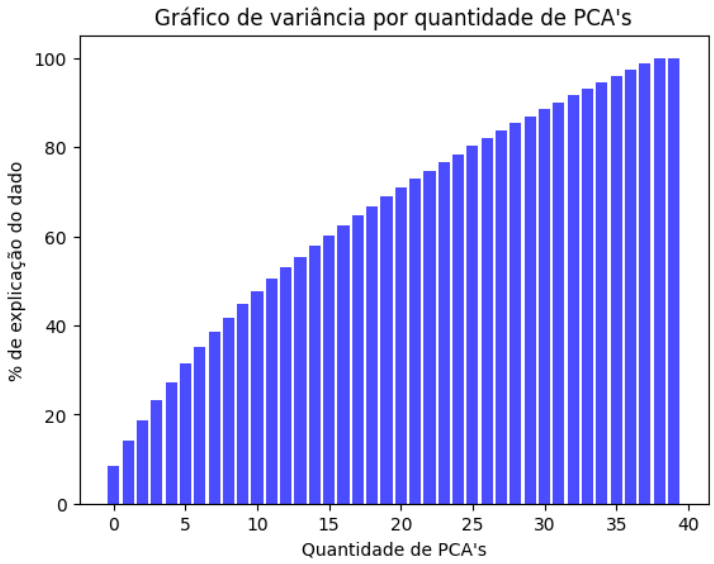
Vamos dar uma olhada em nossas imagens agora centralizadas (subtraídas a média).

[23]:
```
count = 0
fig, ax = plt.subplots(5, 8, figsize=(10, 10))
for linha in range(5):
    for coluna in range(8):
        ax[linha,coluna].imshow(centered_train_face[count].
↪reshape(altura_imagem,comprimento), cmap='gray')
        count += 1
        ax[linha, coluna].set_xticks([])
        ax[linha, coluna].set_yticks([])
        ax[linha, coluna].set_xticklabels([])
        ax[linha, coluna].set_yticklabels([])
        ax[linha, coluna].set_xlabel('')
        ax[linha, coluna].set_ylabel('')
```

A seguir, vamos analisar a variância cumulativa das 40 componentes principais. Desse modo, podemos considerar apenas as mais relevantes a fim de simplificar a análise do dado.

[24]:
```
# O quanto cada autovalor representa do dado?

# Somando todos os autovalores
sum_all = sum(D_Face_Famous)
# Transformando o array com os autovalores numa soma cumulativa
cummulative = (np.cumsum(D_Face_Famous).round(2))*100
# Vendo quanto representa do total
percentage = cummulative/sum_all

# Plotando o gráfico
plt.bar(range(len(percentage)), percentage, color='blue', alpha=0.7)
plt.title("Gráfico de variância por quantidade de PCA's")
plt.xlabel("Quantidade de PCA's")
plt.ylabel("% de explicação do dado")
```

[24]:
```
Text(0, 0.5, '% de explicação do dado')
```



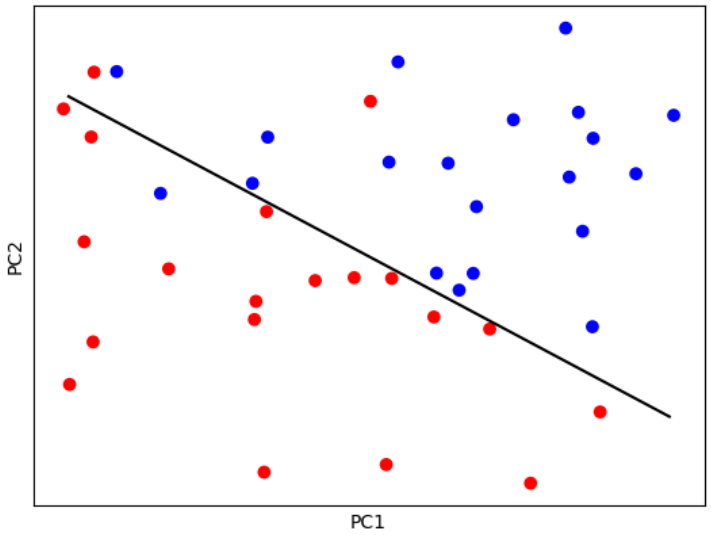
Podemos observar que as 20 primeiras componentes, ou seja, metade do conjunto, é capaz de representar aproximadamente 70% do dado. No entanto, para fins lúdicos, utilizaremos a seguir apenas as 2 primeiras a fim de ser possível plotá-las em um gráfico. Apesar de não serem tão precisas, elas já representam 15% da variação do dado, sendo suficiente para nosso propósito.

[25]:
```
# Nossa ideia agora, será pegar os dois principais componentes encontradas pelo PCA␣
↪e então plotar as nossas imagens nesse subespaço
dois_maiores_PCAs = V_Face_Famous[:2]

# Criando a lista o qual terão as coordenadas das imagens que já trainamos o nosso␣
↪dado
coordinates = []

# Fazendo um loop for para adicionar todas as imagens nessa lista
for img in centered_train_face:
    # Fazendo o produto interno para achar a projeção de cada face no espaço das␣
↪duas principais componentes.
    coordinates_xy = np.dot(dois_maiores_PCAs, img)
    # Adicionando na lista
    coordinates.append(coordinates_xy)
```

Portanto, chequemos a distribuição do dado a partir dessas duas componentes principais. No gráfico abaixo, os pontos vermelhos representam as imagens da Taylor Swift, enquanto que os azuis representam as do Will Smith.

[26]:
```
# Colocando uma lista de cores para diferenciar taylor swift e will smith
colors = []
iteracao = 1

# Adicionando cores para as imagens
for number in range(len(coordinates)):
    # Se for taylor swift vermelho
    if iteracao <= 20:
        colors.append("red")
        iteracao += 1
    # Se for will azul
    elif iteracao <= 40:
```

```
        colors.append("blue")
        iteracao += 1

# Transformando as coordenadas em um array e transpondo
coordinates = np.array(coordinates).T

# Adicionando a linha de separação entre taylor e will
x_line = np.linspace(-8000, 8000, 100)
y_line = -0.55*x_line - 700

# Plotando o gráfico
plt.scatter(coordinates[0], coordinates[1], c=colors)
plt.plot(x_line, y_line, color = "black")
plt.xticks([])
plt.yticks([])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



Apesar da baixa acurácia fornecida apenas por 2 componentes, obtemos uma distribuição relativamente distinguível, com apenas alguns pontos em posições incorretas.

Agora, vamos fazer alguns testes de imagens de fora da base para ver onde elas serão colocadas nessa distribuição. Testaremos outras imagens da própria Taylor Swift e do próprio Will Smith, além de outras 4 celebridades.

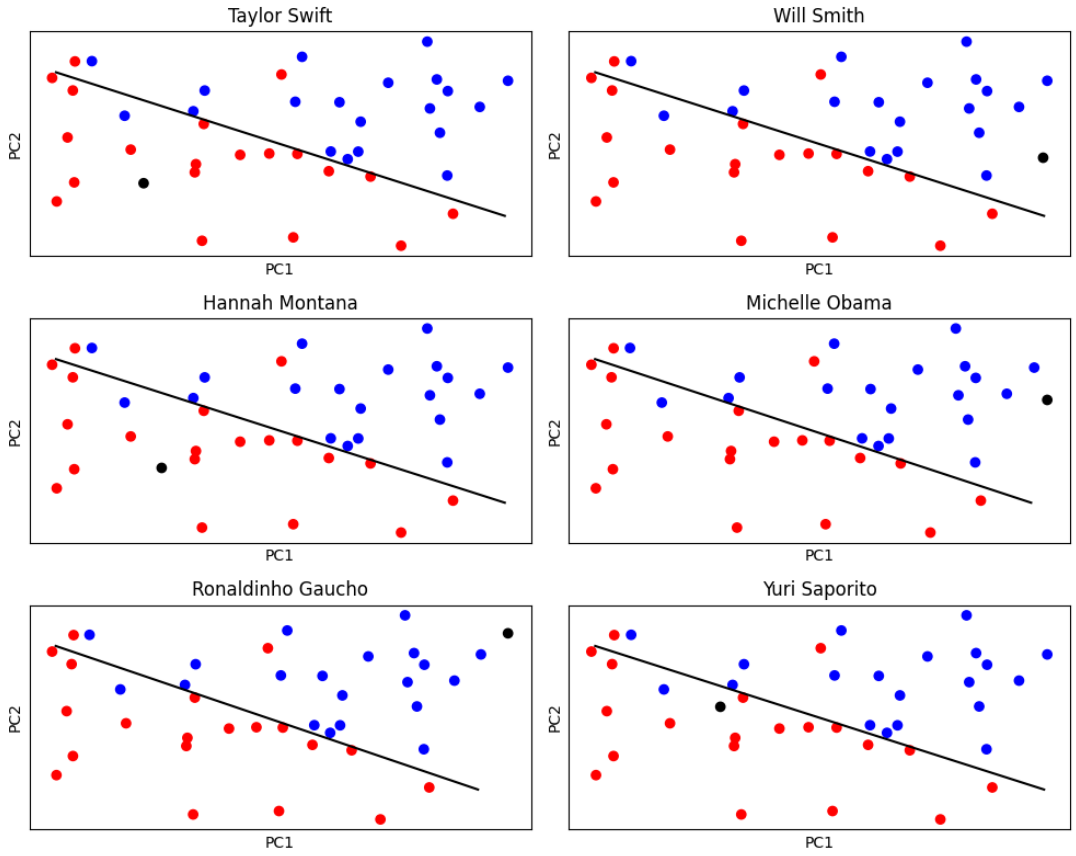
Em cada um dos gráficos abaixo, o ponto preto representa a nova imagem. Lembrando que os pontos vermelhos representam a Taylor Swift e os azuis o Will Smith.

[27]:
```
# Agora que já temos os dados dos elementos que plotamos, vamos tentar pegar␣
↪imagens que não foram utilizadas e então plotar no nosso espaço para determinar␣
↪se a pessoa é mais parecida com o will smith ou com a talor swift.

# Redefinindo as coordenadas para o próximo gráfico, pois já usamos a variável␣
↪coordinates
coordinates = []
for img in centered_train_face:
    coordinates_xy = np.dot(dois_maiores_PCAs, img)
```

```
    coordinates.append(coordinates_xy)

# Adicionando o path
training_set = "imagens_datateste/"

# criando um dicionário que terá o nome das pessoas e as coordenadas
coordinates_new_dict = {}

# Iterando sobre cada imagem da pasta
for image_name in os.listdir(training_set):
    # Pegando o caminho até ela
    image_path = os.path.join(training_set, image_name)

    # Realizando procedimentos pra ler a imagem
    img = cv2.imread(image_path)
    # Transformando num cinza
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Transformando em array
    img = np.array(img)
    # transformando num array unidimensional
    img = img.flatten()
    # retirando a imagem nova
    img = img - mean_train_face

    # Fazendo o produto interno para achar a projeção no espaço que queremos
    coordinates_new = np.dot(dois_maiores_PCAs, img)

    # Colocando as coordenadas num dicionário
    coordinates_new_dict[str(image_name[0:-4])] = coordinates_new


# Colocando uma lista de cores para diferenciar taylor swift e will smith
colors = []
iteracao = 1

for number in range(len(coordinates)):
    if iteracao <= 20:
        colors.append("red")
        iteracao += 1
    elif iteracao <= 40:
        colors.append("blue")
        iteracao += 1

# Definindo a cor da imagem que será adicionada
colors.append("black")
```

[28]:
```
# Criando um grid plot com as pessoas
fig, axes = plt.subplots(3, 2, figsize=(10, 8))
axes = axes.flatten()

# Criando um loop para iterar sobre cada plot
for i, (name, values) in enumerate(coordinates_new_dict.items()):
    ax = axes[i]

    # criando uma cópia das coordenadas
    temp_coordinates = coordinates.copy()
```

```
    # colocando as coordenadas que estão no dicionário na lista
    temp_coordinates.append(values)

    # Transformando em array e transpondo a matriz
    temp_coordinates = np.array(temp_coordinates).T

    # Fazendo o scatter plot
    ax.scatter(temp_coordinates[0], temp_coordinates[1], c=colors)

    # Adicionando a linha de separação
    x_line = np.linspace(-8000, 8000, 100)
    y_line = -0.55*x_line - 700
    ax.plot(x_line, y_line, color = "black")

    # Definindo o nome dos gráficos e formantando-o
    ax.set_title(str(name)[1:])
    axes[i].set_xticks([])
    axes[i].set_yticks([])
    axes[i].set_xticklabels([])
    axes[i].set_yticklabels([])
    axes[i].set_xlabel('PC1')
    axes[i].set_ylabel('PC2')

# Mostrando o gráfico
plt.tight_layout()
plt.show()
```



Discutindo os resultados da aplicação de eigenfaces assim que, apesar de as componentes principais

escolhidas representarem apenas 15% do dado, as imagens selecionadas conseguiram um bom grau de diferenciação no espaço considerado. A imagem do Will Smith que não pertencia à base caiu no lado do Will Smith, e a imagem externa da Taylor também caiu no lado correto. Podemos ver que as celebridades que possuem um tom de pele mais escuro tenderam a cair no lado do Will Smith, enquanto que as celebridades com um tom de pele mais claro tenderam a cair no o lado da Taylor Swift.

Uma hipótese para essa observação pode ser formulada a partir do processo de obtenção das eigenfaces. O procedimento todo se baseia na transformação de pixels em números, dependendo muito das cores dos pixels para a criação dos autoespaços.

4 Conclusão

Portanto, concluímos que o processo de obtenção das eigenfaces é um processo viável e de relativa simplicidade para o processamento de imagens, no qual basta transformarmos as imagens em números em uma matriz com base nas colorações de seus pixels e, com isso, efetuar o processo do PCA, encontrar os autovetores da matriz de covariância, obter os componentes principais e comparar as coordenadas das imagens de interesse nesse novo subespaço.

Os resultados obtidos, mesmo com uma baixa porcentagem de variância das informações fornecida apenas pelos 2 primeiros componentes principais, podem ser suficientes para distinguir algumas imagens e para classificar indivíduos entre Taylor Swift e Will Smith, as faces selecionadas como base para o nosso dado.

Por fim, compreende-se o poder da Álgebra Linear em problemas do cotidiano, podendo agir em questões de reconhecimento facial e de análise de imagens.

5 Bibliografia

[1] TURK, Matthew; PENTLAND, Alex. Eigenfaces for recognition. Journal of cognitive neuroscience, v. 3, n. 1, p. 71-86, 1991.

[2] Hwang, Sung-Wook; Sugiyama, Junji. (2021). Computer vision-based wood identification and its expansion and contribution potentials in wood science: A review. Plant Methods. 17. 10.1186/s13007-021-00746-1.

[3] PESSOA, Wiliam. Reconhecimento de padrões — Eigenfaces. Medium, 11 de jun. de 2019. Disponível em: <https://medium.com/@williangpessoa/reconhecimento-de-padr%C3%B5es-eigenfaces-e4cef8f04919>. Acesso em: 1 de novembro de 2023.

[4] STRANG, Gilbert. Introduction to linear algebra. Wellesley-Cambridge Press, 2022.