AARHUS
UNIVERSITET

Computer Vision & Machine Learning

Alexandros Iosifidis
@
Department of Electrical and Computer Engineering
Aarhus University

This week

Introduction to ML

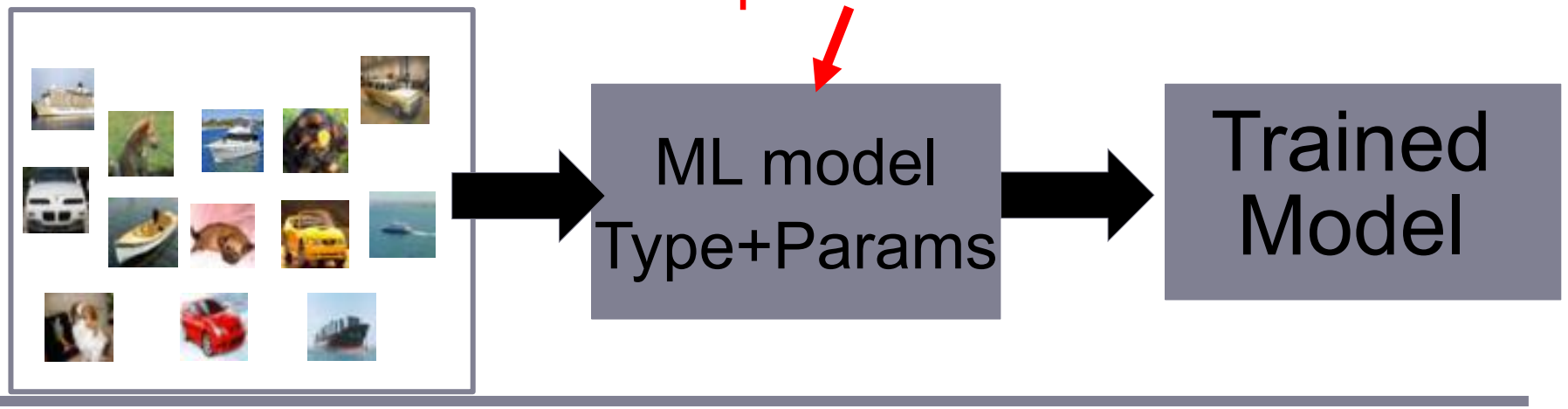
Data clustering

Subspace Learning

Multi-view Learning

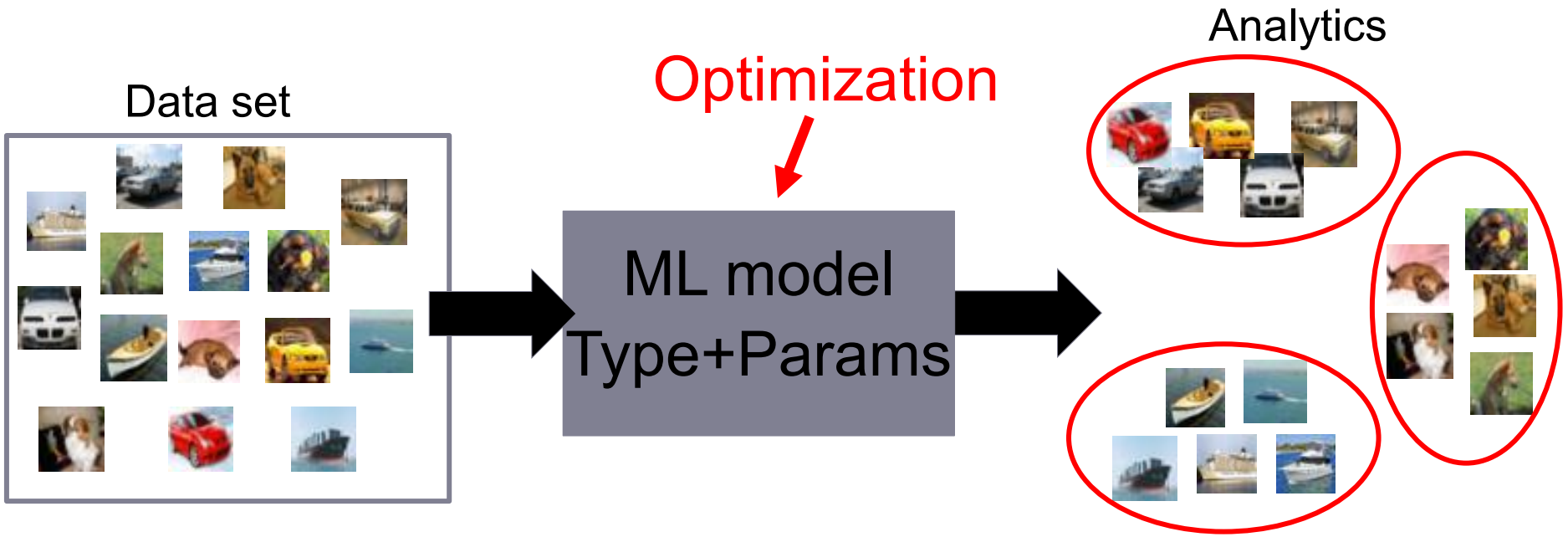
ML for out-of-sample analysis

Training phase



Test phase/Evaluation/Online process

ML for in-sample analysis



General processing steps

In order to apply a ML model we:

General processing steps

In order to apply a ML model we:

1. Identify the type of the problem:
    - Generic categories:
        - Unsupervised learning: We only have data!
        - Supervised learning: We have data and expert-given information (e.g. labels)

General processing steps

In order to apply a ML model we:

1. Identify the type of the problem:
   - Generic categories:
     - Unsupervised learning: We only have data!
     - Supervised learning: We have data and expert-given information (e.g. labels)
   - More specific categories:
     - Clustering (determination of groups of items)
     - Dimensionality reduction (mapping from $\mathbb{R}^D$ to $\mathbb{R}^d$, d<D)
     - Regression (mapping from x ϵ $\mathbb{R}^D$ to y ϵ $\mathbb{R}^C$)
     - Classification (mapping from x ϵ $\mathbb{R}^D$ to y ϵ Z)

General processing steps

In order to apply a ML model we:

1. Identify the type of the problem
2. Define the “nature” of the problem:
   - Linear → the use of linear solutions, like lines and (hyper-)planes, can lead to good result
   - Non-linear → a good solution can be obtained by using nonlinear methods (defining e.g. parabolic solutions)

General processing steps

In order to apply a ML model we:

1. Identify the type of the problem
2. Define the “nature” of the problem
3. Other constraints:
   - Processing time limitations (e.g. for real-time operation)
   - Processing power limitations (e.g. for smart-phone apps)
   - Possible dependences on other software/hardware

General processing steps

In order to apply a ML model we:

1. Identify the type of the problem
2. Define the “nature” of the problem
3. Other constraints
4. Select the ML model to be used:
   - K-Means
   - Principal Component Analysis (PCA)
   - Linear Discriminant Analysis (LDA)
   - Support Vector Machine (SVM)
   - Multilayer Perceptron (MLP)
   - Convolutional Neural Network (CNN)
   - …

General processing steps

In order to apply a ML model we:

1. Identify the type of the problem
2. Define the “nature” of the problem
3. Other constraints
4. Select the ML model to be used
5. Find the data for the learning (or otherwise the optimization of the parameters of the ML model)

General processing steps

In order to apply a ML model we:

1. Identify the type of the problem
2. Define the “nature” of the problem:
3. Other constraints
4. Select the ML model to be used
5. Find the data for the learning (or otherwise the optimization of the parameters of the ML model)
6. Apply the optimization

General processing steps

ML models are (most often) defined as problems trying to optimize an objective (otherwise called criterion)

This is done by:

- Finding exact (and unique) solutions → Convex optimization
  - For example: Linear Regression, Principal Component Analysis, etc.
  - The criteria of such models have a unique extreme (maximum/minimum) point w.r.t. their parameters

General processing steps

ML models are (most often) defined as problems trying to optimize an objective (otherwise called criterion)

This is done by:

- Finding exact (and unique) solutions → Convex optimization
- Finding *good enough* solutions (usually through iterative processes):
  - For example: K-Means, Neural networks, etc.
  - The criteria of these models have many extreme points w.r.t. their parameters
  - We are satisfied by finding a *local minimum/maximum*

General processing steps

ML models are (most often) defined as problems trying to optimize an objective (otherwise called criterion)

This is done by:

- Finding exact (and unique) solutions → Convex optimization
- Finding *good enough* solutions (usually through iterative processes):
  - For example: K-Means, Neural networks, etc.
  - The criteria of these models have many extreme points w.r.t. their parameters
  - We are satisfied by finding a *local minimum/maximum*

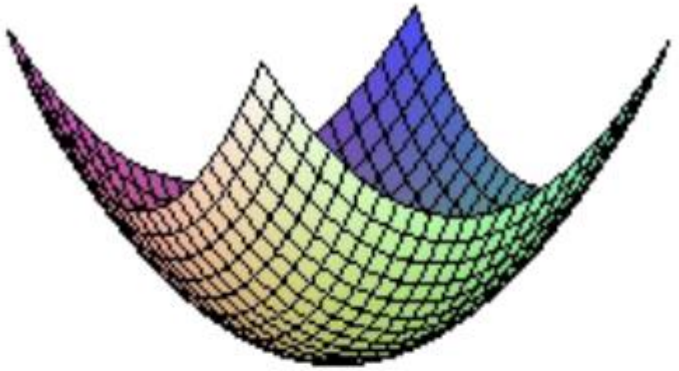
Optimization usually involves:

- Linear Algebra properties/equalities/expressions
- Calculus (differentiation, etc.)

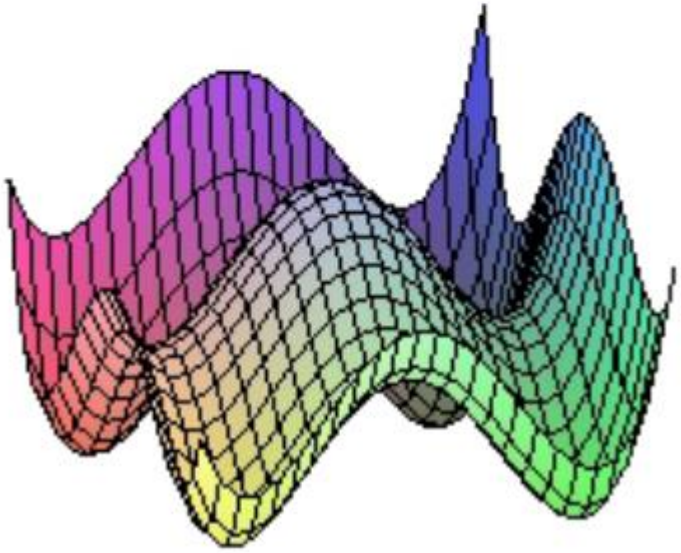
We will discuss how each model is optimized, along with its description

General processing steps

Convex criterion of two parameters



Non-convex criterion of two parameters



K-Means clustering

Unsupervised method:

- Available information is only the samples' representations x_i
- Goal: Define K groups of similar vectors

Mean vector of k-th

Criterion:

$$\min_{C,\{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i)=k} ||x_i - m_k||^2$$

Euclidean distance between two vectors:

$$d(x_i, x_{i'}) = \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 = ||x_i - x_{i'}||^2$$

K-Means clustering

Iterative optimization (non-convex optimization problem)

Algorithm 14.1 *K-means Clustering.*

1. For a given cluster assignment C, the total cluster variance (14.33) is minimized with respect to $\{m_1, \ldots, m_K\}$ yielding the means of the currently assigned clusters (14.32).

2. Given a current set of means $\{m_1, \ldots, m_K\}$, (14.33) is minimized by assigning each observation to the closest (current) cluster mean. That is,

$$C(i) = \underset{1 \le k \le K}{\operatorname{argmin}} ||x_i - m_k||^2. \qquad (14.34)$$

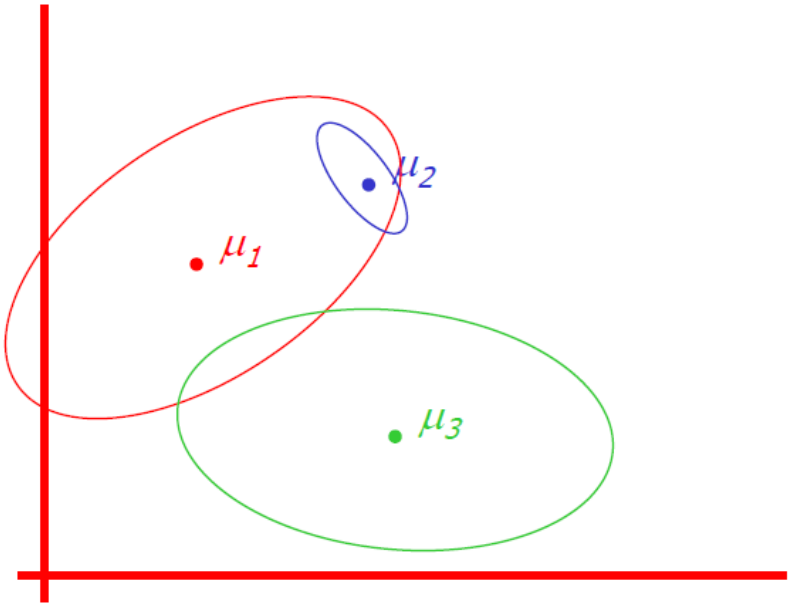
3. Steps 1 and 2 are iterated until the assignments do not change.

Demo

Mixture of Gaussians

K-Means assumes that all groups are represented by the corresponding mean vector m_k and have the same variance:

- This means that a sample x_i will be assigned to the group corresponding to the closest vector m_k, irrespectively to the *shape* of the groups
- An approach which uses such group shape information is the Gaussian Mixture Model (GMM)

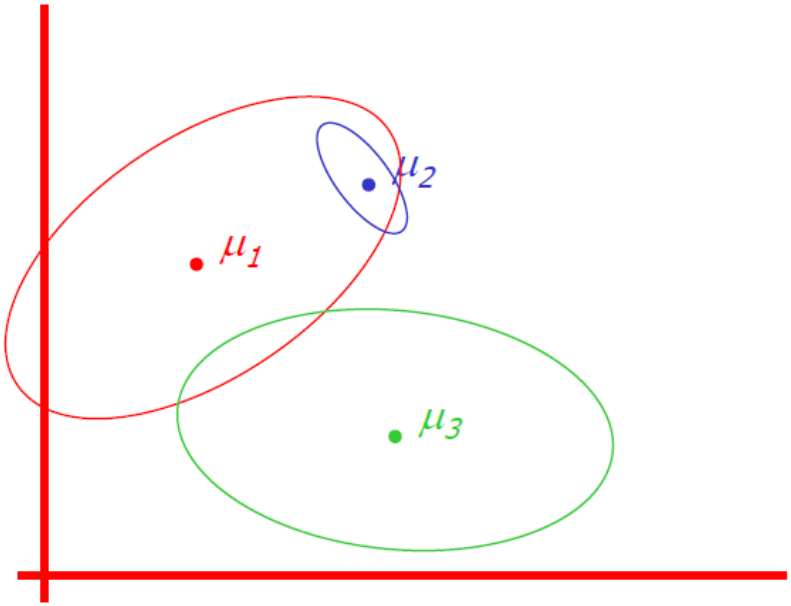


Mixture of Gaussians

GMM:

- There are K components/clusters
- The k-th cluster is associated with a vector m_k (mean vector)
- Each cluster generates data from a Gaussian with mean m_k and covariance Σ_k

Thus, in addition to defining the mean vector m_k as done in K-Means, we also need to determine the covariance Σ_k for each cluster.



Iterative process formed by two steps:

E-step: Compute the expected cluster label for all data

M-step: Update the mean vectors m_k and the covariance matrices Σ_k, k=1,…,K

K-Medoids clustering

K-Means uses the Euclidean distance function. By using different types of distances, it is extended to the K-medoids algorithm:

Algorithm 14.2 *K-medoids Clustering.*

1. For a given cluster assignment C find the observation in the cluster minimizing total distance to other points in that cluster:

$$i_k^* = \underset{\{i:C(i)=k\}}{\operatorname{argmin}} \sum_{C(i')=k} D(x_i, x_{i'}). \tag{14.35}$$

   Then $m_k = x_{i_k^*}$, $k = 1, 2, \ldots, K$ are the current estimates of the cluster centers.

2. Given a current set of cluster centers $\{m_1, \ldots, m_K\}$, minimize the total error by assigning each observation to the closest (current) cluster center:

$$C(i) = \underset{1 \le k \le K}{\operatorname{argmin}} D(x_i, m_k). \tag{14.36}$$

3. Iterate steps 1 and 2 until the assignments do not change.

Hierarchical clustering

The results of applying K-means or K-medoids clustering algorithms depend on the choice for the number of clusters to be searched and a starting configuration assignment.

In contrast, hierarchical clustering methods require the user to specify a measure of dissimilarity between (disjoint) groups of observations, based on the pairwise dissimilarities among the observations in the two groups.

Strategies:

- agglomerative (bottom-up): start at the bottom and at each level recursively merge a selected pair of clusters into a single cluster.
- divisive (top-down): start at the top and at each level recursively split one of the existing clusters at that level into two new clusters.

Both strategies lead to a (N-1)-level hierarchy

Agglomerative clustering

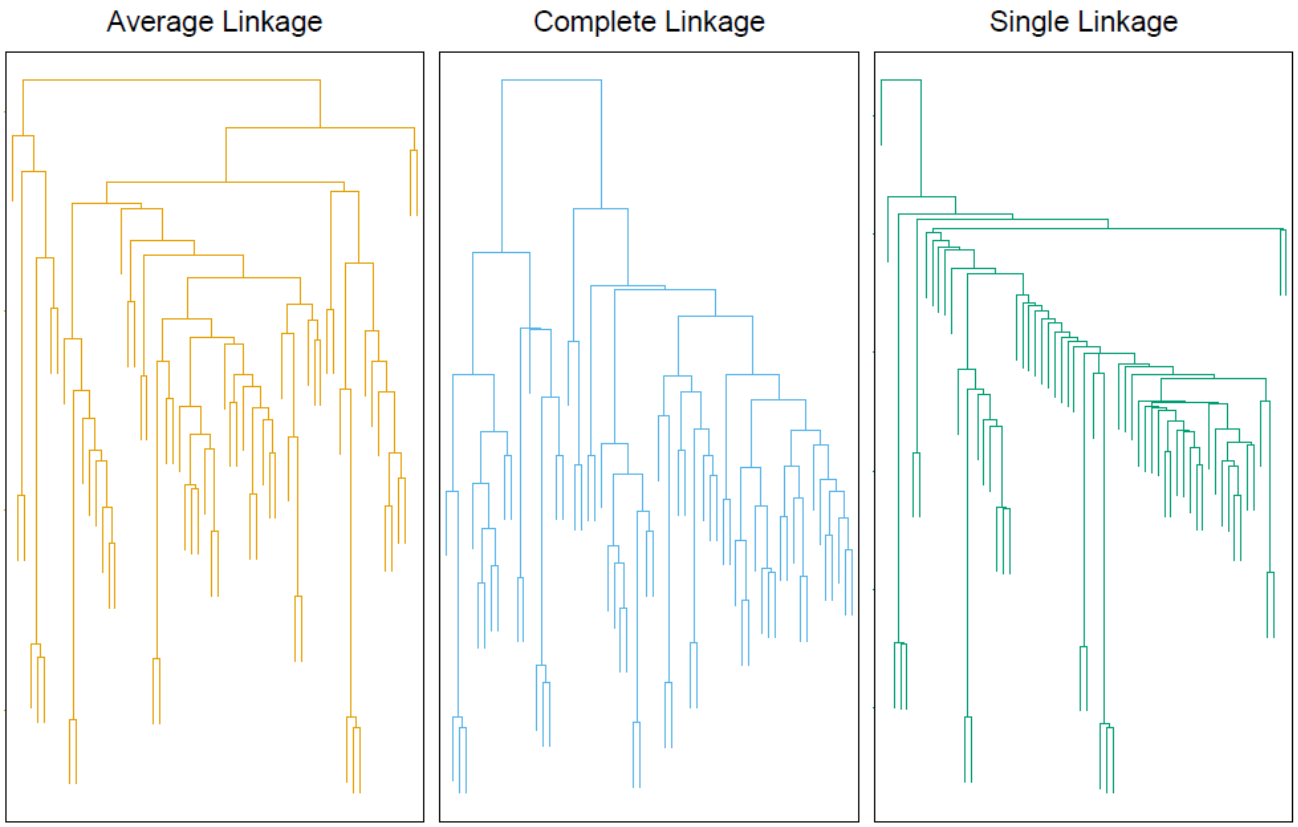
General process:

- Begin with every observation representing a singleton cluster
- At each of the N −1 steps, the closest two (least dissimilar) clusters are merged into a single cluster, producing one less cluster at the next higher level

Dissimilarity between groups G and H:

- Single linkage (or nearest-neighbor) approach: $d_{SL}(G,H) = \min_{\substack{i\in G \\ i'\in H}} d_{ii'}$
- Complete linkage (or furthest-neighbor) approach: $d_{CL}(G,H) = \max_{\substack{i\in G \\ i'\in H}} d_{ii'}$
- Group average approach: $d_{GA}(G,H) = \frac{1}{N_G N_H} \sum_{i\in G} \sum_{i'\in H} d_{ii'}$

Agglomerative clustering



Diameter D_G of a group of observations: the largest dissimilarity among its members:

$$D_G = \max_{\substack{i \in G \\ i' \in G}} d_{ii'}$$

SL → large D_G
CL → small D_G
AL → somewhere between SL-CL

FIGURE 14.13. *Dendrograms from agglomerative hierarchical clustering of human tumor microarray data.*

Divisive clustering

General process:

- Begin with the entire data set as single cluster
- Recursively divide one of the existing clusters into two clusters at each iteration (e.g. by applying K-Means with K=2)

Alternative method (not having randomness issues):

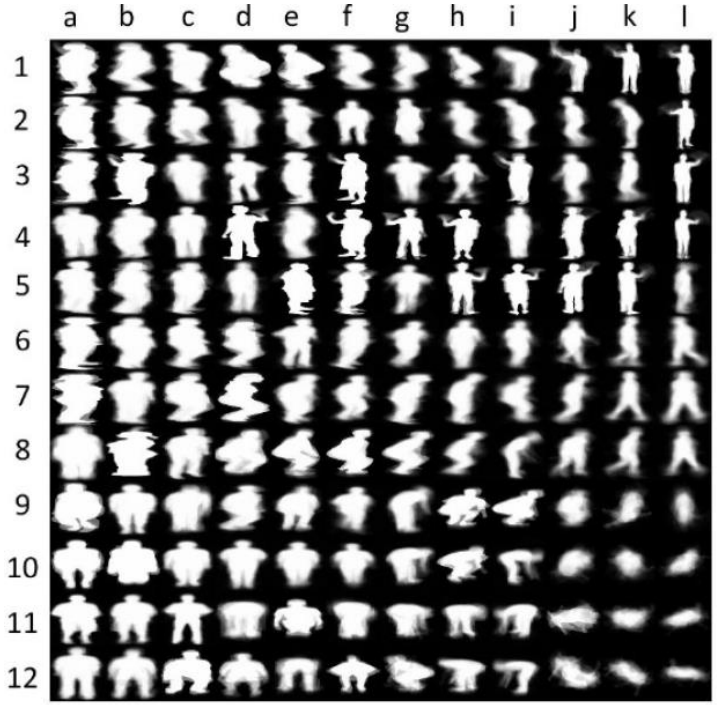
- Place all observations in a single cluster G
- Choose the sample whose average dissimilarity from all the other samples is largest. This is the first member of a second cluster H
- At each successive step the sample in G whose average distance from those in H, minus that for the remaining samples in G is largest, is transferred to H
- Stop when the corresponding difference in averages becomes negative.

Self-Organizing Map

Instead of clustering the input vectors, we want to learn vectors m_k, k=1,…,K that can represent the data well.

These vectors are constrained to belong to a topological map (e.g. a line or a HxW lattice).

A lattice of 12x12 prototypes
learnt using human body poses



Self-Organizing Map

Learning process:

- Initialization of the prototypes (e.g. as random vectors, or random samples)
- Use the input vectors x_i, i=1,…,N multiple times (epochs) for updating the prototypes
- At each iteration, a vector x_i is used to update the prototypes m_k:
  - find the closest (e.g. using Euclidean distance) prototype m_j to x_i
  - Update all prototypes m_k using

$$m_k \leftarrow m_k + \alpha h(\|\ell_j - \ell_k\|)(x_i - m_k)$$

h(·) is a decreasing function operating on the lattice coordinates of prototypes

Effect: For each sample x_i, the closest to it prototypes are updated to be more similar to it according to their similarities in the lattice.

Dimensionality Reduction

Goal: define a mapping from $\mathbb{R}^D$ to $\mathbb{R}^d$, where d < D

This mapping is defined using a set of data $x_i \in \mathbb{R}^D$, i=1,…,N and can be:

- Unsupervised
  - only the vectors x_i are known
  - the objective/criterion defining the mapping involves (geometric) relationships between the data (e.g. variance of the data)
- Supervised
  - vectors x_i are accompanied with class labels l_i
  - the objective/criterion defining the mapping involves class discrimination

Principal Component Analysis

Goal: define a mapping from $\mathbb{R}^D$ to $\mathbb{R}^d$, where d < D, such that the data in $\mathbb{R}^d$ will have the maximal variance.

This mapping is also called projection and for linear mappings is expressed using a matrix W ϵ $\mathbb{R}^{d \times D}$

Principal Component Analysis

Process:

- We define the data representations in $\mathbb{R}^d$ $\quad \mathbf{y}_i = \mathbf{W}^T \mathbf{x}_i$

- We define the variance using the representations y_i , i=1,…,N

$$\mathbf{S}_T = \sum_{i=1}^{N} (\mathbf{y}_i - \mathbf{m})(\mathbf{y}_i - \mathbf{m})^T = \sum_{i=1}^{N} \bar{\mathbf{y}}_i \bar{\mathbf{y}}_i^T = \bar{\mathbf{Y}} \bar{\mathbf{Y}}^T$$

- We express S_T as a function of the parameters **W**

$$\begin{aligned} \mathbf{S}_T &= \sum_{i=1}^{N} \left[\mathbf{W}^T (\mathbf{x}_i - \boldsymbol{\mu}) \right] \left[\mathbf{W}^T (\mathbf{x}_i - \boldsymbol{\mu}) \right]^T = \sum_{i=1}^{N} \mathbf{W}^T \bar{\mathbf{x}}_i \bar{\mathbf{x}}_i^T \mathbf{W} \\ &= \mathbf{W}^T \bar{\mathbf{X}} \bar{\mathbf{X}}^T \mathbf{W} = \mathbf{W}^T \tilde{\mathbf{S}}_T \mathbf{W}, \end{aligned}$$

- We optimize the following criterion $\quad \mathbf{W}^* = \arg\max \; Tr\left(\mathbf{W}^T \tilde{\mathbf{S}}_T \mathbf{W} \right)$

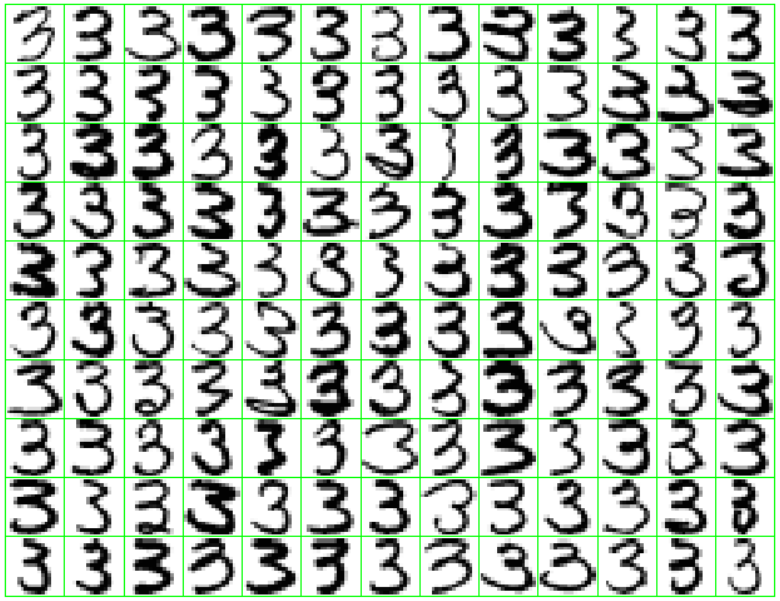
$$subject\ to \quad : \quad \mathbf{W}^T \mathbf{W} = \mathbf{I},$$

Principal Component Analysis

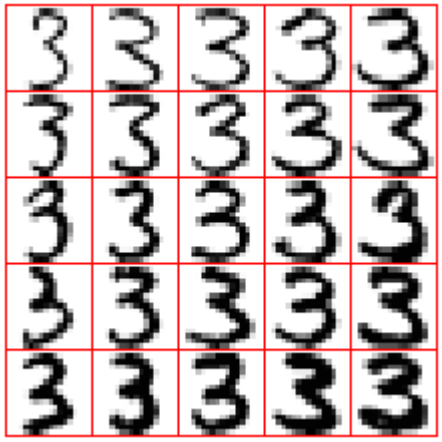
The matrix W is obtained by applying eigenanalysis to the matrix S_T

The same solution can be also obtained by applying Singular Value Decomposition to the matrix $\bar{X} = USV^T$ and keeping the right singular vectors V.

Input data



Principal Components



Demo

$3 = \bar{3} + \lambda_1 \cdot$  $+ \lambda_2 \cdot$  $+ \ldots$

Non-negative Matrix Factorization

NMF is useful for modeling non-negative data, such as images.

It is obtained by calculating a factorization of the matrix X ϵ $\mathbb{R}^{N \times D}$ as follows:

$$\mathbf{X} \approx \mathbf{WH}$$

The matrices W and H are found by minimizing:

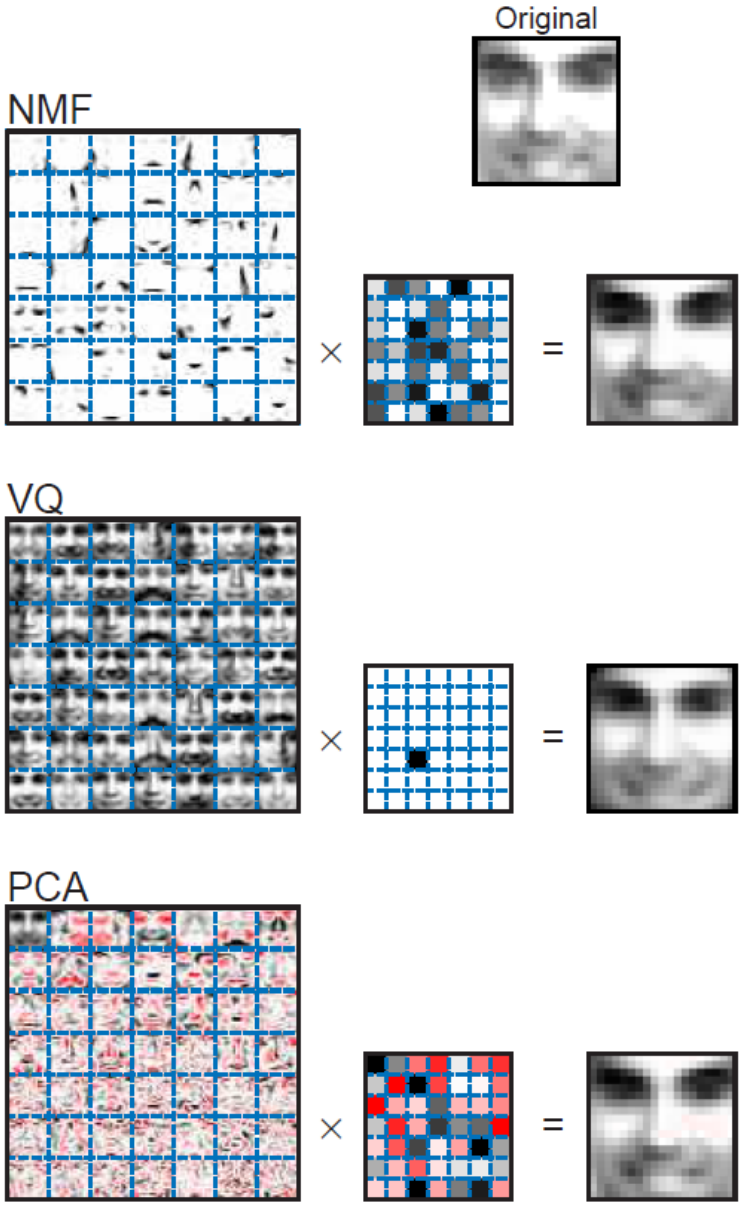
$$L(\mathbf{W}, \mathbf{H}) = \sum_{i=1}^{N} \sum_{j=1}^{p} [x_{ij} \log(\mathbf{WH})_{ij} - (\mathbf{WH})_{ij}]$$

Iterative process formed by two steps:

- Keeping H fixed, update W as $w_{ik} \leftarrow w_{ik} \frac{\sum_{j=1}^{p} h_{kj} x_{ij} / (\mathbf{WH})_{ij}}{\sum_{j=1}^{p} h_{kj}}$

- Keeping W fixed, update H as $h_{kj} \leftarrow h_{kj} \frac{\sum_{i=1}^{N} w_{ik} x_{ij} / (\mathbf{WH})_{ij}}{\sum_{i=1}^{N} w_{ik}}$

Examples

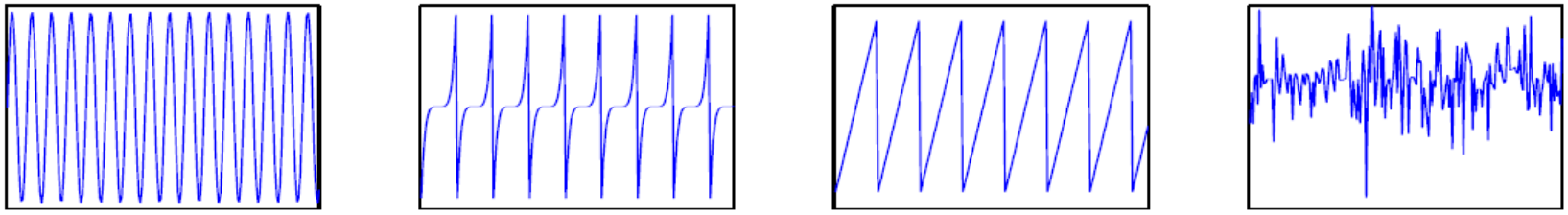


Independent Component Analysis

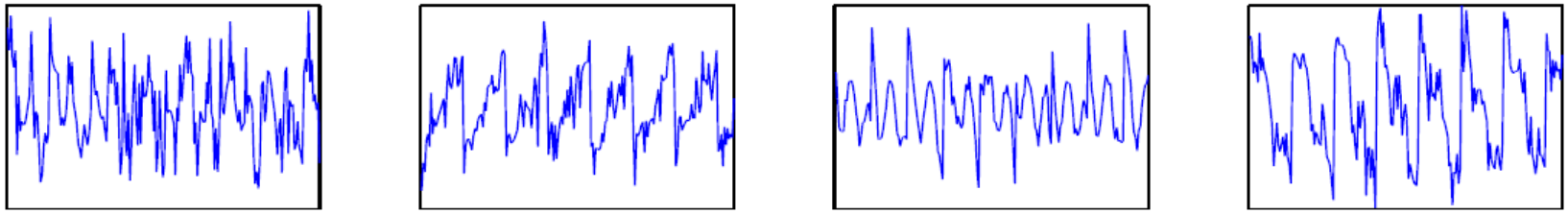
Some motivation first!

Blind source separation problem:

- There is a number of “source signals”:



- Due to some reason, we observe only linear mixtures of the above signals:



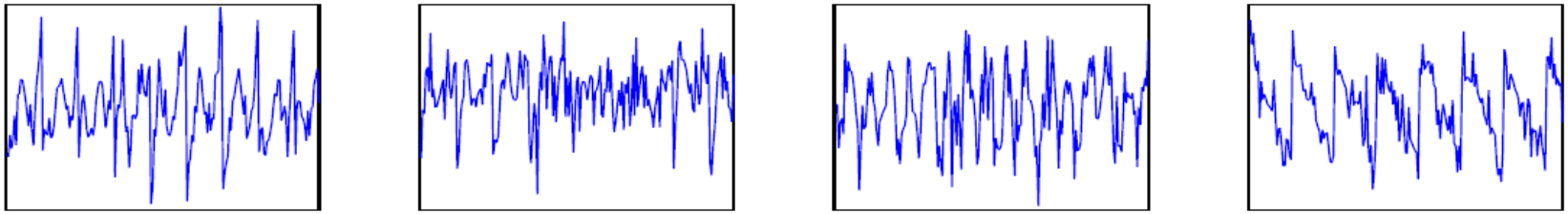
- We want to estimate (separate) the original signals, based on the observations

Independent Component Analysis

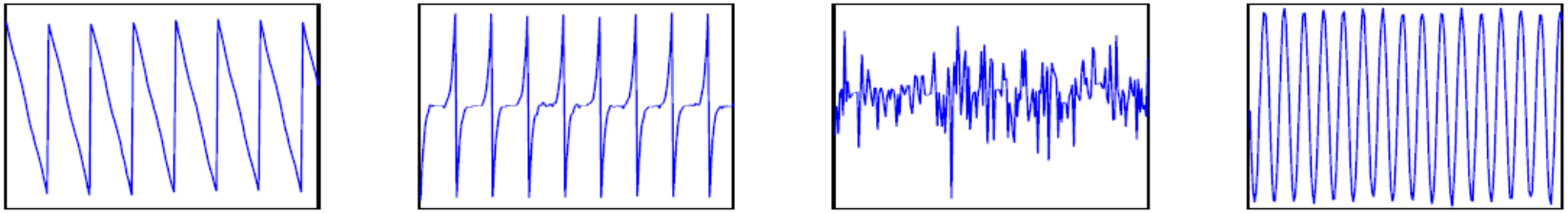
Some motivation first!

Blind source separation problem:

- PCA (optimized to find components with maximal variance) fails:



- If we use information on statistical independence we can recover:



Independent Component Analysis

The observation matrix X is modelled hidden variables s:

$$\mathbf{X} = \mathbf{AS}$$

where:

- the matrix A is called “mixing matrix”
- the vectors forming the matrix S are called “independent components”

Assumptions and restrictions:

- The number of Independent Components is equal to the number of observations
  → A is a square matrix. Then, we can use $S = A^{-1} X = W^T X$
- s_i are statistically independent → $p(s_1,s_2) = p(s_1)p(s_2)$
- Since both A and S are not known, any arbitrary scaling of the first can be cancelled by inverting it to the second → we restrict $||s_i||_2 = 1$.

Independent Component Analysis

Pre-processing steps:

1. Data centering: $x_i \leftarrow x_i - \mu$
2. Data whitening: $S_T = U\Sigma U^T$ and $x_i = (U\Sigma^{-1/2}U^T)\, x_i$

$$Obj(\mathbf{W}) = \sum_{t=1}^{T} G(\mathbf{W}^T \mathbf{x}_t) - \Lambda(\mathbf{W}^T\mathbf{W} - \mathbf{I})$$

$$\frac{\partial Obj}{\partial \mathbf{W}} = \mathbf{X}g(\mathbf{W}^T\mathbf{X})^T - \Lambda\mathbf{W} = \mathbf{0}$$

Fixed Point Algorithm

Input: $\mathbf{X}$

Random init of $\mathbf{W}$

Iterate until convergence:

$$\mathbf{S} = \mathbf{W}^T\mathbf{X}$$

$$\mathbf{W} = \mathbf{X}g(\mathbf{S})^T$$

$$\mathbf{W} = \mathbf{W}\sqrt{(\mathbf{W}^T\mathbf{W})^{-1}}$$

Output: $\mathbf{W}$, $\mathbf{S}$

Independent Component Analysis

Criterion: Find the A that leads to the most independence between the components of $Y = W^T X$.

We use the mutual information I(Y) : $$I(Y) = \sum_{j=1}^{p} H(Y_j) - H(Y)$$

Where H(Y) is the differential entropy of Y with density g(y)

$$H(Y) = -\int g(y) \log g(y) dy$$

Random Projections

The transformation y = W x, when W ϵ $\mathbb{R}^{d \times D}$ is a random matrix is referred to as RP

It has been shown that for a random matrix W such that each element of it is distributed normally with zero mean and variance 1/d, the ratio

$$\frac{\|W\mathbf{x}_1 - W\mathbf{x}_2\|}{\|\mathbf{x}_1 - \mathbf{x}_2\|}$$

is close to 1.

The above means that the pair-wise distances between vectors in $\mathbb{R}^D$ and $\mathbb{R}^d$ are very similar

Linear Discriminant Analysis

Supervised method:

- Each vector x_i is followed by a class label $l_i \in \{1,\ldots,C\}$

Goal: Define a mapping $y_i = W^T x_i$, when $W \in \mathbb{R}^{d \times D}$, enhancing class discrimination:

- increasing class compactness == minimizing within-class variance
- increasing distance between classes == increasing between-class variance

Demo

Linear Discriminant Analysis

Process:

- Define within-class variance (μ_k is the mean vector of k-th class)

$$\mathbf{S}_w = \sum_{k=1}^{K} \sum_{i,l_i=k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T$$

- Define between-class variance

$$\mathbf{S}_b = \sum_{k=1}^{K} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T$$

- Maximize the Fisher ratio

$$\mathcal{J}(\mathbf{W}) = \frac{Tr(\mathbf{W}^T \mathbf{S}_b \mathbf{W})}{Tr(\mathbf{W}^T \mathbf{S}_w \mathbf{W})}$$

Several solutions for J(W) have been proposed. The most common one is the solution of the generalized eigen-value problem

$$\mathbf{S}_b \mathbf{w} = \lambda \mathbf{S}_w \mathbf{w}$$

Graph-based Subspace Learning

Let us consider a graph G = {V,E}:

- V = $\{v_1,\ldots,v_N\}$ is called the vertex set of the graph
- E = $\{e_{ij}\}$ is called the edge set of the graph
  - e_{ij} is an edge connecting v_i and v_j
  - e_{ij} can indicate a link between the i-th and j-th vertices (unweighted graph)
  - e_{ij} can be associated with a weight value w_{ij} showing how strong is the link between the i-th and j-th vertices (weighted graph)

Graph-based Subspace Learning

Let us consider a graph G = {V,E}:

- V = $\{v_1,\ldots,v_N\}$ is called the vertex set of the graph
- E = $\{e_{ij}\}$ is called the edge set of the graph
  - e_{ij} is an edge connecting v_i and v_j
  - e_{ij} can indicate a link between the i-th and j-th vertices (unweighted graph)
  - e_{ij} can be associated with a weight value w_{ij} showing how strong is the link between the i-th and j-th vertices (weighted graph)

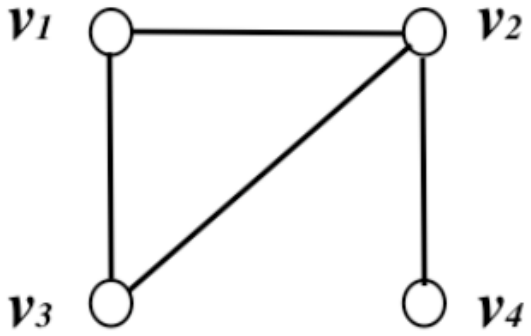
The graph can be:

- fully connected → there is an edge between every pair of nodes (and the associated weight)
- partially connected → each vertex is connected only with some other vertices (those which are more similar to it – according to a similarity/distance metric)

Graph-based Subspace Learning

The adjacency matrix A ϵ $\mathbb{R}^{N \times N}$ of the graph is defined as:

$$\mathbf{A} := \begin{cases} A_{ij} = 1 & \text{if there is an edge } e_{ij} \\ A_{ij} = 0 & \text{if there is no edge} \\ A_{ii} = 0 \end{cases}$$



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The weight matrix W ϵ $\mathbb{R}^{N \times N}$ of the graph has the same form, but each element denotes the strength of connection (similarity) between the corresponding vertices.

Graph-based Subspace Learning

Let us associate a feature vector $f(v_i)$ to each vertex i of an undirected weighted graph:

- We want the relation between the vectors representing the graph vertices to be the same as those expressed by the corresponding weights w_{ij}
- Example of graph weights (Gaussian kernel):

$$w_{ij} = \exp\left(-\|\boldsymbol{v}_i - \boldsymbol{v}_j\|^2/\sigma^2\right)$$

$$0 \leq w_{\min} \leq w_{ij} \leq w_{\max} \leq 1$$

this means that if w_{ij} is high, the representations of vertices i and j should be close using the Euclidean distance, i.e. the vertex representations to minimize:

$$\frac{1}{2}\sum_{e_{ij}} w_{ij}(f(v_i) - f(v_j))^2 = \boldsymbol{f}^\top \mathbf{L} \boldsymbol{f}$$

Graph-based Subspace Learning

$$\frac{1}{2}\sum_{e_{ij}} w_{ij}(f(v_i) - f(v_j))^2 = \boldsymbol{f}^\top \mathbf{L}\boldsymbol{f}$$

L is the Laplacian matrix: L = D – W

D is a diagonal matrix (called Degree matrix) having elements $D_{ii} = \sum_j w_{ij}$

Laplacian Embedding

Define low-dimensional representations f by optimizing:

$$\arg\min_{\boldsymbol{f}} \boldsymbol{f}^\top \mathbf{L}\boldsymbol{f} \text{ with: } \boldsymbol{f}^\top \boldsymbol{f} = 1 \text{ and } \boldsymbol{f}^\top \mathbf{1} = 0$$

Solution is obtained by solving the eigenanalysis problem $\mathbf{L}\boldsymbol{f} = \lambda \boldsymbol{f}$

Graph Embedding

GE defines two graphs:

- an (intrinsic) graph G expressing properties of the data that we want the embedding to enhance/minimize
- a penalty graph G^p, expressing properties of the data that we want the embedding to penalize/maximize

Graph Embedding

GE defines two graphs:

- an (intrinsic) graph G expressing properties of the data that we want the embedding to enhance/minimize
- a penalty graph G^p, expressing properties of the data that we want the embedding to penalize/maximize

Let us assume that L, B are the Laplacian matrices of G and G^p, respectively. Then, GE optimizes the following criterion:

$$y^* = \arg \min_{y^T B y = d} \sum_{i \neq j} \|y_i - y_j\|^2 W_{ij} = \arg \min_{y^T B y = d} y^T L y$$

y_i is the vector associated with vertex i and d is a constant value

Graph Embedding

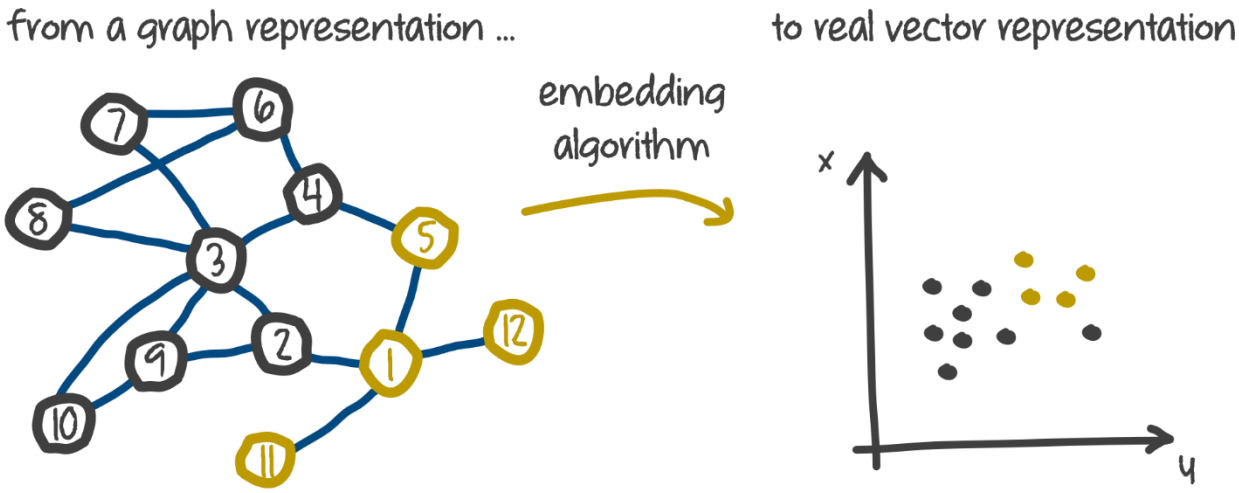
If we define a linear mapping: $y_i = w^T x_i$, then the projection w is obtained by:

$$w^* = \underset{\substack{w^T XBX^T w=d \\ \text{or } w^T w=d}}{\arg\min} \sum_{i \neq j} \|w^T x_i - w^T x_j\|^2 W_{ij} = \underset{\substack{w^T XBX^T w=d \\ \text{or } w^T w=d}}{\arg\min} w^T XLX^T w$$

And is calculated by solving the generalized eigenanalysis problem: $\tilde{L}v = \lambda \tilde{B}v$

where $\tilde{L} = XLX^T$

$\tilde{B} = XBX^T$



Graph Embedding

Let us take again a look at the scatter matrices of LDA:

$$\mathbf{S}_w = \sum_{k=1}^{K} \sum_{i,l_i=k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T$$

Using $\mathbf{1}_k \in \mathbb{R}^N$ a vector having its elements for which $l_i = k$ equal to one and the remaining elements equal to zero and $\mathbf{J}_k \in \mathbb{R}^{N \times N}$ the diagonal matrix of 1_k

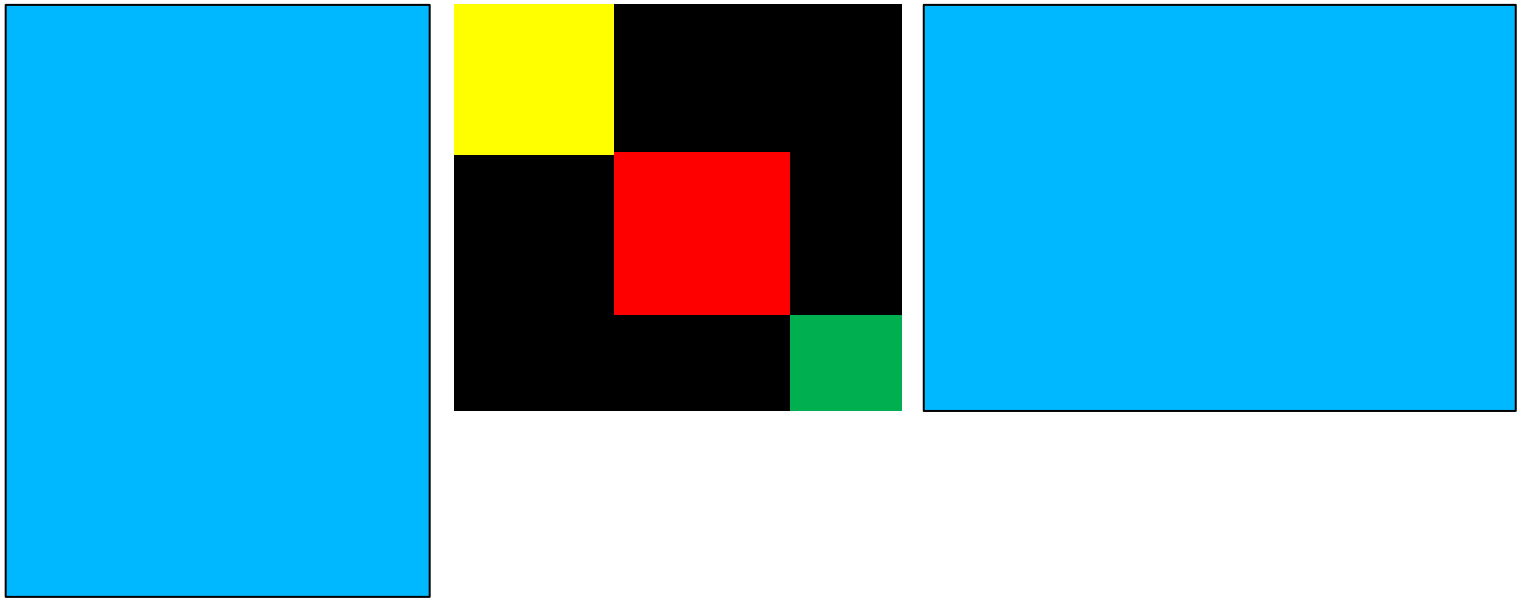
$$\begin{aligned}
\mathbf{S}_w &= \sum_{k=1}^{K} \left(\mathbf{X}\mathbf{J}_k - \frac{1}{N_k}\mathbf{X}\mathbf{1}_k\mathbf{1}_k^T \right) \left(\mathbf{X}\mathbf{J}_k - \frac{1}{N_k}\mathbf{X}\mathbf{1}_k\mathbf{1}_k^T \right)^T \\
&= \sum_{k=1}^{K} \mathbf{X} \left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T \right) \left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T \right)^T \mathbf{X}^T \\
&= \mathbf{X} \left(\sum_{k=1}^{K} \left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T \right) \left(\mathbf{J}_k - \frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T \right)^T \right) \mathbf{X}^T \\
&= \mathbf{X}\mathbf{L}_w\mathbf{X}^T.
\end{aligned}$$

Graph Embedding

Let us take again a look at the scatter matrices of LDA:

$$\mathbf{S}_w = \sum_{k=1}^{K} \sum_{i,l_i=k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T$$

$\mathbf{S}_w =$ $\mathbf{X}$ $\mathbf{L}_w$ $\mathbf{X}^T$



Graph Embedding

Similarly:

$$\mathbf{S}_b = \sum_{k=1}^{K} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T$$

$$\begin{aligned}
\mathbf{S}_b &= \sum_{k=1}^{K} N_k \left(\frac{1}{N_k}\mathbf{X}\mathbf{1}_k\mathbf{1}_k^T - \frac{1}{N}\mathbf{X}\mathbf{1}\mathbf{1}_k^T\right)\left(\frac{1}{N_k}\mathbf{X}\mathbf{1}_k\mathbf{1}_k^T - \frac{1}{N}\mathbf{X}\mathbf{1}\mathbf{1}_k^T\right)^T \\
&= \mathbf{X}\left(\sum_{k=1}^{K} N_k \left(\frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T - \frac{1}{N}\mathbf{1}\mathbf{1}_k^T\right)\left(\frac{1}{N_k}\mathbf{1}_k\mathbf{1}_k^T - \frac{1}{N}\mathbf{1}\mathbf{1}_k^T\right)^T\right)\mathbf{X}^T \\
&= \mathbf{X}\mathbf{L}_b\mathbf{X}^T.
\end{aligned}$$

and

$$\mathcal{J}(\mathbf{W}) = \frac{Tr(\mathbf{W}^T\left(\mathbf{X}\mathbf{L}_b\mathbf{X}^T\right)\mathbf{W})}{Tr(\mathbf{W}^T\left(\mathbf{X}\mathbf{L}_w\mathbf{X}^T\right)\mathbf{W})}$$

Graph Embedding

Similarly:

$$\mathbf{S}_b = \sum_{k=1}^{K} N_k(\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T$$

$\mathbf{S}_b$ = $\mathbf{X}$ $\mathbf{L}_b$ $\mathbf{X}^T$

Graph Embedding

And the criterion of LDA is given by:

$$\mathcal{J}(\mathbf{W}) = \frac{Tr(\mathbf{W}^T \left(\mathbf{X}\mathbf{L}_b\mathbf{X}^T\right)\mathbf{W})}{Tr(\mathbf{W}^T \left(\mathbf{X}\mathbf{L}_w\mathbf{X}^T\right)\mathbf{W})}$$

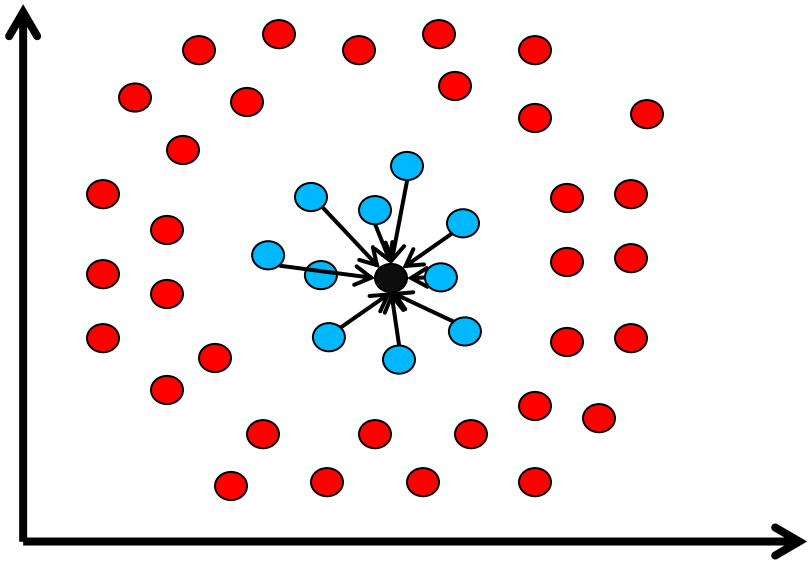
The above expression is simple, yet powerful:

- Use of different $L_{(?)}$ matrices leads to the description of different relationships for the data (and/or classes)
- Examples:
  - Linear Discriminant Analysis
  - Principal Component Analysis
  - Local Fisher Discriminant Analysis
  - Marginal Discriminant Analysis
  - Class-Specific Discriminant Analysis

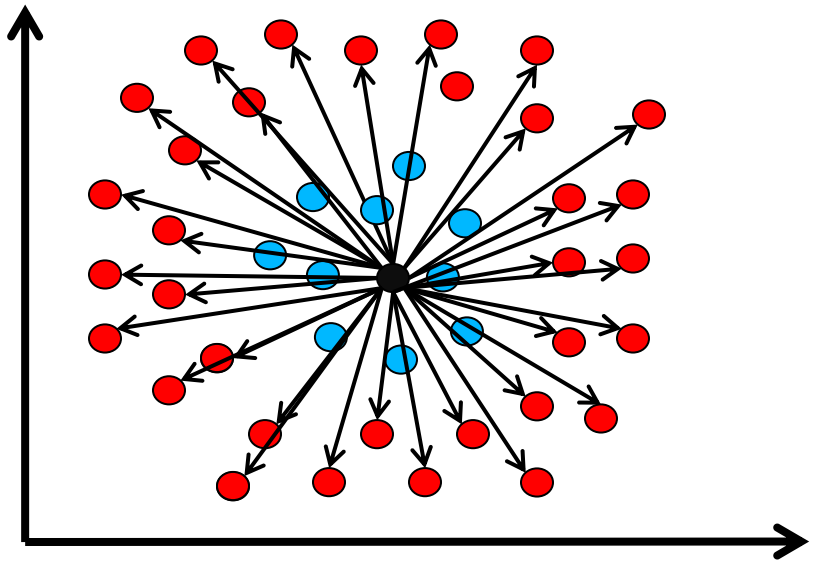
Graph Embedding

Example: Class-specific Discriminant Analysis

Intra-class scatter $\mathbf{S}_i$



Out-of-class scatter $\mathbf{S}_p$



Spectral Clustering

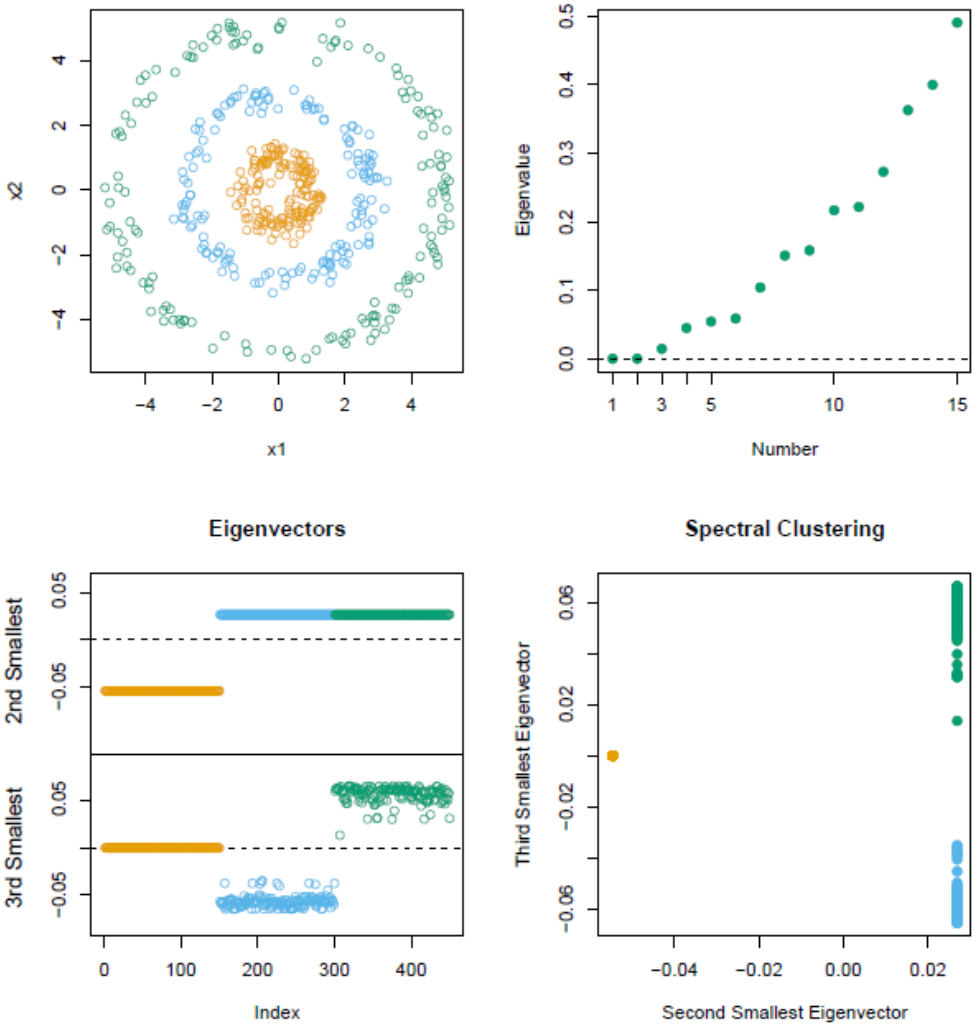
Spectral Clustering exploits a data transformation based on the spectrum (eigenanalysis) of a graph Laplacian matrix for applying non-linear clustering.

Process:

- Construct a fully connected graph G = {V,E,W} using $x_i \in \mathbb{R}^D$, i=1,..,N
- Calculate the graph Laplacian matrix L = D – W
- Apply eigenanalysis to L and keep the eigen-vectors corresponding to the d smallest eigenvalues, in order to form new d-dimensional data representations y_i.
- Apply a clustering algorithm (K-Means) using y_i.
- Assign the cluster labels to x_i.

Spectral Clustering

Example:



Multi-view Embedding

Until now we used one representation for each sample. However, there are cases where a sample can be represented based on different modalities/views (e.g. a video can be represented by a vector encoding visual information and another vector encoding sound information).

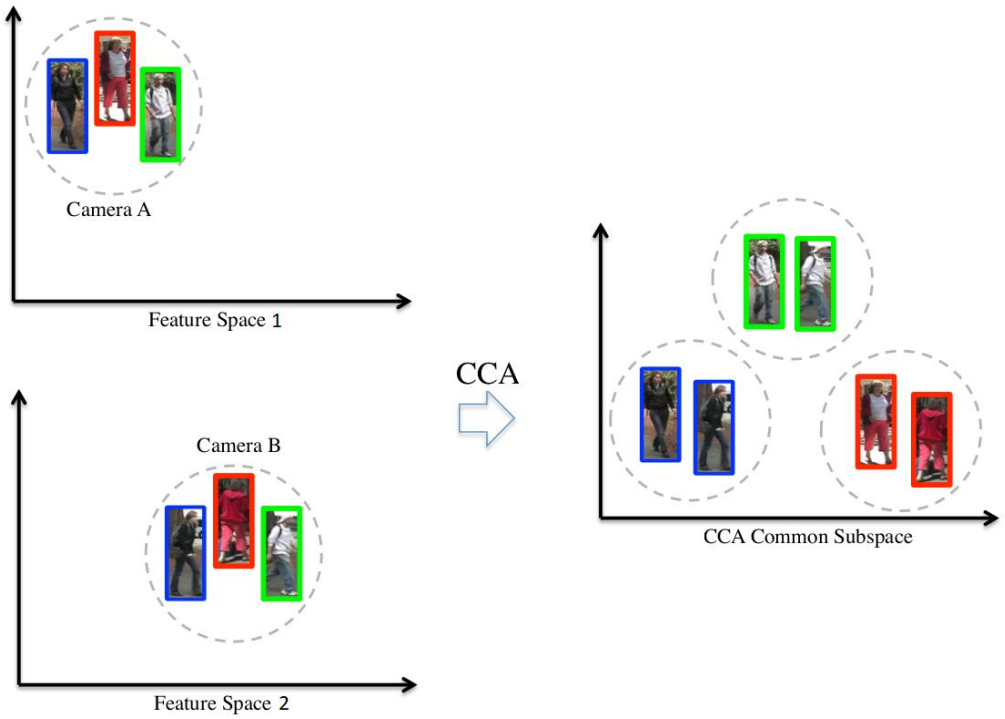
In Multi-view Embedding with V views:

- Each sample is represented by V vectors → V data matrices X_1, X_2, …, X_V
- We want to jointly use the information encoded in all V matrices to enhance performance

Multi-view Embedding

Canonical Correlation Analysis:

- Given two data views ($X_1 \in \mathbb{R}^{D_1 \times N}$ and $X_2 \in \mathbb{R}^{D_2 \times N}$) we want to determine data representations in a common (sub-)space $\mathbb{R}^d$ such that data correlation is maximized



Multi-view Embedding

Canonical Correlation Analysis:

- Given two data views ($X_1 \in \mathbb{R}^{D_1 \times N}$ and $X_2 \in \mathbb{R}^{D_2 \times N}$) we want to determine data representations in a common (sub-)space $\mathbb{R}^d$ such that data correlation is maximized
- For linear projections we have: $Y_1 = W_1^T X_1$ and $Y_2 = W_2^T X_2$
- $W_1 \in \mathbb{R}^{D_1 \times d}$ and $W_2 \in \mathbb{R}^{D_2 \times d}$ are obtained by optimizing:

$$\mathcal{J} = \arg\max_{\mathbf{W}_1, \mathbf{W}_2} \operatorname{corr}(\mathbf{W}_1^\top \mathbf{X}_1, \mathbf{W}_2^\top \mathbf{X}_2)$$

$$= \arg\max_{\mathbf{W}_1, \mathbf{W}_2} \frac{\mathbf{W}_1^\top \boldsymbol{\Sigma}_{12} \mathbf{W}_2}{\sqrt{\mathbf{W}_1^\top \boldsymbol{\Sigma}_{11} \mathbf{W}_1} \cdot \sqrt{\mathbf{W}_2^\top \boldsymbol{\Sigma}_{22} \mathbf{W}_2}}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} = \frac{1}{N} \begin{bmatrix} \bar{\mathbf{X}}_1 \bar{\mathbf{X}}_1^\top & \bar{\mathbf{X}}_1 \bar{\mathbf{X}}_2^\top \\ \bar{\mathbf{X}}_2 \bar{\mathbf{X}}_1^\top & \bar{\mathbf{X}}_2 \bar{\mathbf{X}}_2^\top \end{bmatrix}$$

centered data

Multi-view Embedding

Multi-view Discriminant Analysis:

- Define data projections $Y_1 = W_1^T X_1$ and $Y_2 = W_2^T X_2$ using a generalization of the within-class and between-class variance

$$\mathbf{S}_W^M = \sum_{i=1}^{V}\sum_{j=1}^{V}\mathbf{W}_i^\top\mathbf{X}_i\left(\mathbf{I} - \sum_{c=1}^{C}\frac{1}{N_c}\mathbf{e}_c\mathbf{e}_c{}^\top\right)\mathbf{X}_j^\top\mathbf{W}_j$$

$$\mathbf{S}_B^M = \sum_{i=1}^{V}\sum_{j=1}^{V}\mathbf{W}_i^\top\mathbf{X}_i\left(\sum_{c=1}^{C}\frac{1}{N_c}\mathbf{e}_c\mathbf{e}_c{}^\top - \frac{1}{N}\mathbf{e}\,\mathbf{e}^\top\right)\mathbf{X}_j^\top\mathbf{W}_j$$

- $W_1 \in \mathbb{R}^{D_1 \times d}$ and $W_2 \in \mathbb{R}^{D_2 \times d}$ are obtained by optimizing:

$$\mathcal{J} = \arg\max_{\mathbf{W}} \frac{\mathrm{Tr}(\mathbf{S}_B^M)}{\mathrm{Tr}(\mathbf{S}_W^M)}$$

Multi-view Embedding

Generalized Multi-view Embedding:

- Define two graphs G and G' with graph Laplacians L and L', respectively.
- Define the projections $Y_1 = W_1^T X_1$ and $Y_2 = W_2^T X_2$, where $W_1 \in \mathbb{R}^{D_1 \times d}$ and $W_2 \in \mathbb{R}^{D_2 \times d}$ are obtained by optimizing:

$$\mathcal{J} = \underset{\mathbf{W}^\top \mathbf{W} = \mathbf{I}}{\arg\max} \frac{\sum_{v=0}^{V} \sum_{i=0}^{N} \sum_{j=0}^{N} \mathbf{S}'_{vij} \|\mathbf{W}_v^\top \mathbf{X}_{vi} - \mathbf{W}_v^\top \mathbf{X}_{vj}\|^2}{\sum_{v=0}^{V} \sum_{i=0}^{N} \sum_{j=0}^{N} \mathbf{S}_{vij} \|\mathbf{W}_v^\top \mathbf{X}_{vi} - \mathbf{W}_v^\top \mathbf{X}_{vj}\|^2}$$

$$= \underset{\mathbf{W}^\top \mathbf{W} = \mathbf{I}}{\arg\max} \frac{\mathrm{Tr}(\mathbf{W}^\top \mathbf{X} \mathbf{L}' \mathbf{X}^\top \mathbf{W})}{\mathrm{Tr}(\mathbf{W}^\top \mathbf{X} \mathbf{L} \mathbf{X}^\top \mathbf{W})}.$$

Multi-view Embedding

Generalized Multi-view Embedding:

- Define two graphs G and G' with graph Laplacians L and L', respectively.
- Define the projections $Y_1 = W_1^T X_1$ and $Y_2 = W_2^T X_2$, where $W_1 \in \mathbb{R}^{D_1 \times d}$ and $W_2 \in \mathbb{R}^{D_2 \times d}$
- Generalization of many multi-view methods of the form:

$$\mathcal{J} = \arg\max_{\mathbf{W}} \frac{\mathrm{Tr}(\mathbf{W}^\top \mathbf{P} \mathbf{W})}{\mathrm{Tr}(\mathbf{W}^\top \mathbf{Q} \mathbf{W})}$$

where the projection matrices $W = [W_1^T W_2^T \ldots W_V^T]^T$ are obtained by solving:

$$\mathbf{P}\mathbf{W} = \rho\, \mathbf{Q}\mathbf{W}$$

Multi-view Embedding

$$\mathbf{PW} = \rho\,\mathbf{QW}$$

	$\mathbf{P}$	$\mathbf{Q}$
MvCCA	$\begin{bmatrix} \mathbf{0} & \boldsymbol{\Sigma}_{12} & \cdots & \boldsymbol{\Sigma}_{1V} \\ \boldsymbol{\Sigma}_{21} & \mathbf{0} & \cdots & \boldsymbol{\Sigma}_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\Sigma}_{V1} & \boldsymbol{\Sigma}_{V2} & \cdots & \mathbf{0} \end{bmatrix}$	$\begin{bmatrix} \boldsymbol{\Sigma}_{11} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{22} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{\Sigma}_{VV} \end{bmatrix}$
MvPLS	$\begin{bmatrix} \mathbf{0} & \boldsymbol{\Sigma}_{12} & \cdots & \boldsymbol{\Sigma}_{1V} \\ \boldsymbol{\Sigma}_{21} & \mathbf{0} & \cdots & \boldsymbol{\Sigma}_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\Sigma}_{V1} & \boldsymbol{\Sigma}_{V2} & \cdots & \mathbf{0} \end{bmatrix}$	$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I} \end{bmatrix}$
MvMDA	$\begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} & \cdots & \mathbf{P}_{1V} \\ \mathbf{P}_{21} & \mathbf{P}_{22} & \cdots & \mathbf{P}_{21} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{V1} & \mathbf{P}_{V2} & \cdots & \mathbf{P}_{VV} \end{bmatrix}$	$\begin{bmatrix} \mathbf{Q}_{11} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{22} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{Q}_{VV} \end{bmatrix}$