MATH 7339 - Machine Learning and Statistical Learning Theory 2

Section: Continuous Latent Variables

1. Review PCA
2. Kernel PCA
3. Probabilistic PCA
4. Factor Analysis
5. ICA

Dimensional Reduction

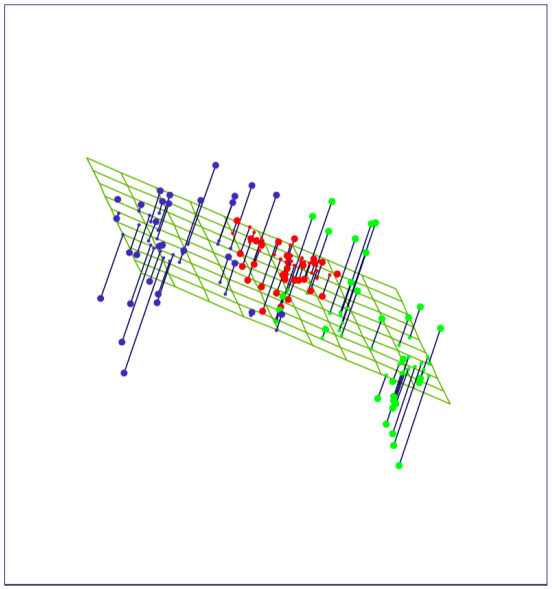
Data $\mathcal{D} = \left\{\left(\vec{x}^{(i)}, y^{(i)}\right) \mid i = 1, \ldots, N\right\} = (X, \vec{y})$

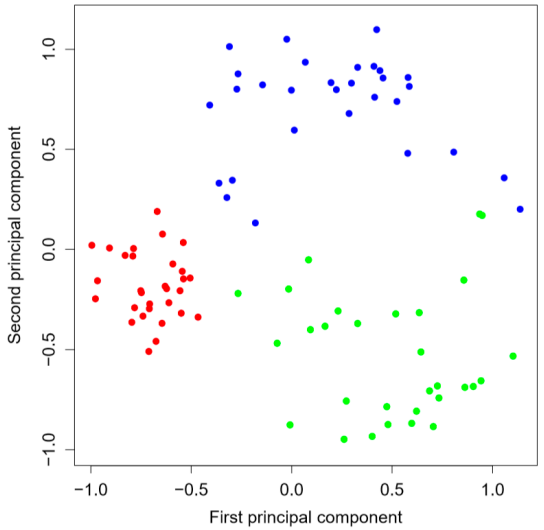
$$\vec{x}^{(i)} \in \mathbb{R}^d$$

Dimensional Reduction is the process of combining high dimensional information $\mathbb{R}^d$ into low dimensional representations $\mathbb{R}^k$. Low dimensional representations can be computationally easier to work with, while hopefully throwing away only extraneous information.

There are two main techniques in dimensional reduction:
1.projection onto linear subspaces and
2. manifold learning.

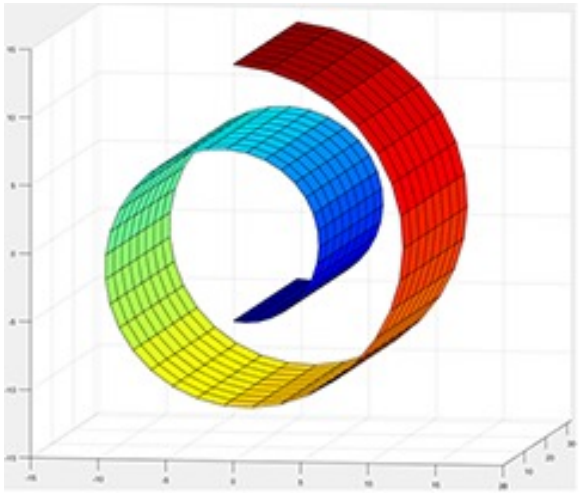
In projection onto linear subspaces, we try to discover the linear combination of features the provide the clearest coordinate system for the dataset, i.e. the component factors.



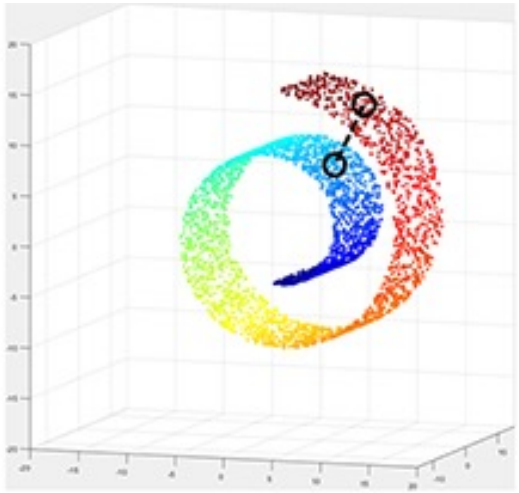


PCA projection: $\mathbb{R}^3 \rightarrow \mathbb{R}^2$.

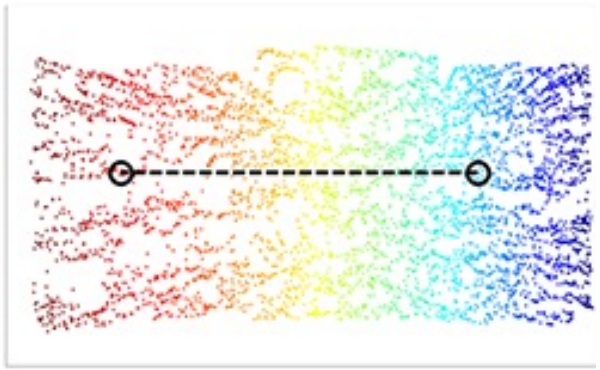
In manifold learning, we try to fit a more complicated manifold to the underlying data.



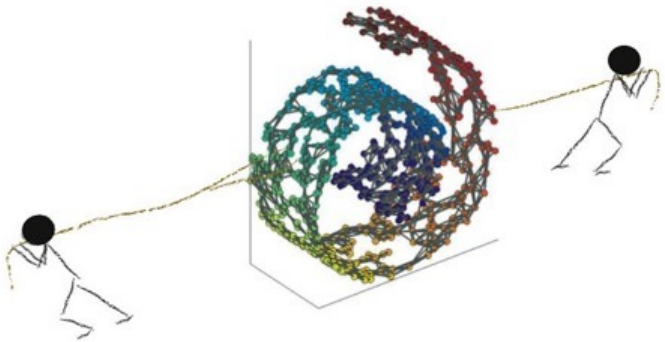
(a)



(b)



(c)



(a) The original 3D surface. **(b)** The data points sampled from (a). The Euclidean distance indicated by dash line between the circled points cannot represent the distance lying on the potential structure. **(c)** The distance measured in the learned low-dimensional can more accurately model the data.

If we knew lower dimensional surface S and could work in it (e.g., regression, classification, etc.) then possibly avoid curse of dimensionality, i.e., if $k \ll d$ gives much lower dimension.

Challenges*:*

(a) Don't know S
(b) Must learn from finite dataset $\mathcal{D} = \{(\vec{x}^{(i)}) \mid i = 1, \ldots N\}$
(c) $\vec{x}^{(i)}$ may live "near" but not "on" the subspace or manifold

This section ill discuss:

- PCA / Factor Analysis
- Kernel PCA
- ICA (Independent Component Analysis)

Other (additional topics not discussed)

- ISOMAP (Tenenbaum and Langford, Stanford) [algorithm for manifold finding]
- Local linear embedding (Saul) Work of Beylkin, Nagoya, Donoho
- MDS (multidimensional scaling) linear embeddings under possibly
- non-Eucidean distributions

➢ Review of Principal Component Analysis (PCA)

Data matrix X, is a $N \times d$ matrix, with n data points and dimension d. (Suppose the **mean** (center) of X is **zero**.)

Goal: find a $N \times k$ matrix Z with n data points and in dimension k.

The **sample covariance matrix** for $\vec{x}$ is

$$Cov(X) = \frac{1}{N-1} X^T X \qquad \text{Or } Cov(X) = \frac{1}{N} X^T X$$

Use estimated means Use known means

Denote $A = X^T X$. It is called **Gram matrix**
Suppose $rank\ A = rank\ X = p$, it has the following properites:

- A is a $d \times d$ symmetric matrix.
- A is positive semidefinite, with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p > \lambda_{p+1} = \cdots = \lambda_d = 0$
- $A = VDV^{-1} = VDV^T$ orthogonally diagonalizable. (Spectral decomposition)

$$D = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_d \end{bmatrix} \qquad V = [\vec{v}_1 \quad \cdots \quad \vec{v}_d] \text{ is orthogonal matrix}$$

❑ Singular value decomposition of X

From the spectral decomposition of $X^T X = VDV^T$, we can solve the following

$$X = U\Sigma V^T = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_p \vec{u}_p \vec{v}_p^T$$

$N \times d$ $\quad$ $N \times N$ $\quad$ $N \times d$ $\quad$ $d \times d$

$V = [\vec{v}_1 \quad \cdots \quad \vec{v}_d]$ is orthogonal matrix

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & 0 & \sigma_p & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{N \times d}$$

Here $\sigma_i = \sqrt{\lambda_i}$ are called singular values.
So $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p$

$U = [\vec{u}_1 \cdots \vec{u}_p \ \vec{u}_{p+1} \cdots \vec{u}_N]$ is orthogonal matrix

where $\vec{u}_i := \dfrac{X\vec{v}_i}{\|X\vec{v}_i\|} = \dfrac{X\vec{v}_i}{\sigma_i}$ for $i = 1, \ldots, p$. And, $\{\vec{u}_{p+1} \cdots \vec{u}_N\}$ basis of $(\text{im } X)^{\perp}$

$X\vec{v}_i = 0$ for $i = p+1, \ldots, d$

Some times, people use a truncated version of SVD:

$$X = \widetilde{U}\tilde{\Sigma}V^T = \sigma_1\vec{u}_1\vec{v}_1^T + \sigma_2\vec{u}_2\vec{v}_2^T + \cdots + \sigma_p\vec{u}_p\vec{v}_p^T$$

$N\times d$ $\quad$ $N\times d$ $\quad$ $d\times d$ $\quad$ $d\times d$

Here $\widetilde{U} = [\vec{u}_1 \cdots \vec{u}_d]$ $\quad$ $\tilde{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & 0 & \sigma_p & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{d\times d}$

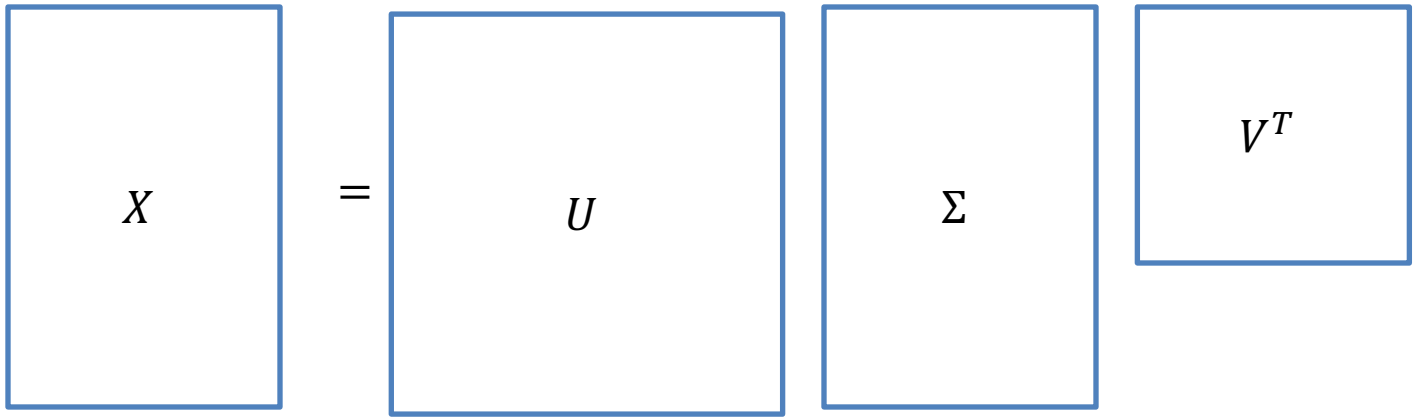
- The ***k*-th truncation** of data matrix X

$$X_k := \sigma_1\vec{u}_1\vec{v}_1^T + \sigma_2\vec{u}_2\vec{v}_2^T + \cdots + \sigma_k\vec{u}_k\vec{v}_k^T$$

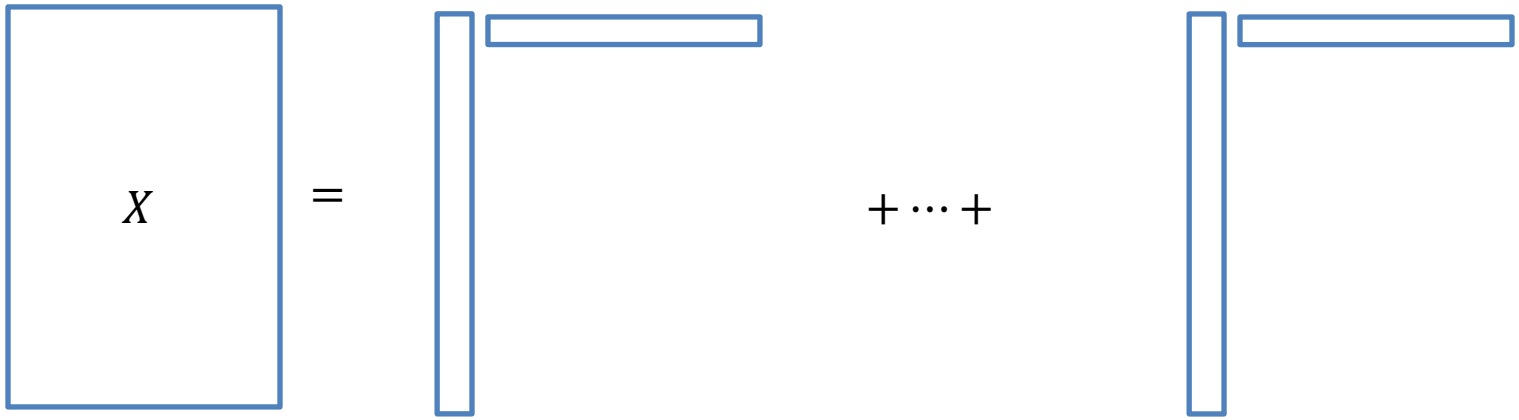
for $k = 1, \ldots, p$.

Graph Explanation of SVD:

$$X = U\Sigma V^T = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_p \vec{u}_p \vec{v}_p^T$$



$$X = U\Sigma V^T = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_p \vec{u}_p \vec{v}_p^T$$



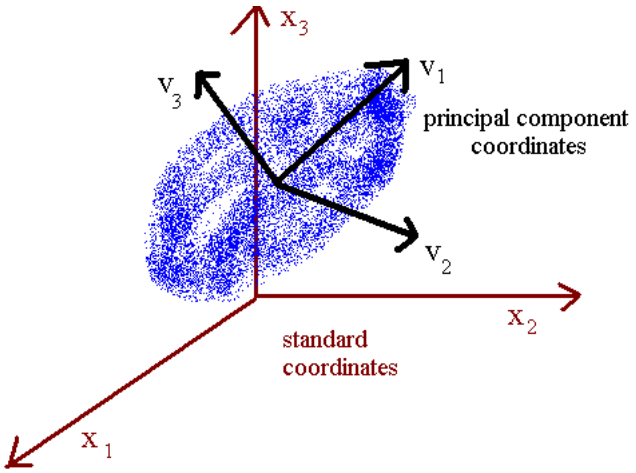
❑ Principal Components

Definition: The unit vectors $\vec{v}_i$ in matrix V is called the ***i*-th principal components** of the data X.

- The vectors $\vec{v}_1, \dots, \vec{v}_d$ form an orthonormal basis (coordinate system) for the feature space $\mathbb{R}^d$.

- Define the **new features $\vec{z} \in \mathbb{R}^d$:**

$$\vec{z} := \begin{bmatrix} z_1 \\ \vdots \\ z_d \end{bmatrix} \qquad \vec{x} = V\vec{z} = z_1\vec{v}_1 + \cdots + z_d\vec{v}_d$$



So, $\vec{z}$ is the **coordinate** of $\vec{x}$ under the basis $\vec{v}_1, \dots, \vec{v}_d$.

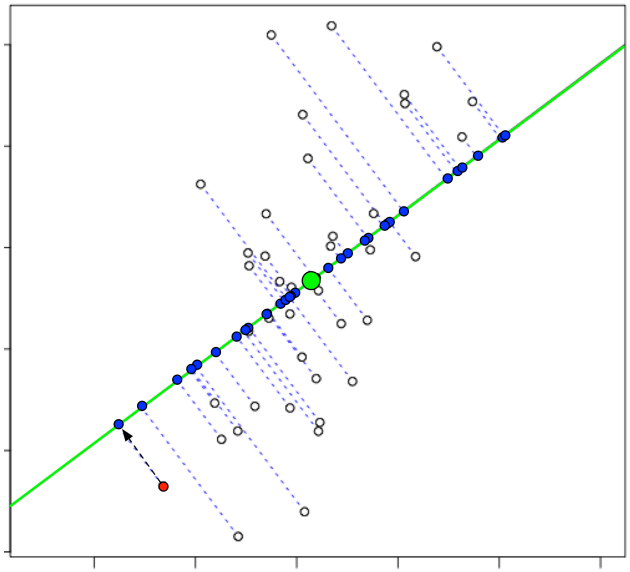
$$\vec{z} = V^{-1}\vec{x} = V^T\vec{x}$$

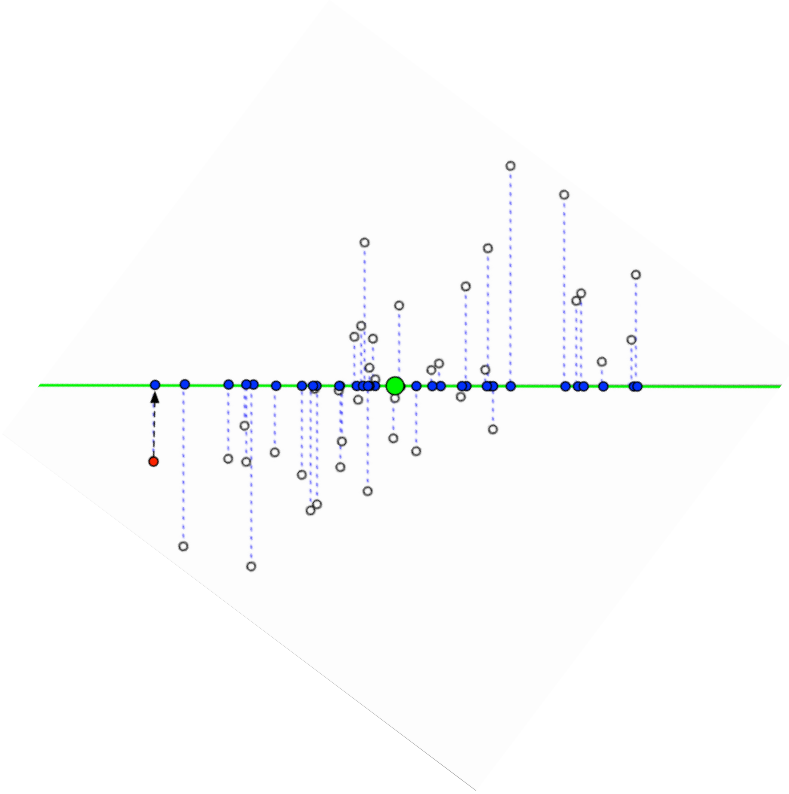
z_i is called the ***i*-th principal component coordinate**.

The map from feature $\vec{x} \in \mathbb{R}^d$ to new feature $\vec{z} \in \mathbb{R}^d$ is a **linear** (**orthogonal**) transformation

$$\mathbb{R}^d \rightarrow \mathbb{R}^d$$

$$\vec{x} \rightarrow \vec{z} = V^T \vec{x}$$

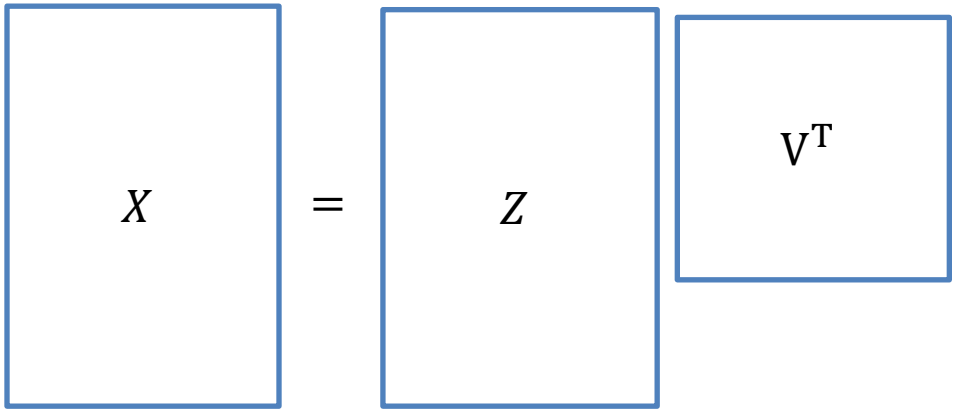




- Define the **new data matrix:**

$$Z = XV$$

$N \times d$ $N \times d$ $d \times d$



- The **sample covariance matrix** for $\vec{z}$ is

$$Cov(Z) = \frac{1}{N-1} Z^T Z = \cdots = \frac{1}{N-1} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_d \end{bmatrix}$$

Each $\frac{\lambda_i}{N-1}$ is the **variance** of data in **direction** $\vec{v}_i$.

The new features z_i and z_j has covariance zero.

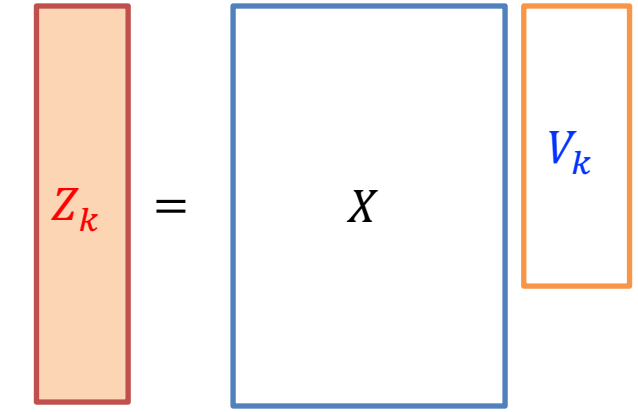
PCA projection

Since $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$, we can **truncate** the new data matrix Z as a $N \times k$ truncated matrix Z_k (which is the first k columns of Z) but preserve most of the variance.

$V_k = [\vec{v}_1 \cdots \vec{v}_k]$ is the first k columns of V. $\quad Z_k \coloneqq XV_k$

Now, k =rank Z_k << rank $X = p$. But data matrix Z_k is in dimension k feature space $\mathbb{R}^k$, with far fewer parameters.

For the truncated feature $\vec{z} \in \mathbb{R}^k$ and the new data matrix Z_k



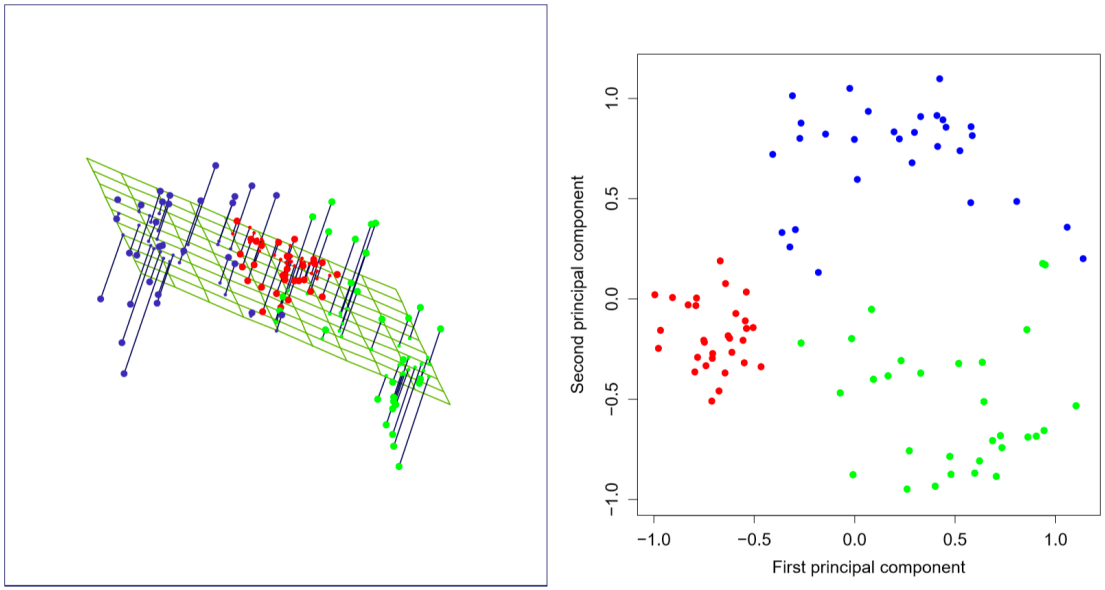
$$Cov(Z_k) = \frac{1}{N-1} Z_k^T Z_k = \cdots$$

$$= \frac{1}{N-1} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_k \end{bmatrix}$$

The truncated PCA projection from feature $\vec{x} \in \mathbb{R}^d$ to new truncated feature $\vec{z} \in \mathbb{R}^k$ is a **linear** transformation

$$\mathbb{R}^d \rightarrow \mathbb{R}^k$$

$$\vec{x} \rightarrow \vec{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix}$$



PCA projection: $\mathbb{R}^3 \rightarrow \mathbb{R}^2$.

Approximation to the original data:

$X = ZV^T$

Approximation

Reconstruction of X

$X \approx Z_k V_k^T$

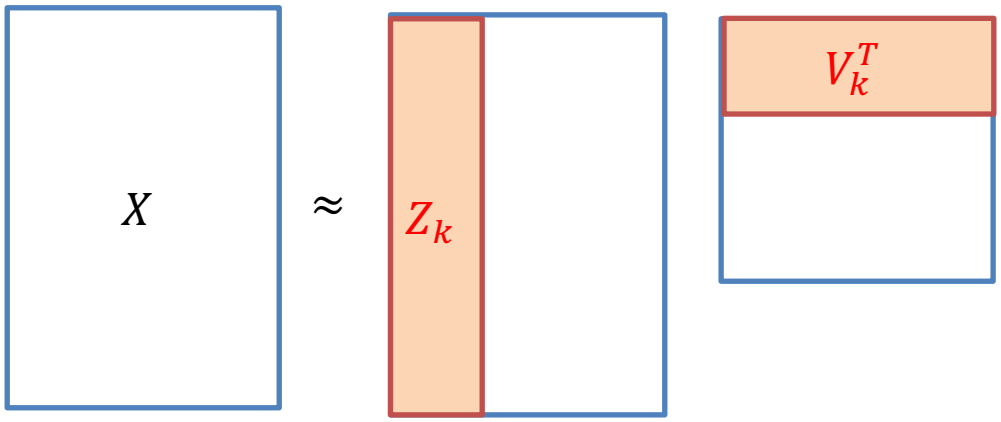
$N \times d$

$N \times k$

latent-variables

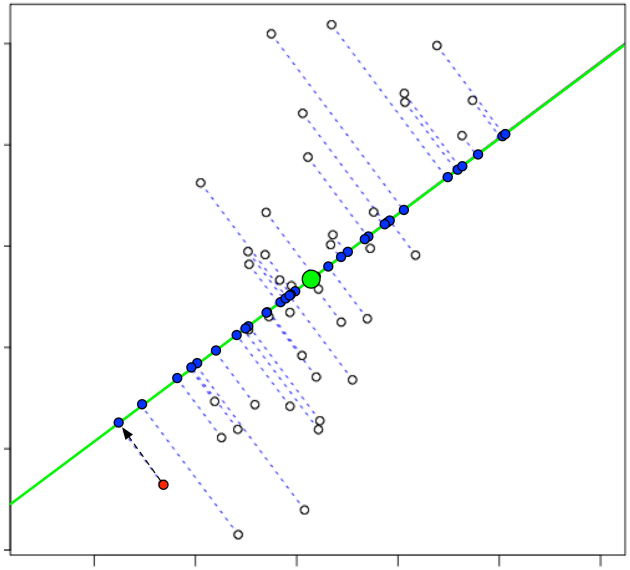
$k \times d$

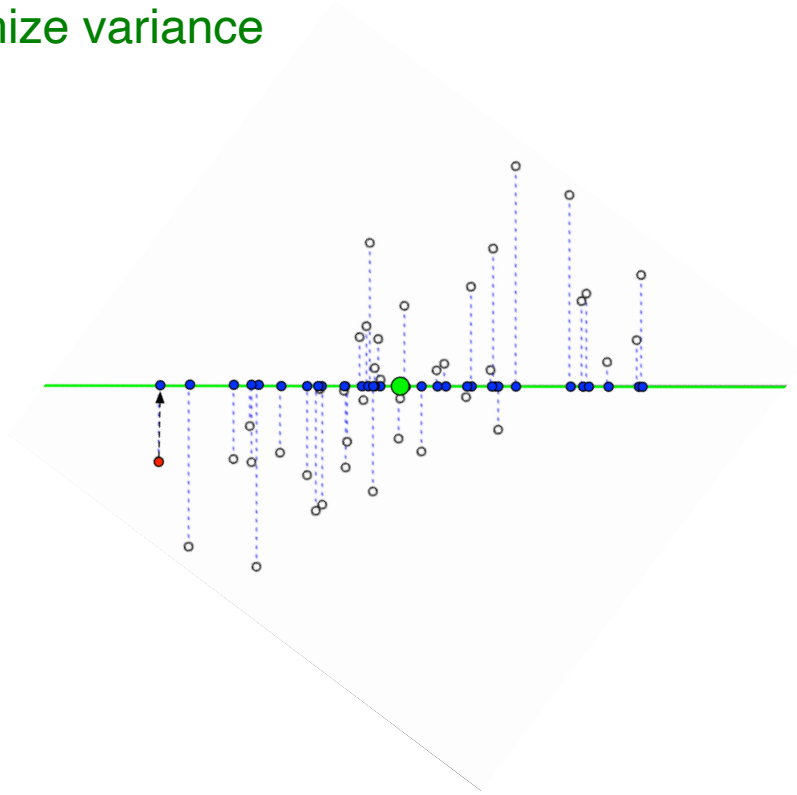
latent-factors



Fix k, two classical **equivalent** interpretations/derivations of PCA projection: $\mathbb{R}^d \to \mathbb{R}^k$:

1. Choose latent-factors V to minimize error.
2. Choose latent-factors V to maximize variance





Minimizing the reconstruction error is a clear goal.
The idea of maximize the projected variance is to keep as much of the data's structure intact as possible. By maximizing the variance we keep the data spread out, which keeps distinct points distinct.

1. Choose latent-factors V to minimize (reconstruction) error.

$$\underset{Z_k, V_k}{\operatorname{argmin}} \left\| X - Z_k V_k^T \right\|^2 = \sum_{i=1}^{N} \sum_{j=1}^{d} \left(x_j^{(i)} - \left(Z_k V_k^T \right)_{ij} \right)^2$$

$$= \underset{V_k}{\operatorname{argmin}} \left\| X - X V_k V_k^T \right\|^2$$

2. Choose latent-factors V to maximize variance.

$$\underset{V_k}{\operatorname{argmax}} \sum_{i=1}^{N} \left\| \vec{z}^{(i)} \right\|^2 = \sum_{i=1}^{N} \left\| V_k \vec{x}^{(i)} \right\|^2$$

$$= \underset{V_k}{\operatorname{argmax}} \|Z_k\|^2 = \underset{V_k}{\operatorname{argmax}} \|X V_k\|^2$$

Here, matrix norm is Frobenius norm

Equivalence comes from the Pythagorean Decomposition.

Reason for equivalence of the above two optimizations:

If we look at one data point $x^{(i)}$ and look at the

(reconstruction) error from $\vec{x}^{(i)}$ to $V_k\vec{z}^{(i)}$

$$\left\|\vec{x}^{(i)} - V_k V_k^T \vec{x}^{(i)}\right\|^2$$

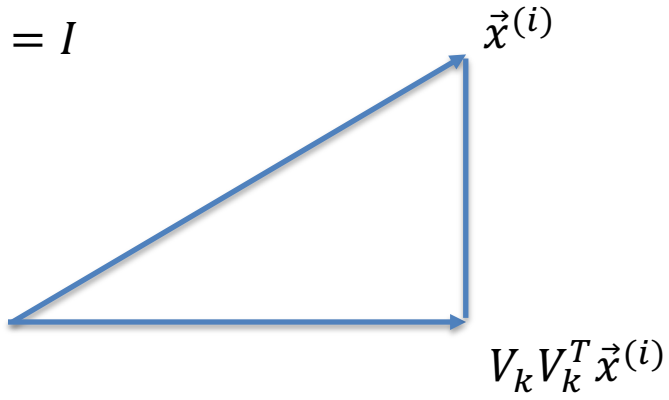
Variance of $\vec{z}^{(i)}$

Here, we assume the mean is zero.

$$\left\|\vec{z}^{(i)}\right\|^2 = \left\|V_k^T \vec{x}^{(i)}\right\|^2 = \left\|V_k V_k^T \vec{x}^{(i)}\right\|^2$$

$V_k V_k^T$ is the projection matrix, since $V_k{}^T V_k = I$

$$\left\|\vec{x}^{(i)}\right\|^2 = \left\|\vec{x}^{(i)} - V_k V_k^T \vec{x}^{(i)}\right\|^2 + \left\|V_k V_k^T \vec{x}^{(i)}\right\|^2$$



An alternative formulation of PCA matrix:

$$Z = XV = U\Sigma \mathrm{V}^{\mathrm{T}}\mathrm{V} = U\Sigma$$

Since $\widetilde{U}^T\widetilde{U} = I_d$, we also have.

$$Z = XV = \mathrm{XV}\widetilde{\Sigma}\,\widetilde{U}^T\widetilde{U}\widetilde{\Sigma}^{-1}$$

$$= XX^T\widetilde{U}\widetilde{\Sigma}^{-1}$$

$$\tilde{\Sigma}^{-1} = \begin{bmatrix} 1/\sigma_1 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & 0 & 1/\sigma_p & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{d\times d}$$

Or we can use

$$Z = XV = \mathrm{XV}\Sigma U^T U\Sigma^{-1}$$

$$= XX^T U\Sigma^{-1}$$

Here we use pseudo-inverse.

$$\Sigma^{-1} = \begin{bmatrix} 1/\sigma_1 & 0 & \cdots & 0 \\ 0 & \ddots & 0 & 0 \\ \vdots & 0 & 1/\sigma_p & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{d\times N}$$

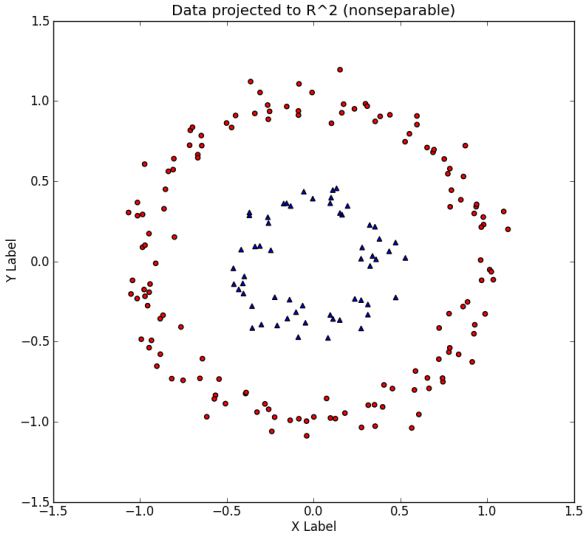
Here XX^T is the kernel matrix.

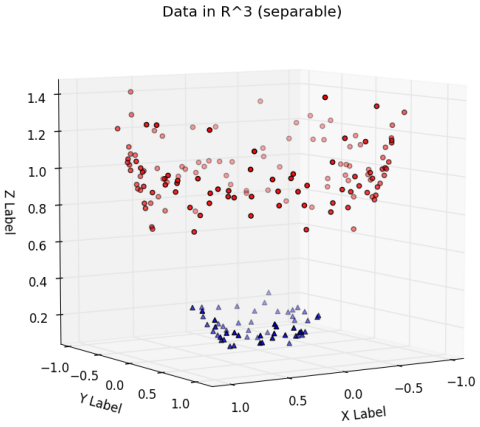
➢ Kernel PCA.

We can consider PCA in the new high dimensional space $\mathbb{R}^D$, by a **feature map:**

$$\phi\colon \mathbb{R}^d \to \mathbb{R}^D$$

$$\vec{x} \;\to\; \phi(\vec{x}) = \begin{bmatrix} \phi_1(\vec{x}) \\ \vdots \\ \phi_D(\vec{x}) \end{bmatrix}$$





$$\phi\colon \mathbb{R}^2 \to \mathbb{R}^3$$

$$\phi(\vec{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix}$$

Kernel Trick

Again, we will use **kernel trick** for the computation.

In construction of PCA, we already have the dual formula for the **new data matrix**:

$$Z = XX^T U\Sigma^{-1}$$

Here,

$$X = \begin{bmatrix} \vec{x}^{(1)^T} \\ \vdots \\ \vec{x}^{(N)^T} \end{bmatrix} \qquad X^T = \begin{bmatrix} \vec{x}^{(1)} & \dots & \vec{x}^{(N)} \end{bmatrix} \qquad XX^T = \begin{bmatrix} \vdots \\ \dots \vec{x}^{(i)^T} \vec{x}^{(j)} \dots \\ \vdots \end{bmatrix}$$

So, in the new feature space, points are replaced as

$$K\left(\vec{x}^{(i)}, \vec{x}^{(j)}\right) = \phi\left(\vec{x}^{(i)}\right)^T \phi\left(\vec{x}^{(j)}\right)$$

For example, we can use our popular kernels:

Gaussian kernel $K(\vec{x}, \vec{y}) \coloneqq \exp\left(-\frac{\|\vec{x}-\vec{y}\|^2}{2\sigma^2}\right)$

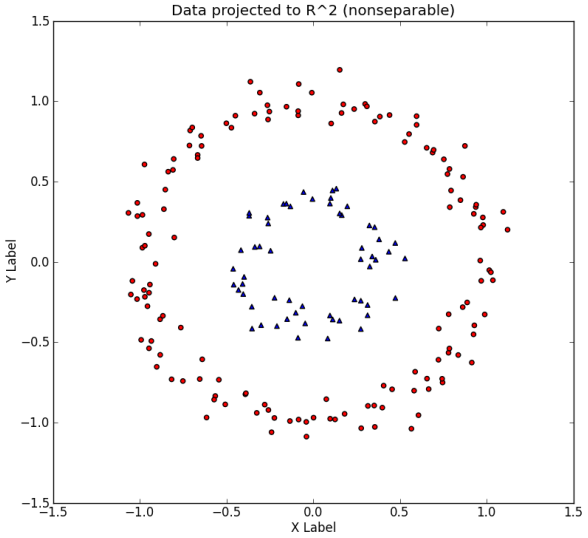
Polynomial Kernel $K(\vec{x}, \vec{y}) \coloneqq (\vec{x}^T\vec{y} + c)^n$

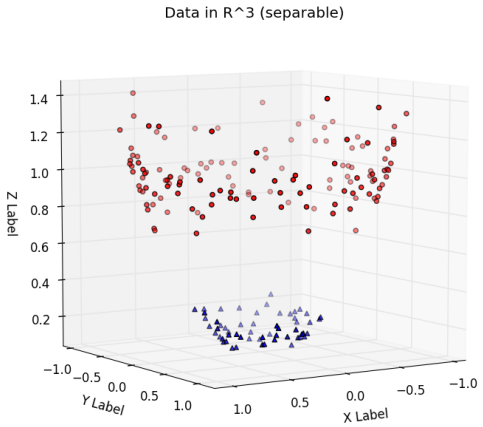
Sigmoid Kernel $K(\vec{x}, \vec{y}) \coloneqq \tanh(\eta\vec{x}^T\vec{y} + c)$

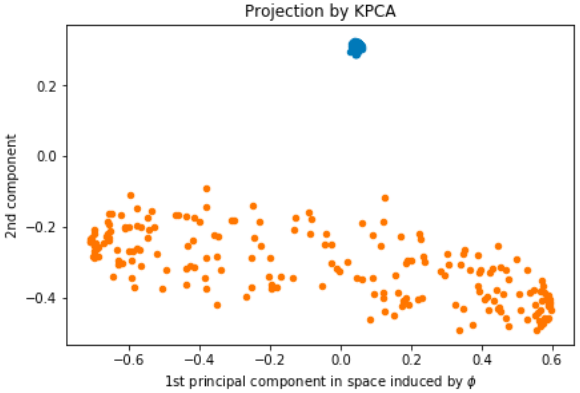
The **kernel PCA projection** matrix is

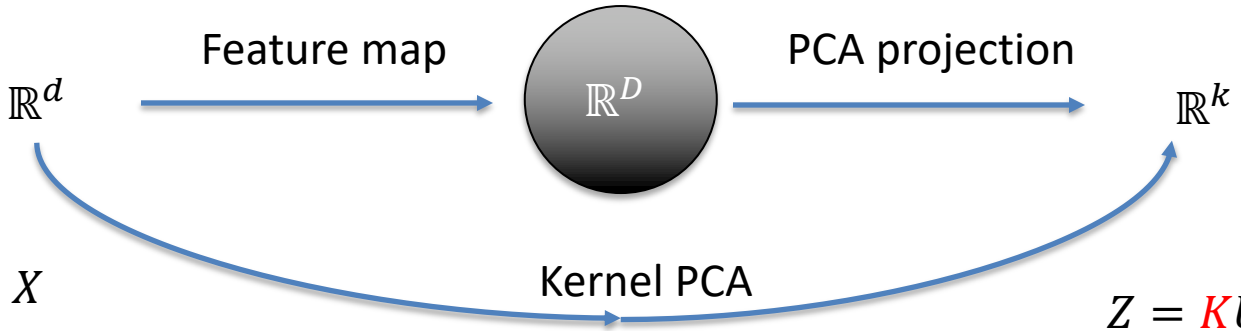
$$Z = KU\Sigma^{-1} \text{ where } K_{\text{ij}} \coloneqq K\left(\vec{x}^{(i)}, \vec{x}^{(j)}\right)$$

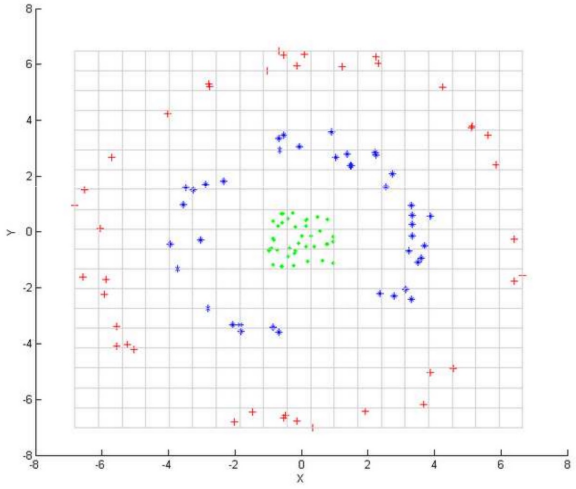
$$z_m^{(i)} = \sum_{j=1}^{N} \frac{1}{\sigma_m} u_{j\text{m}} K_{ij}$$

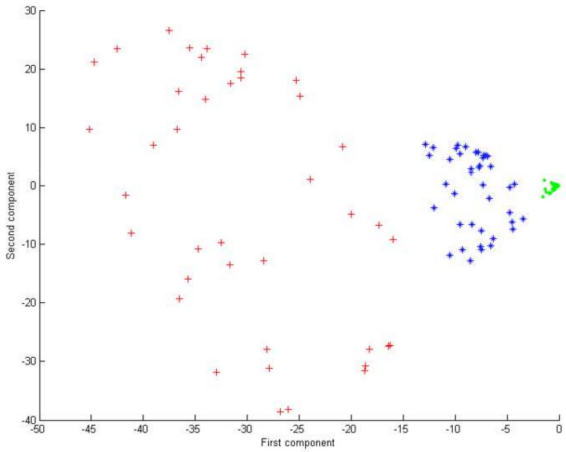

Data projected to R^2 (nonseparable)
Y Label
X Label


Data in R^3 (separable)
Z Label
Y Label
X Label


Projection by KPCA
2nd component
1st principal component in space induced by ϕ


Feature map
$\mathbb{R}^d$
$\mathbb{R}^D$
PCA projection
$\mathbb{R}^k$
X
Kernel PCA
$Z = KU\Sigma^{-1}$


X
Y


Second component
First component

Representation of $\vec{x}$ in PCA coordinate system

Similarly, for each point

$$\vec{z} = V^T\vec{x} = \Sigma^{-1}U^TU\Sigma V^T\vec{x} = \Sigma^{-1}U^TX\vec{x} = \Sigma^{-1}U^T\begin{bmatrix}\vec{x}^{(1)^T}\vec{x}\\ \vdots \\ \vec{x}^{(N)^T}\vec{x}\end{bmatrix}$$

$$= \begin{bmatrix}1/\sigma_1 & 0 & \cdots & 0\\ 0 & \ddots & 0 & 0\\ \vdots & 0 & 1/\sigma_p & 0\\ 0 & 0 & 0 & 0\end{bmatrix}_{d\times N}\begin{bmatrix}\vec{u}_1^{\mathrm{T}}\\ \vdots \\ \vec{u}_{\mathrm{N}}^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\vec{x}^{(1)^T}\vec{x}\\ \vdots \\ \vec{x}^{(N)^T}\vec{x}\end{bmatrix}$$

So, each $z_m = \sum_{j=1}^{N}\frac{1}{\sigma_m}u_{j\mathrm{m}}\,\vec{x}^{(j)^T}\vec{x}$

After applying Kernel tricks, we have the Kernel PCA coordinates

$$\vec{z} = \Sigma^{-1}U^T\begin{bmatrix}K(\vec{x}^{(1)},\vec{x})\\ \vdots \\ K(\vec{x}^{(N)},\vec{x})\end{bmatrix} \qquad z_m = \sum_{j=1}^{N}\frac{1}{\sigma_m}u_{j\mathrm{m}}\,K(\vec{x}^{(j)},\vec{x})$$

How to use (Kernel) PCA:

1. Start with our original data X
2. Apply PCA or Kernel PCA, we have our new data matrix Z
3. Using our new data matrix Z, we can do clustering, regression or classification. (Need labels $\vec{y}$)
4. After the regression, we can replace back to feature $\vec{x}$ in our original feature space.

➢ Sparse principal components

The **first** principal component $\vec{v}$

$$\underset{\|\vec{v}\|=1}{\operatorname{argmax}} \|X\vec{v}\|^2 = \underset{\|\vec{v}\|=1}{\operatorname{argmax}} (X\vec{v})^T X\vec{v} = \underset{\|\vec{v}\|=1}{\operatorname{argmax}} \vec{v}^T X^T X\vec{v}$$

Sparse principal components is a shrinkage method by solving

$$\underset{\|\vec{v}\|=1;\|\vec{v}\|_1 \le t}{\operatorname{argmax}} \quad \vec{v}^T X^T X\vec{v}$$

This is a **lasso** type l_1 constraint $\|\vec{v}\|_1 = \sum_{j=1}^{d} |v_j| \le t$

The constraint forces some of the components to be zero.

The procedure can be done inductively for PCA and for Sparse PCA,

Given $k-1$ principal component directions $\vec{v}_1, \ldots, \vec{v}_{k-1} \in \mathbb{R}^d$, (orthonormal), the k-th principal component directions $\vec{v}_k \in \mathbb{R}^d$ is given by

$$\vec{v}_k := \underset{\substack{\|\vec{v}\|=1 \\ \vec{v}^T \vec{v}_j = 0 \\ \text{for } j = 1, \ldots, k-1}}{\operatorname{argmax}} \vec{v}^T X^T X \vec{v}$$

The vector $X\vec{v}_k \in \mathbb{R}^N$ is *called the kth principal component* ***score*** *of X*

$\sigma_k := \sqrt{\vec{v}_k^T X^T X \vec{v}_k}$ is the ***variance*** *explained by* $\vec{v}_k$

$\vec{u}_k := \dfrac{X\vec{v}_k}{\sigma_k}$ is the normalized *k-th principal component* ***score***

All these coincide with the SVD $X^T X = U\Sigma V^T$

❑ Summary of PCA

PCA replaces X with a lower-dimensional approximation Z.

Matrix Z has N rows, but typically far fewer columns.

PCA is used for:

- Dimensionality reduction: replace X with a lower-dimensional Z.
- Outlier detection: if PCA gives poor approximation of $x^{(i)}$, could be outlier.
- Basis for linear models: use Z as features in regression model.
- Data visualization: display $z^{(i)}$ in a scatterplot.
- Factor discovering: discover important hidden "factors underlying data.

➢ Factor Analysis:

Factor Analysis tries to find low dimensional factors that explain higher dimensional information. The Factor Analysis model is a classical statistical model and is essentially a probabilistic version of PCA. It is a continuous latent variable models.

Assume our data $\{\vec{x}^{(i)}\}_{i=1}^{N}$ has **zero-mean** again. (Centered data)

In **factor analysis**, we try to explain some high dimensional observations $\vec{x}^{(i)} \in \mathbb{R}^d$ in in terms of some lower dimensional observed data $\vec{z}^{(i)} \in \mathbb{R}^k$. Especially, we want to write

$$\vec{x}^{(i)} = A\vec{z}^{(i)} + \vec{\epsilon}$$

where $\vec{\epsilon}$ is a vector of random variables.

Recall for any matrix X have singular value decomposition (SVD):

$$X = \widetilde{U}\tilde{\Sigma}V^T$$

$N \times d$ $\quad$ $N \times d$ $\quad$ $d \times d$ $\quad$ $d \times d$

Orthonormal columns $\quad$ Orthonormal columns

We assume $N \geq d$.

Define new matrices: $S \coloneqq \sqrt{N}\widetilde{U}$ $\quad$ $A^T \coloneqq \dfrac{\tilde{\Sigma}V^T}{\sqrt{N}}$

So $\quad X = SA^T$

Note that A has orthogonal columns since V has orthonormal columns.

This is the matrix relationship between dataset $\left\{\vec{x}^{(i)}\right\}_{i=1}^{N}$ and transformed dataset $\left\{\vec{s}^{(i)}\right\}_{i=1}^{N}$

Recall our data matrix

$$X = \begin{bmatrix} \vec{x}^{(1)^T} \\ \vdots \\ \vec{x}^{(N)^T} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & & x_{2d} \\ \vdots & & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & & x_d^{(2)} \\ \vdots & & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_d^{(N)} \end{bmatrix}$$

Each feature vector $\vec{x}^{(i)} = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{id} \end{bmatrix}$ is a sample from underlying Random Variable

$$\vec{X} = \begin{bmatrix} X_1 \\ \vdots \\ X_d \end{bmatrix}$$

The j-th column of the matrix X is an empirical (estimated) probability distribution of the underlying Random Variable X_j

In $X = SA^T$, write

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{bmatrix} \qquad S = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1d} \\ s_{21} & s_{22} & \cdots & s_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ s_{N1} & s_{N2} & \cdots & s_{Nd} \end{bmatrix}$$

Now consider A fixed,

$$X^T = AS^T$$

We have $\vec{x}^{(i)} = As^{(i)}$ for $i = 1,2, \ldots, N$

Now view each 'new' feature vector $\vec{s}^{(i)}$ as 'old' feature vector $\vec{x}^{(i)}$ in new coordinates.

This empirical relationship is an approximation of an underlying relationship between underlying feature vectors

$$\vec{X} = \begin{bmatrix} X_1 \\ \vdots \\ X_d \end{bmatrix} \quad \text{and} \quad \vec{S} = \begin{bmatrix} S_1 \\ \vdots \\ S_d \end{bmatrix}$$

of the form in random variables $\vec{X} = A\vec{S}$

The map $\mathbb{R}^d \to \mathbb{R}^d$ by $\vec{X} = A\vec{S}$ is a **linear** transformation from new feature $\vec{S} \in \mathbb{R}^d$ to feature $\vec{X} \in \mathbb{R}^d$.

The new coordinate system then has coordinate direction vectors which are the columns $\vec{a}_i$ of

$$A = [\vec{a}_1\ \vec{a}_2\ \ldots \vec{a}_d]$$

The relationship $\vec{X} = A\vec{S}$ is give by

$$\begin{bmatrix} X_1 \\ \vdots \\ X_d \end{bmatrix} = [\vec{a}_1\ \vec{a}_2\ \ldots \vec{a}_d] \begin{bmatrix} S_1 \\ \vdots \\ S_d \end{bmatrix} = \mathrm{S}_1\vec{a}_1 + \mathrm{S}_2\vec{a}_2 + \cdots + S_d\vec{a}_d$$

Here, $A^T \coloneqq \dfrac{\tilde{\Sigma}V^T}{\sqrt{N}}$ is **normalized principal components.** (Orthogonal columns)

So, $\vec{a}_i$ is just multiples of the PCA vectors of $\vec{v}_i$, i.e., the same coordinate system.

From PCA we know therefore that the coordinate values S_i (which are just multiples of PCA coordinates) are uncorrelated.

Thus we have a new coordinate system in the space of the variables $\vec{X}$, the the new **factor analysis** coordinates being $\vec{S}$.

Given a feature vector $\vec{X}$, its factor analysis coordinates $\vec{S}$ represent the values of the *fundamental or latent* underlying factors in the data set.

Factor analysis: $\vec{X} = A\vec{S} = \frac{1}{\sqrt{N}} V\Sigma\vec{S}$

PCA : $\vec{X} = V\vec{Z} = V\Sigma\vec{U}$

So, $\vec{U} = \frac{1}{\sqrt{N}}\vec{S}$

i.e. the PCA and factor analysis coordinates are essentially identical up to a factor $\frac{1}{\sqrt{N}}$

Note that the a_{ij} are called **factor loadings**, while the S_i are **latent variables**.

The components S_i are uncorrelated in the dataset.

The data matrix have the relation:

$$S = \sqrt{N}U$$

The **sample(empirical) covariance** of S is the identity matrix

$$Cov(S) = \frac{1}{N}S^T S = I$$

Thus **assuming** the underlying random variables $\vec{S}$ has covariance

$$Cov(\vec{S}) = I$$

❑ Non-uniqueness of factor analysis coordinates

Suppose R is any $d \times d$ orthogonal matrix.

$$\vec{X} = A\vec{S} = (AR^T)\left(R\vec{S}\right) = A^*\vec{S}^*$$

So, there are lots of choices of latent variables.

In addition:

$$Cov\left(\vec{S}^*\right) = RCov(S)R^T = I$$

A has orthogonal columns, i.e., $A^T A$ is a diagonal matrix.

But A^* does not have orthogonal columns. So, the new coordinate $\vec{S}^*$ go with new coordinate axis $\vec{a}_1, \ldots, \vec{a}_d$, that generally *crooked* (non-orthogonal).

Factor analysis can't even identify factor directions, because of the rotation R.

❑ Truncation.

As in the truncated PCA,

Define new matrices: $S_k := \sqrt{N}\widetilde{U}_k \qquad A^T := \dfrac{\widetilde{\Sigma}_k V_k^T}{\sqrt{N}}$

$N \times k \qquad\qquad k \times d$

So $X \approx S_k {A_k}^T$ gives optimal approximation just using k eigenvalues.

As before we have a corresponding relationship for the underlying random variables:

$$\vec{X} \approx A_k \overrightarrow{S_k}$$

$$\begin{bmatrix} X_1 \\ \vdots \\ X_d \end{bmatrix} \approx [\vec{a}_1\ \vec{a}_2\ \ldots \vec{a}_k] \begin{bmatrix} S_1 \\ \vdots \\ S_k \end{bmatrix} = \mathrm{S}_1\vec{a}_1 + \mathrm{S}_2\vec{a}_2 + \cdots + S_k\vec{a}_k$$

We assume $\vec{X} = A_k \overrightarrow{S_k} + \vec{\epsilon}$ More precise,

$$X_1 = a_{11}S_1 + a_{12}S_2 + \cdots + a_{1k}S_k + \epsilon_1$$

$$\vdots$$

$$X_d = a_{d1}S_1 + a_{d2}S_2 + \cdots + a_{dk}S_k + \epsilon_d$$

where we assume the noise ϵ_i are **independent**. They are also independent from S_j.

So now we have a **truncated factor analysis:**

Covariance matrix of the random variable $\vec{X}$ is

$$Cov(\vec{X}) = Cov(A_k \overrightarrow{S_k}) + Cov(\vec{\epsilon}) = A_k Cov(S_k) A_k^T + Cov(\vec{\epsilon})$$

$$= A_k A_k^T + D_\epsilon$$

$$Cov(S_k) = \mathrm{I} \qquad Cov(\vec{\epsilon}) = D_\epsilon = \begin{bmatrix} V(\epsilon_1) & 0 & \cdots & 0 \\ 0 & V(\epsilon_2) & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & V(\epsilon_d) \end{bmatrix}$$

The columns of A are the factor loadings. S_i are still the **latent variables**.

$$Cov(\vec{X}) = A_k A_k^T + D_\epsilon$$

But the previous **non-uniqueness** issue is still a problem.

➢ Independent Components Analysis

Independent component analysis (ICA) is a more recent approach (1984). Under certain assumptions it can identify factors.

As in PCA/FA, ICA is a matrix factorization method,

$$X \approx SA^T$$

Assume that $X = SA^T$ for a true A with $k = d$. (In PCA, we assume $k < d$.)

There are 3 issues stopping us from finding "true" A.

1. Label switching. (Multiply a permutation matrix)
2. Scaling (multiply a diagonal matrix)
3. Rotation. (Multiply a orthogonal matrix) $\vec{X} = A\vec{S} = (AR^T)\left(R\vec{S}\right) = A^*\vec{S}^*$

We only care about the direction. So, only rotation is a big problem.

We now search for independent as opposed to just uncorrelated components; this will typically give uniqueness. (In several applications, the the source signals are independent of each other.)

When we found the principal component variables

$$\vec{S} = \begin{bmatrix} S_1 \\ \vdots \\ S_d \end{bmatrix}$$

We rotated the original $\vec{X}$ coordinate system such that $Cov(\vec{S}) = I$, i.e., the S_i are uncorrelated, so $E(S_i S_j) = E(S_i)E(S_j)$ for $i \neq j$.

But, they may not be independent, for example, $E(S_i^2 S_j^2) \neq E(S_i^2)E(S_j^2)$

If we have **independence** of S_j, that would determine all cross-moment.

But: if the variables are Gaussian then just the second moments (i.e. just their correlations) determine them. We still have factor analysis non-uniqueness problem.

Thus typically need to assume (at least hope, for this algorithm) that S_j that are **independent** and **non-Gaussian**.

Consider a **prior** that assumes the S_j are independent,

$$p(\vec{S}) = \prod_{j=1}^{k} p(S_j)$$

In PCA and FA, we assume that S_j is $N(0,1)$.

If $p(\vec{S})$ is rotation-invariant, $p(R\vec{S}) = p(\vec{S})$, then it must be Gaussian. (why?)

The (non-intuitive) magic behind ICA:
If the priors are all non-Gaussian, it isn't rotationally symmetric.

Implication: we can identify factors A if at most 1 factor is Gaussian.
Up to permutation/sign/scaling (other rotations change distribution).

❑ ICA procedure

Assume the random vector $\vec{X}$ has been **pre-whitened**, i.e.,

$$\vec{X} \to B\vec{X}$$

such that $Cov(B\vec{X}) = B\left(Cov(\vec{X})\right)B^T = I$. This can be done with SVD.

Thus we are assuming we already have moved to **uncorrelated** factors like in $\vec{S}$. We also assume that $\vec{X}$ has **mean zero.**

So saying $\vec{X}$ has been pre-whitened is saying it has been replaced by its factor analysis variable $\vec{S}$, which we know is uncorrelated and has covariance matrix I.

Now view all in terms of the underlying random variables X_j

❑ Entropy

If Y is a random variable, we will for simplicity assume it has a density function $f(y)$, and then we define its **entropy** (uncertainty) as

$$H(Y) \coloneqq -\int f(y) \ln f(y)\, dy$$

If Y is restricted to interval $[a, b]$, then Y has **maximum entropy** if Y is uniform on $[a, b]$.

Y has smallest entropy if Y is a point mass at some (i.e., if $P(Y = c) = 1, \text{then } H(Y) = 0$)

Note: Gaussian variables are the ones with maximum entropy, conditioned on having mean 0 and variance 1.

If $\vec{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_d \end{bmatrix}$, then **mutual information** of the components of

$$I(\vec{Y}) := \left[\sum_{i=1}^{d} H(Y_i)\right] - H(\vec{Y})$$

This is the **Kullback-Leibler distance** between the joint density $p(\vec{y})$of Y and the product of the marginals (independent version)

$$\prod_{i=1}^{d} p_{Y_i}(y_i)$$

Recall we had an underlying random vector $\vec{X} = \begin{bmatrix} X_1 \\ \vdots \\ X_d \end{bmatrix}$ which we pre-whitened to have uncorrelated components.

This is equivalent to extracting (using factor analysis) a random vector with uncorrelated components via what we did earlier:

$$\vec{X} = A\vec{S}$$

with $\vec{S}$ having uncorrelated components.

So, $\vec{S} = A^{-1}X \equiv B\vec{X}$ in above notation. (We assume A is invertible.)

So (as stated above) $\vec{X}$ has been replaced by $\vec{S}$ (we still call it $\vec{X}$) and has uncorrelated components.

Then any orthogonal transformation A will keep the components uncorrelated.

Goal now: Find orthogonal transformation A that makes the (new) components of $\vec{Y} = A^T \vec{X}$ the most(maximally) independent.

Still can treat A as matrix containing loadings as in factor analysis.

To get most independent components of $\vec{Y}$ we want to minimize mutual information:

$$I(\vec{Y}) := \left[\sum_{i=1}^{d} H(Y_i)\right] - H(\vec{Y})$$

$$= \left[\sum_{i=1}^{d} H(Y_i)\right] - H(\vec{X}) - \ln|\det(A)|$$

$$= \left[\sum_{i=1}^{d} H(Y_i)\right] - H(\vec{X})$$

since A is orthogonal

Thus need to **minimize** sum of entropies of components:

$$\left[\sum_{i=1}^{d} H(Y_i)\right]$$

The idea: We claim that if there exist any (approximately) independent component Y_i of our feature vectors in some new coordinate system (call it the 'independent' coordinate system), then the choice of Y_i will form be the most '**non-Gaussian**' components.

Thus to find the 'independent' coordinate system we should choose it as the one in which the components are 'least' Gaussian.
Note however that if the independent components are already (too close to) Gaussian, this will not work.

❑ Fast ICA –Another way (algorithm) to construct ICA

Define **negentropy**

$$J(Y_i) \coloneqq H(Z_i) - H(Y_i)$$

where Z_i is Gaussian with same mean and variance as Y_i.

We want to maximize $J(Y_i)$ to make Y_i far from Gaussian. (Maximization of Non-Gaussianity.)

FastICA - algorithm which approximates

$$J(Y_i) \approx \left[E\left(G(Y_i) - E\big(G(Z_i)\big)\right)\right]^2$$

with $G(u) \coloneqq \ln\cosh u$

Hyperbolic cosine:

$$\cosh u \coloneqq \frac{e^x + e^{-x}}{2}$$

[Hyvarinen and Oja, 2000] proved above Approximation. (FastICA)

Note that for actual datasets $\{\vec{x}^{(i)}\}_{i=1}^{N}$ the expectations as integrals are replaced by sums over the datasets.

There are **other methods/algorithms** to perform ICA:

- **Infomax-based** ICA was described by Bell and Sejnowski, and Nadal and Parga in 1995.
- **Kernel-independent component analysis**
- Joint Approximation Diagonalization of Eigen-matrices **(JADE)**

Lots of applications for each algorithm.

Application: Cocktail Party Problem.

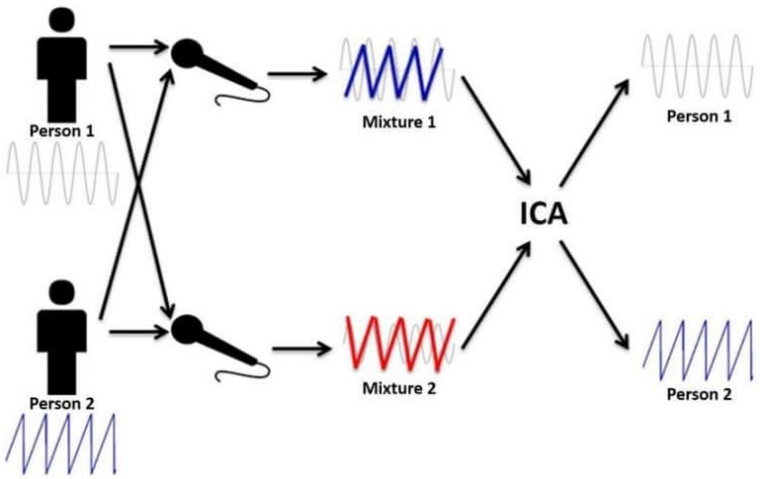
Multiple microphones recording multiple sources. Each microphone gets different mixture of the sources.

Goal is to reconstruct sources (factors) from the measurements.

$\vec{s}$ A $\vec{x}$ A^{-1} $\vec{s}$

Source signal:

Independent,
Non-Gaussian



Data is $N \times 2$ matrix X.

Goal: Find $X = SA^T$, or $\vec{x} = A\vec{s}$ with loading(mixing matrix) A.

$\vec{s}$ is the latent variable (source vector)

Up to permutation/scaling/sign (Order of people, voice volume), we want to find unique A.

We need to prewhite(sphere) the data first. (Zero Center, Identity Coariance)

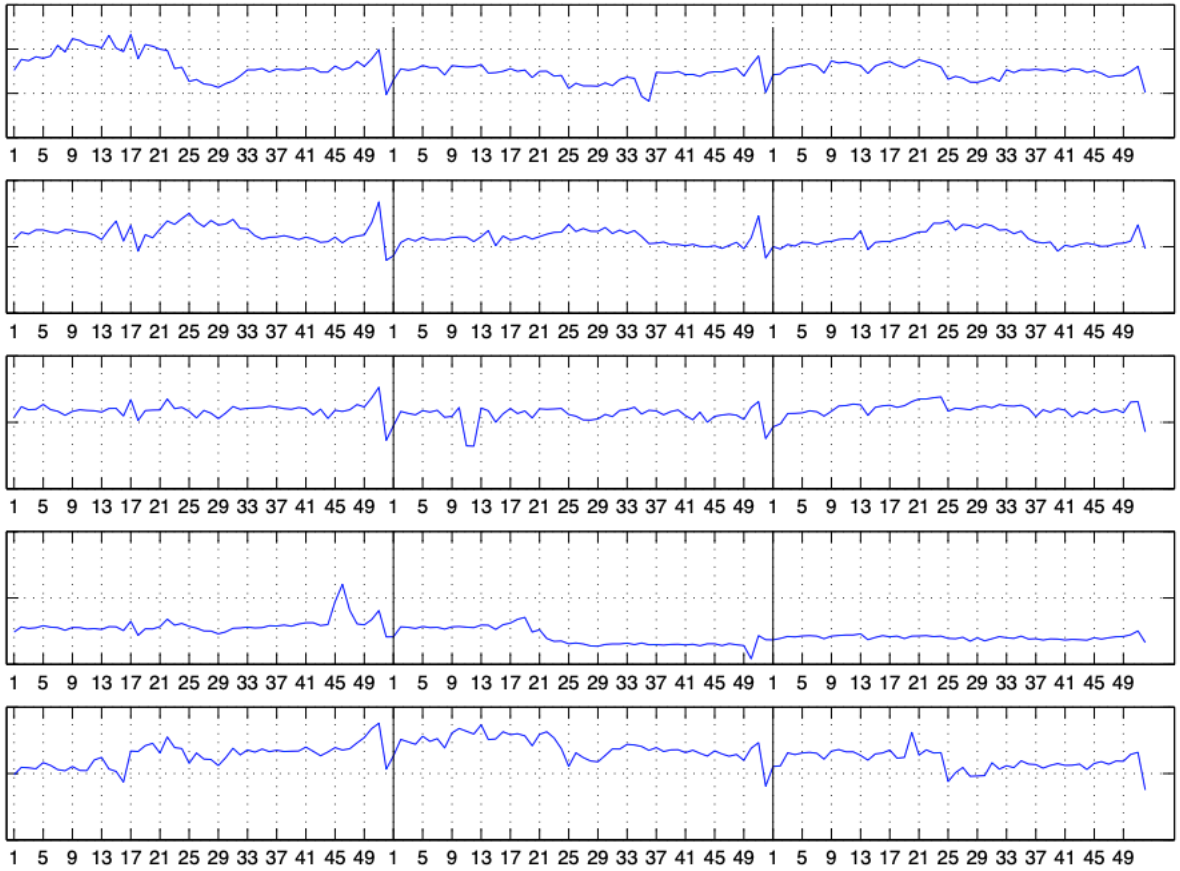
To find A and source data S

Then apply an algorithm to perform ICA (i.e., use FastICA)

ICA: A Demo http://research.ics.aalto.fi/ica/icademo/

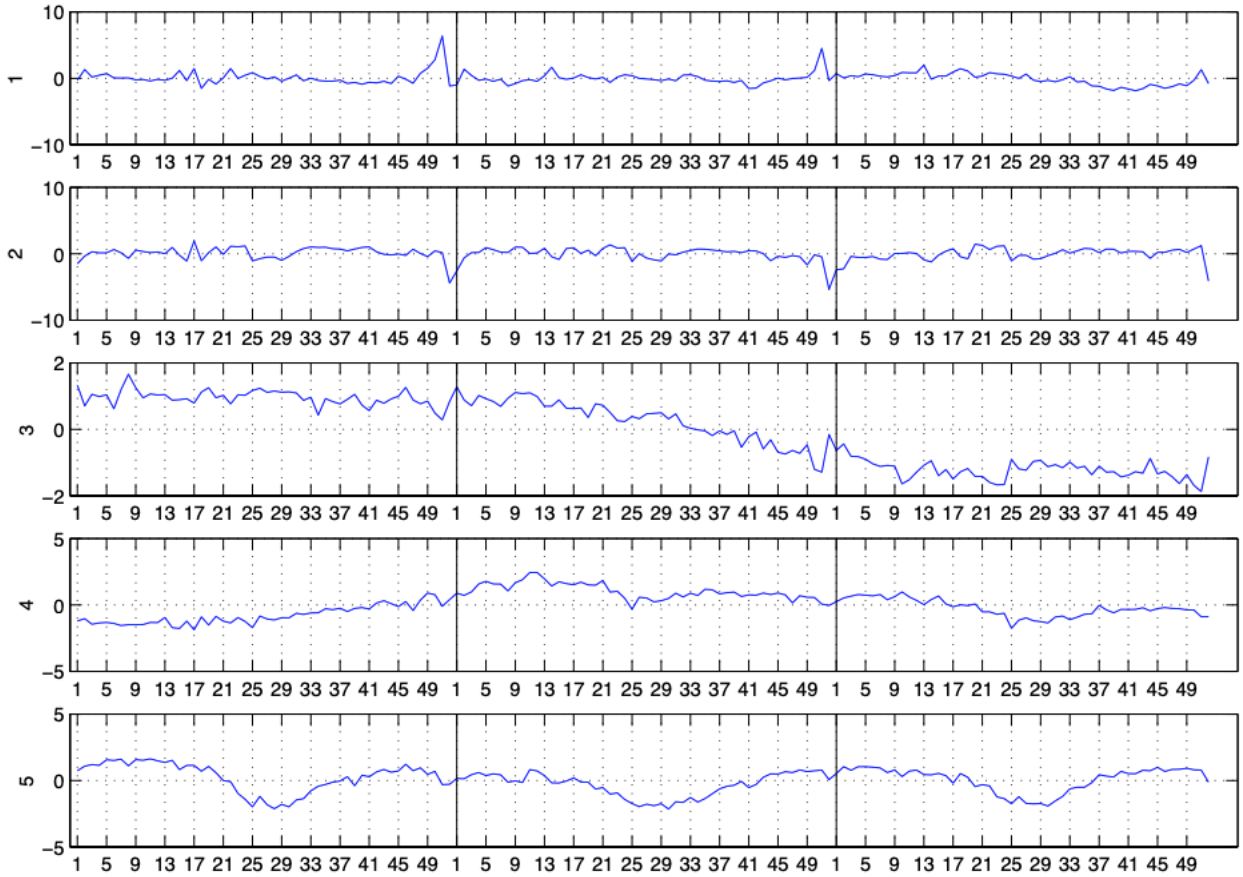
Example: ICA on Retail Purchase Data

Cash flow from 5 different stores over 3 years:



https://www.cs.jhu.edu/~ayuille/courses/Stat161-261-Spring14/HyvO00-icatut.pdf

Factors found using ICA.
1-2 reflect “holiday season”, 3-4 are year-to-year, and 5 is summer dip in sales.



Example: Handwritten digits [Hastie]

Handwritten 3's – dataset $\left\{\vec{x}^{(i)}\right\}_{i=1}^{658}$ of digitized 3's on a 16 by 16 grid, so

$$\vec{x}^{(i)} \in \mathbb{R}^{256}$$

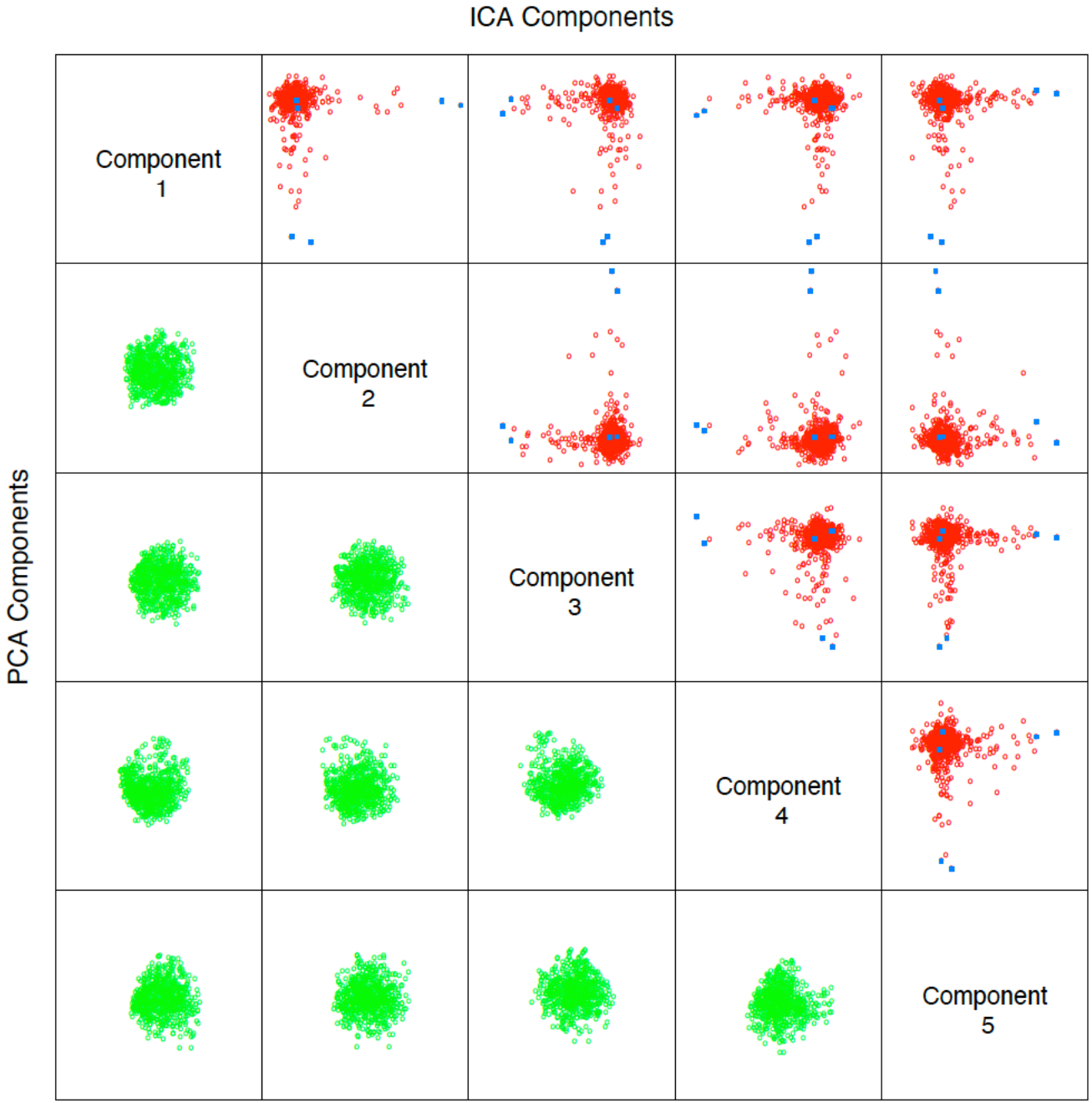
Identify first 5 principal components (normalized to have same length and width -- don't get narrower).

Compare to first 5 independent components:

A comparison of the first five ICA components computed using `FastICA` (above diagonal) with the first five PCA components(below diagonal). Each component is standardized to have unit variance.

Each plot is a two-dimensional projection from a 256-dimensional space.

Note ICA departures from Gaussianity in the independent components.

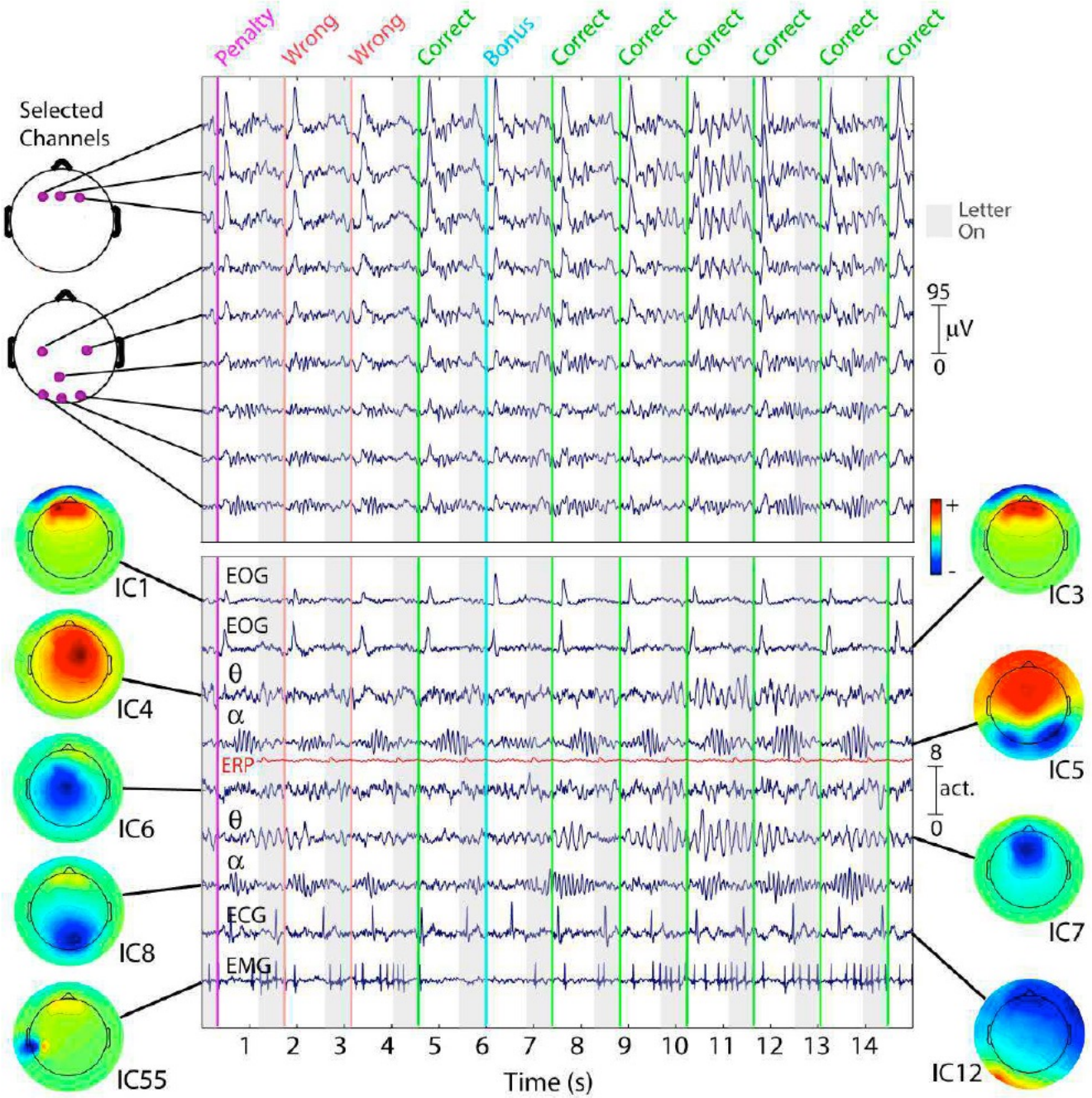

ICA Components
PCA Components
Component 1
Component 2
Component 3
Component 4
Component 5

Example: *ElectroEncephaloGraphic (***EEG) [Hastie]**

15 seconds of EEG output from 9 electrodes turned into feature vectors; single subject. Get a collection $\{\vec{x}^{(i)}\}_{i=1}^{n}$ time snapshots of the nine dimensional signal; thus $\vec{x}^{(i)} \in \mathbb{R}^9$ and $n = 15{\times}1000$ (e.g. assuming 1000 measurements per second).

Find independent components and their time signatures. This is a single subject over a single 15 second period, with the original signals above and the ICA components below:

Feature: Fifteen seconds of EEG data at nine (of 100) scalp channels (top panel), as well as nine ICA components (lower panel). While nearby electrodes record nearly identical mixtures of brain and non-brain activity, ICA components are temporally distinct. The colored scalps represent the ICA unmixing coefficients a_j as a heatmap, showing brain or scalp location of the source.


Penalty
Wrong
Wrong
Correct
Bonus
Correct
Correct
Correct
Correct
Correct
Correct
Selected Channels
Letter On
95
μV
0
EOG
EOG
θ
α
ERP
θ
α
ECG
EMG
8
act.
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Time (s)
IC1
IC4
IC6
IC8
IC55
IC3
IC5
IC7
IC12

Note that the above 9 dimensional feature vectors giving the EEG signatures at each time point can be assumed collected 1000 times per second for 15 seconds, yielding 15000 feature vectors $\vec{x} \in \mathbb{R}^9$.

The top 9 time series are these 9 features before ICA transformation; The bottom 9 time series are the 9 'independent' features after ICA transformation.

Note that there are loadings as well for ICA -- the heat map in the diagram indicates the loadings of the independent components.

More applications: see Wikipedia
https://en.wikipedia.org/wiki/Independent_component_analysis

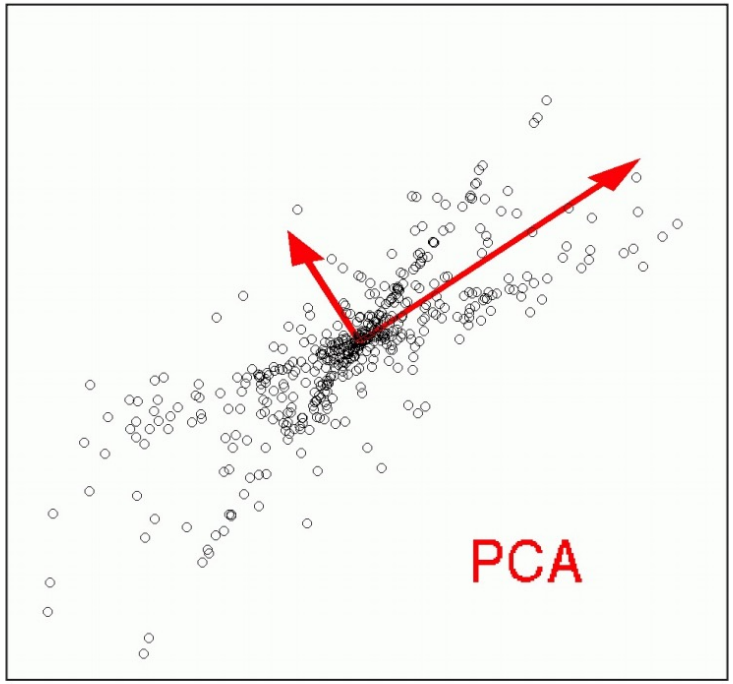
PCA v.s. Factor Analysis v.s. ICA

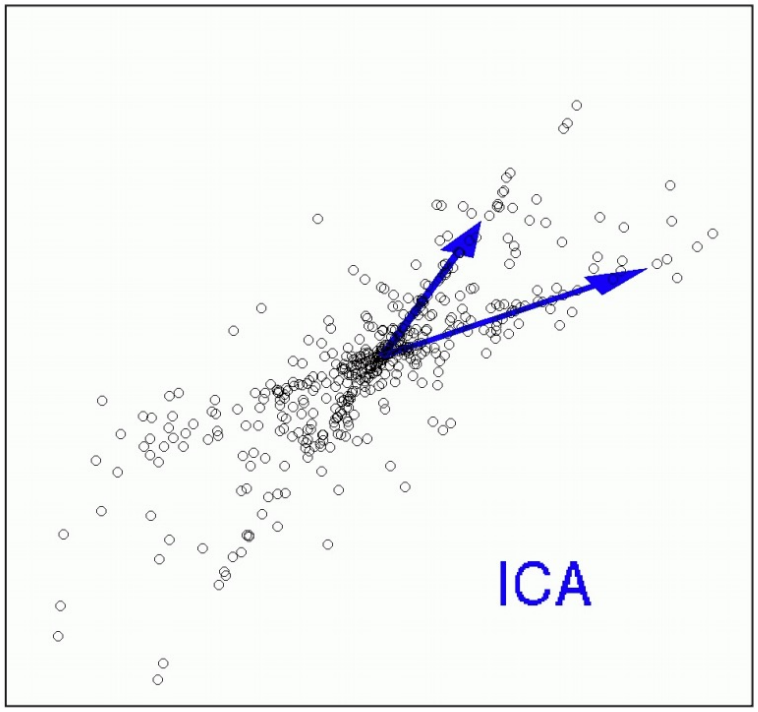
The most common technique for factor analysis is Principle Component Analysis(PCA).

Factor analysis extends PCA, with different noise in each dimension. It is in fact so common that "factor analysis" usually means "factor analysis that isn't PCA."

ICA: allows identifying non-Gaussian Latent factors.

PCA v.s. ICA





Z

S

PCA tries to find a transformation of a matrix X to a new set of latent variables $\vec{Z}$ such that the new latent variables are **orthogonal**.
ICA finds sources S which are **independent**.

➢ Probabilistic PCA (*Generative model approach*)

We have discussed two equivalent methods of defining PCA. Choose latent-factors V such that data matrix $X \approx XVV^T$, or features $\vec{Z} = V^T\vec{X}$

1. minimize reconstruction error, or equivalently,
2. maximize variance

Now, let us formulate PCA as a generative model and thereby glean some intuition for why the eigenvectors of the empirical covariance matrix are good principal components.

With zero-mean (centered data) $\left\{\vec{x}^{(i)}\right\}_{i=1}^{N}$, in PCA, we assume that

$$\vec{x}^{(i)} \approx V\vec{z}^{(i)}$$

There was no noise in the reconstruction error formulation; the error only stemmed from the fact that we had fewer principal components than dimensions $(k < d)$. Hence, in the generative model, $\vec{x}$ would be deterministic given $\vec{z}^{(i)}$ and V.

In probabilistic PCA we can relax this assumption a bit and add isotropic Gaussian noise, **assume** that

$$\vec{x}^{(i)}|\vec{z}^{(i)} \sim N(V\vec{z}^{(i)}, \sigma^2 I) \qquad \text{and } \vec{z}^{(i)} \sim N(0, I)$$

A different perspective is that these models assume

$$\vec{x}^{(i)} = V\vec{z}^{(i)} + \vec{\epsilon}$$

such that $\vec{z}^{(i)} \sim N(0, I)$ and $\vec{\epsilon} \sim N(0, \sigma^2 I)$

It turns out that in the limit of $\sigma^2 \to 0$, the **MLE** estimate V and $\vec{z}^{(i)}$ recovers the classical PCA solution. (Show next)

(In general, we can actually use any Gaussian density for $\vec{z}$)

We can treat $\vec{z}^{(i)}$ as nuisance parameters integrate over them in likelihood, take the Marginal distribution of the joint distribution:

$$p\left(\vec{x}^{(i)}\middle|V\right) = \int_{\vec{z}^{(i)}} p\left(\vec{x}^{(i)}, \vec{z}^{(i)}\middle|V\right) d\vec{z}^{(i)}$$

Notice: Marginal of Gaussian is Gaussian. We don't have to calculate the integral.

From the assumptions we have

$$p(\vec{x}, \vec{z}|V) = p(\vec{x}|\vec{z}, V)\, p(\vec{z}|V)$$

$$= p(\vec{x}|\vec{z}, V)\, p(\vec{z})$$

$$\propto \exp\left(-\frac{(\vec{x} - V\vec{z})^T(\vec{x} - V\vec{z})}{2\sigma^2}\right)\exp\left(-\frac{\vec{z}^T\vec{z}}{2}\right)$$

$$= \exp\left(-\frac{(\vec{x} - V\vec{z})^T(\vec{x} - V\vec{z})}{2\sigma^2} - \frac{\vec{z}^T\vec{z}}{2}\right)$$

Re-write the exponent as a quadratic form,

$$p(\vec{x}, \vec{z}|V) \propto \exp\left(-\frac{1}{2} [\vec{x}^T \;\; \vec{z}^T] \begin{bmatrix} \frac{1}{\sigma^2} I & -\frac{1}{\sigma^2} V \\ -\frac{1}{\sigma^2} V^T & \frac{1}{\sigma^2} V^T V + I \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{z} \end{bmatrix} \right)$$

So, the covariance matrix of $(\vec{x}, \vec{z}|V)$ is Σ such that

$$\Sigma^{-1} = \begin{bmatrix} \frac{1}{\sigma^2} I & -\frac{1}{\sigma^2} V \\ -\frac{1}{\sigma^2} V^T & \frac{1}{\sigma^2} V^T V + I \end{bmatrix}$$

So, $\Sigma = \begin{bmatrix} VV^T + \sigma^2 I & V \\ V^T & I \end{bmatrix}$

which gives that solution to integrating over $\vec{z}$ is $\left(\vec{x}^{(i)} \middle| V\right) \sim N(0, VV^T + \sigma^2 I)$

Negative Log Likelihood of the observed data is

$$-\log\big(p(X|V)\big) = \frac{N}{2}Tr(SA) - \frac{N}{2}\log|A| + constant$$

where $A = (VV^T + \sigma^2 I)^{-1}$ and $S = \frac{1}{N}X^T X$ is the sample covariance.

In PCA, we also assume that V is a $d \times k$ matrix with orthonormal columns, i.e., $V^T V = I_k$ and VV^T is the $d \times d$ projection matrix.

$$\det(VV^T + \sigma^2 I) = (1+\sigma^2)^r (\sigma^2)^{N-r} = constant$$

$$A = (VV^T + \sigma^2 I)^{-1} = \frac{1}{\sigma^2} I - \frac{1}{\sigma^2(\sigma^2+1)} VV^T$$

MLE is the same as **minimize** $Tr(SA)$, which is the same as **maximize**

$$Tr(VV^T X^T X) = Tr(V^T X^T X V) = \|XV\|^2 = \text{data variance}$$

(By linear algebra properties)

❑ Probabilistic Factor Analysis

It turns out that both PCA and FA can be viewed as special cases of the generative model described above. In factor analysis, however, we have the following model:

$$\vec{z}^{(i)} \sim N(0, I)$$

$$\vec{x}^{(i)} | \vec{z}^{(i)} \sim N(V\vec{z}^{(i)}, D)$$

where D is a diagonal matrix.

The difference with PCA is that you can have a noise variance for each dimension.

Similarly as Probabilistic PCA calculation, we have

$$\vec{x}^{(i)} | V \sim N\left(\vec{0}, VV^T + D\right)$$

Factor Analysis has extra degrees of freedom in variance of individual variables.

Given training data, we can write down the

Negative Log Likelihood of the observed data:

$$-\log\big(p(X|V)\big) = \sum_{i=1}^{N} \frac{1}{2}\big(\vec{x}^{(i)}\big)^T (VV^T + D)^{-1}\vec{x}^{(i)} + N\log|VV^T + D| + constant$$

Minimize the above is not easy.

We can apply EM algorithm to factor analysis.

General case with mean

In the whole section, we assume our data is centered, i.e., with mean zero. In general, we assume

$$\vec{x}^{(i)} = V\vec{z}^{(i)} + \vec{\mu} + \vec{\epsilon}$$

such that $\vec{z}^{(i)} \sim N(0, I)$ and $\vec{\epsilon} \sim N(\vec{\mu}, D)$

Or, equivalently

$$\vec{x}^{(i)} | \vec{z}^{(i)} \sim N(V\vec{z}^{(i)} + \vec{\mu}, D)$$

In mixture models, we have a **discrete** latent variable z_i.
In mixture of Gaussians, if you know the cluster z_i then $p\left(x^{(i)} \middle| z_i\right)$ is a Gaussian.

In latent-factor models, we have **continuous** latent variables $z^{(i)}$:
In probabilistic PCA, if you know the latent-factors $z^{(i)}$ then $p\left(x^{(i)} \middle| z^{(i)}\right)$ is a Gaussian.

Learning: Use EM algorithm to learn the parameters $V \; and \; D$, and infer the latent factor $\vec{z}$

In the E step we update the latent factors given the current weights and noise matrices, and in the M step we set the weights and noise matrices to their MAP estimates under the current factors.
One benefit of the EM algorithm is that it is easy to handle missing data. We simply estimate it during the E step.

References:

[Hastie]: Sec14.5-14.7

MATLAB: https://www.mathworks.com/help/stats/dimensionality-reduction.html

https://github.com/UMD-ISL/Matlab-Toolbox-for-Dimensionality-Reduction

https://lvdmaaten.github.io/drtoolbox/

Python: https://scikit-learn.org/stable/modules/decomposition.html

https://scikit-learn.org/stable/modules/manifold.html

MATLAB example: Analyze Stock Prices Using Factor Analysis

https://www.mathworks.com/help/stats/analyze-stock-prices-using-factor-analysis.html

FastICA

MATLAB: **FastICA**: http://research.ics.aalto.fi/ica/fastica/

- FastICA in R: https://cran.r-project.org/web/packages/fastICA/
- FastICA in C++ (part of IT++ package) https://itpp.sourceforge.net/devel/fastica_8cpp.html
- FastICA in Python as part of MDP package https://mdp-toolkit.sourceforge.net/
- FastICA in Python as part of scikit-learn package https://scikit-learn.org/dev/auto_examples/decomposition/plot_ica_blind_source_separation.html

Further: References:

Independent Component Analysis
A. Hyvärinen, J. Karhunen, E. Oja http://research.ics.aalto.fi/ica/book/

Paper: Independent Component Analysis: Algorithms and Applications
https://www.cs.helsinki.fi/u/ahyvarin/papers/NN00new.pdf

Independent Component Analysis A Tutorial
https://www.cs.jhu.edu/~ayuille/courses/Stat161-261-Spring14/HyvO00-icatut.pdf