exercise_11_notebook

January 14, 2020

1 Programming task 11: Dimensionality Reduction

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA
%matplotlib inline
```

1.1 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.2 PCA

Given the data in the matrix X your tasks is to: * Calculate the covariance matrix Σ. * Calculate eigenvalues and eigenvectors of Σ. * Plot the original data X and the eigenvectors to a single diagram. What do you observe? Which eigenvector corresponds to the smallest eigenvalue? * Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace. * Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

1.2.1 The given data X

[2]:
```
X = np.array([(-3,-2),(-2,-1),(-1,0),(0,1),
              (1,2),(2,3),(-2,-2),(-1,-1),
              (0,0),(1,1),(2,2), (-2,-3),
              (-1,-2),(0,-1),(1,0), (2,1),(3,2)])
```

1.2.2 Task 1: Calculate the covariance matrix Σ

[3]:
```
def get_covariance(X):
    """Calculates the covariance matrix of the input data.

    Parameters
    ----------
    X : array, shape [N, D]
        Data matrix.

    Returns
    -------
    Sigma : array, shape [D, D]
        Covariance matrix

    """
    # TODO
    N = X.shape[0]
    mu = X.mean(0)        #shape (D,)
    mu_2d = mu[:,None]    #shape (D,1)
    mu_matrix = np.dot(mu_2d, mu_2d.T)  #shape (D,D)
    cov = (1/N)*np.dot(X.T,X)-mu_matrix  #shape (D,D)
    return cov
```

1.2.3 Task 2: Calculate eigenvalues and eigenvectors of Σ.

[4]:
```
def get_eigen(S):
    """Calculates the eigenvalues and eigenvectors of the input matrix.

    Parameters
    ----------
    S : array, shape [D, D]
        Square symmetric positive definite matrix.

    Returns
    -------
    L : array, shape [D]
        Eigenvalues of S
    U : array, shape [D, D]
        Eigenvectors of S

    """
    # TODO
    L, U = LA.eig(S)
    return L, U
```

1.2.4 Task 3: Plot the original data X and the eigenvectors to a single diagram.

Note that, in general if u_i is an eigenvector of the matrix M with eigenvalue λ_i then $\alpha \cdot u_i$ is also an eigenvector of M with the same eigenvalue λ_i, where α is an arbitrary scalar (including $\alpha = -1$).

Thus, the signs of the eigenvectors are arbitrary, and you can flip them without changing the meaning of the result. Only their direction matters. The particular result depends on the algorithm used to find them.

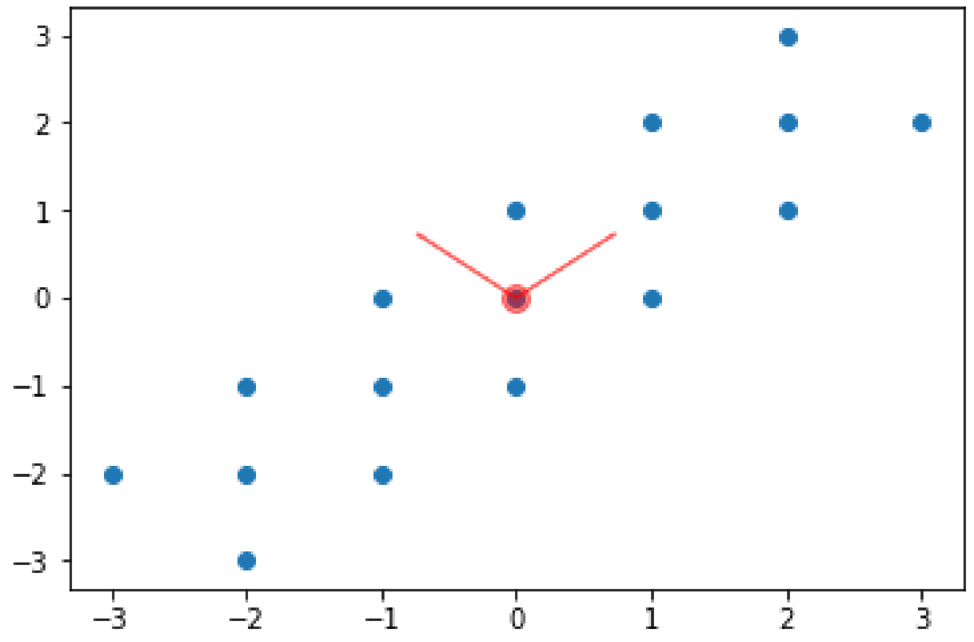
```
[5]: # plot the original data
plt.scatter(X[:, 0], X[:, 1])

# plot the mean of the data
mean_d1, mean_d2 = X.mean(0)
plt.plot(mean_d1, mean_d2, 'o', markersize=10, color='red', alpha=0.5)

# calculate the covariance matrix
Sigma = get_covariance(X)

# calculate the eigenvector and eigenvalues of Sigma
L, U = get_eigen(Sigma)

plt.arrow(mean_d1, mean_d2, U[0, 0], U[1, 0], width=0.01, color='red', alpha=0.
 ↪5)
plt.arrow(mean_d1, mean_d2, U[0, 1], U[1, 1], width=0.01, color='red', alpha=0.
 ↪5);
```



What do you observe in the above plot? Which eigenvector corresponds to the smallest eigenvalue?

Write your answer here:

Only one eigen vector [[0.70710678],[0.70710678]] is sufficient to describe the data

Eigen vector [[-0.70710678],[0.70710678]] corresponds to smallest eigen value

1.2.5 Task 4: Transform the data

Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace. Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

[6]:
```
def transform(X, U, L):
    """Transforms the data in the new subspace spanned by the eigenvector␣
 ↪corresponding to the largest eigenvalue.

    Parameters
    ----------
    X : array, shape [N, D]
        Data matrix.
    L : array, shape [D]
        Eigenvalues of Sigma_X
    U : array, shape [D, D]
        Eigenvectors of Sigma_X

    Returns
    -------
    X_t : array, shape [N, 1]
        Transformed data

    """
    # TODO
    new_basis = U[:,L.argmax()]     #shape (D,)
    new_basis = new_basis[:,None]   #shape (D,1)
    X_t = np.dot(X,new_basis)
    return X_t
```

[7]:
```
X_t = transform(X, U, L)
```

[8]:
```
X_t
```

[8]:
```
array([[-3.53553391],
       [-2.12132034],
       [-0.70710678],
       [ 0.70710678],
       [ 2.12132034],
       [ 3.53553391],
       [-2.82842712],
       [-1.41421356],
       [ 0.        ],
       [ 1.41421356],
       [ 2.82842712],
       [-3.53553391],
       [-2.12132034],
       [-0.70710678],
       [ 0.70710678],
       [ 2.12132034],
       [ 3.53553391]])
```

1.3 SVD

1.3.1 Task 5: Given the matrix M find its SVD decomposition $M = U \cdot \Sigma \cdot V$ and reduce it to one dimension using the approach described in the lecture.

[9]:
```
M = np.array([[1, 2], [6, 3],[0, 2]])
```

[10]:
```
def reduce_to_one_dimension(M):
    """Reduces the input matrix to one dimension using its SVD decomposition.

    Parameters
    ----------
    M : array, shape [N, D]
        Input matrix.

    Returns
    -------
    M_t: array, shape [N, 1]
        Reduce matrix.

    """
    # TODO
    # M= U Σ V.T
    temp = np.dot(M.T,M)
    lamba,V = get_eigen(temp)              # lamba = Σ^2
    max_index = np.argmax(lamba)           # get index of maximum eigen value
    one_dim = V[:,max_index]               # 1D translation vector
    one_dim = one_dim[:,None]              # shape (D,1)
    M_t = np.dot(M,one_dim)
    return M_t
```

[11]:
```
M_t = reduce_to_one_dimension(M)
```

[12]:
```
M_t
```

[12]:
```
array([[1.90211303],
       [6.68109819],
       [1.05146222]])
```