Dimensionality Reduction

Lab

PCA/MDS/ISOMAP

Apostolos N. Papadopoulos – Christos Giatsidis

What are we going to do?

- Implement:
  - PCA
  - MDS
  - Isomap
- Compare implementation results to sklearn equivalents
- Extra: Linear Discriminative Analysis

Dimensionality Reduction

- We need a smaller space to project data
  - How do we project data?
    - What is the result of the projection
  - Where do we find the smaller space?
- We do not want to lose information from the original space
  - The new space must maintain properties of the original one (e.g. distances, variance e.t.c.)

Basics: SVD

- SVD is standard in matrix manipulation packages
  - e.g. in **numpy** *svd* is in sub-package *linalg*
- **Reminder:**
  - $X = U\Sigma V^T$
  - U: eigenvectors of XX^T
  - V:eigenvectors of X^TX
  - Σ: eigen values of XX^T & X^TX
- Keep the first k rows of U,V and eigenvalues of Σ
- We try to maintain the magnitude of the values

Back-Ground Info

- There are multiple versions of dimensionality reduction techniques that carry the same name
  - Small improvements
  - Different approaches
  - Slightly different implementations
    - Might be more convent under specific platform
- The majority of them has the same "plan"
  - Map a property of your data into Matrix Form
  - Apply SVD/eigen decomposition

PCA

- Try to maintain the variance
- Suppose matrix **A** with the **data** (mxn)

1. C = A – M *Where* ***M*** *contains the mean per column*
2. $W = C^T C$ *(W holds the variance)*
3. *E,V=eig(W) get the eigenvectors E & eigenvalues V of W*
4. *Keep the* ***k*** *(k<n) largest eigenvectors* E_k *(Principal Components)*
5. $A' = CE_k$

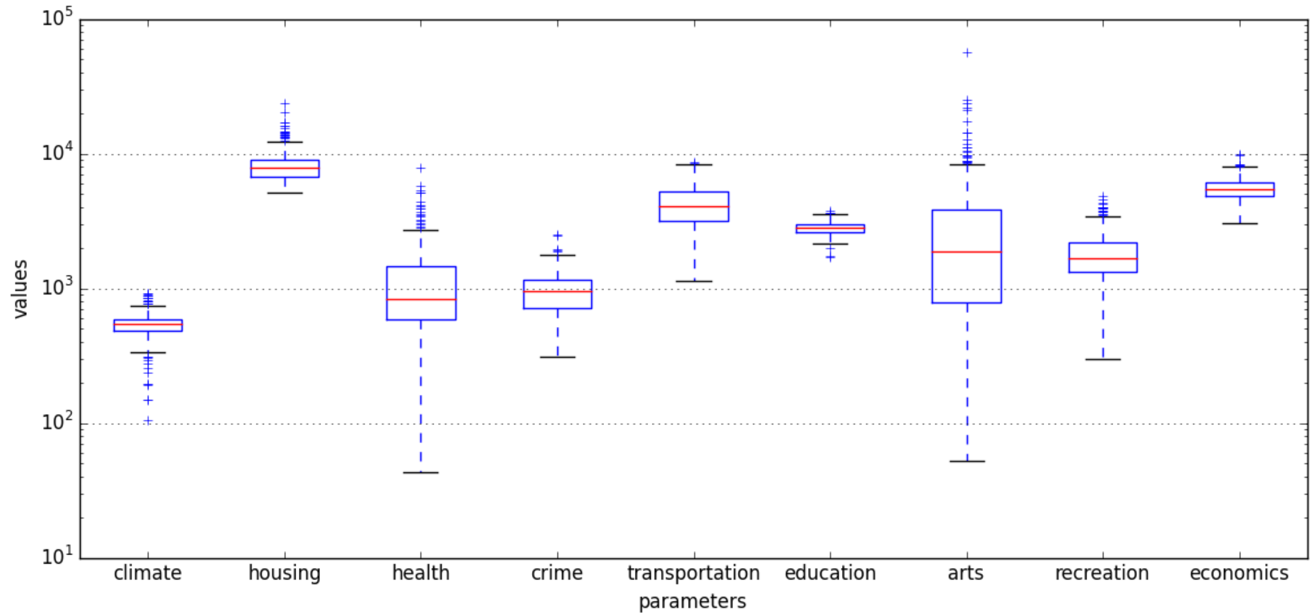
- A' is the new data (mxk)
- "largest" refers to the corresponding eigenvalues
- Remember SVD?
  - We could take "V" of svd(C)
- What do eigenvalues represent here?

Cities data

- We have a dataset of cities and some metrics that identify the quality of life
  - climate, housing, health, crime, transportation, education, arts, recreation, economics
  - File *cities.txt* (csv format)
- Python files:
  - *pcaImp.py* : **We have to fill in the code here**
  - *pca_example.py* : **The main file to run**

Implementing PCA

- Try running *pca_example.py*

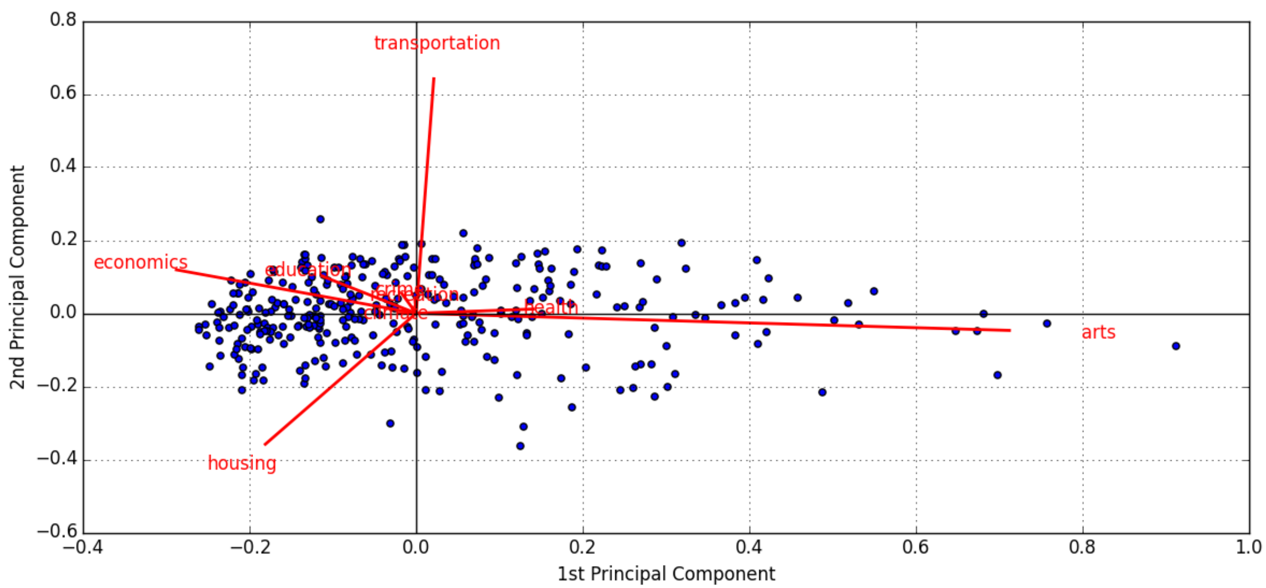


- We want to see how the cities are “spread” over these features
  - Get the first 2 PCs and plot only those two
- What if you project the transpose of your data onto the PCs?
  - For i,j : $(X[:,i]^T PC[:,j])$

Fill in the code

1. C = A – M *Where* **M** *contains the mean per column*
2. $W = C^T C$ *(W holds the variance)*
3. *E=eig(W) get the eigenvalues of W*
4. *Keep the* ***k*** *(k<n) largest eigenvectors* E_k *(Principal Components)*
5. $A' = CE_k$

What you are supposed to get from pca_example.py



MDS(classic)

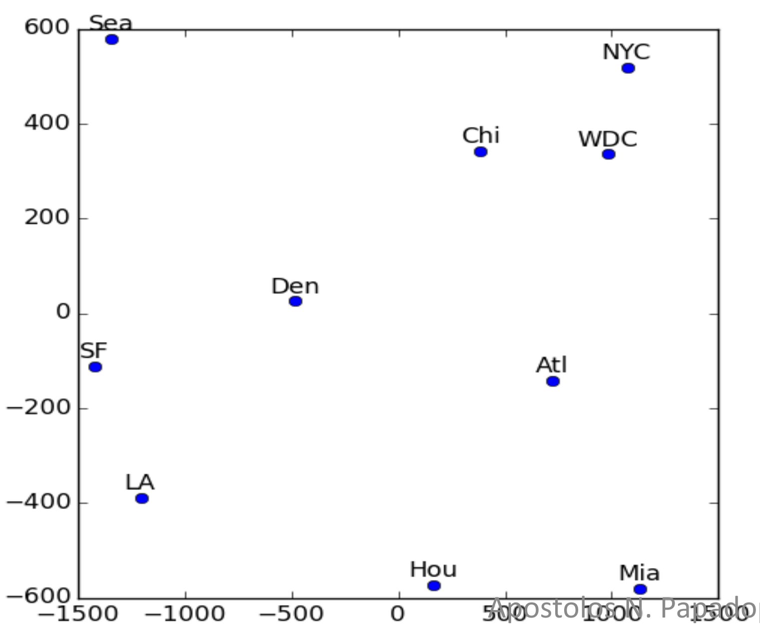
- Maintain the distances / similarities / dissimilarities among all pairs of elements

1. Suppose matrix A with **distances**. $D = A^2$
2. $J = I_N - (\mathbf{1}\mathbf{1}^{\boldsymbol{T}})/N$
   - where I_N the NxN identity matrix (one's in diagonal and zeros elsewhere) and $(\mathbf{1}\mathbf{1}^{\boldsymbol{T}})$ is the NxN matrix with one's at each cell
3. $B = -\frac{1}{2}JDJ$
4. $U\Sigma V^T = svd(B)$
5. $A' = U_k {\Sigma^{1/2}}_k$ (k largest eigen values and corresponding vectors from U)

- U from svd : distances become similarities

Implementing MDS

- Files:
  – *distanceMatrix.csv :* distances among cities
  – *mds_example.py :* main file to run
    - Has the names of the cities
    - Map to compare the new data to an actual map
  – *mdsImp.py :* code to fill in

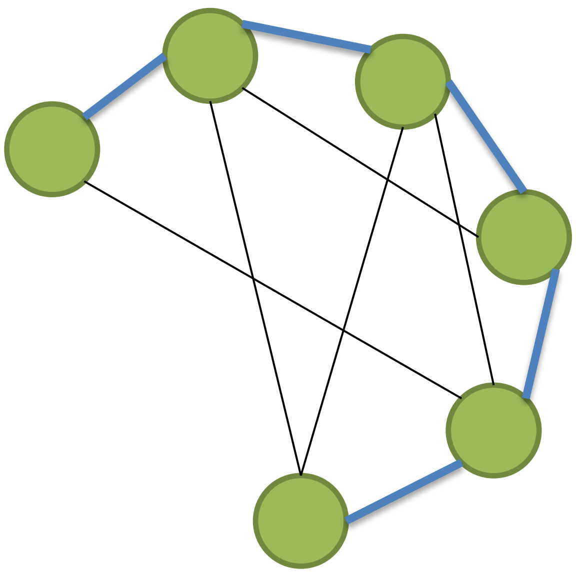


MDS & PCA overview

- The classic MDS algorithm works the similarity of the data
- PCA works on the similarity of the features
- PCA and classic MDS have the same results **if** Euclidean distance is used.
  - The eigen values of XX^T and XX^T are the same.
    - *[Cox & Cox (2001), p 43-44]*
  - *Starting from the distance matrix we can perform PCA too*
    - *[The Elements of Statistical Learning, section 18.5.2.]*

What about other approaches?(1)

- Other distances?
  - Local information



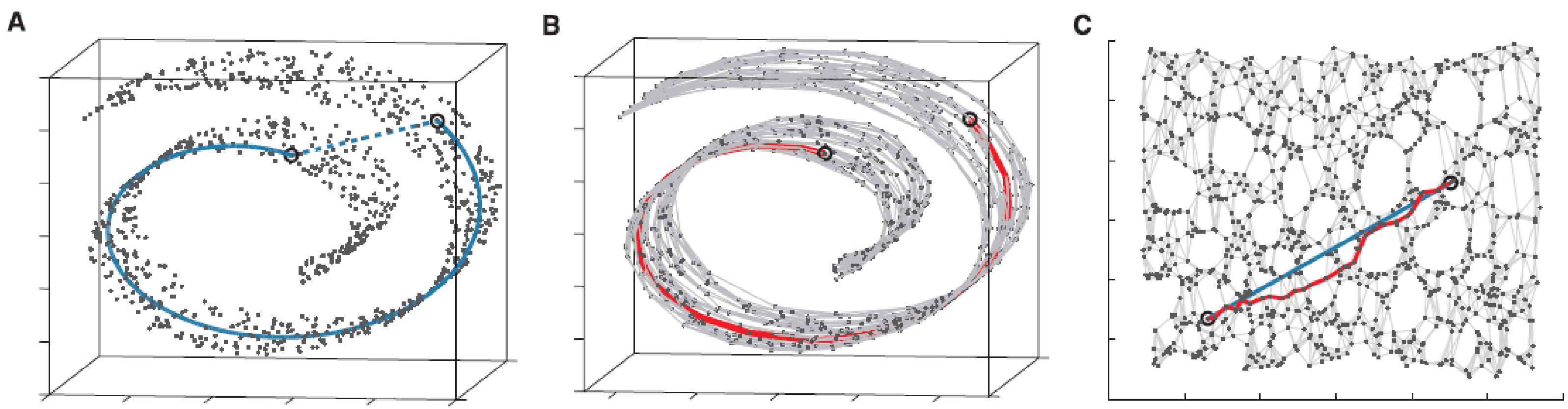
In the previous approaches , these dots are a “cloud”

But they follow a curved path

Approaches like Isomap try to work on finding the plane (path) the points define and “unrolling it”

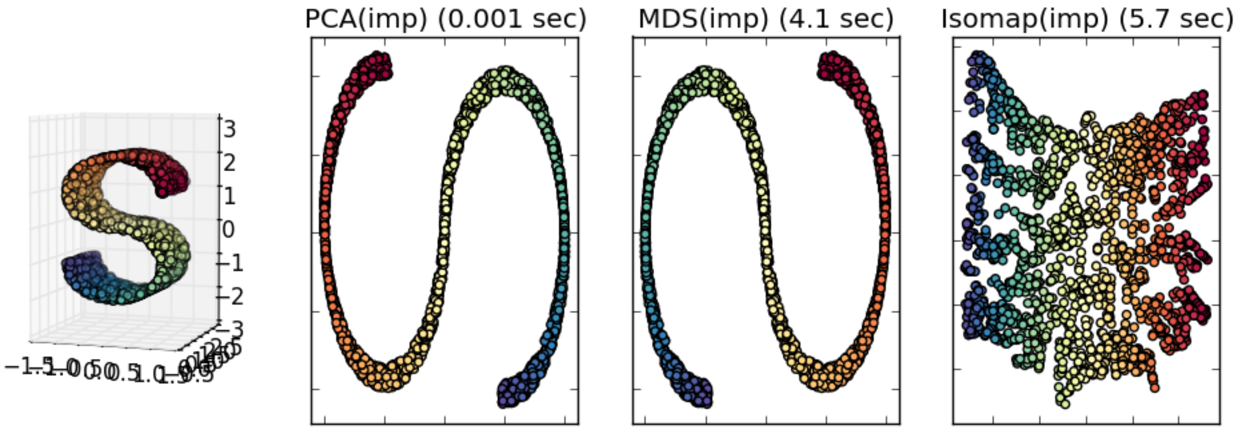
Isomap

1. Find the k-nearest neighbors of each point
   - **k** is a parameter of the algorithm
2. Construct a graph where each node is connected only to the k-nearest neighbours. Graph Matrix D_g
   - The links are typically weighted with the Euclidian distance
3. *Find graph distances between all points in the graph.* ***Shortest paths :*** $\boldsymbol{D_g}$
4. *Apply MDS on* $\boldsymbol{D_g}$
   - (or PCA)



Implementing Isomap

- Files :
  - *isomap.py* : to fill in the code
  - *compare.py :* Main file to run
    - *A big part of it is commented leave it for now*
    - *Try the S curve and the swissroll*
    - *What if you change the number of neighbors?*
    - *Benefits of isomap vs. PCA/MDS?*



Existing implementations

- sklearn contains many dimensionality techniques
- Remove multiline comment and run it again
  - Compares the results of our implementations with the ones ok sklearn
- What do you notice?
  - There is quite a difference for MDS
  - Why do you think sklean PCA is different?
    - Is it a coincidence it matches our MDS?

https://github.com/scikit-learn/scikit-learn/blob/a95203b/sklearn/decomposition/pca.py

MDS variations

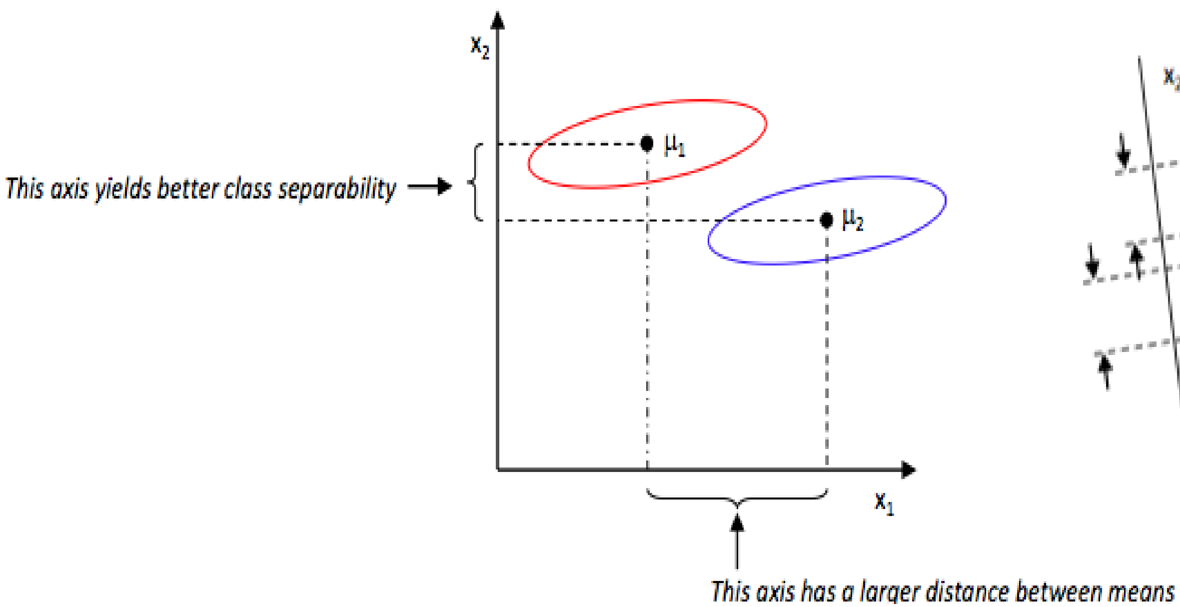
- MDS has a large history
  - *[Past, Present, and Future of Multidimensional Scaling, Patrick J. F. Groenen]*
- *Sklearn utilizes another method for MDS*
  - *Minimizing Stress with Majorization (an optimization technique)*
    - **S**caling by **MA**jorizing a **CO**omplicated **F**unction (SMACOF)
    - *The function is the one of stress*
  - https://en.wikipedia.org/wiki/Stress_majorization

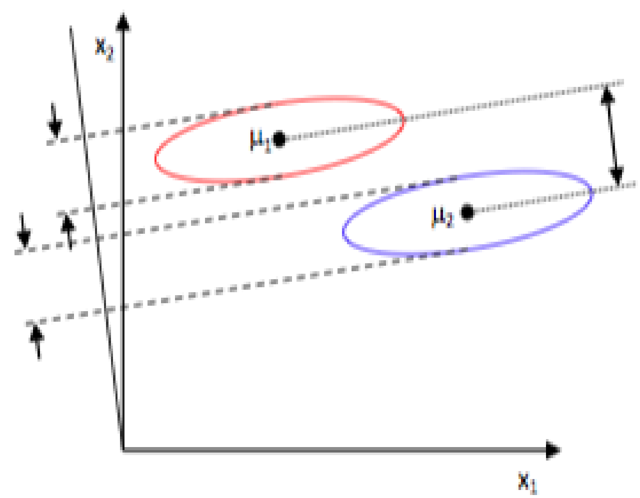
Non linear dimensionality reduction

- Isomap belongs in the non linear techniques
  - It has a few weaknesses
    - Sparse data don't work so well
- There are better techniques to consider (a bit more complicated to implement)
  - E.g. : Locally linear embedding (LLE)
    - Same starting point as Isomap (nearest neighbors)
    - Pseudo code : https://www.cs.nyu.edu/~roweis/lle/algorithm.html
    - It has many variations (e.g. Hessian LLE)

Other approaches (2)

- The previous techniques focus on properties of the feature space
  - They don't take into account any labels/classes the data may have
  - What if we tried to take into account properties of the data per class?
- Linear Discriminant Analysis (LDA):
  - **maximize** the distance of the class centers
  - **minimize** the within-class variance





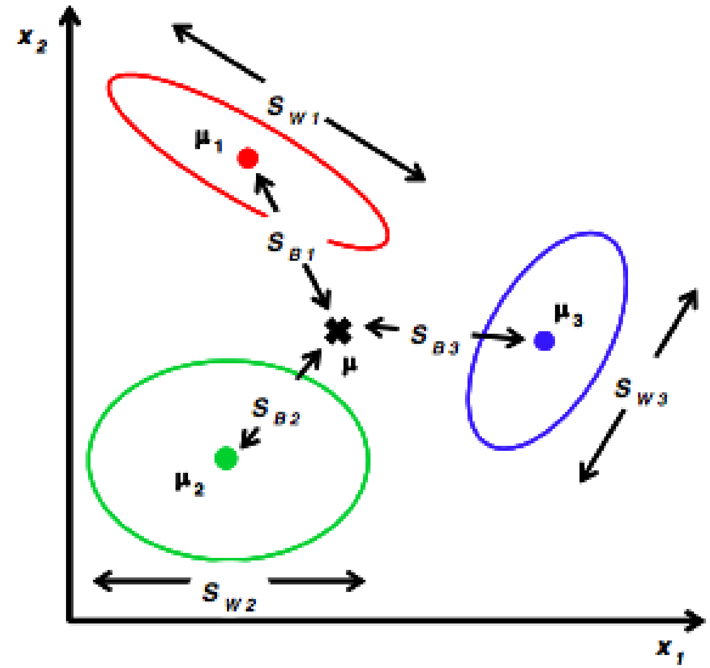
LDA (1)

- Here we look at the general case of K classes (C) (where k>2)
- Within class variance:
  - $S_w = \sum_{j=1}^{K} \sum_{x \in C_j} (x - m_j)(x - m_j)^T$
    - $m_j = 1/N_j \sum_{x \in C_j} x$ (the mean per class)
  - covariance(x)= $\dfrac{\sum_{x \in C_j} (x - m_j)\left(x - m_j\right)^T}{N_j}$
- $S_w = \sum_{j=1}^{K} \text{covariance}\left(x_{C_j}\right) * N_j$

LDA(2)

- Distance between classes (between class scatter)
  - $S_B = \sum_{i=1}^{K} N_i (m_i - m)(m_i - m)^T$
    - Where m is the TOTAL mean

- We try to maximize the distance from the total mean



LDA(3)

- Find projection matrix W
  - $W_{LDA} = argmax_w \frac{|W^T S_b W|}{|W^T S_w W|}$
    - Maximizes numerator and minimizes denominator
    - The ratio is known as Fischer's criterion
- The solution is given by the **K-1** largest eigenvectors of:${S_w}^{-1} S_B$

LDA Algorithm

Given data X and labels Y

1. Compute :
   a) S_w
   b) S_B
   c) $T = {S_w}^{-1} S_B$
2. *E,V=eig(T)*
3. *Keep the **K-1** (K =number of Classes) largest eigenvectors* $W = E_k$
4. *Project the data on W :* X_{LDA}
   a) Project the mean vectors of each class (centers of the class)

- The projection of the mean vectors is useful for simple classification
  - We can classify new data based on the minimum distance from the new projected centers
- We don't choose the number of eigenvectors

Example Data

- File:wine_data.csv
  - Chemical analysis of wines
  - 178 different wines
  - 13 features describing the chemical analysis
  - 3 classes of wines
- We want to compare LDA with PCA
  - They both deal with variance in general
  - Plot new space under both approaches

Implementing LDA

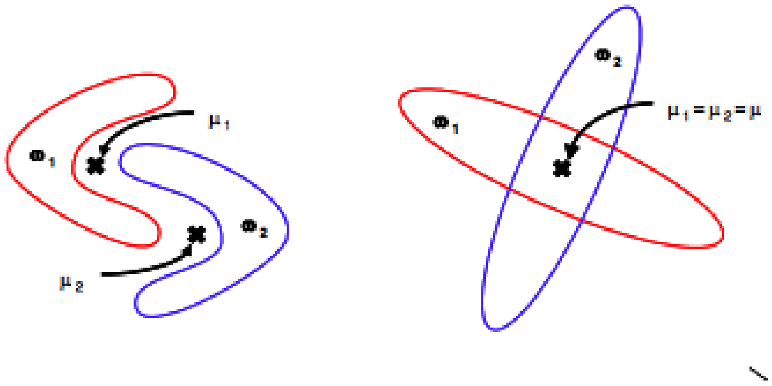
- Files:
  - *LDAvsPCA.py* : main file to run
    - At the beginning plots a few combinations of the features
  - *LDAImp.py* : fill in the code

Given data X and labels Y

1. Compute :
   a) S_w
   b) S_B
   c) $T = {S_w}^{-1} S_B$
2. *E,V=eig(T)*
3. *Keep the **K-1** (K =number of Classes) largest eigenvectors* $W = E_k$
4. *Project the data on W :* X_{LDA}
   a) *Project the mean vectors of each class (centers of the class)*

Is LDA better?

- PCA does not care about the class attribute
  - But LDA produces a fixed number of features
    - Might not be enough
- What happens if the mean of the classes is the same as the total mean?



THANK YOU