Tarefa 1

April 11, 2021

1 Tarefa 1

Alunos: Andreza(164213), Gil(225323) e Yan(118982)

1.1 Importando as bibliotecas

[1]:
```
import time
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
plt.rcParams['figure.figsize'] = [12, 8]
```

1.2 Ler arquivo dados.npy

Arquivo de 10500 linhas e 784 colunas, mas cada linha é uma matriz 28x28

[2]:
```
X = np.load("dados.npy")
X.shape
```

[2]:
```
(10500, 784)
```

Se formos traduzir esta matriz, diríamos que ela apresenta em suas colunas o total de dimensões de cada imagem, e um total de 10500 imagens.

1.3 Imprima a imagem dos 3 primeiros dígitos

Passo 1: cada linha da matriz precisa ser transformada em outra matriz 28x28;

Passo 2: codificação das cores para cinza;

Passo 3: imprimir as 3 primeiras imagens;

[3]:
```
fig=plt.figure(figsize=(8, 8))
for i in range(0, 3):
    img = np.reshape(X[i], (28,28))
    fig.add_subplot(1, 3, i+1)
    img=plt.imshow(img)
    plt.axis('off')
    img.set_cmap('gray')
plt.show()
```

1.4 Faça a fatoração SVD da matrix

3.1: normalizar os dados para média O e desvio-padrão diferente de 1;

3.2: fazer fatoração full_matrix e compacta;

3.3: verificar o tamanho das matrizes;

1.4.1 Normalização

[4]:
```
scaler = StandardScaler(with_std=False)
Xmz=scaler.fit_transform(X)
Xmz.shape
```

[4]:
```
(10500, 784)
```

1.4.2 SVD Matriz Full

[5]:
```
# Fatoração SVD full matrix
start = time.time()
U, S, VT = np.linalg.svd(Xmz, full_matrices=True)
S = np.diag(S)
time_elapsed = time.time() - start
print("Tempo de execução SVD full matrix: {:.3f} seconds".format(time_elapsed))
print(U.shape, S.shape, VT.shape)
```

```
Tempo de execução SVD full matrix: 166.914 seconds
(10500, 10500) (784, 784) (784, 784)
```

1.4.3 SVD Matriz Compacta

[6]:
```
# Fatoração SVD compacta
start = time.time()
U, S, VT = np.linalg.svd(Xmz, full_matrices=False)
S = np.diag(S)
```

```
time_elapsed = time.time() - start
print("Tempo de execução SVD compacto: {:.3f} seconds".format(time_elapsed))
print(U.shape, S.shape, VT.shape)
```

```
Tempo de execução SVD compacto: 18.040 seconds
(10500, 784) (784, 784) (784, 784)
```

1.5 SVD truncado

Usar a redução para 100 dimensões;

4.1 Computar a matriz projetada. Será (10500, 100). Imprimir as dimensões;

4.2 Computar a matriz reconstruída.Será (10500,784). Imprimir as dimensões;

1.5.1 Matriz Projetada

Dado que os dados projetados são representados pela multiplicação das matrizes Ur e Drxr, então selecionamos as r colunas da matriz U e as r linhas e colunas da matriz Sigma

[7]:
```
r=100
Aproj = U[:,:r] @ S[:r,:r]
Aproj.shape
```

[7]:
```
(10500, 100)
```

1.5.2 Matriz Reconstruída

Já a matriz reconstruída será obtida multiplicando-se a matriz projetada pelas primeiras r linhas da matriz VT da seguinte forma

[8]:
```
Arec = U[:,:r] @ S[:r,:r] @ VT[:r,]
Arec.shape
```

[8]:
```
(10500, 784)
```

1.6 Imprima a imagem reconstruída dos 3 primeiros dígitos

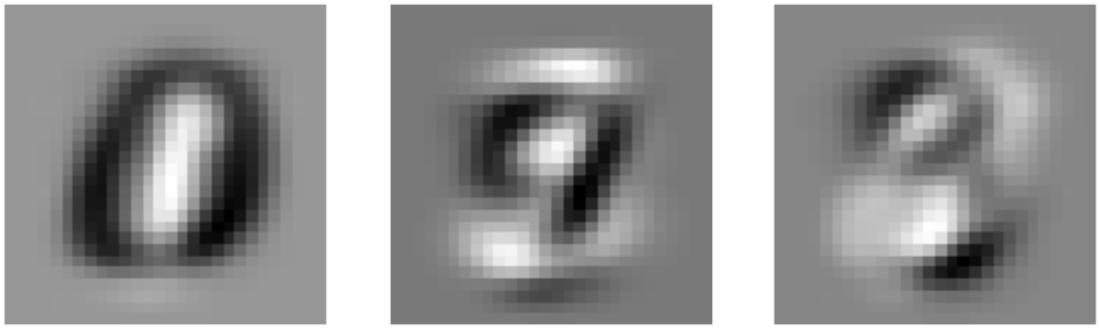
Compare com as imagens impressas acima

[63]:
```
fig=plt.figure(figsize=(8, 8))
for i in range(0,3):
    A = Arec[i].reshape((28,28))
    fig.add_subplot(1, 3, i+1)
    img = plt.imshow(A,cmap='gray')
    plt.axis('off')
plt.show()
```

- Ao comparar com as imagens impressas anteriormente, podemos perceber que informações foram perdidas, reduzindo a nitidez dos dígitos e alterando a cor do fundo, porém ainda conseguimos visualizar e identificar os dígitos de forma correta.

1.7 Imprima os 3 primeiros eigen-dígitos

[64]:
```
fig=plt.figure(figsize=(8, 8))
for i in range(0,3):
    A = VT[i].reshape((28,28))
    fig.add_subplot(1, 3, i+1)
    img = plt.imshow(A,cmap='gray')
    plt.axis('off')
plt.show()
```



1.8 Decidindo o número de dimensões

7.1 Quantas dimensões usando a regra de singular values maior que 1?

7.2 Quantas dimensões manter para capturar 80% da variância dos dados?

7.3 Quantas dimensões manter para capturar 95% da variância dos dados?

1.8.1 Para Singular Values > 1

[54]:
```
print('Necessitamos {} dimensões'.format(S[S>1].shape[0]))
```

```
Necessitamos 671 dimensões
```

1.8.2 Para 80% da variância

[55]:
```
i=0
while np.cumsum(np.diag(S)/np.sum(np.diag(S)))[i]<0.8:
    i=i+1
print('Necessitamos {} dimensões'.format(i))
```

```
Necessitamos 234 dimensões
```

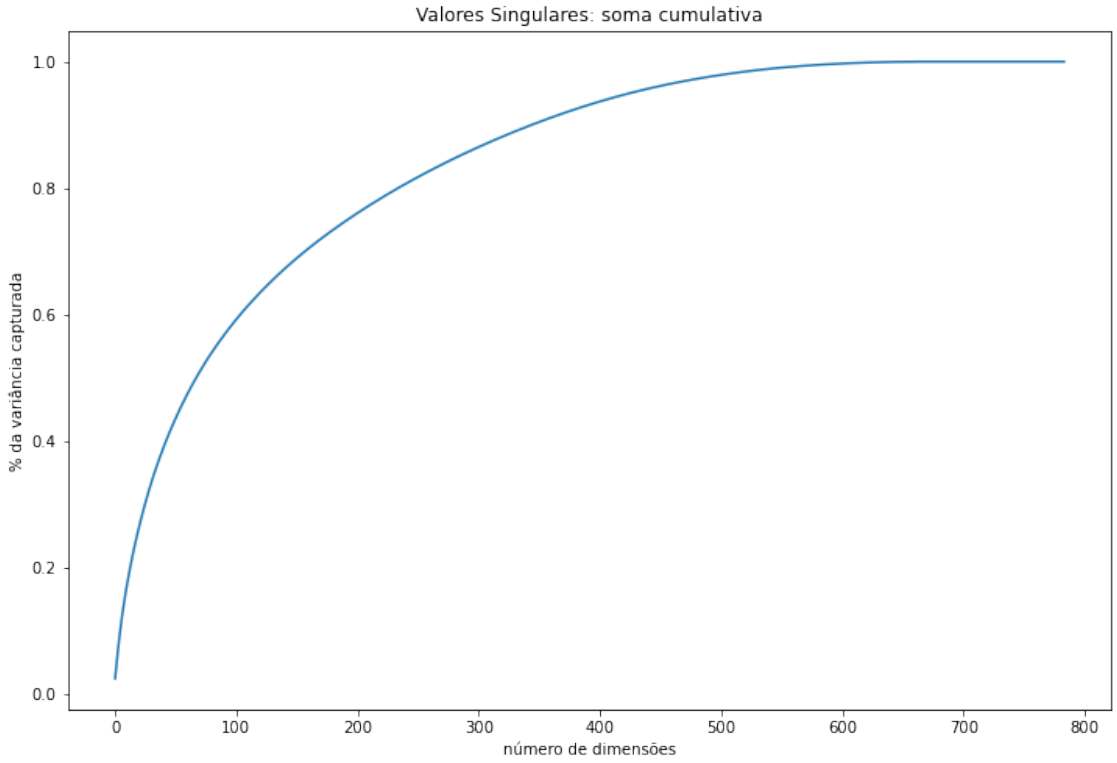
1.8.3 Para 95% da variância

[56]:
```
i=0
while np.cumsum(np.diag(S)/np.sum(np.diag(S)))[i]<0.95:
    i=i+1
print('Necessitamos {} dimensões \n'.format(i))
```

```
Necessitamos 425 dimensões
```

1.9 Extra: Gráfico das dimensões necessárias para as % de variâncias capturadas

[47]:
```
plt.figure(1)
plt.plot(np.cumsum(np.diag(S)/np.sum(np.diag(S))))
plt.title('Valores Singulares: soma cumulativa')
plt.xlabel('número de dimensões')
plt.ylabel('% da variância capturada')
plt.show()
```


Valores Singulares: soma cumulativa
% da variância capturada
número de dimensões
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500
600
700
800