Applied Machine Learning

Project Three – Clustering and PCA

Python tutorial: http://learnpython.org/

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

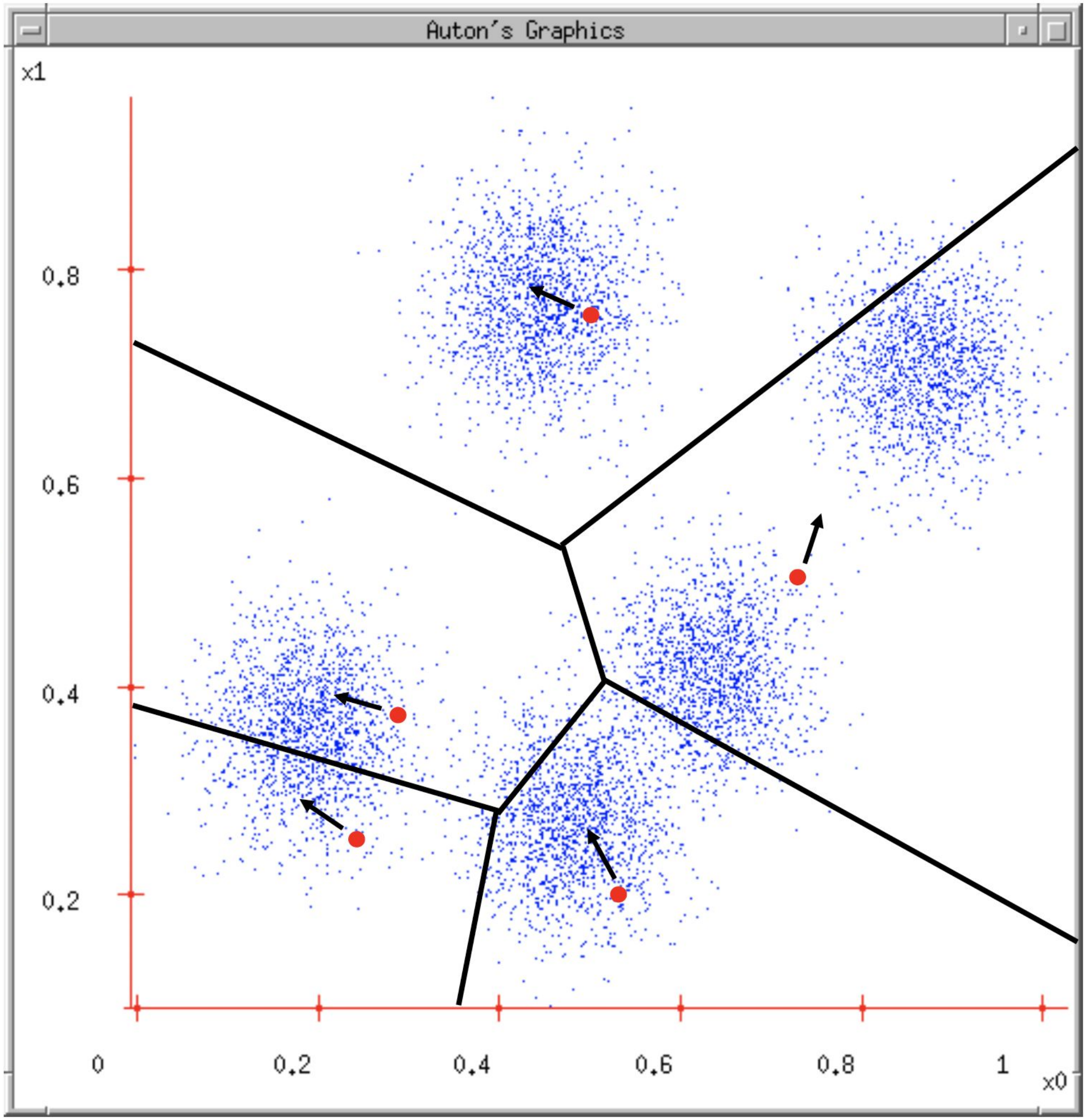
PyTorch tutorial: https://pytorch.org/tutorials/

Prof. Yi Fang (yfang@nyu.edu)

K-means

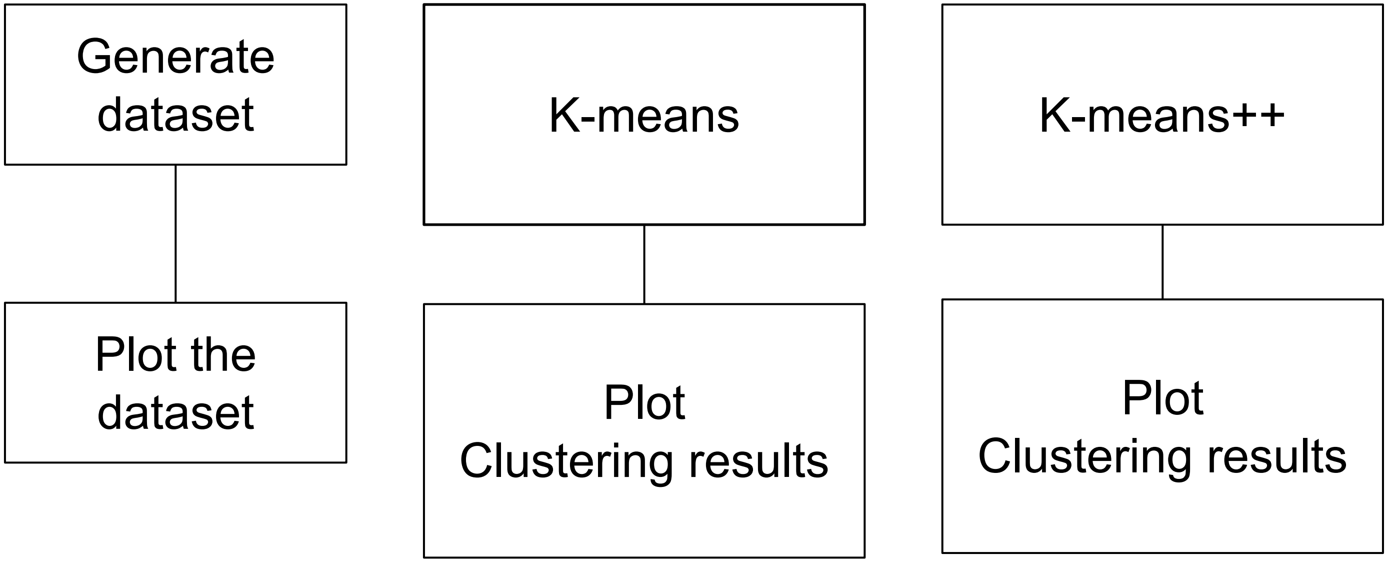
1. Ask user how many clusters they'd like. (e.g. k = 5)
2. Randomly guess k cluster center locations
3. Each datapoint finds out which center it's closest to.
4. Each center finds the centroid of the points it owns, and moves there.

Repeat steps 3-4 until convergence!



Thanks to Andrew Moore for providing this example.

Main Modules for Clustering



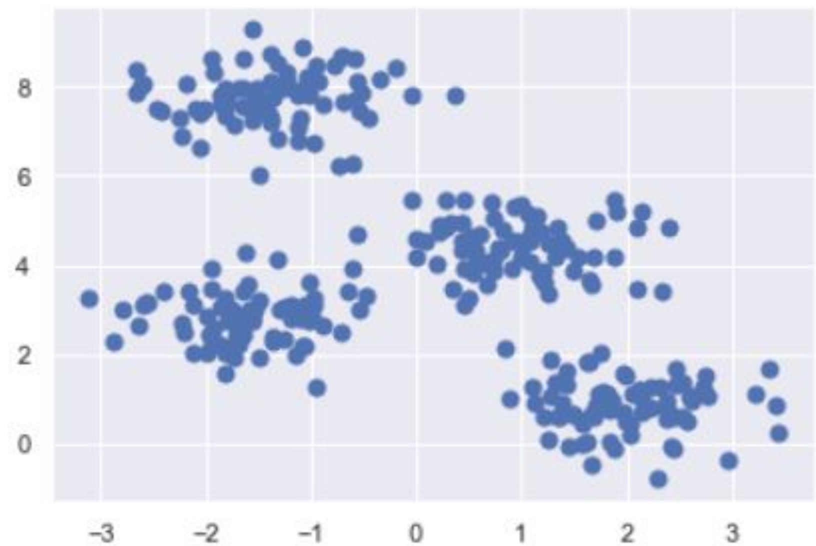
Main Modules for Clustering

Generate dataset

Plot the dataset

In [2]:
```
# manually generate dataset

# use the make_blobs() function with n_samples=300,centers=4,cluster_std=0.6 and random_state=0
# store the return value to X and y_true
# plot the dataset using plt.scatter()
```

Example result:



Main Modules for Clustering

K-means

Write the function k_means(X, k, rseed) that:

- Randomly select *k* points from X with rseed as initial centers
- Assign points in X to closest center and update the centers until convergence
- return final centers and cluster label for each point in X

In [5]:
```
# def k_means(X, n_clusters, rseed=2):
    # 1. Randomly choose clusters
#     using np.random.RandomState first to set the seed and store it to a variable r
#     using r.permutation(data shape) to choose first k data point index as initial center.
#     store the center to a list.
#     repeat until convergence:
#         Assign labels based on closest center using pairwise_distances_argmin()
#         Find new centers from means of points:
#             Update centroid of each cluster to be the average(mean) of examples assigned to cluster k
#         check for convergence:
#             convergence if old center is new center
#     return the centers and labels
```

Main Modules for Clustering

K-means

Plot
Clustering results

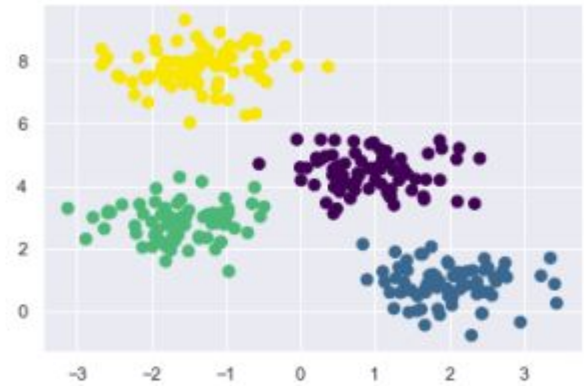
Write the function k_means(X, k, rseed) that:

- Randomly select *k* points from X with rseed as initial center
- Assign points in X to closest center and update the centers until convergence
- return final centers and cluster label for each point in X

In [ ]:
```
# fit our function to the data set with the starting point rseed=0.
# plot the figures
```

In [6]:
```
# fit our function to the data set with the starting point rseed=2.
# plot the figure
```

Example result:



Main Modules for Clustering

K-means++

Write the function k_meanspp(X, k, rseed) that:

- Randomly select ***a*** point from X with rseed as initial center
- Repeat until all ***k*** centers have been found
  - For each point in X, calculate the distance to closest center we found so far, then calculate the probability
  - Randomly choose a new center based on probability
- Run k-means with selected centers as initialization

Algorithm k-means++

$\mu_1 = \mathbf{x}^{(j)}$ for j chosen uniformly at random *// randomly initialize first point*

for k''=2 **to** k **do**

$$d_j = \min_{k' < k''} \left\| \boldsymbol{x}^{(j)} - \boldsymbol{\mu}_{k'} \right\|, \forall j$$ *// compute distances*

$$p_j = \frac{d_j^2}{\sum_{i=1}^{m} d_i^2}, \forall j$$ *// normalize to probability distribution*

j = random chosen with probability p_j

$\mu_{k''} = \mathbf{x}^{(j)}$

run k-means using μ as initial centers

Try to find a point far away from all the other centers as a new center

Main Modules for Clustering

K-means++

Write the function k_meanspp(X, k, rseed) that:

- Randomly select ***a*** point from X with rseed as initial center
- Repeat until all ***k*** centers have been found
  - For each point in X, calculate the distance to closest center we found so far, then calculate the probability
  - Randomly choose a new center based on probability
- Run k-means with selected centers as initialization

In [8]:

```
# def eucl_dist(a, b, axis=1):
#     def the function that calculate the L2 distance

# def the init function for kmean++:
#
# def init_center(k,X,rseed):
#     create a empty list store centers
#     random choose a center:
#         random choose a index:
#         using np.random.RandomState first to set the seed and store it to a variable r
#         using r.permutation(data shape) to choose first data point index as initial center.
#     append this center to the center list
#     while the length of the list less than k:
#         calculate dj for all data point: dj=min(||x^j-c_k||) whiere dj store the distance to the cloest center
#         calculate pj: pj=dj^2/sum_all(d^2) for all data point
#         random choose j using the probability:
#             using np.random.choice()
#         set the new center to be x^j
#         append the new center to center list
#     return all centers
```

In [9]:

```
# def the kmean++:
# def k_meanspp(X, n_clusters):
#     first init centers
#     then, run the k-means with the initialized centers.
```

Main Modules for Clustering

K-means++

Plot
Clustering results

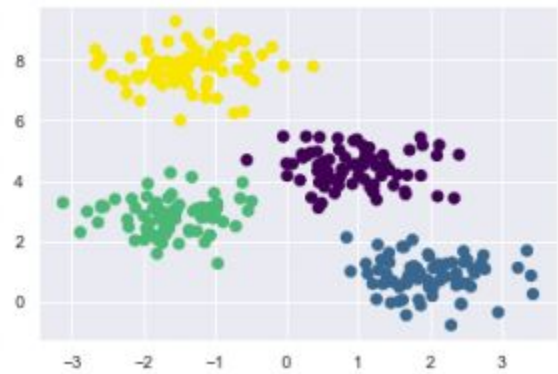
Write the function k_meanspp(X, k, rseed) that:

- Randomly select ***a*** point from X with rseed as initial center
- Repeat until all ***k*** centers have been found
  - For each point in X, calculate the distance to closest center we found so far, then calculate the probability
  - Randomly choose a new center based on probability
- Run k-means with selected centers as initialization

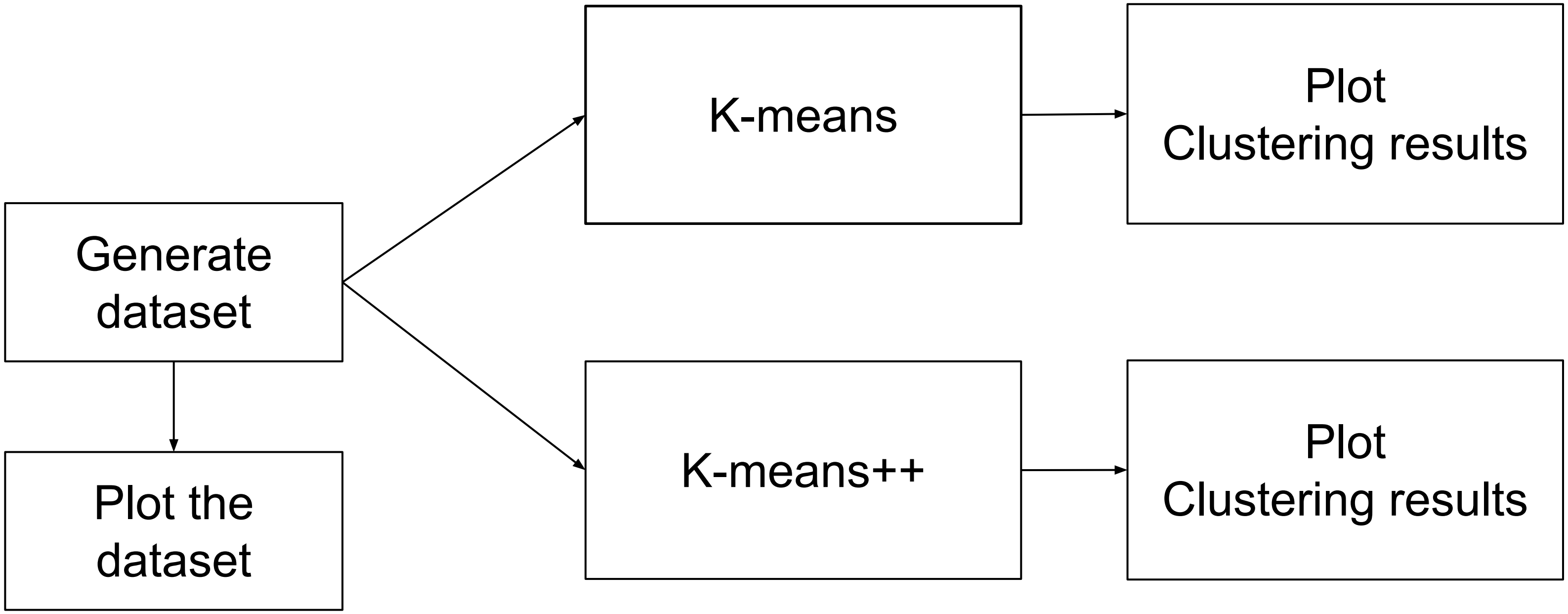
```
In [11]: # fit our kmean++ function to the data set with rseed=0.
         # plot the figure
```

```
In [10]: # fit our kmean++ function to the data set with rseed=2.
         # plot the figure
```

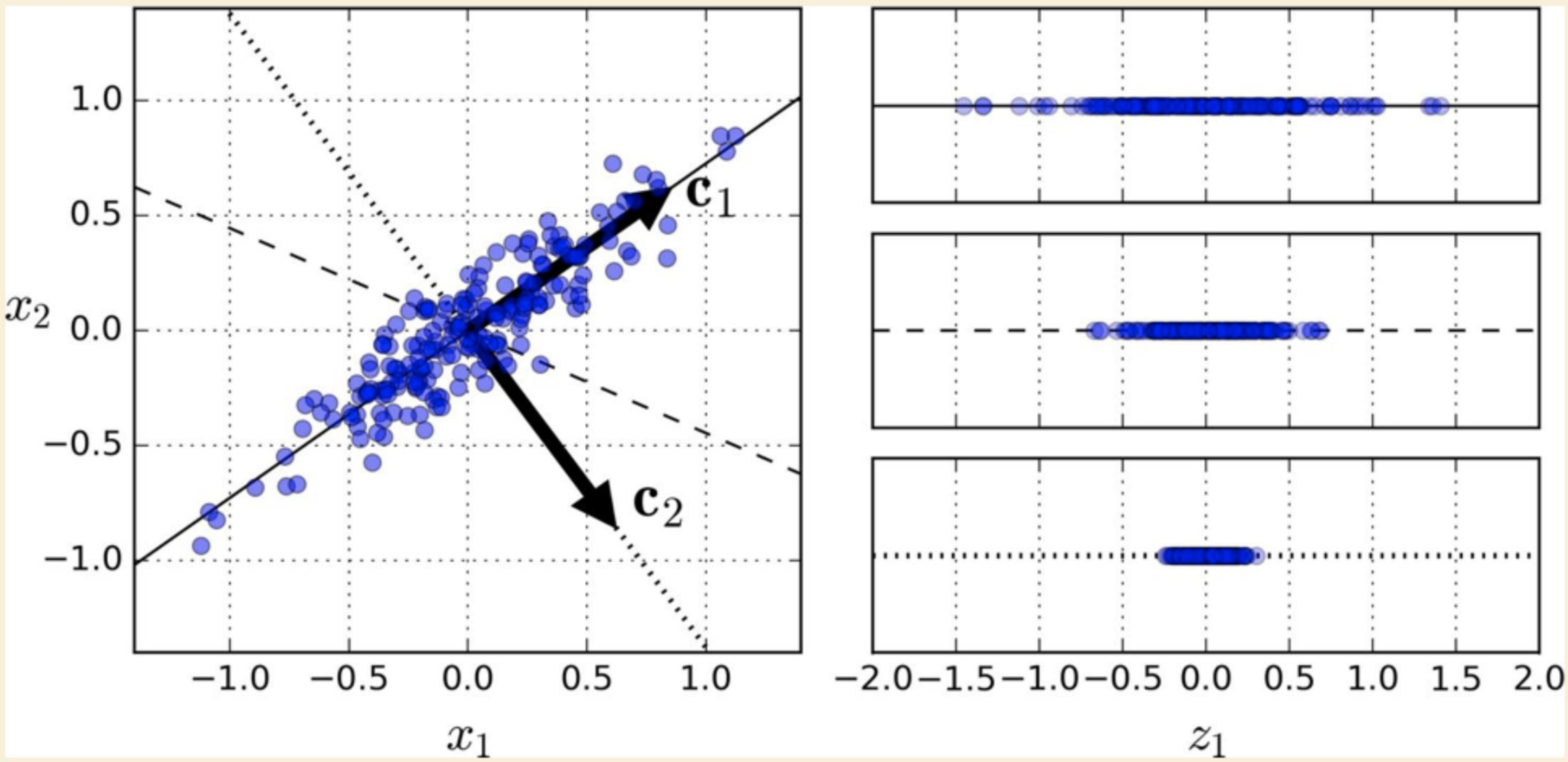
Example result:



Pipeline for Clustering



PCA – Dimensionality Reduction



Main Modules for PCA

Task 1: Users to Movies

- Understand the data
  - Plot the data
- PCA
  - Solution1: Singular Value Decomposition (SVD)
  - Solution2: Computing eigenvectors of covariance matrix
- Select number of components
  - Plot eigenvalues

Task 2: Human Faces

- Load dataset
- Display face
- PCA
- Display face after PCA

Task 1: Users to Movies

Understand the data

Plot the data

$$
\begin{array}{c|c}
 & \begin{array}{ccccc} \text{Matrix} & \text{Alien} & \text{Serenity} & \text{Casablanca} & \text{Amelie} \end{array} \\
\begin{array}{c} \\ \\ \text{SciFi} \\ \\ \\ \text{Romance} \\ \\ \end{array} &
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 \\
3 & 3 & 3 & 0 & 0 \\
4 & 4 & 4 & 0 & 0 \\
5 & 5 & 5 & 0 & 0 \\
0 & 2 & 0 & 4 & 4 \\
0 & 0 & 0 & 5 & 5 \\
0 & 1 & 0 & 2 & 2
\end{bmatrix}
\end{array}
$$

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

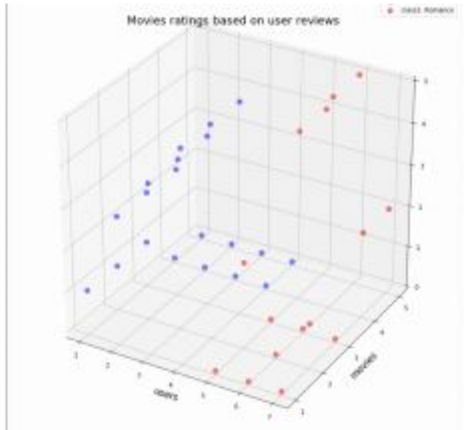
Task 1: Users to Movies

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

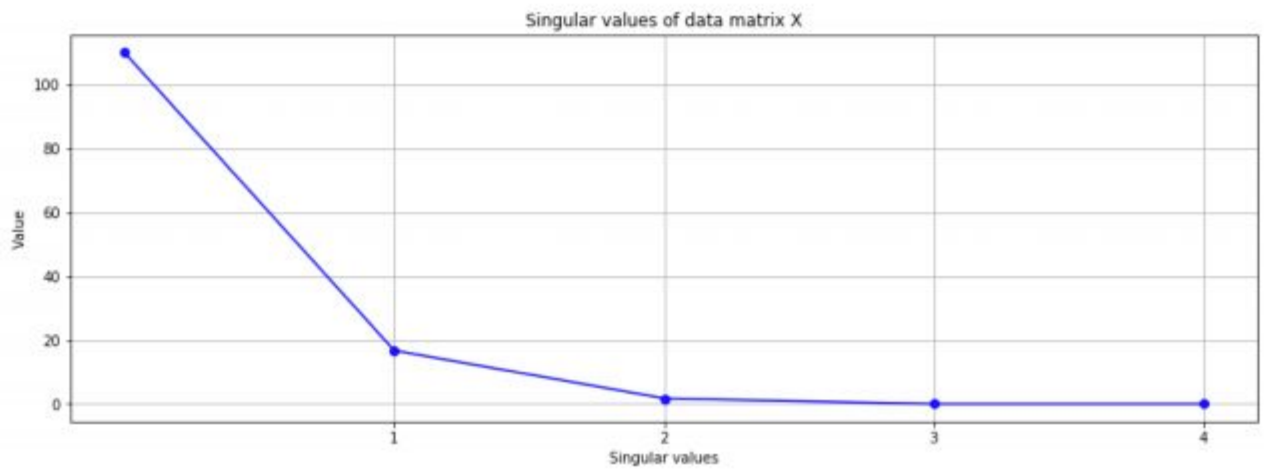
Select number of components

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

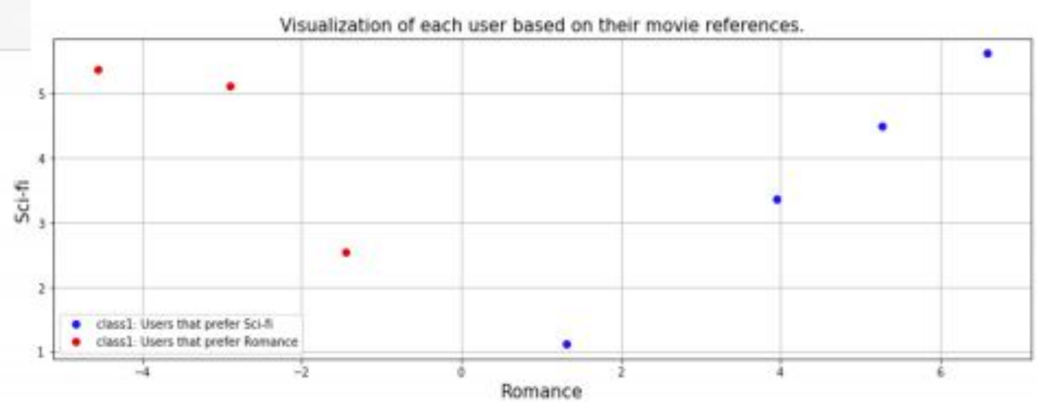
Plot eigenvalues

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

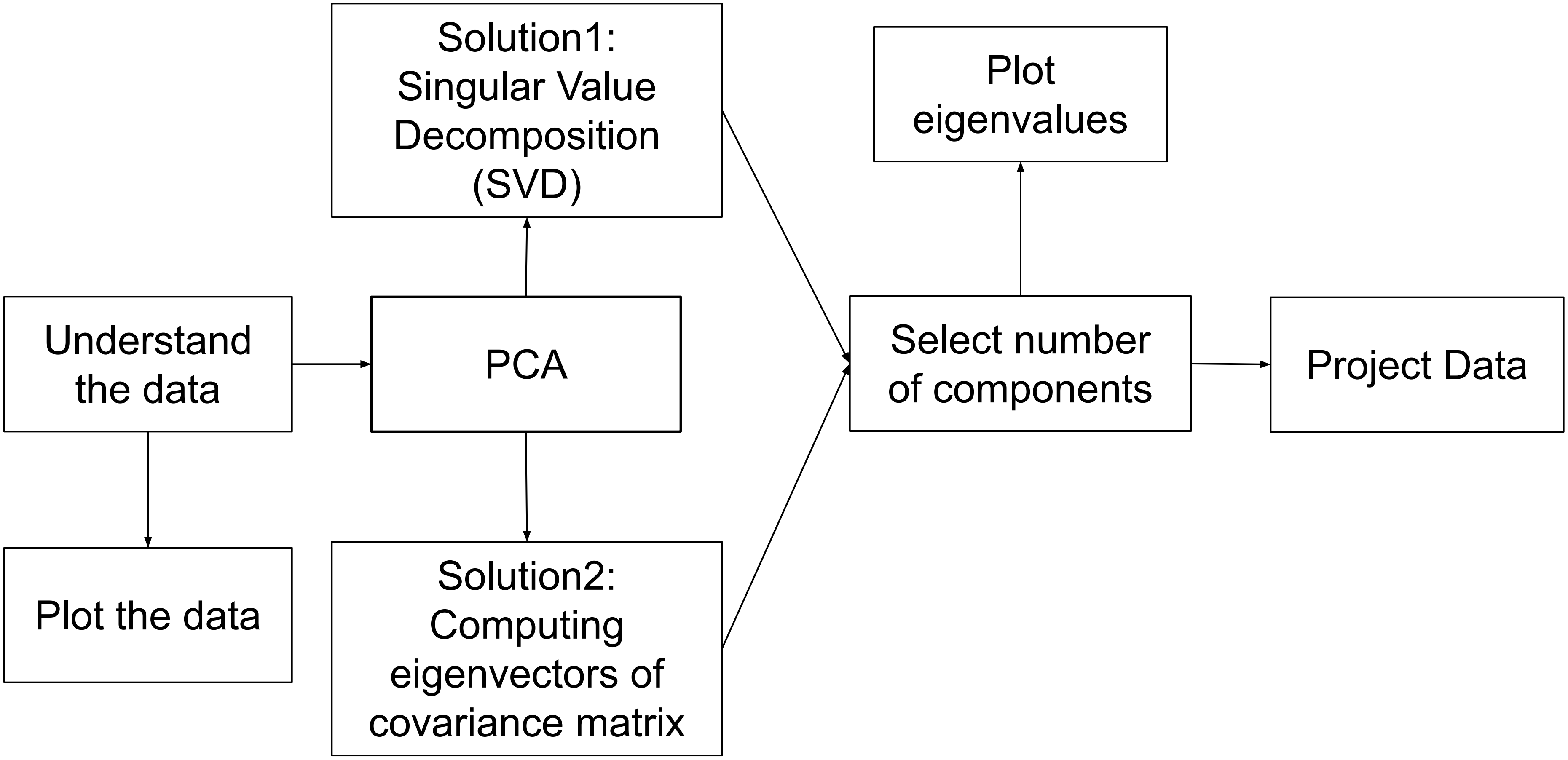
```
In [8]: # Obtaining our compressed data representation:
        # 1. Determine at least k singular values are needed to represent the data set from the fig above
        # 2. Obtain the first k of v^T and store it
        # 3. Calculate the compressed data using np.matmul(), X and stored first k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

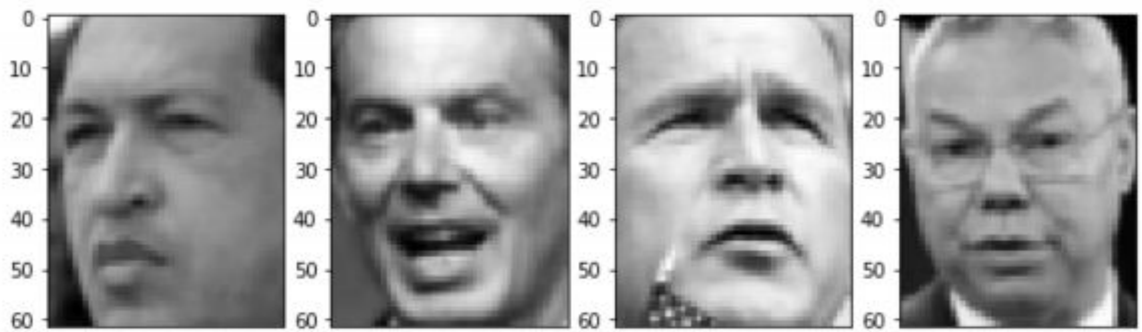
Display face

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

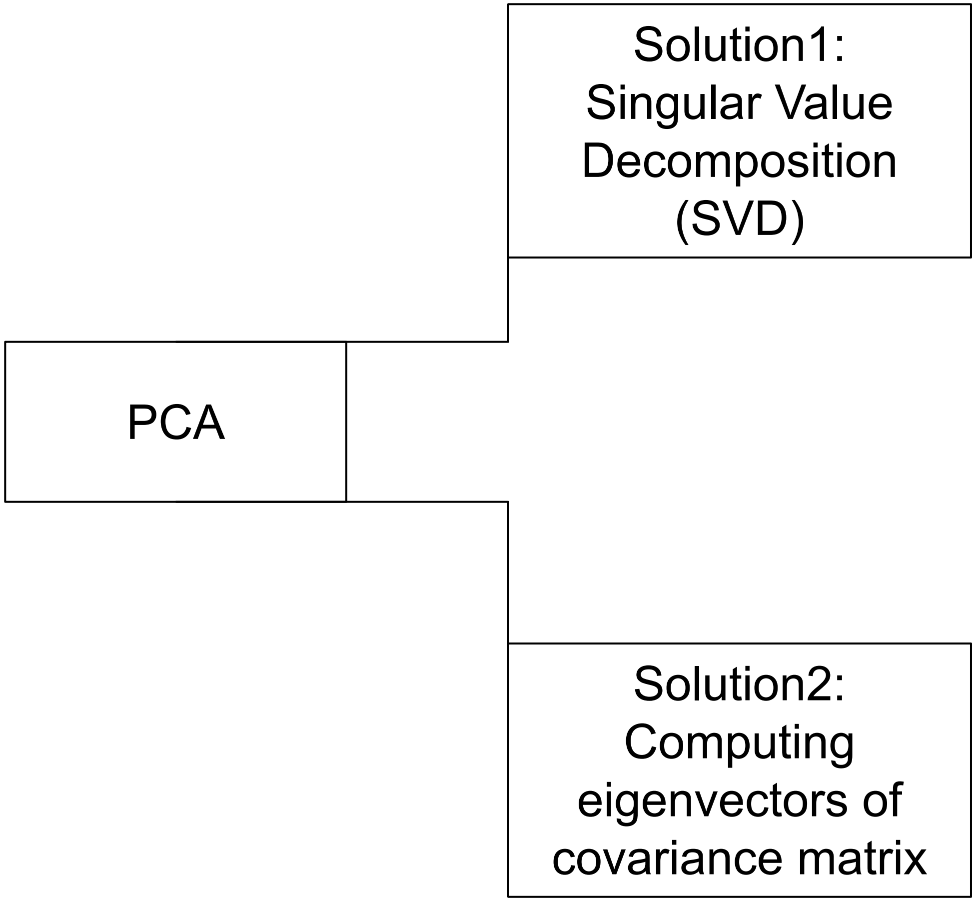
```
In [ ]: # Data set:
        # 1. Load the dataset using fetch_lfw_people() with min_faces_per_person setted to be 70
        #     detail of min_faces_per_person please refer to https://scikit-learn.org/stable/modules/generat
        # 2. Store the number of images and its hight, width using lfw_people.images.shape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

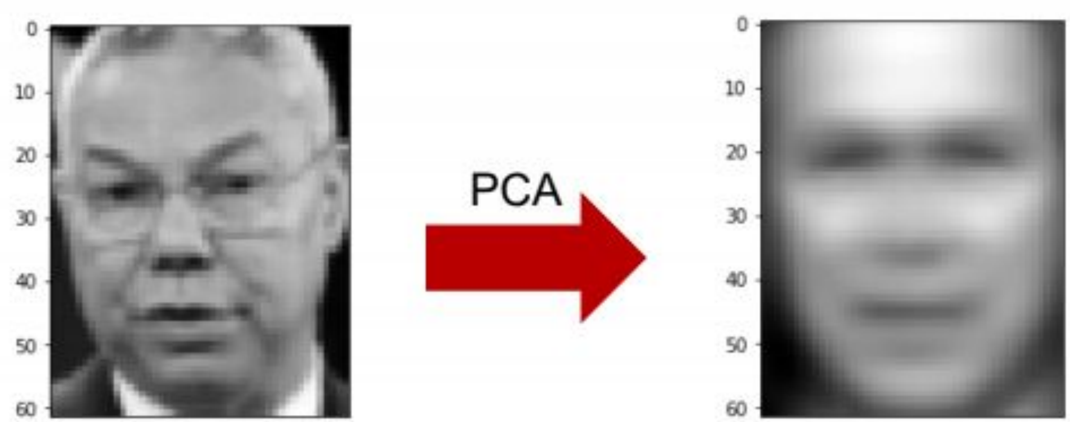
Task 2: Human Faces

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