Machine Learning, Spring 2020

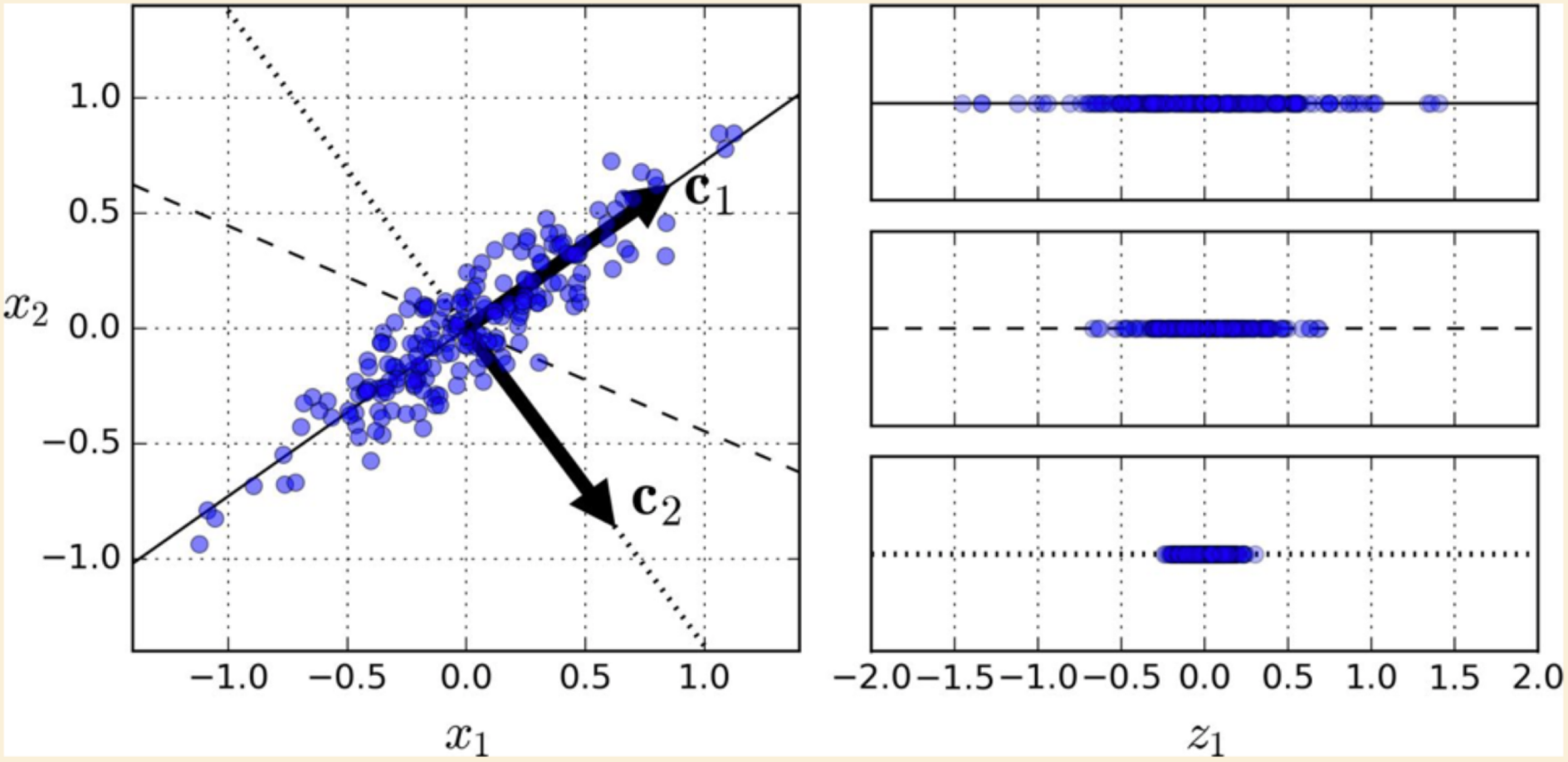
Project Four – PCA

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

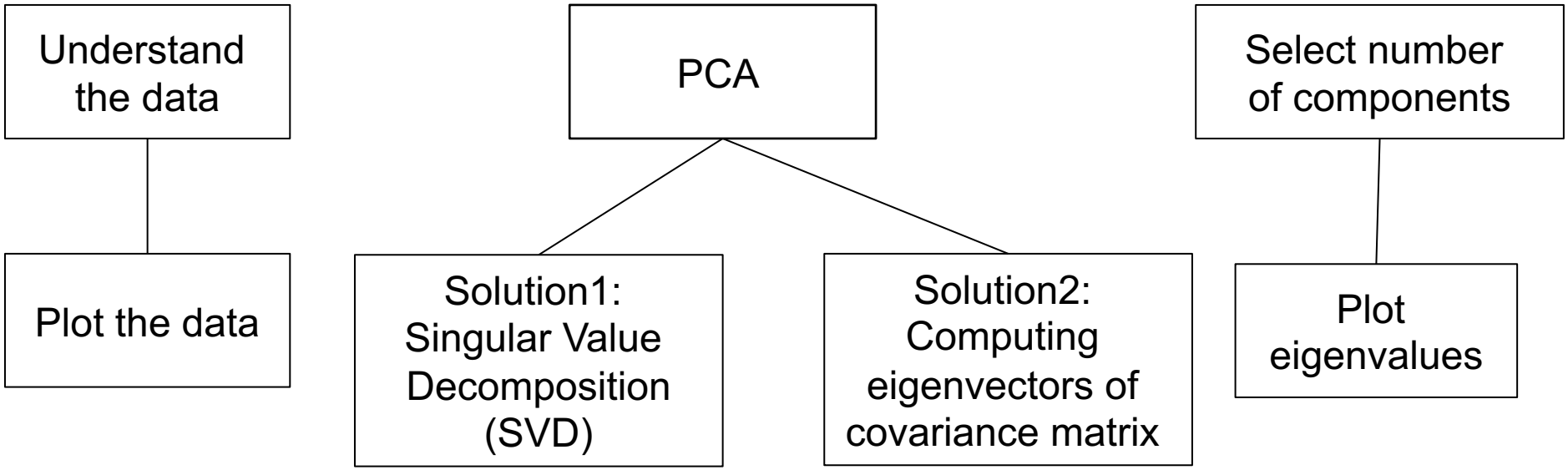
PyTorch tutorial: https://pytorch.org/tutorials/

PCA – Dimensionality Reduction

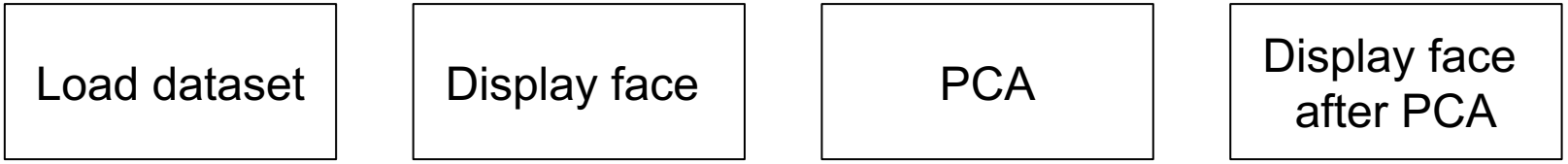


Main Modules for PCA

Task 1: Users to Movies



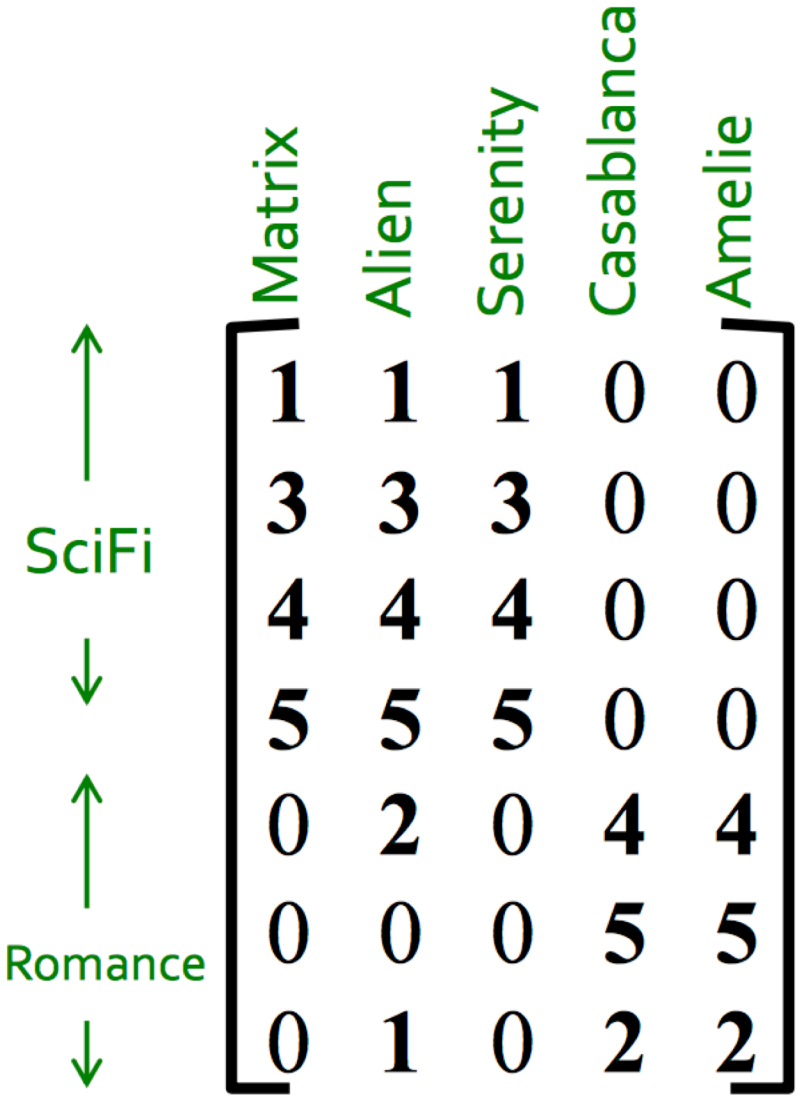
Task 2: Human Faces



Task 1: Users to Movies

Understand the data

Plot the data



Ratings matrix
-- each **column** corresponds to a **movie**
-- each **row** to a **user**.
-- First 4 users prefer **SciFi**, while others prefer **Romance**.

Task 1: Users to Movies

Understand the data

Plot the data in 3D

```
In [3]: # Plot the data set:

# 1. Create three arrays: users, movie, and reviews. to represent the data matrix
#     that is users[0], movie[0] and reviews[0] represent the review of the first user on the first movie.
# tips: use np.array() and flatten() function.

# 2. Set the figure size to (13,13) by using the function plt.figure().

# 3. Add the subplot that point the 1*1 grid by using the function add_subplot() on the figure object.
#     set the first positional arguments to 111 and projection to 3d.

# 4. Set the font size of the legend to be 10 by using plt.rcParams with 'legend.fontsize' as the key.

# 5. Plot the dataset using plot() for the Sci-fi movie and set x to be the user list, y to be the movie list and z to
#     moreover, set resonalbe color and label legend.

# 6. Plot the dataset using plot() for the Romance follow the pervious instruction.

# 7. Set the legend to a proper position using ax.legend(loc=?)

# 8. Set label for the x and y axis with proper front size using plt.xlabel(...)

# 9. Set the title of this fig using plt.title()

# 10. Set the ticks for x axis and y aixs by using plt.xticks()/yticks()

# 11. plot and present the fig using plt.show()
```

Example result:

Task 1: Users to Movies

PCA

Solution1: Singular Value Decomposition (SVD)

- Centering the data : X_centered = X – X.mean
- Implement PCA using SVD
  - Obtain **U, S, V^T** using *np.linalg.svd()*
  - U: each column is the eigenvectors of X^TX
  - S: the square roots of eigenvalues from X^TX or XX^T
  - V: each column is the eigenvectors of XX^T

In [4]:
```
# Data Preprocessing:

# 1. Calculate the mean of the data set
# 2. Subtract the mean from the data set
# 3. Store the new centered data set
```

In [5]:
```
# Calculate the U, S, V^T:
# 1. Use the singular value decomposition from numpy.
# 2. np.linalg.svd()
# 3. Store the u,s,v^T values
```

Task 1: Users to Movies

PCA

Solution2: Computing eigenvectors of covariance matrix

- Centering the data : X_centered = X – X.mean
- Implement PCA by direct computation:
  - Compute the covariance matrix ($\mathbf{X}^T\mathbf{X}$) of X_centered
  - Compute **V** (eigenvectors), and the diagonal elements of **D** (eigenvalues) using *np.linalg.eig()*

In [10]:
```
# Alternative implementation:
# Directly computing V and D from X and X^T
# 1. Comput XTX using np.matmul() and store it.
# 2. Apply np.linalg.eig() to clculate the eigen vectors and values
```

Task 1: Users to Movies

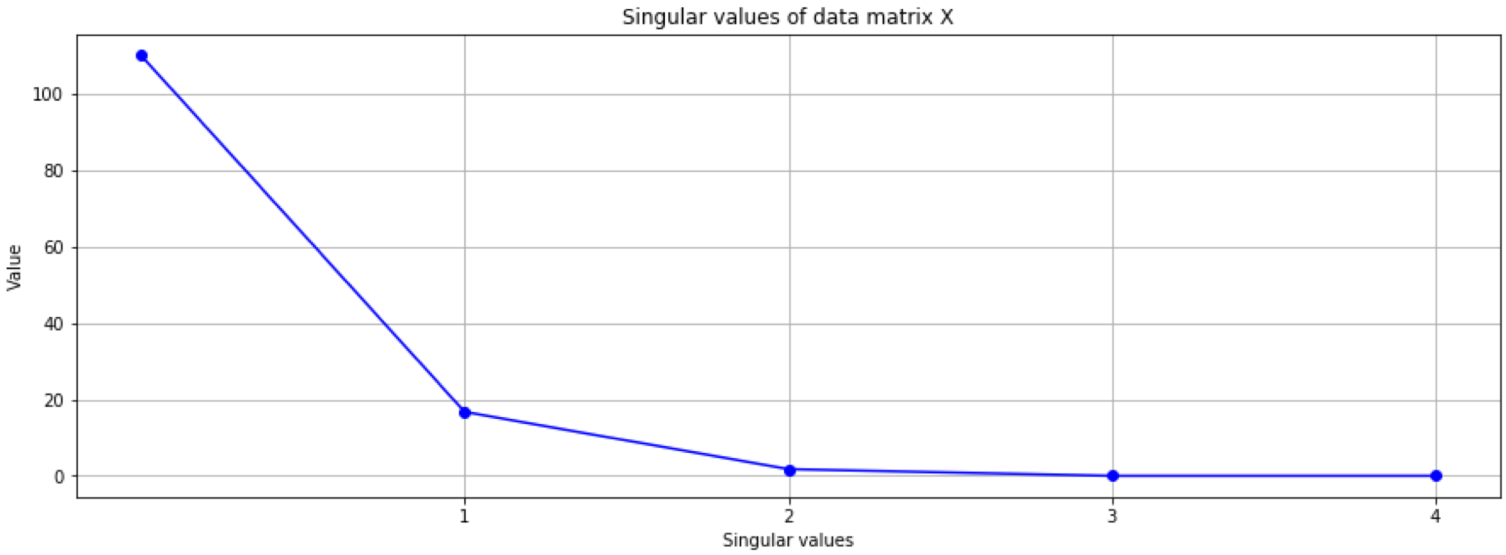
Select number of components

Plot eigenvalues

- Select *K* of principal components based on the eigenvalues
- Plot the eigenvalues

```
In [7]: # plot the singlar values for the  D  matrix.
        # 1. Calculate the D matrix using s: D is s*s
        # 2. Set the fig size to (15,5)
        # 3. Add the line chart using plt.plot( ?? ,'bo-')
        # 3. Add proper tital, ticks, axis labels
```

Example result:



Task 1: Users to Movies

Select number of components

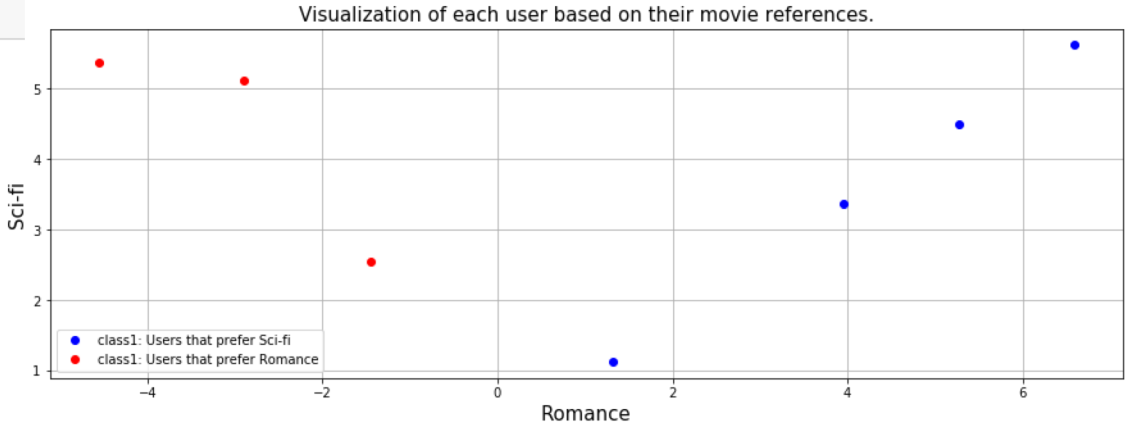
Plot eigenvalues

- Select *K* of principal components based on the eigenvalues
- Plot the eigenvalues
- Project data onto the space with reduced dimensions: X_pca = $X*V_k$

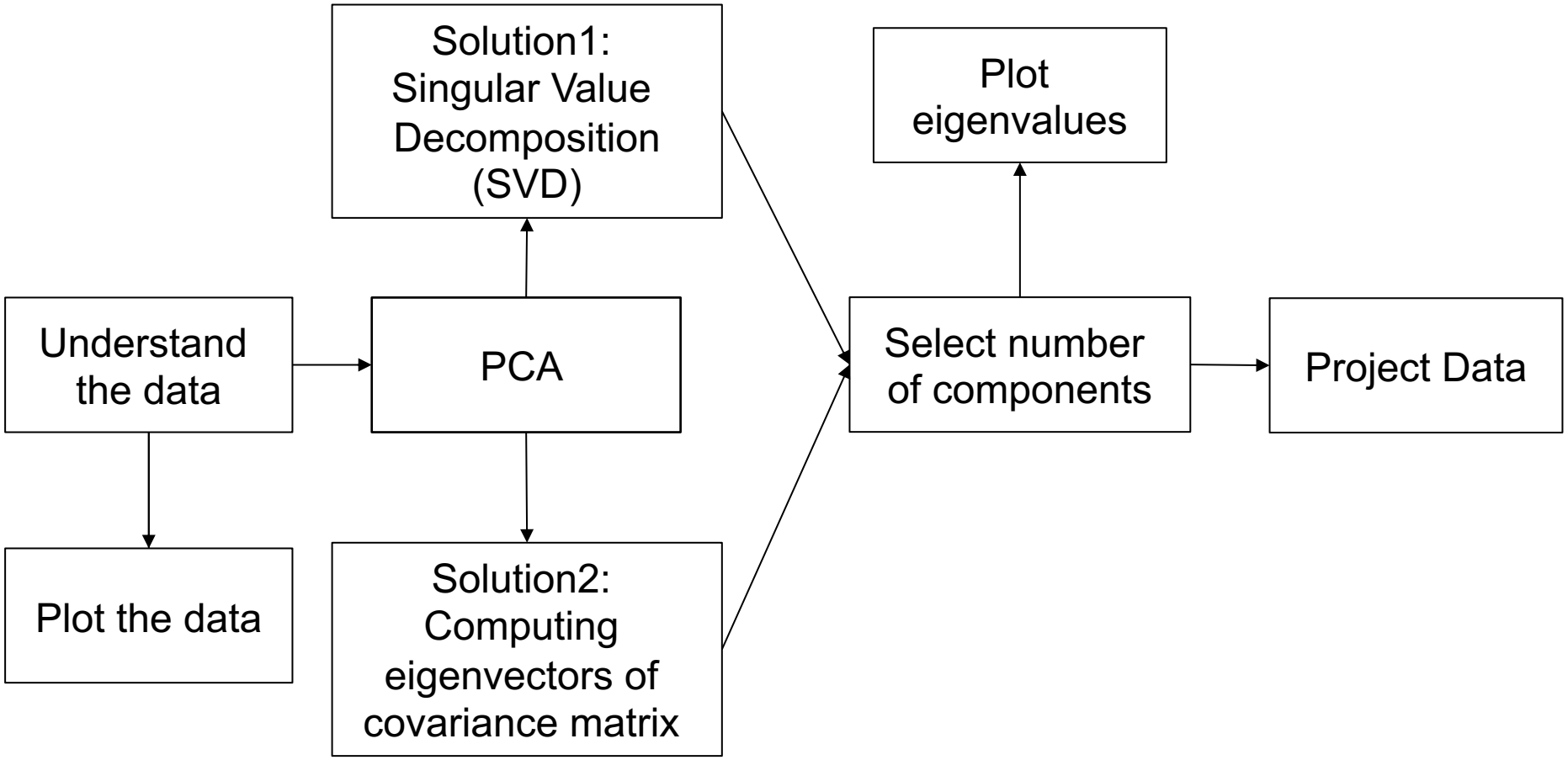
```
In [8]: # Obtaining our compressed data representation:
        # 1. Determine at least k singular values are needed to represent the data set from the fig above
        # 2. Obtain the first k of v^T and store it
        # 3. Calculate the compressed data using np.matmul(), X and stored first k of v^T
        # 4. Print the compressed value of X
```

```
In [9]: # Visualize what just happened:
        # 1. Set the fig size to (15,5)
        # 2. Create propor title, axis and legend
        # 3. Plot the data
```

Example result:



Pipeline for Task 1: Users to Movies



Task 2: Human Faces

Load dataset

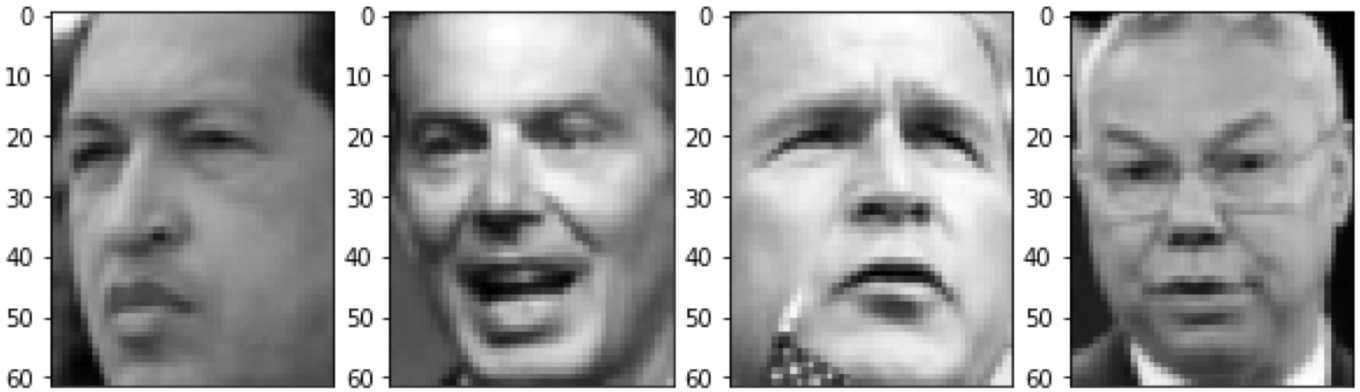
Display face

- Import dataset *fetch_lfw_people* from sklearn.dataset
- Show the faces in the dataset using *plt.imshow()*

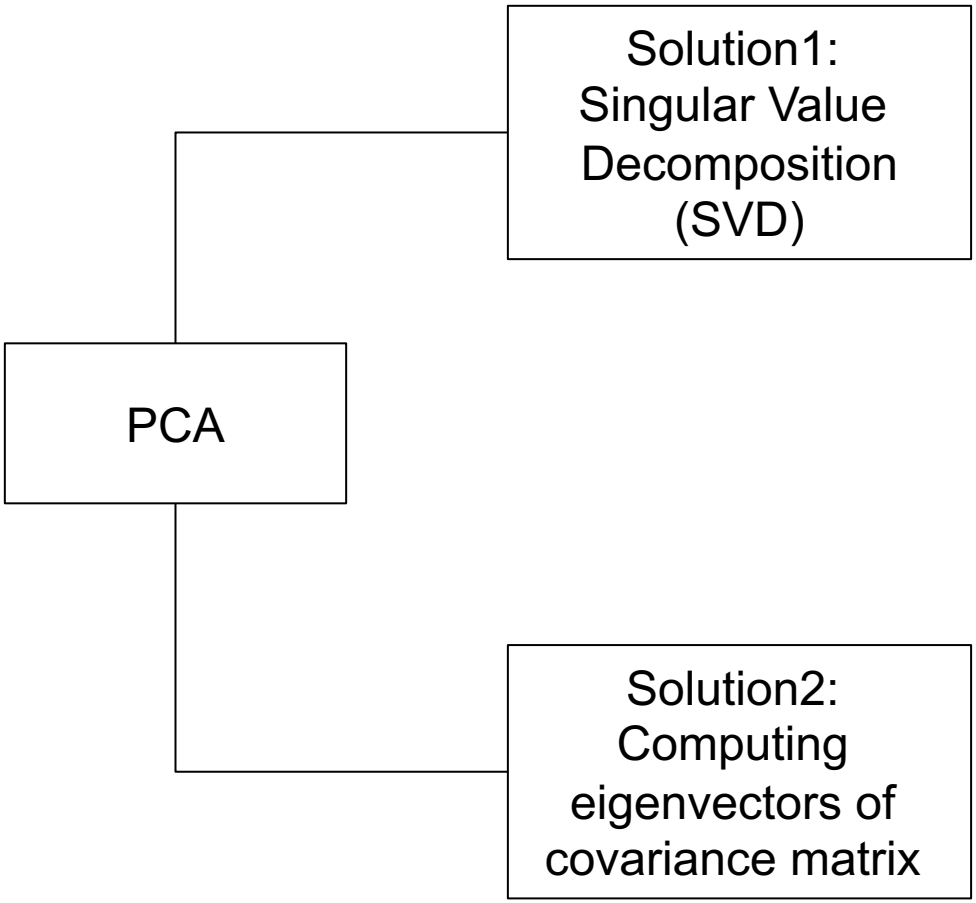
```
In [ ]: # Data set:
        # 1. Load the dataset using fetch_lfw_people() with min_faces_per_person setted to be 70
        #     detail of min_faces_per_person please refer to https://scikit-learn.org/stable/modules/generat
        # 2. Store the number of images and its hight, width using lfw_people.images.shape
        # 3. Calculate number of pixels
        # 4. Store the pixel values using lfw_people.data
```

```
In [ ]: def plt_face(x):
            global h,w
            plt.imshow(x.reshape((h, w)), cmap=plt.cm.gray)
            plt.xticks([])
```

Example result:



Task 2: Human Faces



- Utilize either of the implemented solutions to compute the PCA
- Get the results with the Top 5 PCA and Top 50 PCA

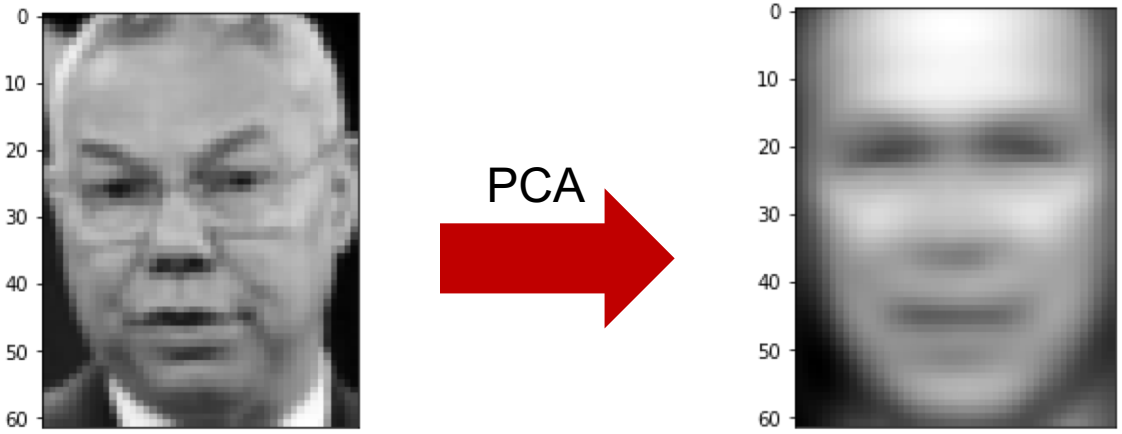
Task 2: Human Faces

Display face after PCA

- Project image data onto the space with reduced dimensions: $X\_pca = X*V_k$
- Project back to image with the features after PCA:
  $X' = X\_pca * V_k^T + X\_mean$

```
In [21]: # project back to the image space where d=5
         # X'= X_pca * VT  + X_mean
```

Example result:



Task 2: Human Faces