Trabalho #1: Álgebra Linear MO431A

Universidade Estadual de Campinas (UNICAMP), Instituto de Computação (IC)

Prof. Jacques Wainer, 2021s1

```
print('225293: ' + 'Elias Batista Ferreira')
print('225242: ' + 'Maurício Pereira Lopes')
print('211518: ' + 'Stephane de Freitas Schwarz')
```

```
    225293: Elias Batista Ferreira
    225242: Maurício Pereira Lopes
    211518: Stephane de Freitas Schwarz
```

```
import numpy as np
import sklearn
import skimage
import scipy
import requests
import io
import matplotlib.pyplot as plt
from sklearn.decomposition import TruncatedSVD, PCA
```

1 Leia o arquivo e carregue a matriz X

Carregando o arquivo via requests.

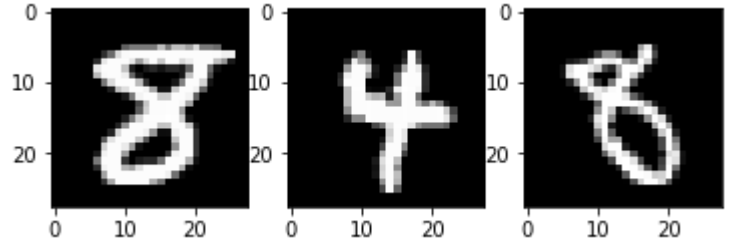
```
request_response = requests.get('https://www.ic.unicamp.br/~wainer/cursos/1s2021/431/dados.npy')
request_response.raise_for_status()
X = np.load(io.BytesIO(request_response.content))
```

2 Imprimindo imagens da matriz X

Para imprimir as matrizes, inicialmente, as imagens devem ser transformadas em vetores bidimensionais.

```
def show_digits(digits_dataset, qty):

  plt.figure()
  fig, ax = plt.subplots(len(digits_dataset), qty)

  for idx, matrix in enumerate(digits_dataset):

    for i in range(qty):

      img = np.reshape(matrix[i], (28, 28))
      if len(digits_dataset) > 1:
        ax[idx, i].imshow(img, cmap='gray')
      else:
        ax[i].imshow(img, cmap='gray')

  plt.show()


show_digits([X], 3)
```

```
<Figure size 432x288 with 0 Axes>
```



.

3 Normalizar e fatorar a matriz

A fatorização $A\Sigma V^H$ é chamada de SVD completo. A forma compacta do SVD retém apenas as r colunas de U e V, associadas com valores singulares não zero. A formulação escrita como: $A = U_r \Sigma_r V_r$ [1].

```
X_norm = X - np.mean(X, axis=0)

u_full, s_full, vh_full = np.linalg.svd(X_norm, full_matrices=True)
print('SVD Full matriz \nU -> {} | D -> {} | V -> {}'.format(u_full.shape, s_full.shape, vh_full.shape))

u_comp, s_comp, vh_comp = np.linalg.svd(X_norm, full_matrices=False)
print('\n\nSVD compacta \nU -> {} | D -> {} | V -> {}'.format(u_comp.shape, s_comp.shape, vh_comp.shape))
```

```
SVD Full matriz
U -> (10500, 10500) | D -> (784,) | V -> (784, 784)


SVD compacta
U -> (10500, 784) | D -> (784,) | V -> (784, 784)
```

4 Redução de dimensionalidade

```
n_components=100

t_svd = TruncatedSVD(n_components=n_components)
matriz_svd_100d = t_svd.fit_transform(X)
print('Dimensão da matriz com SVD truncado: {}'.format(matriz_svd_100d.shape))
```

```
Dimensão da matriz com SVD truncado: (10500, 100)
```

Aqui, reconstruímos a matriz utilizando a função inverse_transform do SVD.

```
X_reconstruida = t_svd.inverse_transform(matriz_svd_100d)
print('Tamanho da matriz reconstruída com SVD: {}'.format(X_reconstruida.shape))
```

```
    Tamanho da matriz reconstruída com SVD: (10500, 784)
```

Também realizamos a reconstrução manualmente. Como é possível notar através do código, para isso consideramos apenas n componentes. Onde n refere-se ao número requerido no exercício anterior (no caso, 100).

Observem que as dimensões da matriz reconstruída é exatamente igual a matriz original.

```
reconstructed_img = np.matrix(u_full[:, :n_components]) * \
                    np.diag(s_full[:n_components]) * \
                    np.matrix(vh_full[:n_components, :])

print('Tamanho da matriz reconstruída na mão: {}'.format(reconstructed_img.shape))
```

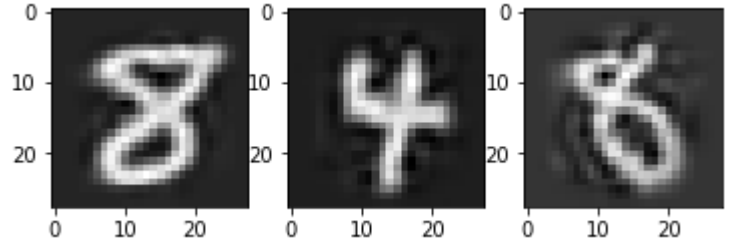
```
    Tamanho da matriz reconstruída na mão: (10500, 784)
```

5 Imprimir a matriz reconstruída

Na figura abaixo mostramos as images após a reconstrução usando o método disponibilizado pela biblioteca Sklearn.

```
show_digits([X_reconstruida], 3)
```

```
<Figure size 432x288 with 0 Axes>
```



A ideia da figura abaixo é mostrar um comparativo entre a imagem original, e as duas abordagens que exploramos para a reconstrução das matrizes.

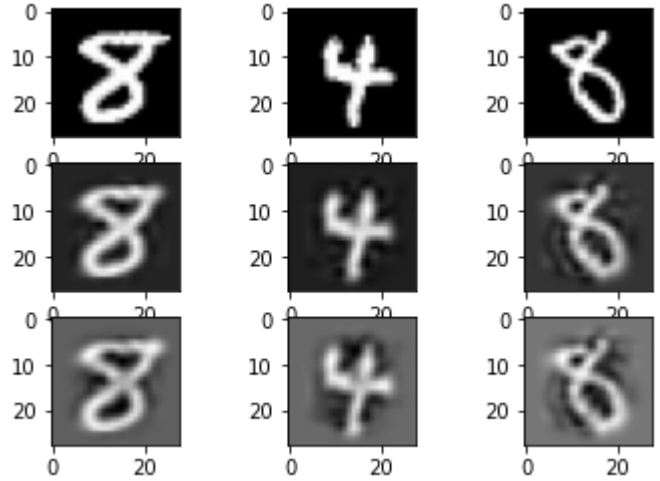
Na primeira linha, apresentamos as três primeiras figuras da base de dados original.

Na segunda linha, a reconstrução através dos vetores gerados pelo SVD truncado.

Por fim, na última linha apresentamos as três primeiras amostras da matriz reconstruída através da matriz completa, encontrada anteriormente.

```
show_digits([X, X_reconstruida, reconstructed_img], 3)
```

```
<Figure size 432x288 with 0 Axes>
```



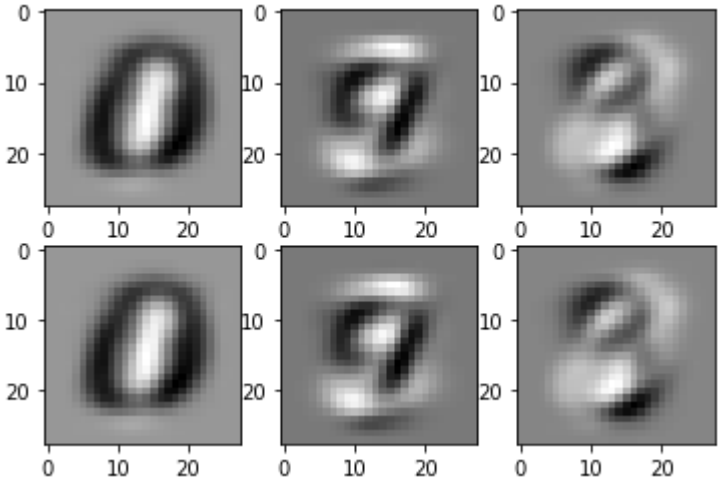
6 Imprimir os eigen-values

Na figura abaixo mostramos os eigen-dígios dos três primeiros dígitos do dataset.

Notem que esses valores salientam características discriminativas das classes das imagens.

```
show_digits([vh_full, vh_comp], 3)
```

```
<Figure size 432x288 with 0 Axes>
```



7 Decidindo número de dimensões

Uma das formas mais simples para encontrar a quantidade de dimensões ótimas para a redução de dimensionalidade é observando a quantidade de valores superiores a 1 a matriz D.

```
print('Usando a regra do 1, são necessários {} componentes.'.format(s_full[s_comp>1].shape[0]))
```

```
Usando a regra do 1, são necessários 671 componentes.
```

```
t_svd = TruncatedSVD(n_components=X.shape[1]-1)
X_tsvd = t_svd.fit(X)


def get_nComponents_variance(variance_array, variance_goal):

  n_components = 0
  acc_variance = 0

  for variance in variance_array:

    acc_variance += variance
    n_components += 1

    if acc_variance >= variance_goal:
      break

  return acc_variance, n_components
```

É possível encontrar a quantidade de componentes através da especificação da variância dos dados. Para isso, basta computar a variância média da base de dados e fazer a soma acumulada dos componentes até alcançar o limear previamente definido.

Realizamos três experimentos para confirmar a quantidade de componentes necessários.

No primeiro, calculamos manualmente. No segundo utilizamos o PCA para confirmar nosso achado inicial. E, finalmente, computamos a quantidade de componentes atrvés da variância média expandida do SVD.

```
X_trans = t_svd.transform(X)
vari = np.var(X_trans, axis=0)
evr = vari/np.var(X, axis=0).sum()

v, c = get_nComponents_variance(evr, 0.8)
print('CALCULANDO NA MÃO -> São necessários {} componentes para obter {:.2f}% de variância.'.format(c, v*100))
```

```
CALCULANDO NA MÃO -> São necessários 43 componentes para obter 80.07% de variância.
```

```
variance = 0.8
pca = PCA(variance)
x_pca = pca.fit_transform(X)

print('USANDO PCA -> São necessários {} componentes para obter {:.2f}% de variância.'.format(x_pca.shape[1], varia
```

```
    USANDO PCA -> São necessários 43 componentes para obter 80.00% de variância.
```

```
v, c = get_nComponents_variance(t_svd.explained_variance_ratio_, 0.8)
print('USANDO SVD -> São necessários {} componentes para obter {:.2f}% de variância.'.format(c, v*100))
```

```
    USANDO SVD -> São necessários 43 componentes para obter 80.07% de variância.
```

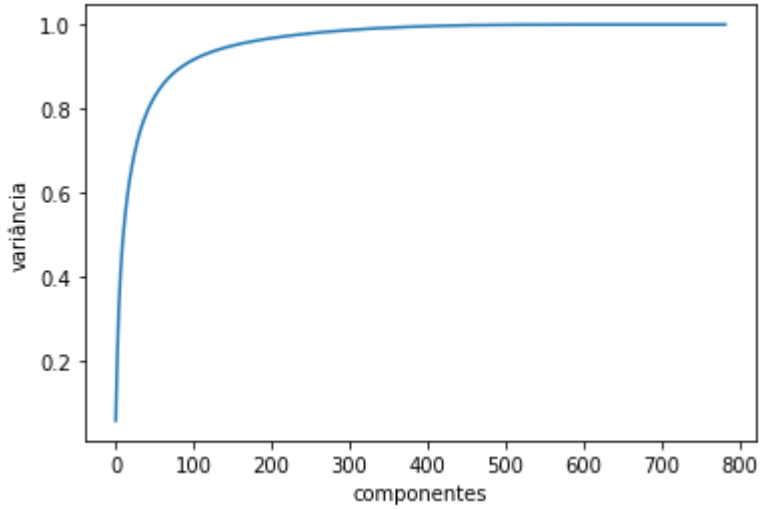
```
v, c = get_nComponents_variance(t_svd.explained_variance_ratio_, 0.95)
print('São necessários {} componentes para obter {:.2f}% de variância.'.format(c, v*100))
```

```
    São necessários 151 componentes para obter 95.00% de variância.
```

No gráfico abaixo mostramos como o número de componentes varia de acordo com a variância dos dados.

```
plt.plot(np.cumsum(t_svd.explained_variance_ratio_))
plt.xlabel('componentes')
plt.ylabel('variância')
```

```
Text(0, 0.5, 'variância')
```



A seguir demostramos uma outra abordagem para o cálculo da quantidade de componentes por variância.

```
# por essa regra o número de dimensões será aquele que somar 80% da variância
# total acumulada nos single values da matriz D

# variância total
var_total = s_comp.sum()
```

```
# fazer a soma das variâncias linha a linha até que atinja 80% da total
for i in range(s_comp.shape[0]):
    var_trunc = s_comp[:i].sum() / var_total
    if var_trunc >= 0.8:
        break

print("Variância acumulada com", i, "dimensões:", var_trunc)
```

```
Variância acumulada com 235 dimensões: 0.8004557116328009
```

```
# por essa regra o número de dimensões será aquele que somar 95% da variância
# total acumulada nos single values da matriz D

# variância total
var_total = s_comp.sum()

# fazer a soma das variâncias linha a linha até que atinja 95% da total
for i in range(s_comp.shape[0]):
    var_trunc = s_comp[:i].sum() / var_total
    if var_trunc >= 0.95:
        break
print("Variância acumulada com", i, "dimensões:", var_trunc)
```

```
Variância acumulada com 426 dimensões: 0.9502576388383304
```