Feature extraction techniques

Vanessa Gómez Verdejo

Applications of Machine Learning
Master in Multimedia and Communications
Academic year 2014-2015

Contents

- Introduction: why feature extraction?
- Feature extraction techniques:
  - Unsupervised approaches: PCA
  - Supervised approaches:
    - Multivariate Analysis: PLS, CCA
    - LDA, ..
  - Kernel extensions
  - Compacted approaches

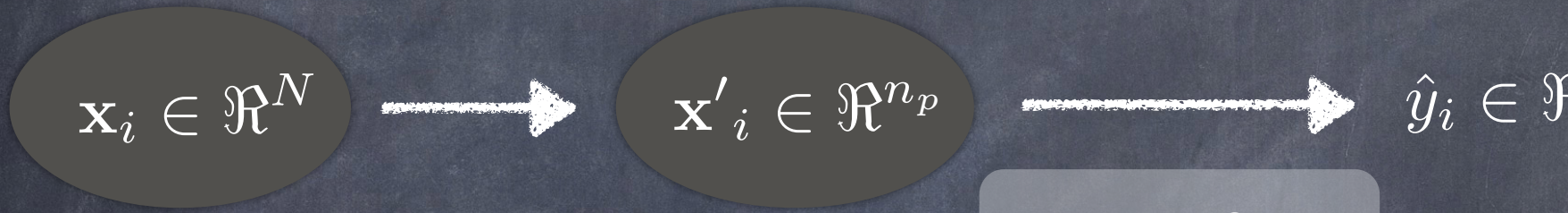
Introduction

Why feature extraction?

- Reduce dimensionality of the input space
- Compact representation of data (crucial for large datasets)
- Minimize the number of parameters in the classifier (curse of dimensionality) (e.g.: polynomial model)
- Use only relevant data
  - Remove irrelevant/noisy/correlated components
  - Discover good combinations of input variables (features)
  - "Bend" the input space to better fit our task
- Goal in FE: Simplify ML stage & minimize the loss of RELEVANT information

Preprocessing stage

Input data Projected data Output data



$X' = U^T X^T$

Classifier

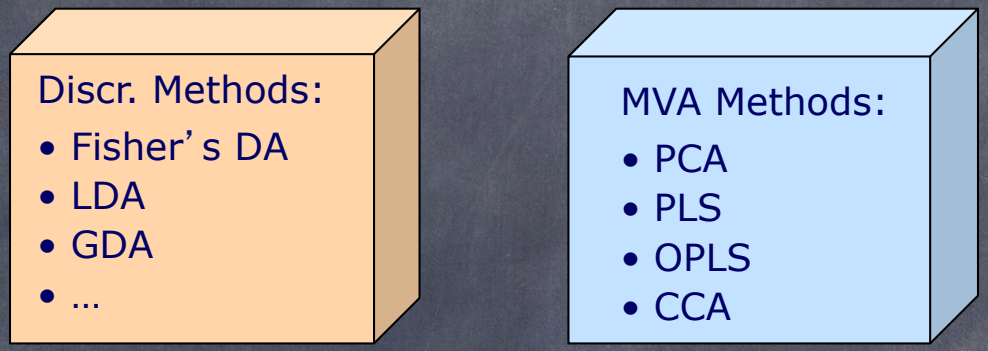
Feature extraction $n_p < N$

$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_L]^T \quad (L \times N) \quad \mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_L]^T \quad (L \times M)$

$U = [\mathbf{u_1}, \ldots, \mathbf{u_{n_p}}] \quad (N \times n_p)$

Feature extraction

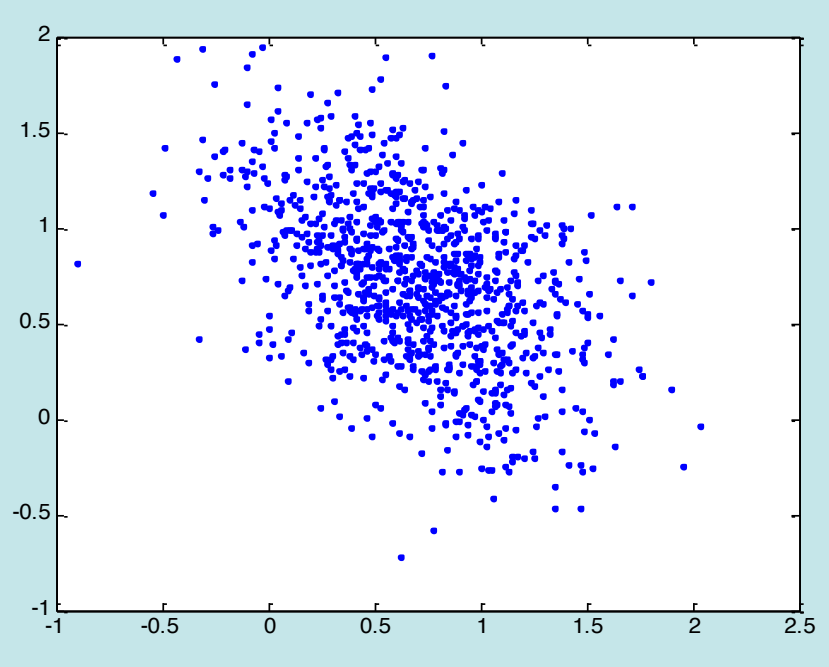
Feature Extraction



- Both families provide linear projections for FE
- Algorithms require classical linear algebra methods (EIG, GEN. EIG, SVD)
- Certain equivalencies are known under a classification context ...

Principal Component Analysis (PCA)

- Goal: Find projections maximizing the variance of the projected data:



- Xu_1 projects the maximum variance of the data
- Xu_2 the second one, ...
- $n_p < N$; i.e., we remove the projections with less variance

Principal Component Analysis

- Find projections maximizing the variance of the projected data

$$\mathbf{U} = \underset{\mathbf{U}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T\mathbf{X}^T\mathbf{X}\mathbf{U}\right\} = \underset{\mathbf{U}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T C_{\mathbf{X}\mathbf{X}}\mathbf{U}\right\}$$

$$\text{s.t.} \quad \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

- Which leads to the eigenvalue problem

$$C_{XX}\mathbf{u} = \lambda\mathbf{u}$$

  - U consists of the first eigenvectors of Cxx (i.e., those associated with largest eigenvalues)

$$\mathbf{U} = \operatorname{eigs}(C_{XX}) \quad \text{or} \quad \mathbf{U} = \operatorname{svd}(\mathbf{X})$$

Principal Component Analysis (PCA)

- PCA is usually implemented by extracting the projection vectors one by one -> DEFLATION
- The following sequential method is applied:

1. The leading eigenvector of Cxx (and its eigenvalue) is extracted -> $\mathbf{u}_i,\ \lambda_i$
2. Cxx is deflated to remove u_i:

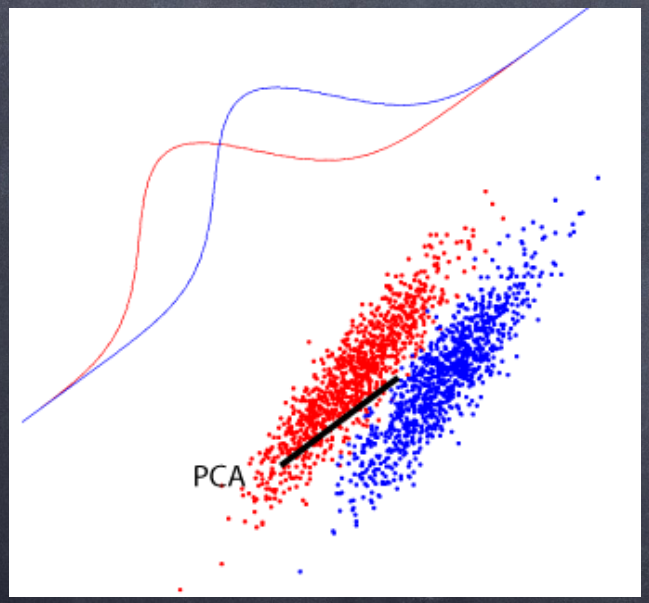
$$C_{XX} \leftarrow C_{XX} - \lambda_i \mathbf{u}_i \mathbf{u}_i^T$$

We are projecting the data onto the orthogonal complement of the direction given by u_i

$$\mathbf{X} \leftarrow \mathbf{X}\left[\mathbf{I} - \mathbf{u}_i \mathbf{u}_i^T\right]$$

Principal Component Analysis (PCA)

- It is an **unsupervised** algorithm!!!!



- Which direction will PCA consider as the most relevant one ?
- If we had to extract only one projection, which is the most relevant for the task?
- Clearly, when dealing with supervised problems, we should consider the labels to obtain good features -> PLS & CCA algorithms

Lab session

- Load the data
  - Split train & test
  - Do not remove water bands
  - Normalize (remove means!!!)
- Create output coding matrix Y
- Linear FE + classifier
  - Starting with PCA

PCA in Python

- from sklearn.decomposition import PCA
- pca = PCA(n_components=N_feat_max)
  - n_components == 'mle' #Minka's MLE algorithm
  - whiten == (False)
- Attributes:
  - .components_
  - .explained_variance_ratio_
- Methods:
  - .fit(),.fit_transform()
  - .transform()

Create some utilities

- You can create a function:
  - to evaluate the extracted features:
  - to plot the error evolution (according to the number of extracted features)
  - to plot the projection vectors
  - to plot the most important extracted data

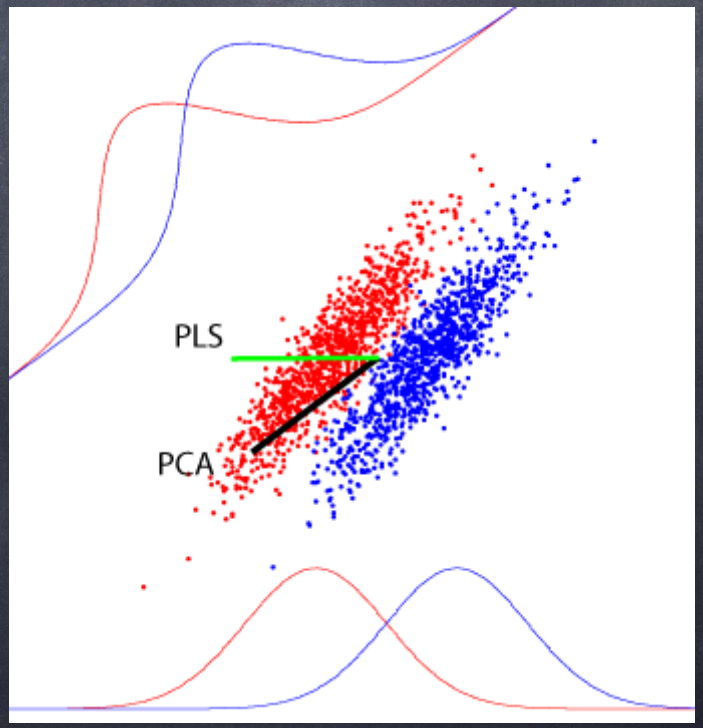
Supervised feature extraction

Partial Least Squares (PLS)

- The goal of PLS is to find the projections of the input and output data with maximum **covariance**

$$\mathbf{U}, \mathbf{V} = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T \mathbf{X}^T \mathbf{Y} \mathbf{V}\right\} = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T C_{\mathbf{XY}} \mathbf{V}\right\}$$

$$\mathbf{U}^T \mathbf{U} = \mathbf{V}^T \mathbf{V} = \mathbf{I}$$



- Its solutions is given by:

$$\mathbf{U}, \mathbf{V} = \operatorname{svd}(C_{XY})$$

Partial Least Squares (PLS)

- We can find 2 deflation processes

1. PLS Mode A (equivalent to block solution PLS-SB)

$$\mathbf{X} \leftarrow \mathbf{X}\left[\mathbf{I} - \mathbf{u}_i\mathbf{u}_i^T\right] \qquad \mathbf{Y} \leftarrow \mathbf{Y}\left[\mathbf{I} - \mathbf{v}_i\mathbf{v}_i^T\right]$$

$$C_{XY} \leftarrow C_{XY} - \lambda_i \mathbf{u}_i\mathbf{v}_i^T$$

as many new features as classes!!!!!

2. PLS2 projects X and Y over the orthogonal complement of Xu_i

$$\mathbf{X} \leftarrow \left(\mathbf{I} - \frac{\mathbf{X}\mathbf{u}_i\mathbf{u}_i^T\mathbf{X}^T}{\|\mathbf{X}\mathbf{u}_i\|_2^2}\right)\mathbf{X} \qquad \mathbf{Y} \leftarrow \left(\mathbf{I} - \frac{\mathbf{X}\mathbf{u}_i\mathbf{u}_i^T\mathbf{X}^T}{\|\mathbf{X}\mathbf{u}_i\|_2^2}\right)\mathbf{Y}$$

as many new features as input dimensions!!!!!

Partial Least Squares (PLS)

- Different implementations (sklearn.cross_decomposition):
  - In block (PLS-SB): .PLSSVD
  - Deflating (PLS-mode A): .PLSCanonical
    - It deflates both input and output spaces
    - We are not interested in output space dimension reduction
  - Input space deflating (PLS 2): .PLSRegression
    - It only deflects the input space
    - You can compute as many features as original ones

Canonical Correlation Analysis (CCA)

- CCA searches for the directions of maximum **correlation** between input and output data

$$\mathbf{u}, \mathbf{v} = \underset{\mathbf{u},\mathbf{v}}{\operatorname{argmax}} \frac{\left(\mathbf{u}^T C_{XY} \mathbf{v}\right)^2}{\mathbf{u}^T C_{XX} \mathbf{u} \mathbf{v}^T C_{YY} \mathbf{v}}$$

- or equivalent to:

$$\mathbf{U}, \mathbf{V} = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T C_{XY} \mathbf{V}\right\}$$

$$\mathbf{U}^T C_{\mathbf{xx}} \mathbf{U} = \mathbf{V}^T C_{\mathbf{xx}} \mathbf{V} = \mathbf{I}$$

Canonical Correlation Analysis (CCA)

- This problem can be solved as a generalized eigenvalue problem
- As many extracted features as output classes
- It is usually applied to obtain a common space to work with input and output features
- For classification purposes, it tends to outperform PLS approaches
- Python: from sklearn.cross_decomposition import CCA

LDA (review)

- Each likelihood is given by a gaussian d.d.p. (same covariance matrix over all the classes)
- Estimate the gaussian parameters with the training data
- We can define linear discriminant functions

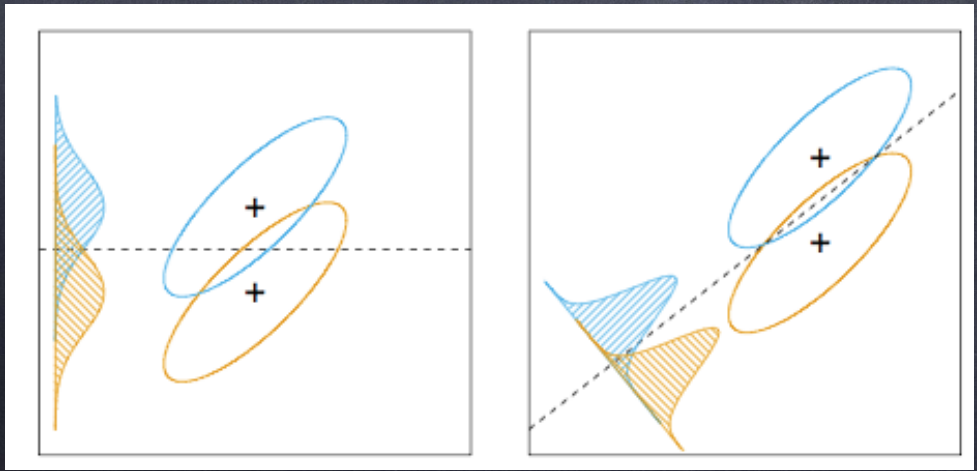
$$\delta_j(\mathbf{x}) = \mathbf{x}^T V^{-1} \mathbf{m}_j - \frac{1}{2} \mathbf{m}_j^T V^{-1} \mathbf{m}_j + \log P_H(j)$$

- And classify according to

$$D(\mathbf{x}) = \underset{j}{\operatorname{argmax}}\ \delta_j(\mathbf{x})$$

LDA as feature extractor

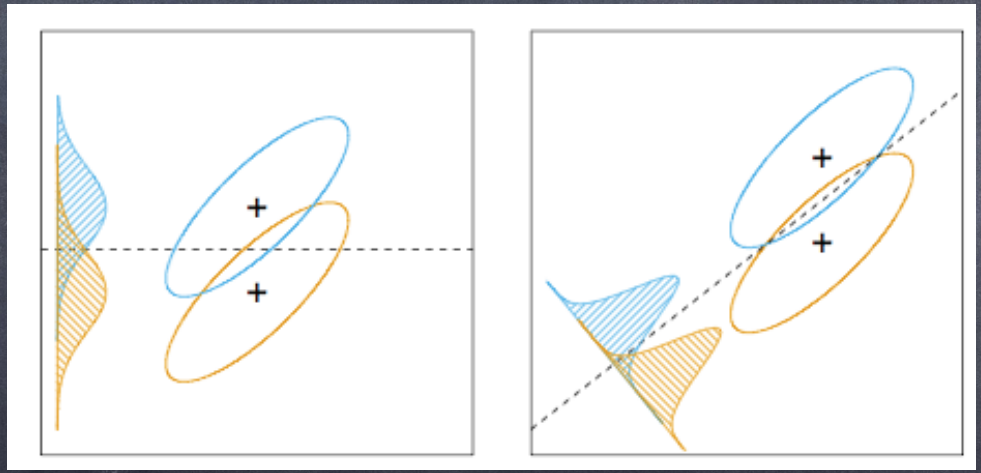
- We have defined M gaussian lying in a subspace of dimension $\leq M - 1$
- (If N is much larger than M) this will be a considerable drop in dimension.
- Thus, we might project x onto this centroid-spanning subspace, and make distance comparisons there.



- LDA finds the direction of minimum overlap among classes

LDA as feature extractor

- **Fisher discriminant analysis** arrived at the same result following a different approach:
  "Find the linear combination $Z = aX$ such that the between-class variance is maximized relative to the within-class variance."



- They are also referred to as **canonical variates**: CCA (over classification problems) provides the same result

LDA (as FE) in Python

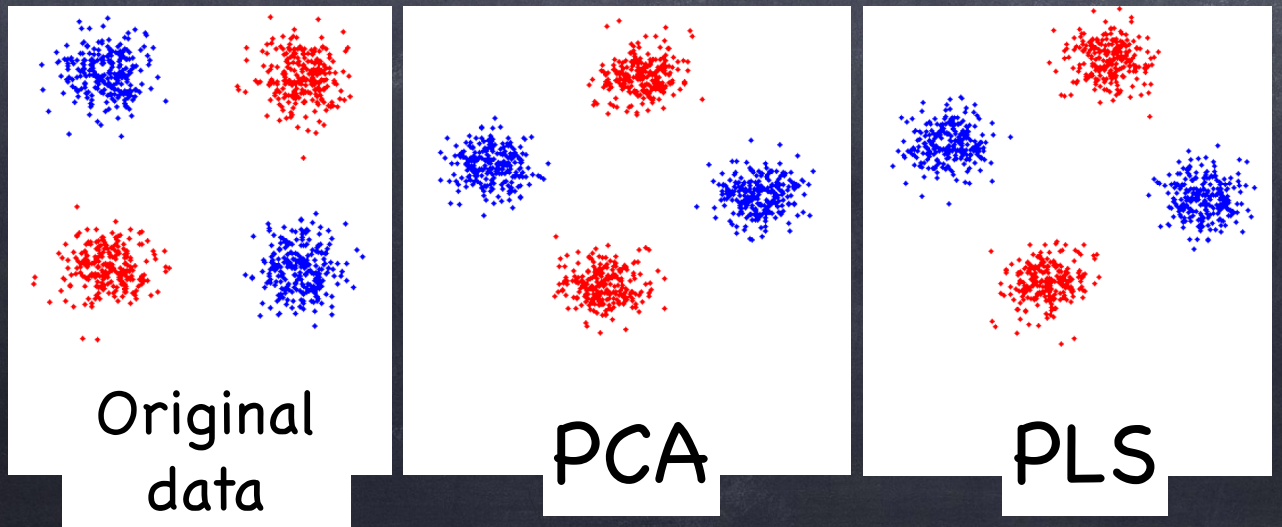
- from sklearn.lda import LDA
- N_feat_max = classes.shape[0] - 1 # As many new features as classes minus 1
- lda = LDA(n_components=N_feat_max)
  - .fit()
  - .transform()

No linear feature extraction

Linear methods

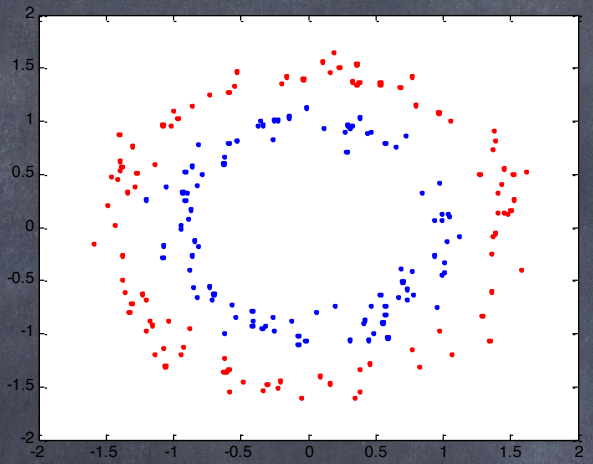
- Linear methods have some nice properties:
  - Simplicity
  - Easy to understand
  - Robust
  - Lead to convex problems

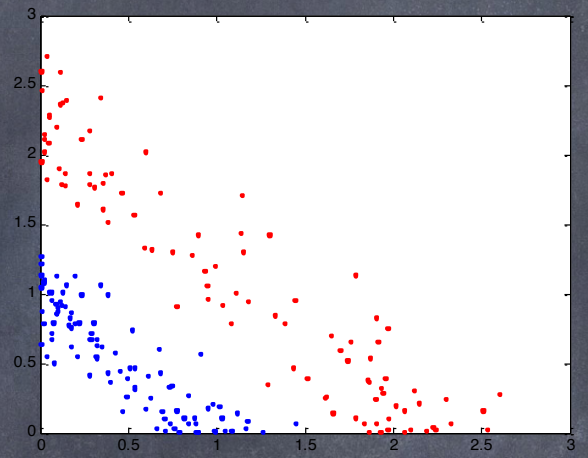
... but lack expressive power



Kernel methods

- Idea: Project Data in a High Dimensional Space





$$\mathbf{x}_i \quad \rightarrow \quad \phi(\mathbf{x}_i)$$

... so that a linear algorithm run in the "Feature Space" is non-linear in the original input space

- Examples: polynomial, gaussian, ...

Working with kernels

- **Kernel trick**: it is possible to (cheaply) compute inner products in many ∞-dimensional spaces

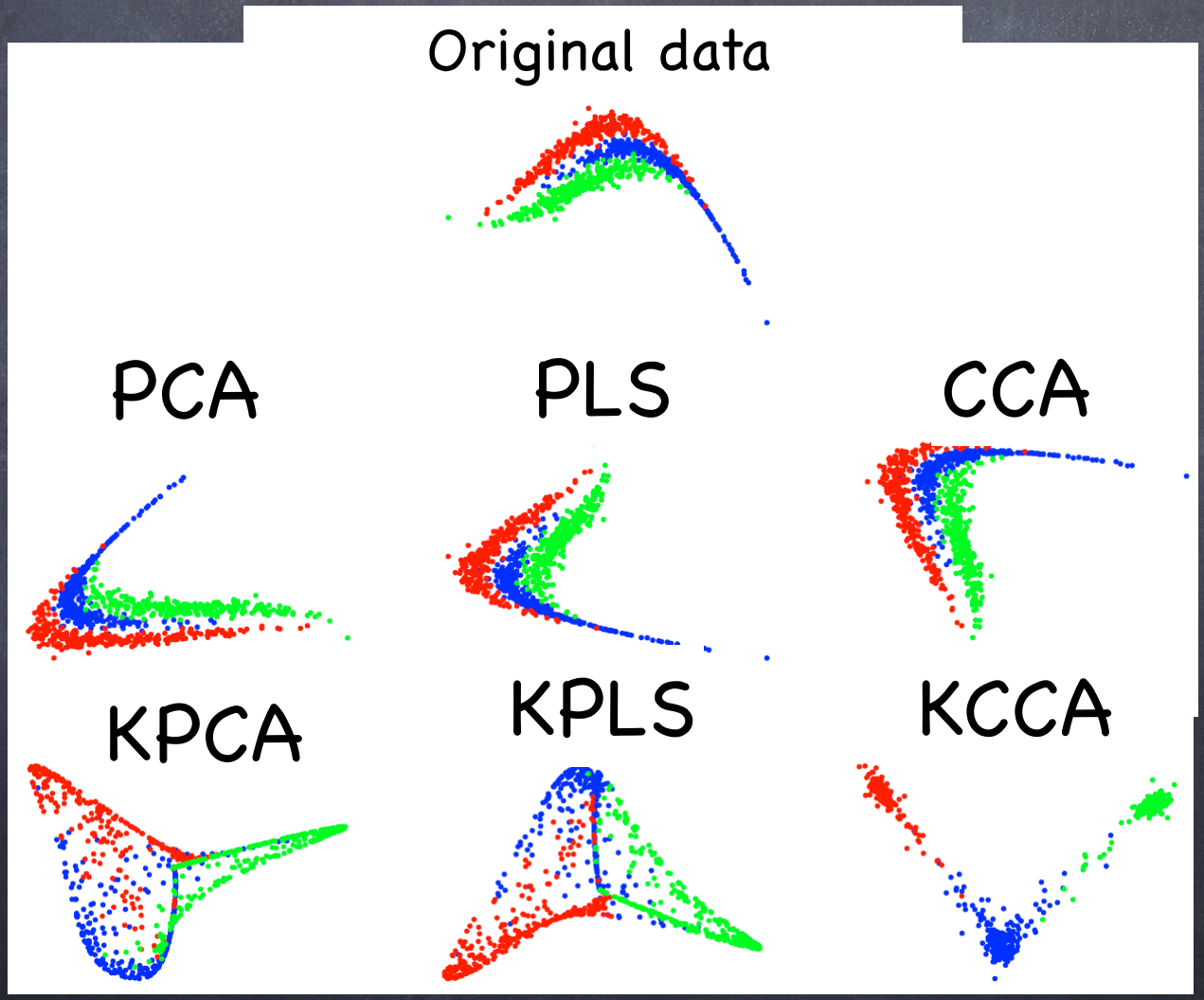
$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$$

If the linear algorithm can be reformulated in terms of inner products only, computing the kernels will make the job.

- **Representer theorem** states that the solutions of certain optimization problems can be written as an expansion in terms of training samples

$$\mathbf{u} = \sum_{i=1}^{L} a_i \phi(\mathbf{x}_i) = \Phi^T \mathbf{a}$$

Working with kernels



Kernel PCA

- Find projections maximizing the variance of the data in FEATURE SPACE:

$$\mathbf{U} = \underset{\mathbf{U}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T \mathbf{\Phi}^T \mathbf{\Phi} \mathbf{U}\right\} \qquad \text{s.t.} \;\; \mathbf{U}^T \mathbf{U} = I$$

- Introducing now $\mathbf{U} = \mathbf{\Phi}^T \mathbf{A}$

$$\mathbf{U} = \underset{\mathbf{U}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{A}^T \mathbf{\Phi}\mathbf{\Phi}^T \mathbf{\Phi}\mathbf{\Phi}^T \mathbf{A}\right\} \qquad \text{s.t.} \;\; \mathbf{A}^T \mathbf{\Phi}\mathbf{\Phi}^T \mathbf{A} = I$$

$$\mathbf{U} = \underset{\mathbf{U}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{A}^T K K \mathbf{A}\right\} \qquad \text{s.t.} \;\; \mathbf{A}^T K \mathbf{A} = I$$

- Which leads to the eigenvalue problem $K\mathbf{a} = \lambda \mathbf{a}$

$$\mathbf{A} = \operatorname{eigs}(K)$$

Kernel PCA in Python

- OPTION 1: use PCA with the kernel matrix of X
- OPTION 2:
  - from sklearn.decomposition import KernelPCA
  - kpca = KernelPCA( )
    - n_components=N_feat_max
    - kernel: "linear" | "poly" | "rbf" | "sigmoid" | "cosine" | "precomputed"
  - kpca.inverse_transform <-preimage computation
  - http://scikit-learn.org/stable/auto_examples/decomposition/plot_kernel_pca.html

KPLS

- Linear formulation:

$$\mathbf{U}, \mathbf{V} = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T \mathbf{X}^T \mathbf{Y} \mathbf{V}\right\} \quad \text{s.t.} \quad \mathbf{U}^T \mathbf{U} = \mathbf{V}^T \mathbf{V} = \mathbf{I}$$

- Kernel extension $\mathbf{U} = \mathbf{\Phi}^T \mathbf{A}$

$$\mathbf{A}, \mathbf{V} = \underset{\mathbf{A},\mathbf{V}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{A}^T K \mathbf{Y} \mathbf{V}\right\} \quad \text{s.t.} \quad \mathbf{A}^T K \mathbf{A} = \mathbf{V}^T \mathbf{V} = \mathbf{I}$$

- Which can be solved as: $\mathbf{A}, \mathbf{V} = \operatorname{svd}(K\mathbf{Y})$
- Python: use PLSRegression with the Kernel matrix of X

KCCA

- Linear formulation:

$$\mathbf{U}, \mathbf{V} = \underset{\mathbf{U},\mathbf{V}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{U}^T\mathbf{X}^T\mathbf{Y}\mathbf{V}\right\}$$

$$\text{s.t.} \quad \mathbf{U}^T\mathbf{X}^T\mathbf{X}\mathbf{U} = \mathbf{V}^T\mathbf{Y}^T\mathbf{Y}\mathbf{V} = \mathbf{I}$$

- Kernel extension $\mathbf{U} = \mathbf{\Phi}^T\mathbf{A}$

$$\mathbf{A}, \mathbf{V} = \underset{\mathbf{A},\mathbf{V}}{\operatorname{argmax}} \operatorname{Tr}\left\{\mathbf{A}^T K\mathbf{Y}\mathbf{V}\right\} \quad \text{s.t.} \quad \mathbf{A}^T KK\mathbf{A} = \mathbf{V}^T\mathbf{V} = \mathbf{I}$$

- Python: use CCA with the Kernel matrix of X

Remarks: computing kernel matrix

- Compute training & test matrix
- Gaussian kernel: you can estimate sigma

$$\sigma^2 = \frac{1}{2}\frac{1}{(L-1)!}\sum_{i \neq j}(\mathbf{x}_i - \mathbf{x}_j)^2$$

- or validate it
- Center the kernel

Centring kernel matrix

$$\phi_c(\mathbf{x}) = \phi(\mathbf{x}) - \frac{1}{L}\sum_{i=1}^{L}\phi(\mathbf{x}_i)$$

$$\kappa_c(\mathbf{x},\mathbf{y}) = \langle \phi_c(\mathbf{x}), \phi_c(\mathbf{y}) \rangle = \left\langle \phi(\mathbf{x}) - \frac{1}{L}\sum_{i=1}^{L}\phi(\mathbf{x}_i), \phi(\mathbf{y}) - \frac{1}{L}\sum_{i=1}^{L}\phi(\mathbf{x}_i)) \right\rangle =$$

$$\kappa(\mathbf{x},\mathbf{y}) - \frac{1}{L}\sum_{i=1}^{L}\kappa(\mathbf{x},\mathbf{x}_i), -\frac{1}{L}\sum_{i=1}^{L}\kappa(\mathbf{z},\mathbf{x}_i) + \frac{1}{L^2}\sum_{i,j=1}^{L}\kappa(\mathbf{x}_i,\mathbf{x}_j)$$

$$\mathbf{K}_c = \mathbf{K} - \frac{1}{L}\mathbf{1}\mathbf{1}^T\mathbf{K} - \frac{1}{L}\mathbf{K}\mathbf{1}\mathbf{1}^T + \frac{1}{L^2}\left(\mathbf{1}^T\mathbf{K}\mathbf{1}\right)\mathbf{1}\mathbf{1}^T$$

Compacted approaches

Limitations of KMVA

- Kernel MVA overcome the lack of expressiveness of the linear versions, but have serious scalability limitations:
  - Computation of the Kernel Matrix K (L $\times$ L)
  - Storage requirements: $O(L^2)$
  - Computational requirements: $O(L^2)$
  - Application to new data is expensive: O(L)
- OVERFITTING problems can emerge

Compact solution

- Reduce de number of possible "support data"

$$\mathbf{U} = \mathbf{\Phi}_R^T \mathbf{A}$$

- where Φ_R is a subset of the training data with R < L points
- We obtain a reduced kernel matrix

$$K_R = \Phi_R \Phi^T \qquad (R \times L)$$

- Are we subsampling the data??
  - K_R still contains information about all samples !!

Python routines to compute kernel matrix

- sklearn.kernel_approximation.Nystroem
  - Constructs an approximate feature map for an arbitrary kernel using a subset of the data as basis.
  - Nystroem(kernel='rbf', gamma=None, coef0=1, degree=3, kernel_params=None, n_components=100, random_state=None)
  - Williams, C.K.I. and Seeger, M. "Using the Nystroem method to speed up kernel machines", Advances in neural information processing systems 2001

Python routines to compute kernel matrix

- sklearn.kernel_approximation.RBFSampler
  - sklearn.kernel_approximation.RBFSampler(gamma=1.0, n_components=100, random_state=None)¶
  - "Random Features for Large-Scale Kernel Machines" by A. Rahimi and Benjamin Recht.
  - Approximates feature map of an RBF kernel by Monte Carlo approximation of its Fourier transform