W04_LabAssesmentGraded [CODES]

December 30, 2024

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse.linalg
```

[2]:
```python
import utils
import w4_unittest
```

[3]:
```python
P = np.array([

    [0, 0.75, 0.35, 0.25, 0.85],
    [0.15, 0, 0.35, 0.25, 0.05],
    [0.15, 0.15, 0, 0.25, 0.05],
    [0.15, 0.05, 0.05, 0, 0.05],
    [0.55, 0.05, 0.25, 0.25, 0]
])

X0 = np.array([[0],[0],[0],[1],[0]])

### START CODE HERE ###

# Multiply matrix P and X_0 (matrix multiplication).
X1 = P @ X0

### END CODE HERE ###

print(f'Sum of columns of P: {sum(P)}')
print(f'X1:\n{X1}')
```

```
Sum of columns of P: [1. 1. 1. 1. 1.]
X1:
[[0.25]
 [0.25]
 [0.25]
 [0.  ]
 [0.25]]
```

[4]:
```
# Test your solution.
w4_unittest.test_matrix(P, X0, X1)
```

```
All tests passed
```

[5]:
```
X = np.array([[0],[0],[0],[1],[0]])
m = 20

for t in range(m):
    X = P @ X

print(X)
```

```
[[0.39392366]
 [0.13392366]
 [0.11407667]
 [0.0850993 ]
 [0.27297672]]
```

[6]:
```
eigenvals, eigenvecs = np.linalg.eig(P)
print(f'Eigenvalues of P:\n{eigenvals}\n\nEigenvectors of P\n{eigenvecs}')
```

```
Eigenvalues of P:
[ 1.         -0.70367062  0.00539505 -0.08267227 -0.21905217]

Eigenvectors of P
[[-0.76088562 -0.81362074  0.10935376  0.14270615 -0.39408574]
 [-0.25879453  0.050269   -0.6653158   0.67528802 -0.66465044]
 [-0.2204546   0.07869601 -0.29090665  0.17007443  0.35048734]
 [-0.1644783   0.12446953  0.19740707 -0.43678067  0.23311487]
 [-0.52766004  0.56018621  0.64946163 -0.55128793  0.47513398]]
```

[7]:
```
X_inf = eigenvecs[:,0]

print(f"Eigenvector corresponding to the eigenvalue 1:\n{X_inf[:,np.newaxis]}")
```

```
Eigenvector corresponding to the eigenvalue 1:
[[-0.76088562]
 [-0.25879453]
 [-0.2204546 ]
 [-0.1644783 ]
 [-0.52766004]]
```

[9]:
```
# This is organised as a function only for grading purposes.
def check_eigenvector(P, X_inf):
    ### START CODE HERE ###
    X_check = P @ X_inf
    ### END CODE HERE ###
```

```
    return X_check

X_check = check_eigenvector(P, X_inf)
print("Original eigenvector corresponding to the eigenvalue 1:\n" + str(X_inf))
print("Result of multiplication:" + str(X_check))

# Function np.isclose compares two NumPy arrays element by element, allowing␣
↪for error tolerance (rtol parameter).
print("Check that PX=X element by element:" + str(np.isclose(X_inf, X_check,␣
↪rtol=1e-10)))
```

```
Original eigenvector corresponding to the eigenvalue 1:
[-0.76088562 -0.25879453 -0.2204546  -0.1644783  -0.52766004]
Result of multiplication:[-0.76088562 -0.25879453 -0.2204546  -0.1644783
-0.52766004]
Check that PX=X element by element:[ True  True  True  True  True]
```

[10]:
```
# Test your solution.
w4_unittest.test_check_eigenvector(check_eigenvector)
```

```
 All tests passed
```

[11]:
```
X_inf = X_inf/sum(X_inf)
print(f"Long-run probabilities of being at each webpage:\n{X_inf[:,np.
↪newaxis]}")
```

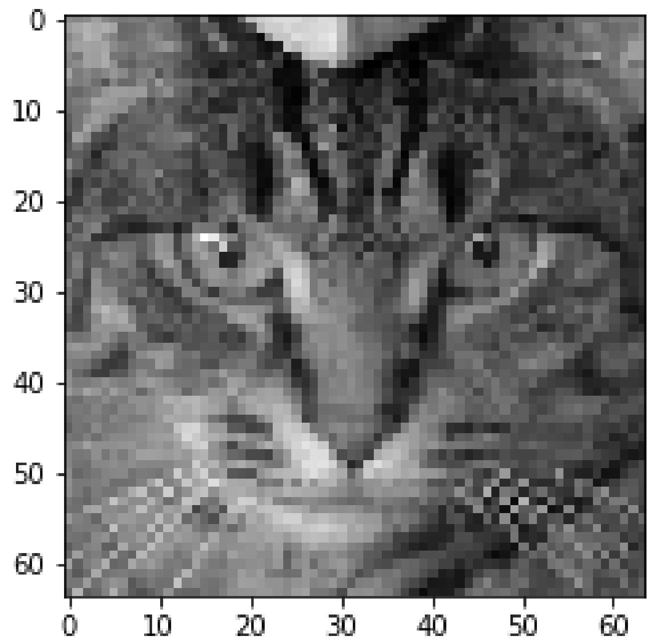
```
Long-run probabilities of being at each webpage:
[[0.39377747]
 [0.13393269]
 [0.11409081]
 [0.08512166]
 [0.27307736]]
```

[12]:
```
imgs = utils.load_images('./data/')
height, width = imgs[0].shape

print(f'\nYour dataset has {len(imgs)} images of size {height}x{width}␣
↪pixels\n')
plt.imshow(imgs[0], cmap='gray')
```

```
Your dataset has 55 images of size 64x64 pixels
```

[12]: <matplotlib.image.AxesImage at 0x7f3aec29aa60>



[13]:
```python
imgs_flatten = np.array([im.reshape(-1) for im in imgs])

print(f'imgs_flatten shape: {imgs_flatten.shape}')
```

```
imgs_flatten shape: (55, 4096)
```

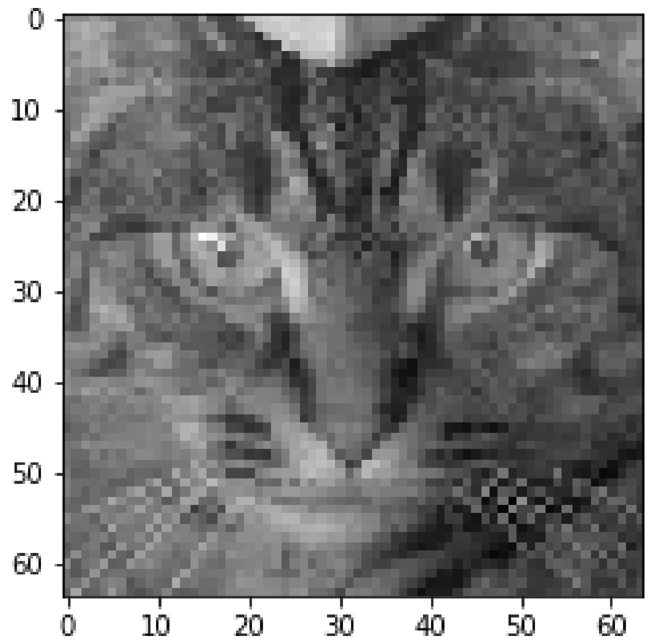
[14]:
```python
# Graded cell
def center_data(Y):
    """
    Center your original data
    Args:
          Y (ndarray): input data. Shape (n_observations x n_pixels)
    Outputs:
        X (ndarray): centered data
    """
    ### START CODE HERE ###
    mean_vector = np.mean(Y, axis=0)

    # use np.reshape to reshape into a matrix with the same size as Y. Remember␣
 ↪to use order='F'
    mean_matrix = np.reshape(mean_vector, (1, -1), order='F')

    X = Y - mean_matrix
    ### END CODE HERE ###
    return X
```

[15]:
```python
X = center_data(imgs_flatten)
plt.imshow(X[0].reshape(64,64), cmap='gray')
```

[15]: <matplotlib.image.AxesImage at 0x7f3aec2472b0>



[16]:
```python
# Test your solution.
w4_unittest.test_center_data(center_data)
```

```
All tests passed
```

[17]:
```python
def get_cov_matrix(X):
    """ Calculate covariance matrix from centered data X
    Args:
        X (np.ndarray): centered data matrix
    Outputs:
        cov_matrix (np.ndarray): covariance matrix
    """

    ### START CODE HERE ###
    cov_matrix = np.dot(X.T, X) / (X.shape[0] - 1)
    ### END CODE HERE ###

    return cov_matrix
```

[18]:
```python
cov_matrix = get_cov_matrix(X)
```

[19]:
```python
print(f'Covariance matrix shape: {cov_matrix.shape}')
```

```
Covariance matrix shape: (4096, 4096)
```

[20]:
```python
# Test your solution.
w4_unittest.test_cov_matrix(get_cov_matrix)
```

```
 All tests passed
```

[21]:
```python
scipy.random.seed(7)
eigenvals, eigenvecs = scipy.sparse.linalg.eigsh(cov_matrix, k=55)
print(f'Ten largest eigenvalues: \n{eigenvals[-10:]}')
```

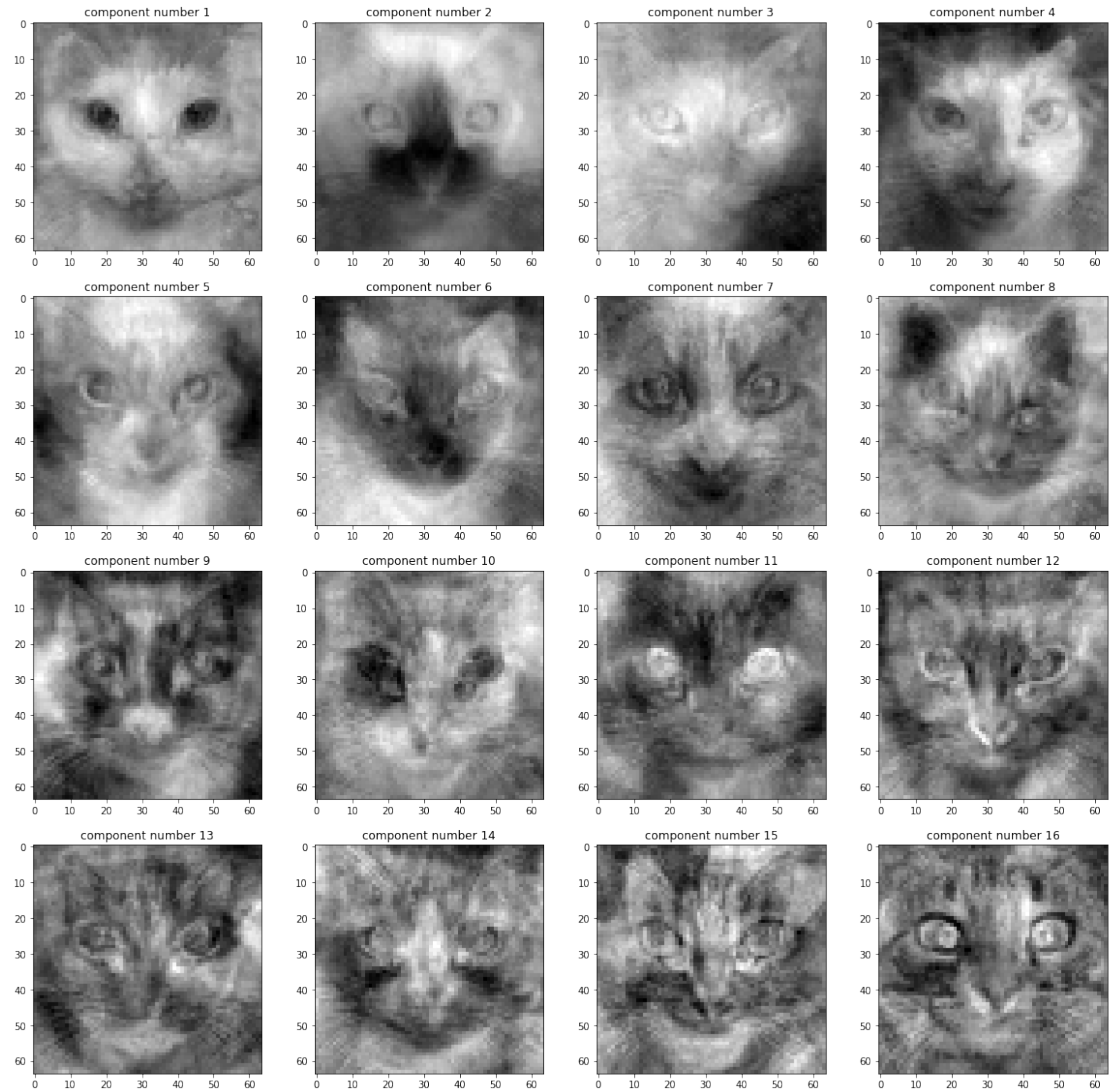
```
Ten largest eigenvalues:
[ 293297.76716381  383558.95285037  399091.64921256  479564.23517501
  839756.42124326  879138.93723794 1011092.7845815  1536790.5408648
 2484055.10309963 4198829.23262023]
```

[22]:
```python
eigenvals = eigenvals[::-1]
eigenvecs = eigenvecs[:,::-1]

print(f'Ten largest eigenvalues: \n{eigenvals[:10]}')
```

```
Ten largest eigenvalues:
[4198829.23262023 2484055.10309963 1536790.5408648  1011092.7845815
  879138.93723794  839756.42124326  479564.23517501  399091.64921256
  383558.95285037  293297.76716381]
```

[23]:
```python
fig, ax = plt.subplots(4,4, figsize=(20,20))
for n in range(4):
    for k in range(4):
        ax[n,k].imshow(eigenvecs[:,n*4+k].reshape(height,width), cmap='gray')
        ax[n,k].set_title(f'component number {n*4+k+1}')
```



[24]:
```python
# GRADED cell
def perform_PCA(X, eigenvecs, k):
    """
    Perform dimensionality reduction with PCA
    Inputs:
        X (ndarray): original data matrix. Has dimensions␣
↪(n_observations)x(n_variables)
        eigenvecs (ndarray): matrix of eigenvectors. Each column is one␣
↪eigenvector. The k-th eigenvector
                            is associated to the k-th eigenvalue
        k (int): number of principal components to use
    Returns:
        Xred
```

```
    """

    ### START CODE HERE ###
    V = eigenvecs[:, :k]
    Xred = X @ V
    ### END CODE HERE ###
    return Xred
```

[25]:
```
Xred2 = perform_PCA(X, eigenvecs,2)
print(f'Xred2 shape: {Xred2.shape}')
```

```
Xred2 shape: (55, 2)
```

[26]:
```
# Test your solution.
w4_unittest.test_check_PCA(perform_PCA)
```

```
 All tests passed
```

[ ]:
```

```