Dimensionality Reduction
and
Principle Component Analysis

Felix Brockherde 2015, TU Berlin and MPI Halle,
Stefan Haufe 2016, TU Berlin

This Lecture

1. Dimensionality reduction

2. Principle Component Analysis

    1. What are Principle Components?

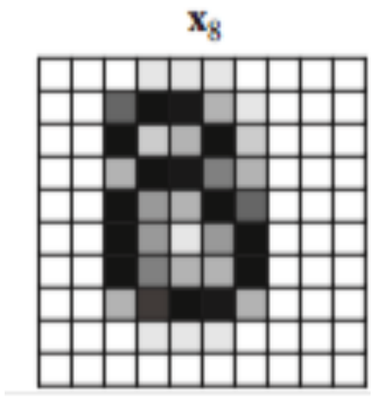
    2. How to find/calculate them

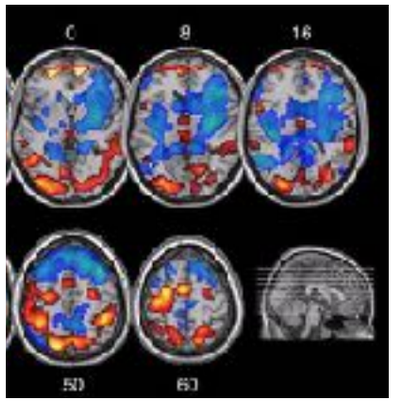
        1. Lagrange Multipliers

    3. What can we do with them? / Applications

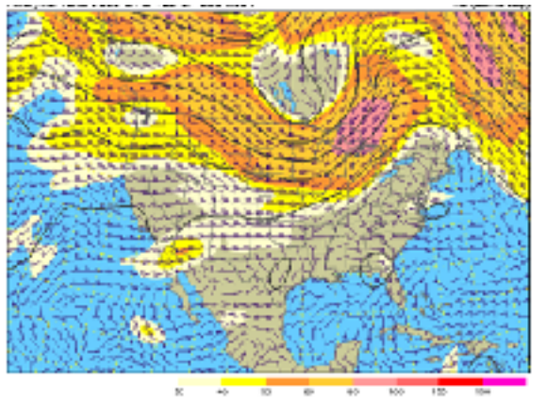
Curse of dimensionality

In many machine learning problems, the data are high-dimensional





From Limb and Braun, 2008

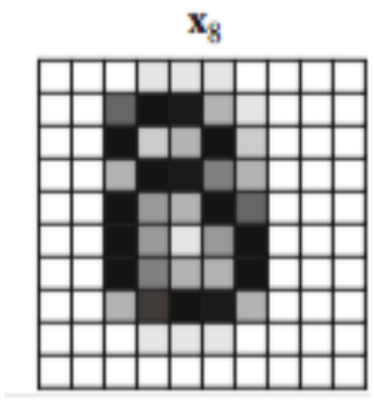


University of Colorado

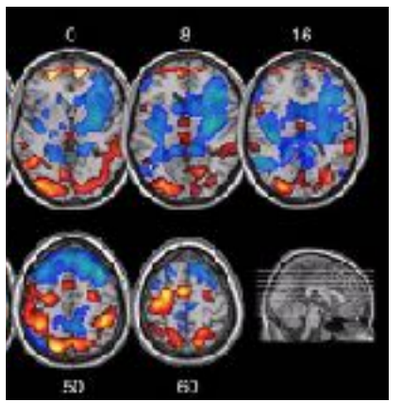
IEEE Spectrum

Curse of dimensionality

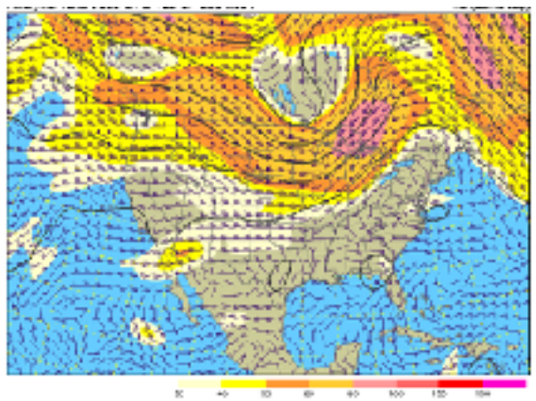
In many machine learning problems, the data are high-dimensional



From Limb and Braun, 2008



University of Colorado



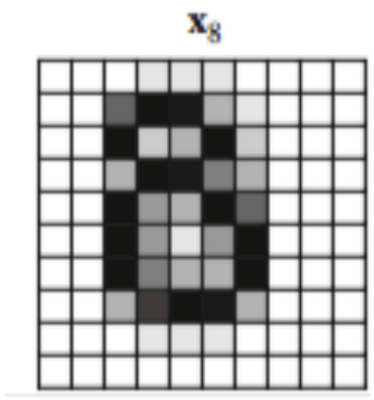
IEEE Spectrum

Standard regression/classification techniques can become

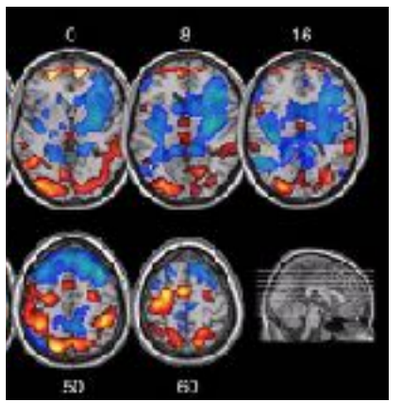
- ill-defined for M >> N
- ill-conditioned/numerically unstable even for M < N

Curse of dimensionality

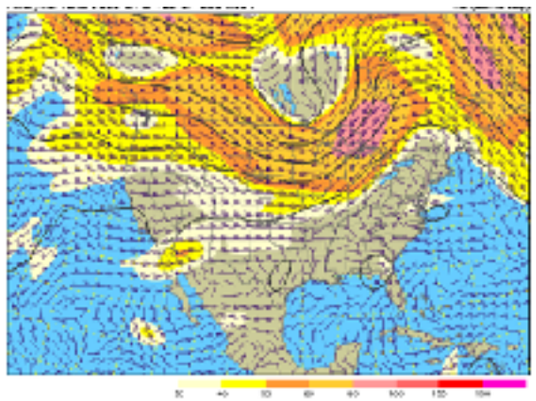
In many machine learning problems, the data are high-dimensional



From Limb and Braun, 2008



University of Colorado



IEEE Spectrum

Standard regression/classification techniques can become

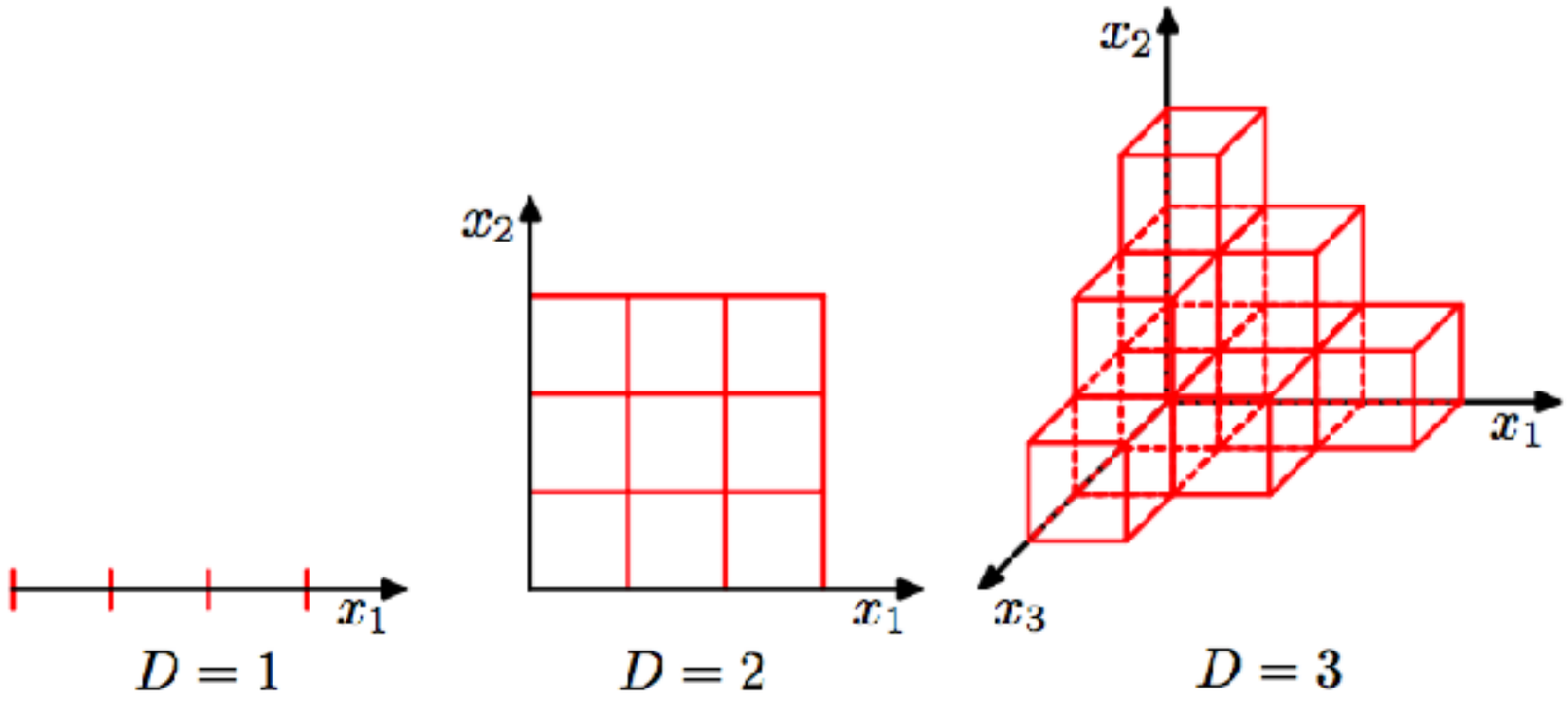
- ill-defined for M >> N
- ill-conditioned/numerically unstable even for M < N

$$\mathbf{w} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

Singular, ill-conditioned

Curse of dimensionality

When the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse



The amount of data needed for a reliable result often grows exponentially with the dimensionality

Regularization

Idea: impose constraints on the parameters to stabilize solution.

One way: introduce prior probability

Regularization

Idea: impose constraints on the parameters to stabilize solution.

One way: introduce prior probability

Linear regression setting: $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}\ , \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)\ , \hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$

Regularization

Idea: impose constraints on the parameters to stabilize solution.

One way: introduce prior probability

Linear regression setting: $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}\ , \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)\ , \hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$

Prior: $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \sigma_{\boldsymbol{\beta}}^2 I)$

Regularization

Idea: impose constraints on the parameters to stabilize solution.

One way: introduce prior probability

Linear regression setting: $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}\ , \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)\ , \hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$

Prior: $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \sigma_{\boldsymbol{\beta}}^2 I)$

= favor a “simple” model

Regularization

Idea: impose constraints on the parameters to stabilize solution.

One way: introduce prior probability

Linear regression setting: $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}\ , \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)\ , \hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$

Prior: $\beta \sim \mathcal{N}(\mathbf{0}, \sigma_{\boldsymbol{\beta}}^2 I)$

= favor a “simple” model

Maximum a-posteriori (MAP) approach:

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \arg\max_{\boldsymbol{\beta}} p(\boldsymbol{\beta}|\mathcal{D}) \\
&= \arg\max_{\boldsymbol{\beta}} p(\mathcal{D}|\boldsymbol{\beta}) p(\boldsymbol{\beta}) \\
&= \arg\max_{\boldsymbol{\beta}} ||\mathbf{y} - X\boldsymbol{\beta}||^2 + \frac{\sigma_{\boldsymbol{\epsilon}}^2}{\sigma_{\boldsymbol{\beta}}^2} ||\boldsymbol{\beta}||^2 \\
&= (X^\top X + \frac{\sigma_{\boldsymbol{\epsilon}}^2}{\sigma_{\boldsymbol{\beta}}^2} I)^{-1} X^\top \mathbf{y}
\end{aligned}$$

Regularization

Idea: impose constraints on the parameters to stabilize solution.

One way: introduce prior probability

Linear regression setting: $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}\ , \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)\ , \hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$

Prior: $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \sigma_{\boldsymbol{\beta}}^2 I)$

= favor a “simple” model

Maximum a-posteriori (MAP) approach:

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \arg\max_{\boldsymbol{\beta}} p(\boldsymbol{\beta}|\mathcal{D}) \\
&= \arg\max_{\boldsymbol{\beta}} p(\mathcal{D}|\boldsymbol{\beta}) p(\boldsymbol{\beta}) \\
&= \arg\max_{\boldsymbol{\beta}} ||\mathbf{y} - X\boldsymbol{\beta}||^2 + \frac{\sigma_{\boldsymbol{\epsilon}}^2}{\sigma_{\boldsymbol{\beta}}^2} ||\boldsymbol{\beta}||^2 \\
&= (X^\top X + \frac{\sigma_{\boldsymbol{\epsilon}}^2}{\sigma_{\boldsymbol{\beta}}^2} I)^{-1} X^\top \mathbf{y}
\end{aligned}$$

Regular, better-conditioned

Regularization

Idea: impose constraints on the parameters to stabilize solution.

One way: introduce prior probability

Linear regression setting: $\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon}\ , \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)\ , \hat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y}$

Prior: $\beta \sim \mathcal{N}(0, \sigma_\beta^2 I)$

= favor a “simple” model

Maximum a-posteriori (MAP) approach:

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= \arg\max_{\boldsymbol{\beta}} p(\boldsymbol{\beta}|\mathcal{D}) \\
&= \arg\max_{\boldsymbol{\beta}} p(\mathcal{D}|\boldsymbol{\beta}) p(\boldsymbol{\beta}) \\
&= \arg\max_{\boldsymbol{\beta}} ||\mathbf{y} - X\boldsymbol{\beta}||^2 + \frac{\sigma_{\boldsymbol{\epsilon}}^2}{\sigma_{\boldsymbol{\beta}}^2} ||\boldsymbol{\beta}||^2 \\
&= (X^\top X + \frac{\sigma_{\boldsymbol{\epsilon}}^2}{\sigma_{\boldsymbol{\beta}}^2} I)^{-1} X^\top \mathbf{y}
\end{aligned}$$

Regular, better-conditioned

= Ridge Regression

Dimensionality reduction

Problem: we still have to invert an M-dimensional matrix → expensive

Goal: reduce data to features most relevant for the learning task

Dimensionality reduction

Problem: we still have to invert an M-dimensional matrix → expensive

Goal: reduce data to features most relevant for the learning task

One way: significance test for single features

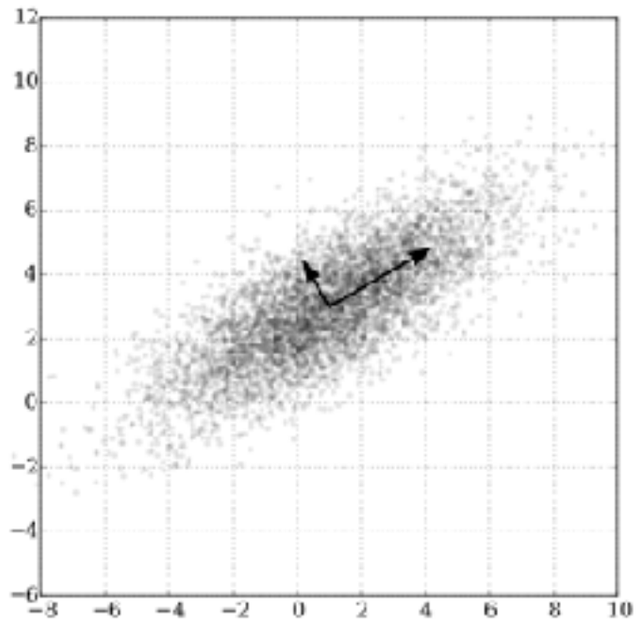
Dimensionality reduction

Problem: we still have to invert an M-dimensional matrix → expensive

Goal: reduce data to features most relevant for the learning task

One way: significance test for single features

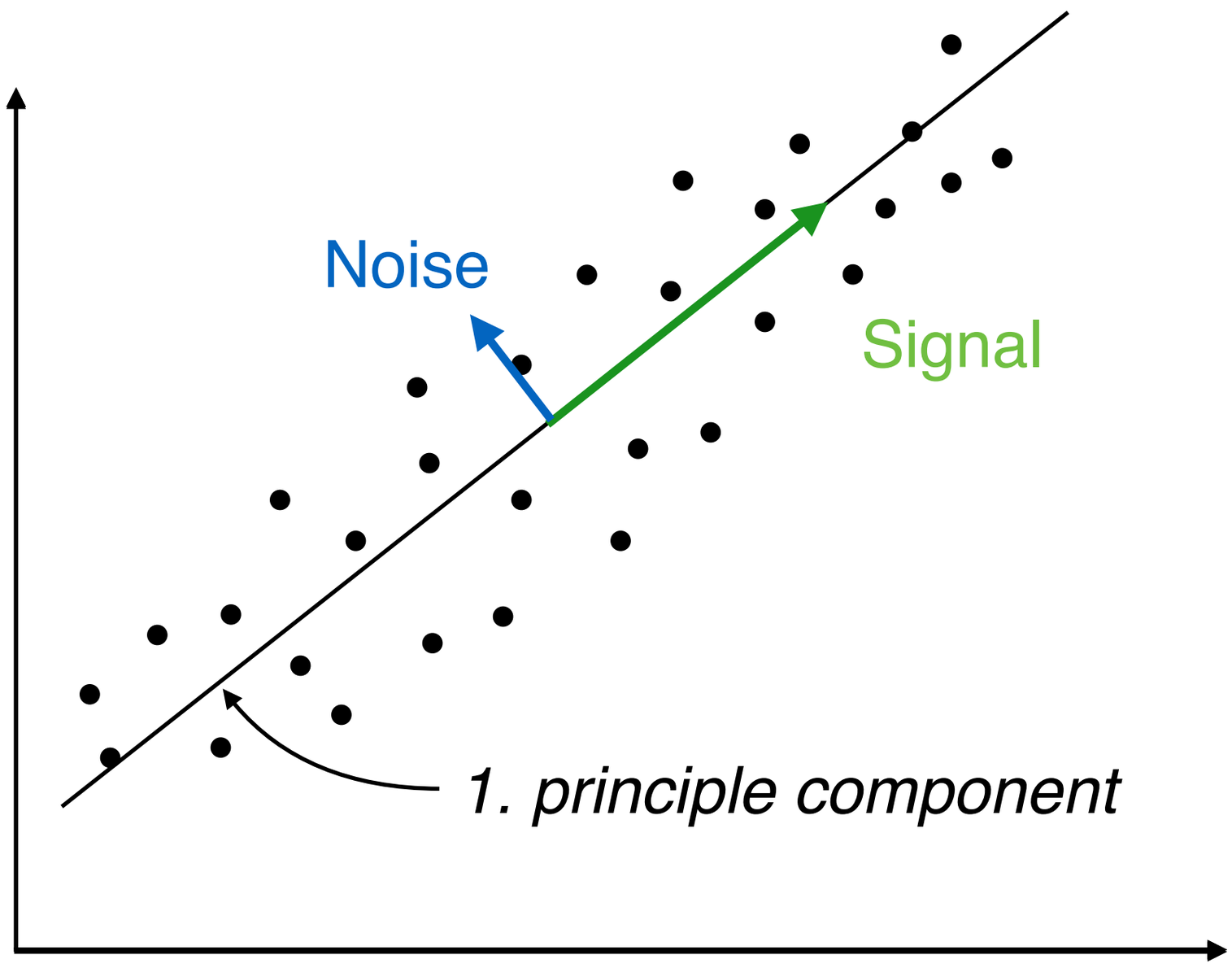
Another way: find ‘relevant’ directions/subspaces in correlated data



Why dimensionality reduction?

- **Visualization**:
  Insights into high-dimensional structures in the data
- **Better Generalization**:
  Fewer dimensions $\rightarrow$ less chances of overfitting
- **Speeding up** learning algorithms:
  Most algorithms scale badly with increasing data dimensionality
- **Data compression**:
  Less storage requirements

Principle Component Analysis (PCA)



Which line fits data best?

The line *w* that minimizes the noise and maximizes the signal [Pearson 1901].

Reference: Bishop, Pattern Recognition and Machine Learning, Chapter 12.1

Problem Definition

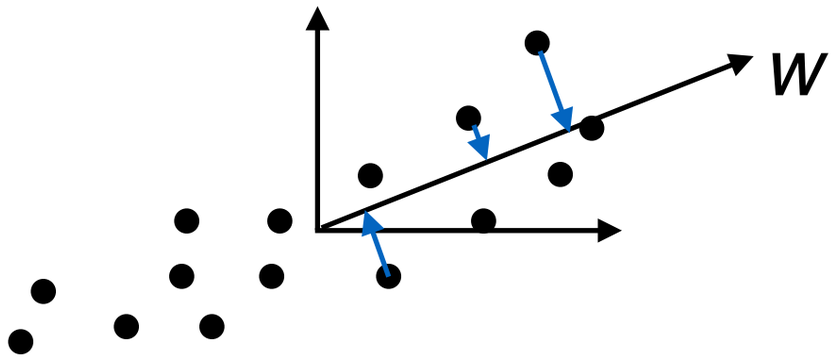
Given $x_1, \ldots, x_n \in \mathrm{R}^d$ find a *k*-dimensional subspace, so that the data projected on that subspace

1) is as close to the original data as possible (minimum noise)
2) has maximum variance (maximum signal).

2 Motivations: same result?

Assume ***the data is centered***: $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i = 0$

Minimum distance



We are only interested in the direction of w, not its length, i.e. set $||w|| = w^{\mathrm{T}} w = 1$.

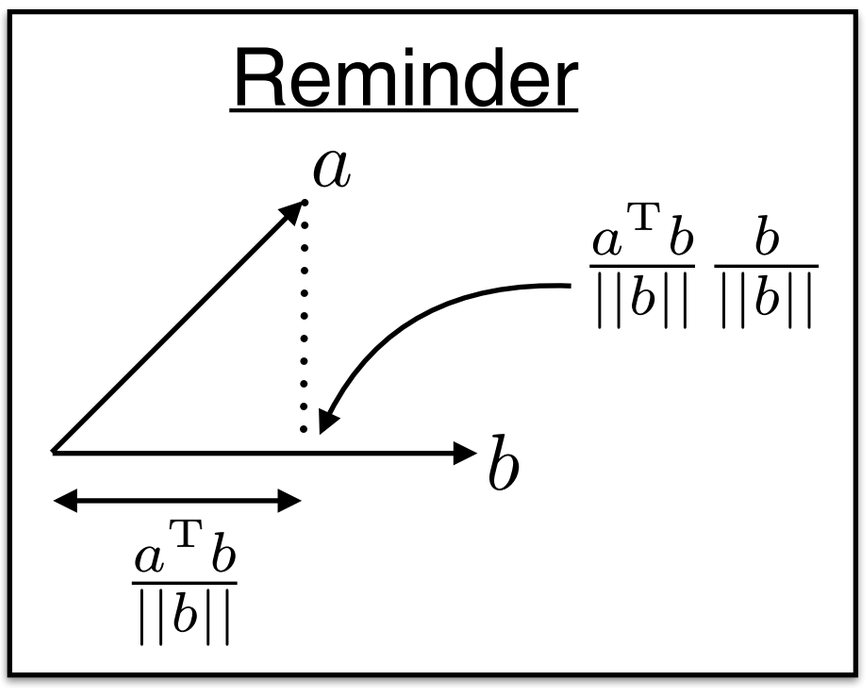
$$\min_w \sum_{i=1}^{n} ||(w^{\mathrm{T}} x_i) w - x_i||^2$$

$$= \min_w \sum_{i=1}^{n} w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w w^{\mathrm{T}} w - 2 w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w + x_i^{\mathrm{T}} x_i$$

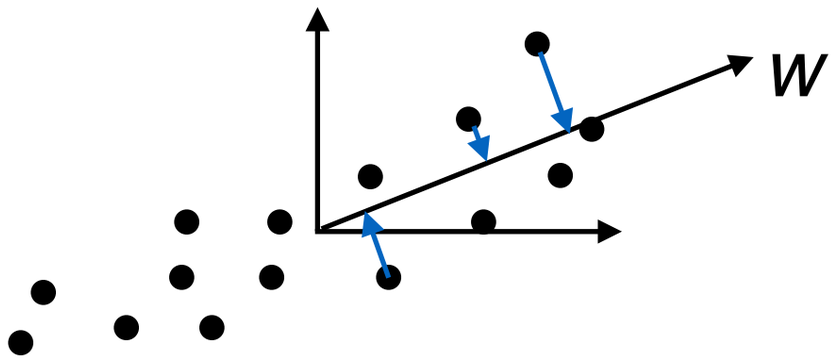
$$= \min_w - \sum_{i=1}^{n} w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w$$

$$= \max_w \sum_{i=1}^{n} w^{\mathrm{T}} x_i x_i^{\mathrm{T}} w$$

$$= \max_w w^{\mathrm{T}} X X^{\mathrm{T}} w$$



Maximum Variance



X is centered

$$\max_w \mathrm{Var}(w^{\mathrm{T}}X) = \mathrm{E}[(w^{\mathrm{T}}X)^2]$$

$$= \max_w \frac{1}{n}\sum_{i=1}^{n} w^{\mathrm{T}}x_i x_i^{\mathrm{T}} w$$

$$= \max_w w^{\mathrm{T}} X X^{\mathrm{T}} w$$

✓ (same as minimum distance)

Introduce the *scatter matrix* $S = XX^{\mathrm{T}}$. This leads to the constrained optimization problem:

$$\max_w \; w^{\mathrm{T}} S w$$

$$\text{s.t. } ||w|| = 1$$

Lagrange Multipliers

- Imagine you have a constrained optimization problem:

Read: subject to

$$\max_x f(x) \quad \text{(Objective function)}$$

$$\text{s.t. } g(x) = 0 \quad \text{(Constraint)}$$

- Example:

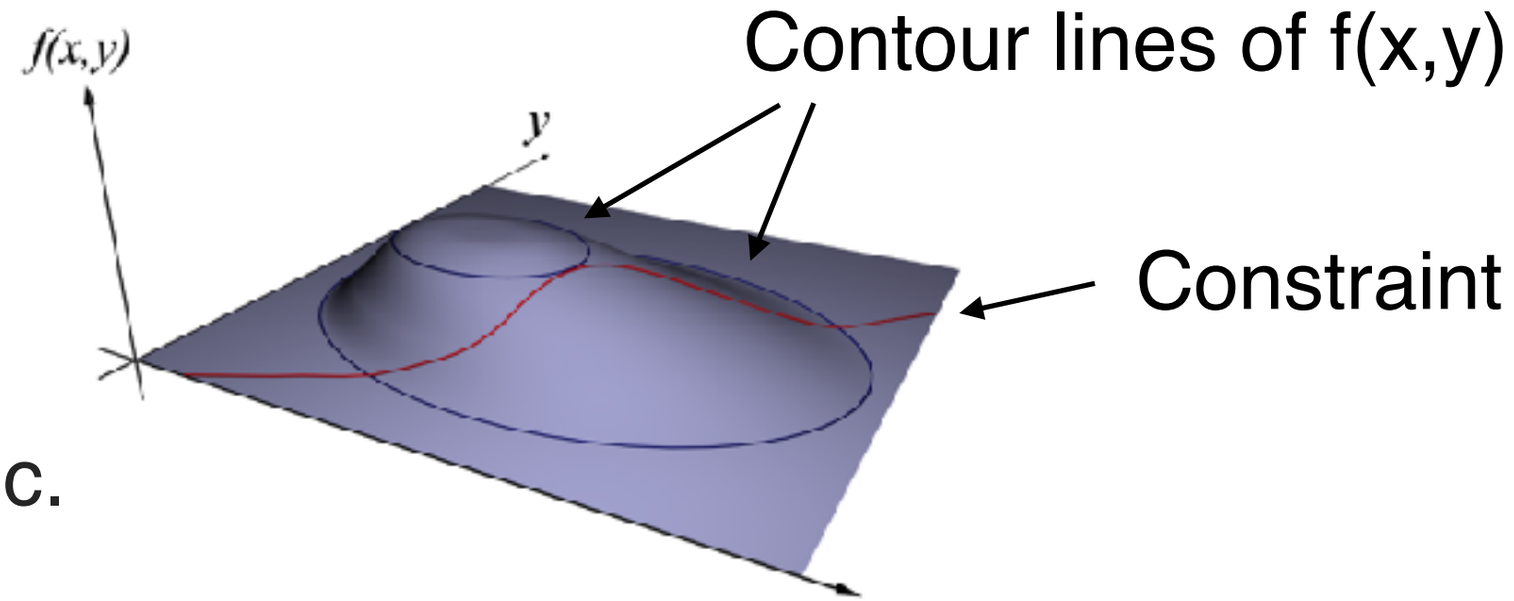
$$\max_{\mathbf{x}} 1 - x_1^2 - x_2^2$$

$$\text{s.t. } x_1 + x_2 - 1 = 0$$

Reference: Bishop, Pattern Recognition and Machine Learning, Appendix E

Geometric intuition

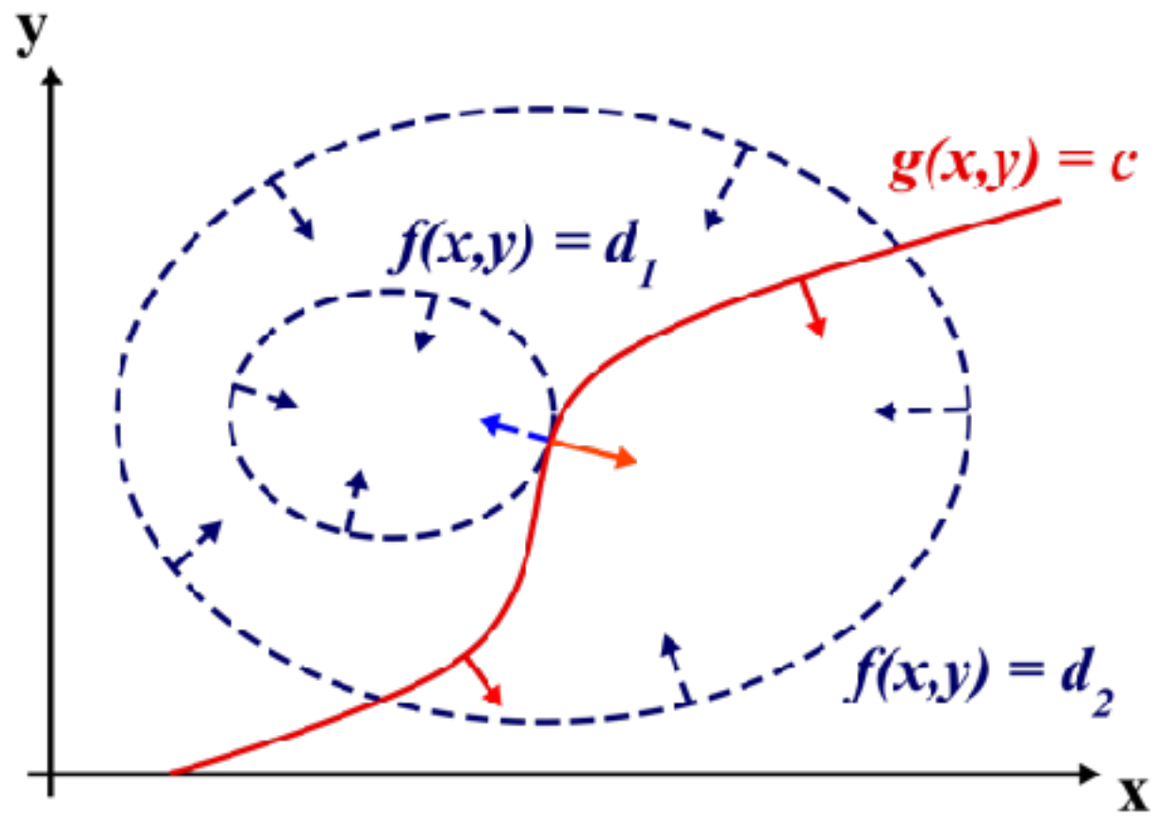
$$\max_x f(x)$$
$$\text{s.t. } g(x) = 0$$



Intuition 1: ∇g must be normal to the line g(x, y) = c.

Intuition 2: At an optimum f(x*,y*) can not increase in the direction of a neighboring point where g(x, y) = c. Thus, at (x*, y*) ∇f must be perpendicular to the line g(x, y) = c.

It follows: <u>In (x*, y*) the gradients ∇g and ∇f are parallel!</u>

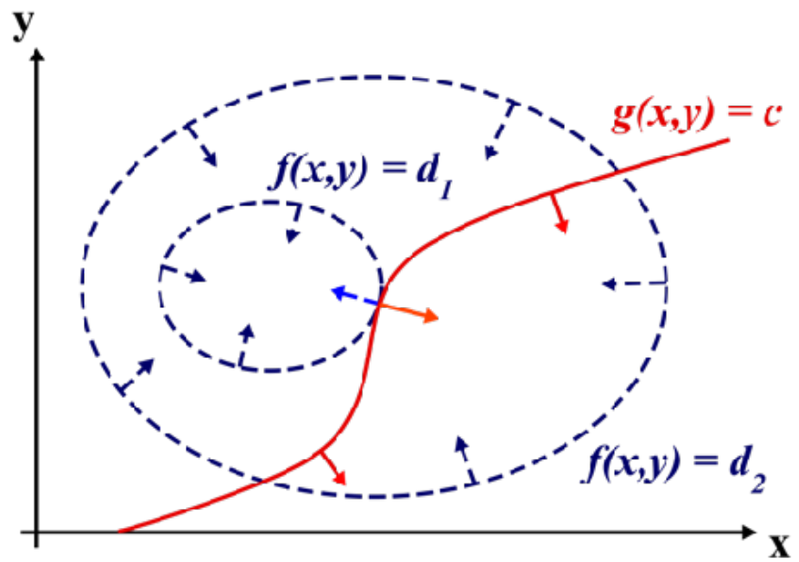


The Lagrange Multiplier

In (x*, y*) the gradients ∇g and ∇f are parallel. Thus for a maximum:

$$\nabla f = \lambda \nabla g$$

Lagrange Multiplier



Lagrangian function:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda g(x)$$

∇*L* = 0 is a necessary (but not sufficient) condition for the optimization solution.

Thus, to solve a constrained optimization problem, we can define the Lagrangian and look for the points where its gradient vanishes.

Example

$$\max_{\mathbf{x}} 1 - x_1^2 - x_2^2$$

$$\text{s.t. } x_1 + x_2 - 1 = 0$$

1. Define Lagrangian:

$$\mathcal{L}(x, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1)$$

2. Set gradient of Lagrangian to zero:

$$-2x_1 + \lambda = 0$$

$$-2x_2 + \lambda = 0$$

$$x_1 + x_2 - 1 = 0$$

3. Solve equations: $(x_1^*, x_2^*) = (\frac{1}{2}, \frac{1}{2}) \quad \lambda = 1$

PCA optimization problem

$$\max_{w} \; w^{\mathrm{T}} S w$$
$$\text{s.t. } ||w|| = 1$$

1. Define Lagrangian: $\mathcal{L}(w, \lambda) = w^{\mathrm{T}} S w + \lambda(1 - w^{\mathrm{T}} w)$

2. Compute gradient: $\dfrac{\partial \mathcal{L}(w, \lambda)}{\partial w} = 2Sw - 2\lambda w$

3. Set to zero: $Sw = \lambda w$

PCA optimization problem

$$\max_{w} \; w^{\mathrm{T}} S w$$
$$\text{s.t. } ||w|| = 1$$

1. Define Lagrangian: $\mathcal{L}(w, \lambda) = w^{\mathrm{T}} S w + \lambda(1 - w^{\mathrm{T}} w)$

2. Compute gradient: $\frac{\partial \mathcal{L}(w, \lambda)}{\partial w} = 2Sw - 2\lambda w$

3. Set to zero: $Sw = \lambda w$

Eigenvalue problems

$$Sw = \lambda w$$

$$(S - \lambda I)w = 0$$

Eigenvalue problems

$$Sw = \lambda w$$

$$(S - \lambda I)w = 0$$

Solutions are found at roots of characteristic polynomial

$$p(\lambda) := \det(S - \lambda I) = 0$$

Eigenvalue problems

$$Sw = \lambda w$$

$$(S - \lambda I)w = 0$$

Solutions are found at roots of characteristic polynomial

$$p(\lambda) := \det(S - \lambda I) = 0$$

In general: d roots (eigenvalues) $\lambda_1, \ldots, \lambda_d$

Corresponding eigenvectors: $w_1, \ldots, w_d$

Eigenvalue problems

$$Sw = \lambda w$$

$$(S - \lambda I)w = 0$$

Solutions are found at roots of characteristic polynomial

$$p(\lambda) := \det(S - \lambda I) = 0$$

In general: d roots (eigenvalues) $\lambda_1, \ldots, \lambda_d$

Corresponding eigenvectors: $w_1, \ldots, w_d$

Leads to the decomposition $W^\top S W = \Lambda \quad S = W \Lambda W^\top$

where $W = [w_1, \ldots, w_d], W^\top W = I$ is orthogonal and
$\Lambda, \Lambda_{ii} = \lambda_i$ is diagonal.

PCA algorithm

The i-th eigenvalue is the variance in the direction of the i-th eigenvector:

$$\mathrm{Var}(w_i^{\mathrm{T}} x) = \frac{1}{n} w_i^{\mathrm{T}} S w_i = \frac{1}{n} \lambda_i w_i^{\mathrm{T}} w_i = \frac{1}{n} \lambda_i$$

PCA algorithm

The i-th eigenvalue is the variance in the direction of the i-th eigenvector:

$$\mathrm{Var}(w_i^{\mathrm{T}} x) = \frac{1}{n} w_i^{\mathrm{T}} S w_i = \frac{1}{n} \lambda_i w_i^{\mathrm{T}} w_i = \frac{1}{n} \lambda_i$$

The direction of largest variance corresponds to the largest eigenvector (= the eigenvector with largest eigenvalue).

PCA algorithm

The i-th eigenvalue is the variance in the direction of the i-th eigenvector:

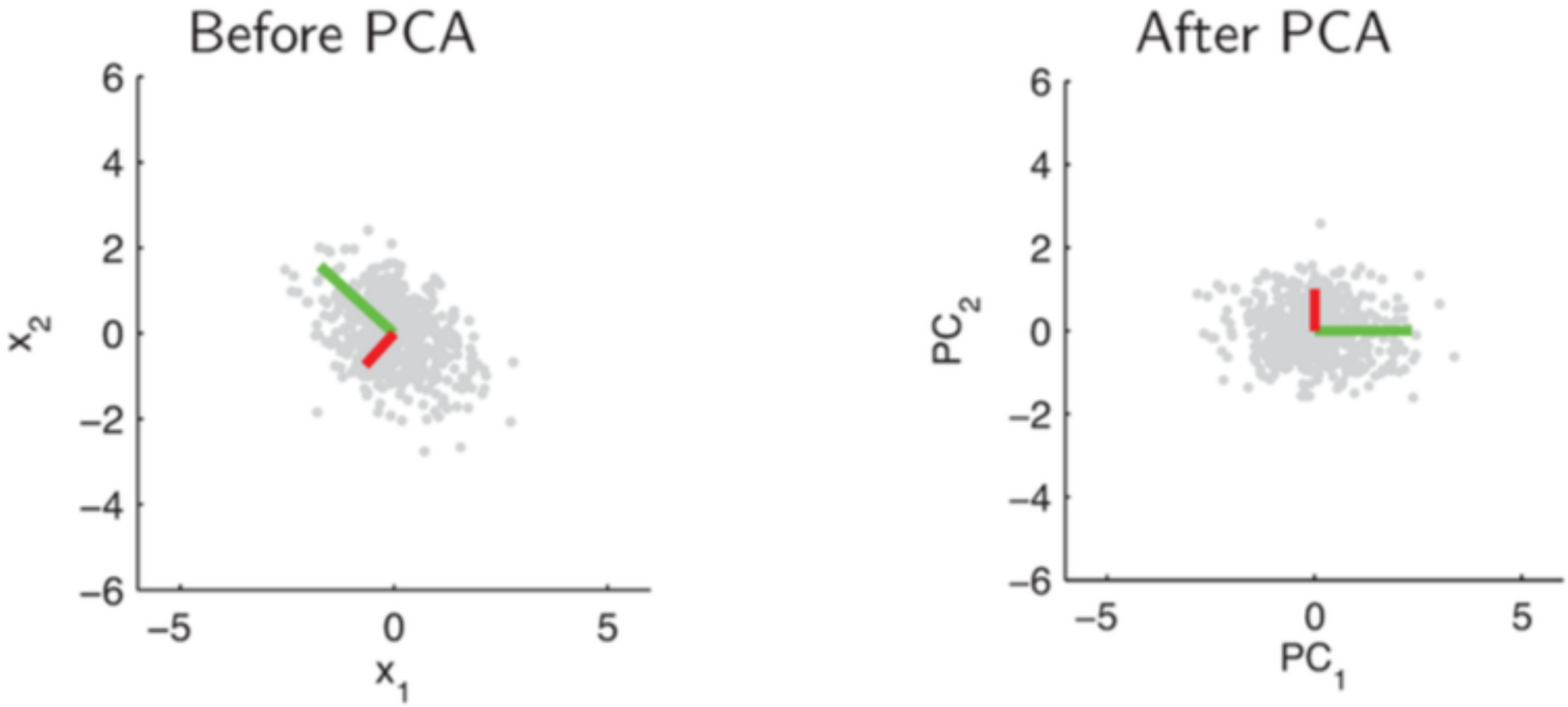
$$\mathrm{Var}(w_i^{\mathrm{T}} x) = \frac{1}{n} w_i^{\mathrm{T}} S w_i = \frac{1}{n} \lambda_i w_i^{\mathrm{T}} w_i = \frac{1}{n} \lambda_i$$

The direction of largest variance corresponds to the largest eigenvector (= the eigenvector with largest eigenvalue).

Algorithm 1: Principal Component Analysis

Require: data $x_1, \ldots, x_N \in \mathbb{R}^d$, number of principal components k

1: # Center Data
2: $X = X - 1/N \sum_i x_i$
3: # Compute Covariance Matrix
4: $C = 1/N \; XX^{\top}$
5: # Compute largest k eigenvectors
6: $W = \mathrm{eig}(C)$
7: **return** W



PCA rotates data into new coordinate system with the directions of largest variances being the new coordinate axes.

Questions

1. What happens when the data are not centered?
2. What happens if data are already standardized (features have zero mean and unit variance)?

Solving PCA with SVD

A singular value decomposition factorizes a matrix as:

$$U\Lambda V = M$$

where

U are the Eigenvectors of MM^*.
V are the Eigenvectors of M^*M.
The square roots of the Eigenvalues of MM^* are on the diagonal of Λ.

Solving PCA with SVD

A singular value decomposition factorizes a matrix as:

$$U\Lambda V = M$$

where

U are the Eigenvectors of MM^*.
V are the Eigenvectors of M^*M.
The square roots of the Eigenvalues of MM^* are on the diagonal of Λ.

Instead of calculating the EV-decomposition of S, we can compute the SVD-decomposition of X. This is computationally much more stable.

E.g. Läuchli Matrix $\begin{pmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{pmatrix}^T$

SVD in numpy

numpy.linalg.svd

numpy.linalg.svd(*a*, *full_matrices=1*, *compute_uv=1*) [source]

Singular Value Decomposition.

Factors the matrix *a* as `u * np.diag(s) * v`, where *u* and *v* are unitary and *s* is a 1-d array of *a*'s singular values.

Parameters :

a *: (..., M, N) array_like*
A real or complex matrix of shape (*M*, *N*) .

full_matrices *: bool, optional*
If True (default), *u* and *v* have the shapes (*M*, *M*) and (*N*, *N*), respectively. Otherwise, the shapes are (*M*, *K*) and (*K*, *N*), respectively, where *K* = min(*M*, *N*).

compute_uv *: bool, optional*
Whether or not to compute *u* and *v* in addition to *s*. True by default.

Returns :

u *: { (..., M, M), (..., M, K) } array*
Unitary matrices. The actual shape depends on the value of `full_matrices`. Only returned when `compute_uv` is True.

s *: (..., K) array*
The singular values for every matrix, sorted in descending order.

v *: { (..., N, N), (..., K, N) } array*
Unitary matrices. The actual shape depends on the value of `full_matrices`. Only returned when `compute_uv` is True.

Raises :

LinAlgError
If SVD computation does not converge.

Power iteration

The SVD has computational complexity $O(\min(n^2 d, d^2 n))$.

But: We often only need a few largest principle components and not all of them.

or power iteration

Algorithm 6.1 Simple vector iteration

1: Choose a starting vector $\mathbf{x}^{(0)} \in \mathbb{F}^n$ with $\|\mathbf{x}^{(0)}\| = 1$.
2: $k = 0$.
3: **repeat**
4: $\quad k := k + 1$;
5: $\quad \mathbf{y}^{(k)} := A\mathbf{x}^{(k-1)}$;
6: $\quad \mu_k := \|\mathbf{y}^{(k)}\|$;
7: $\quad \mathbf{x}^{(k)} := \mathbf{y}^{(k)}/\mu_k$;
8: **until** a convergence criterion is satisfied

The solution is the first eigenvector of A.

Power iteration intuition

Assume a diagonalizable matrix $A = U\Lambda U^{\mathrm{T}}$.

The power iteration computes after step k: $A^k x$

where x is a random start vector.

We have $A^k = U\Lambda^k U^{\mathrm{T}}$ because $U^{\mathrm{T}}U = I$.

Transform x into a new coordinate system: $\tilde{x} = U^{\mathrm{T}}x$

$$A^k x = U\Lambda^k U^{\mathrm{T}} x = U\Lambda^k \tilde{x} = \sum_{i=1}^{d} (\lambda_i^k \tilde{x}_i) u_i = \lambda_1^k \sum_{i=1}^{d} (\frac{\lambda_i^k}{\lambda_1^k} \tilde{x}_i) u_i$$

$$\Rightarrow \frac{A^k x}{||A^k x||} \xrightarrow{k\to\infty} u_1$$

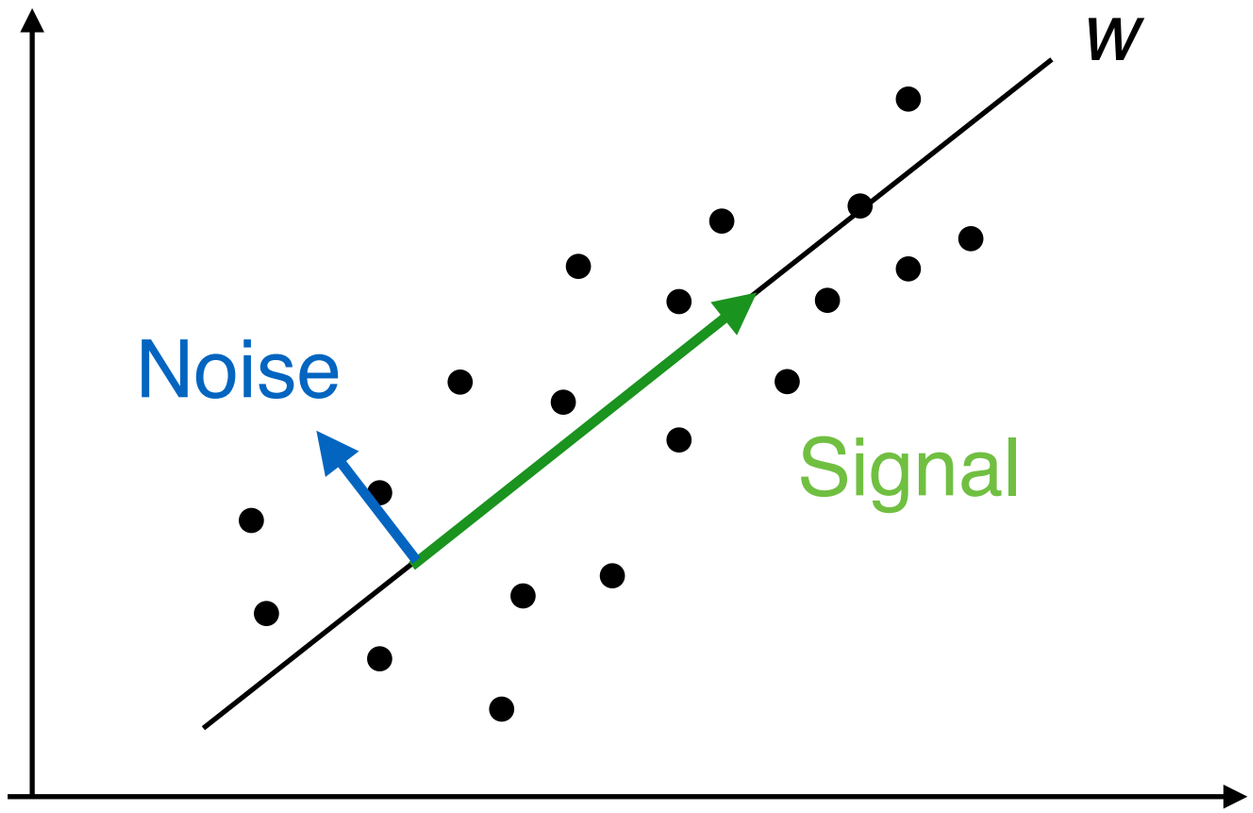
$$\text{because } \frac{\lambda_i}{\lambda_1} \leq 1 \text{ for } \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$$

Deflation

To find the second EV, remember $S = W\Lambda W^{\mathrm{T}} = \sum_{i=1}^{d} \lambda_i w_i w_i^{\mathrm{T}}$.

Thus, do power iteration on $\hat{S} = S - \lambda_1 w_1 w_1^{\mathrm{T}}$.

Application: Dimensionality Reduction



We can reduce the dimensionality of our dataset by projecting on the first k principle components.

How much signal are we going to loose?

$$\sum_{i=k+1}^{d} \mathrm{Var}(w^{\mathrm{T}}x) = \frac{1}{n} \sum_{i=k+1}^{d} \lambda_i$$

number of components

Projection on k principle components:

$$\begin{pmatrix} w_1^{\mathrm{T}}x \\ \vdots \\ w_k^{\mathrm{T}}x \end{pmatrix} = \begin{bmatrix} | & & | \\ w_1 & \cdots & w_k \\ | & & | \end{bmatrix}^{\mathrm{T}} x$$

Application: Eigenfaces

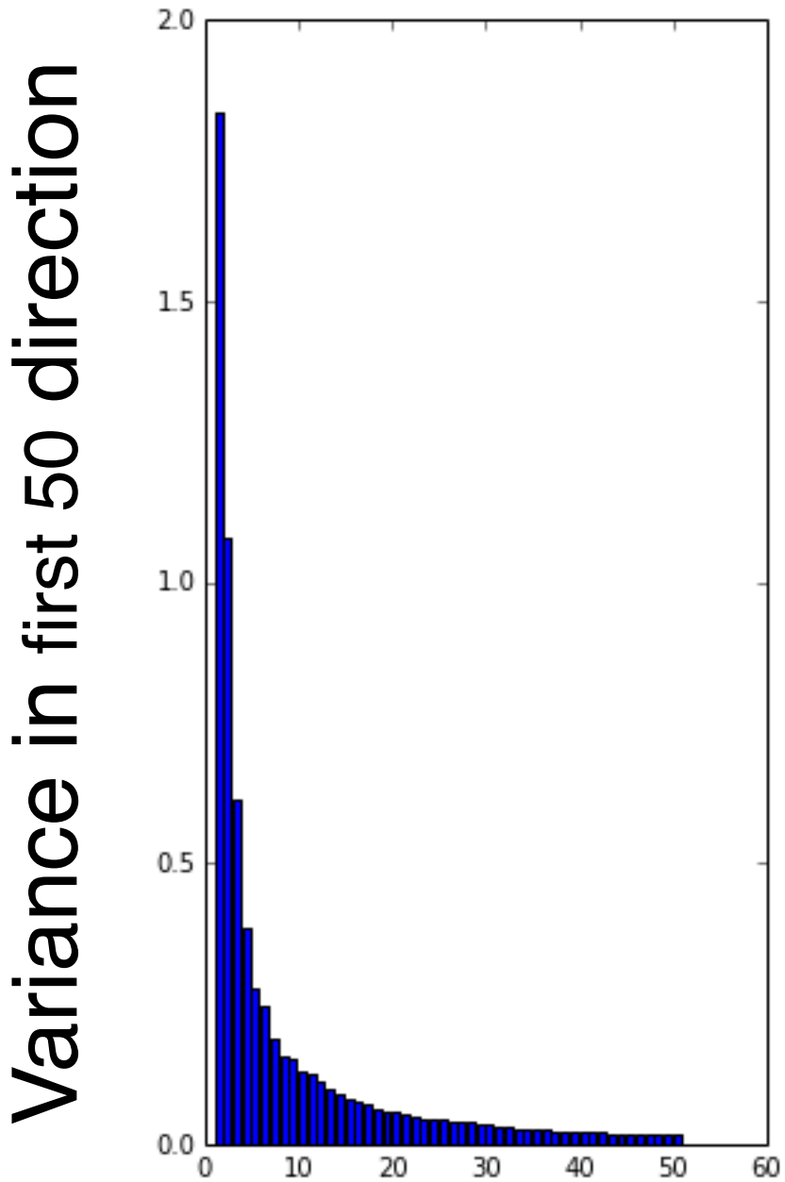
Idea: Faces look very similar in comparison to other random images. How many principle components would we need to accurately describe all faces?

An 64x64 pixel image of a face can be represented as a 4096 dimensional vector where each entry has the pixel's grayscale value.

Reference: Turk, Matthew A and Pentland, Alex P. Face recognition using eigenfaces. Computer Vision and Pattern Recognition, 1991.

Eigenfaces

The principle components are directions in our faces space.
Thus, each principle component is a face representation, too.



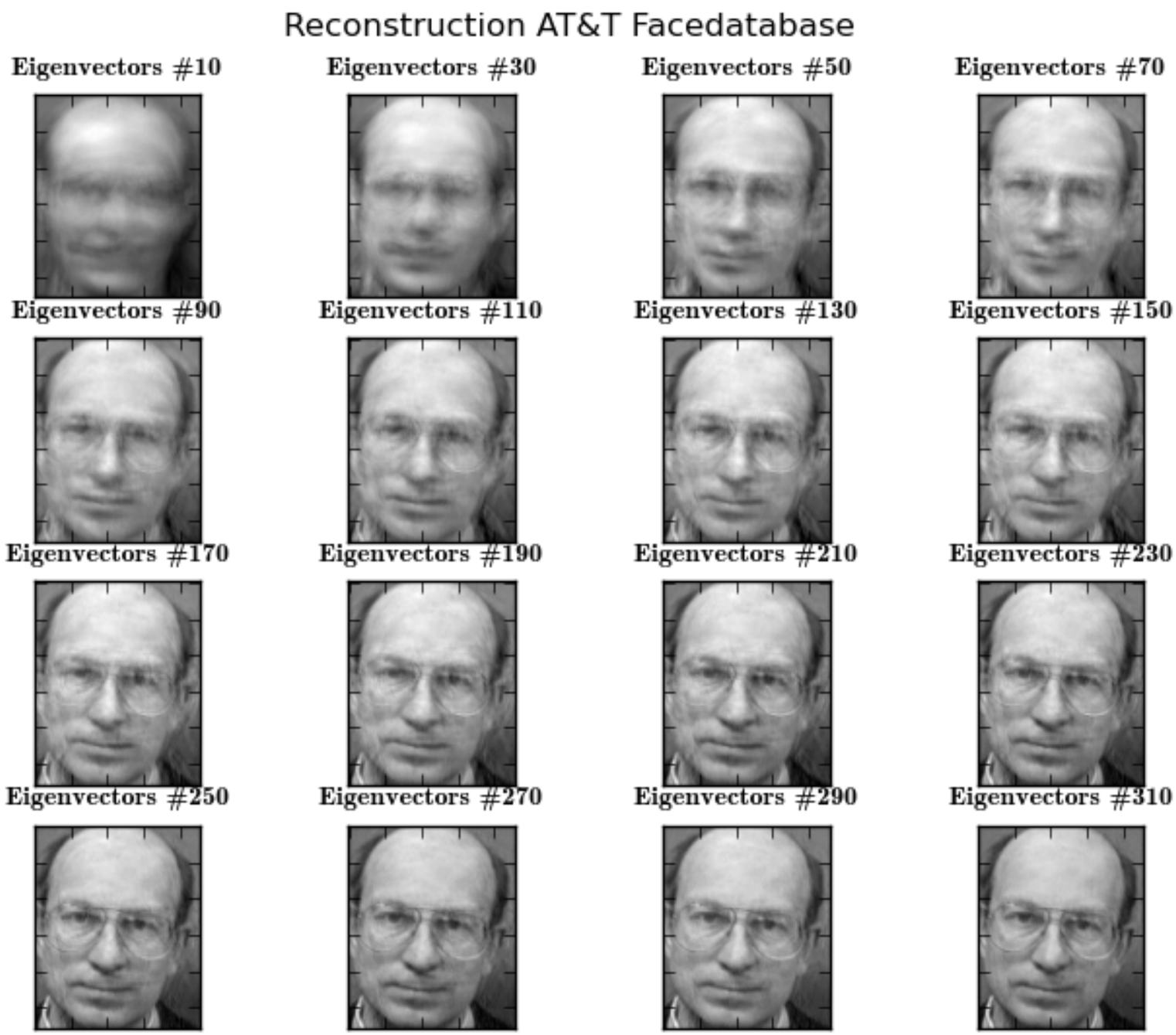
Principle component 1

Principle component 2

Principle component 3

Principle component 350

Eigenfaces


Reconstruction AT&T Facedatabase
Eigenvectors #10
Eigenvectors #30
Eigenvectors #50
Eigenvectors #70
Eigenvectors #90
Eigenvectors #110
Eigenvectors #130
Eigenvectors #150
Eigenvectors #170
Eigenvectors #190
Eigenvectors #210
Eigenvectors #230
Eigenvectors #250
Eigenvectors #270
Eigenvectors #290
Eigenvectors #310

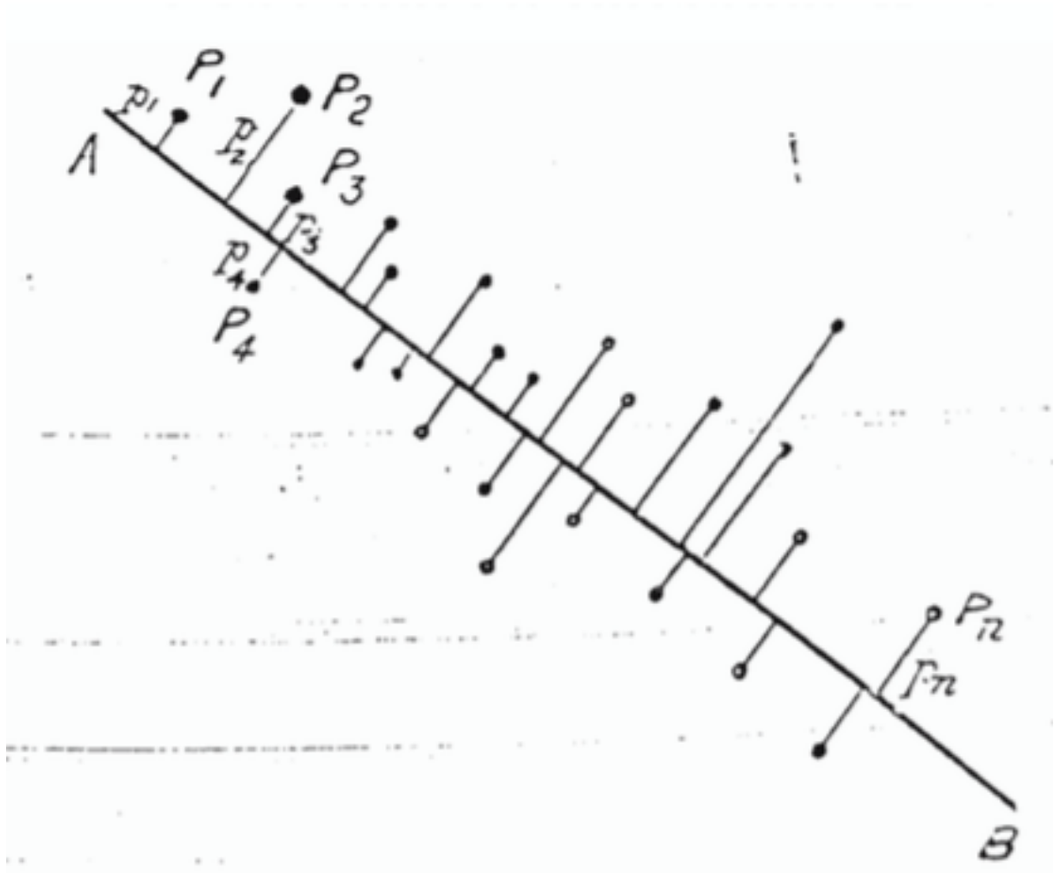
Application: Denoising

We can use PCA to denoise data:

Step 1: Reduce dimensionality to filter out "noise" directions: $(w_1^{\mathrm{T}}x, \ldots, w_k^{\mathrm{T}}x)^{\mathrm{T}}$

Step 2: Project back into original space:

$$\sum_{i=1}^{k}(w_i^{\mathrm{T}}x)w_i$$



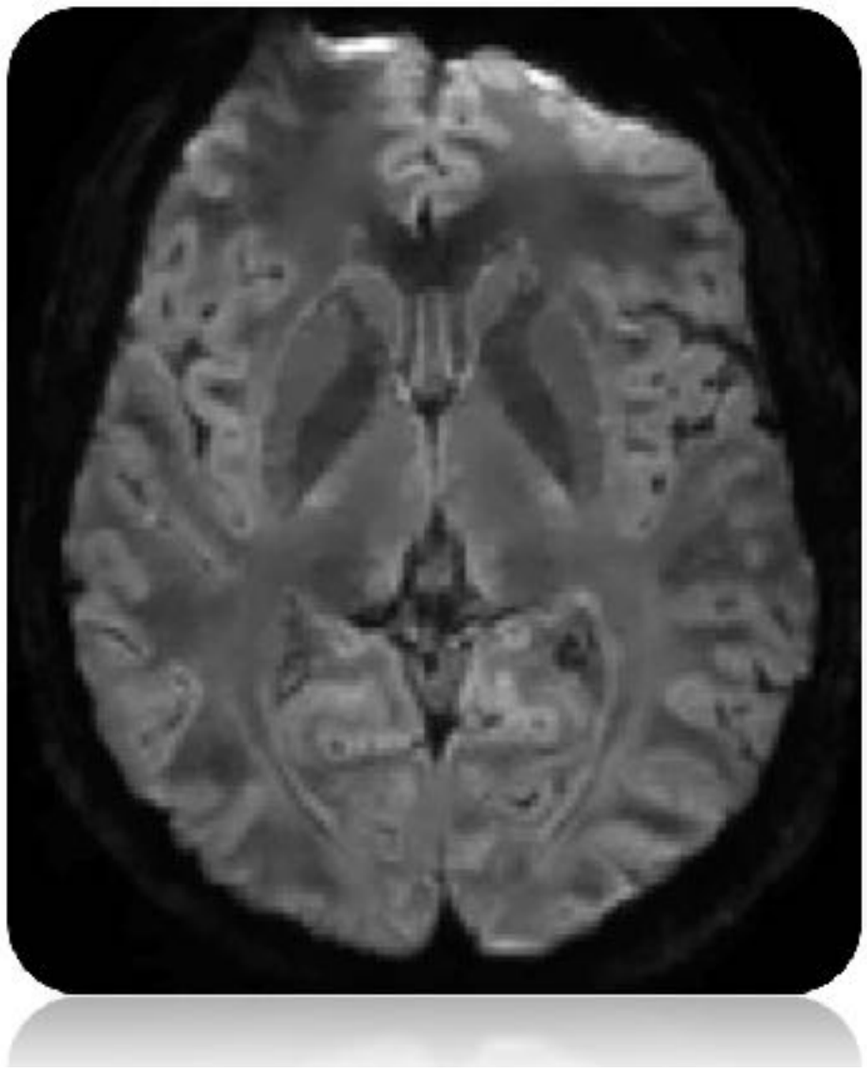
Step 3: Undo centering:

$$\sum_{i=1}^{k}(w_i^{\mathrm{T}}x)w_i + \sum_{i=1}^{n}x_i$$

Application: Denoising

Original DWI

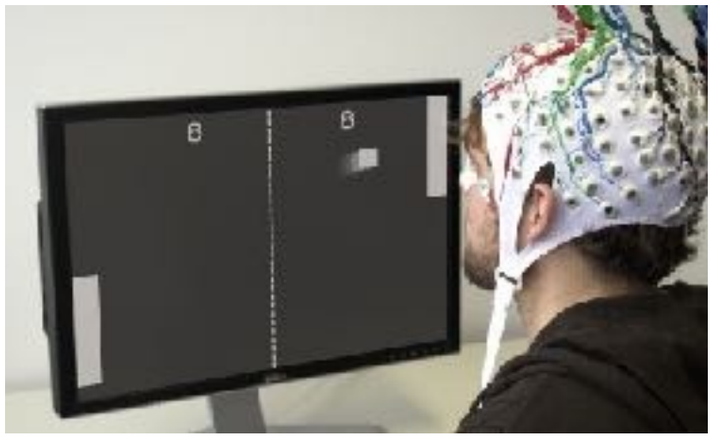
Denoised DWI using the LPCA filter

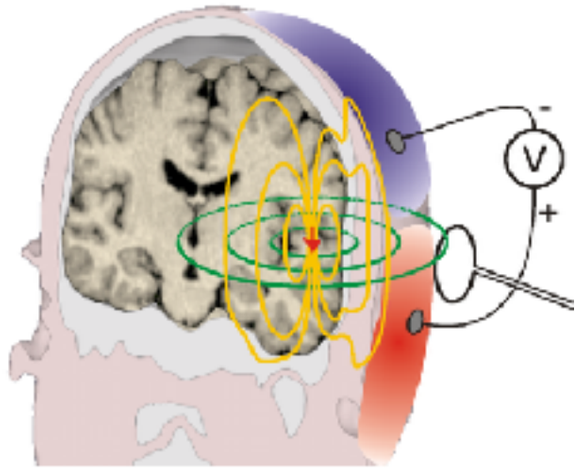


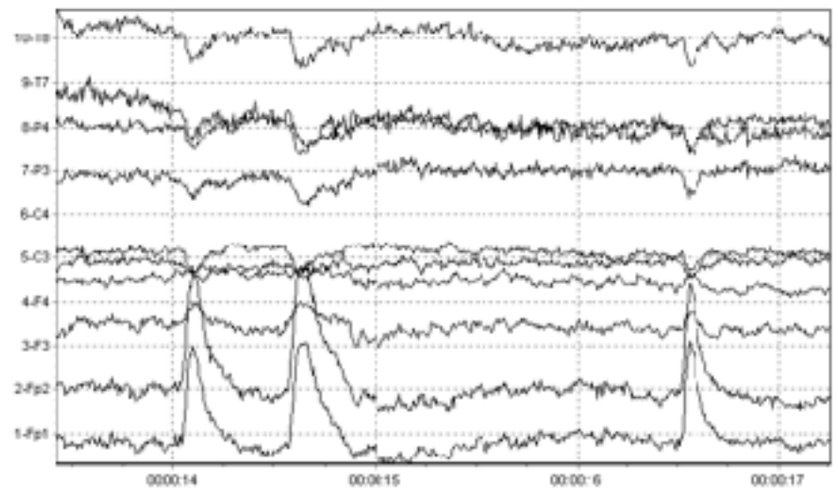
Mann et al., 2013

Application: EEG artifacts

In electroencephalographic (EEG) recordings, eye blink artifacts can be stronger than the neuronal activity.



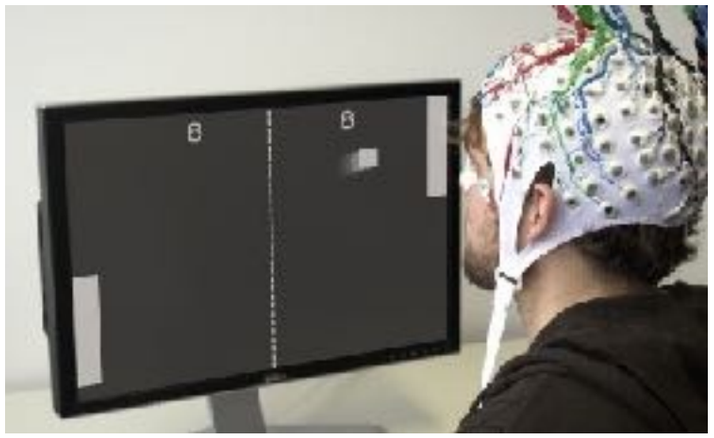


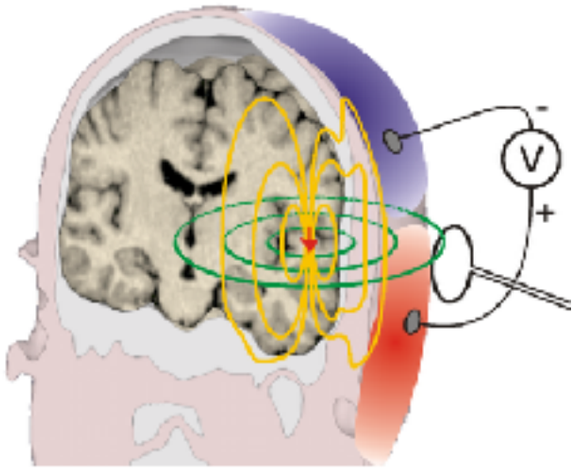


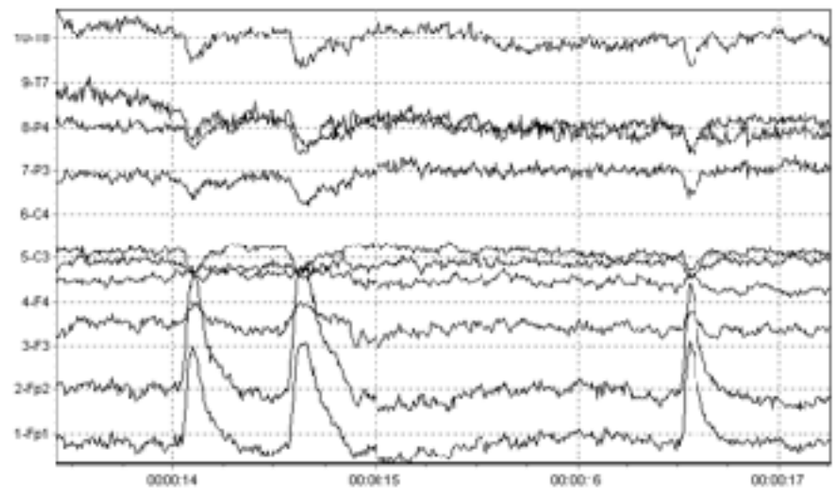
BCI2000

Application: EEG artifacts

In electroencephalographic (EEG) recordings, eye blink artifacts can be stronger than the neuronal activity.

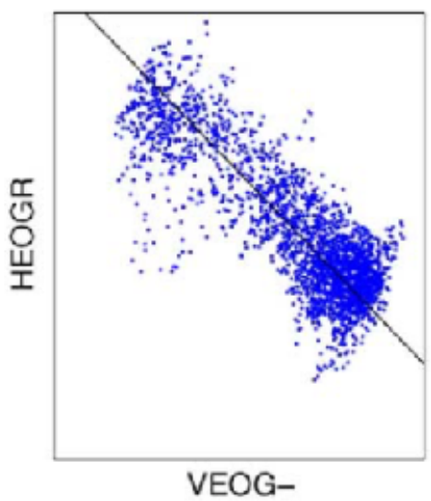


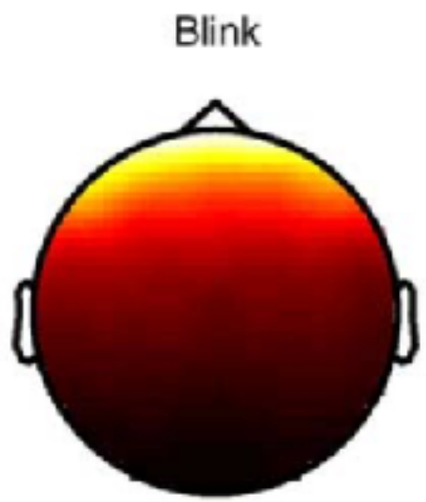




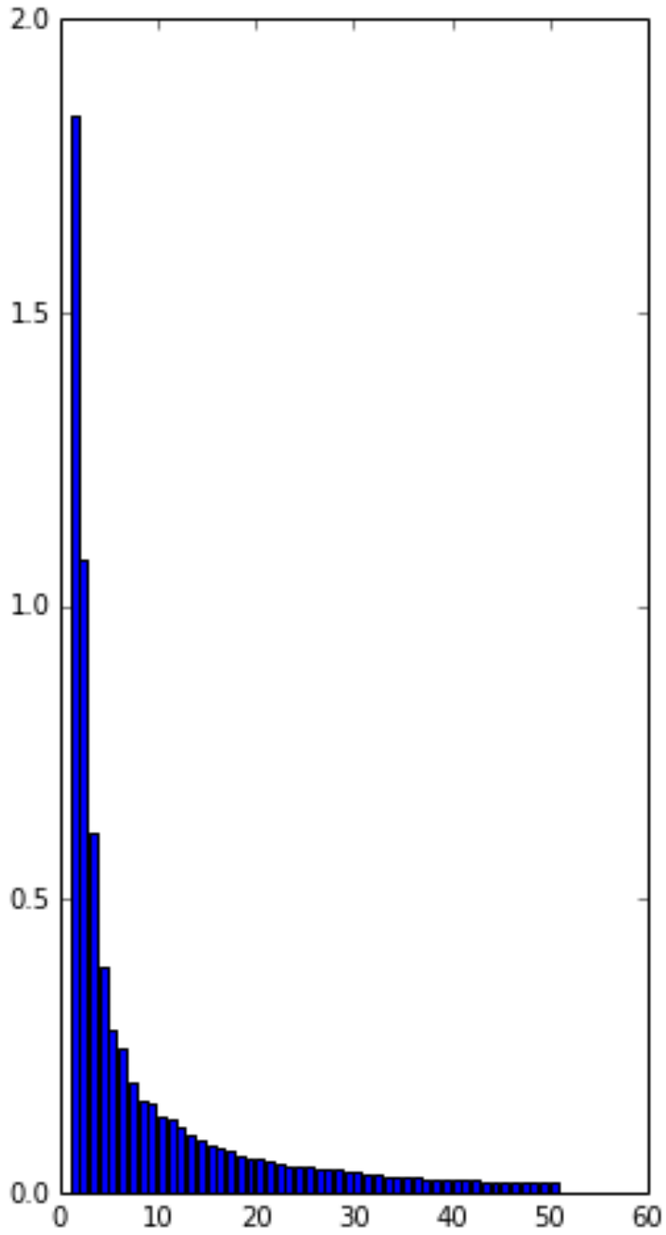
BCI2000

→ reasonable to remove first principal components

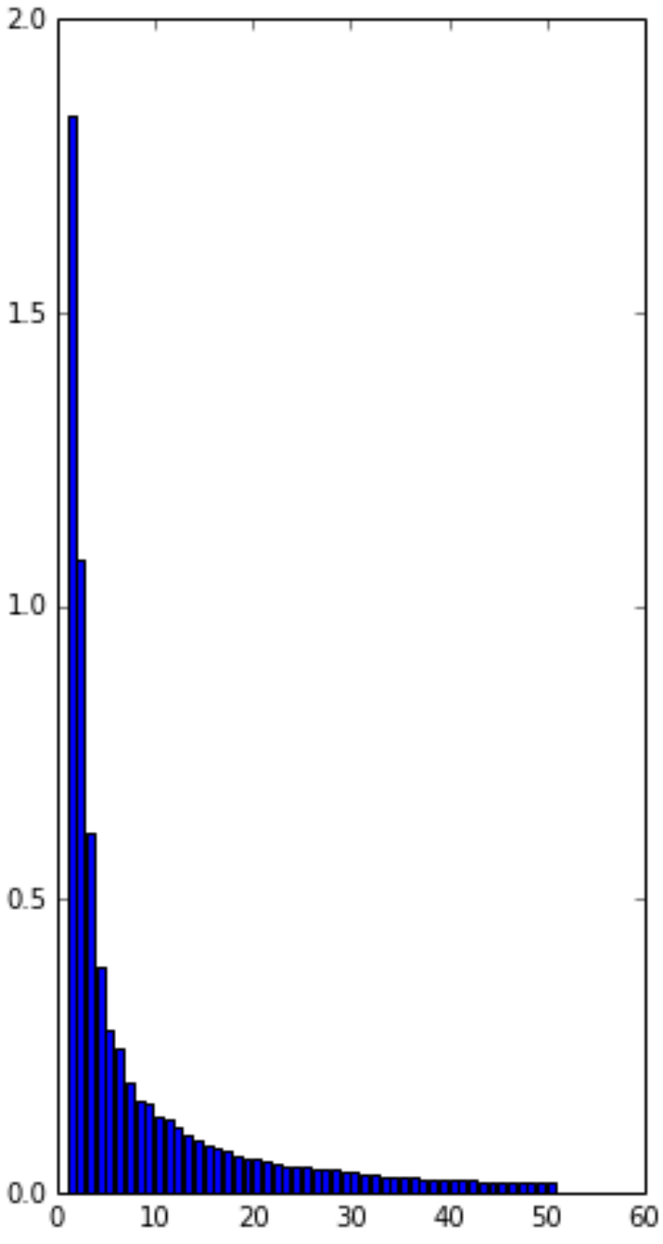




How many principal components?

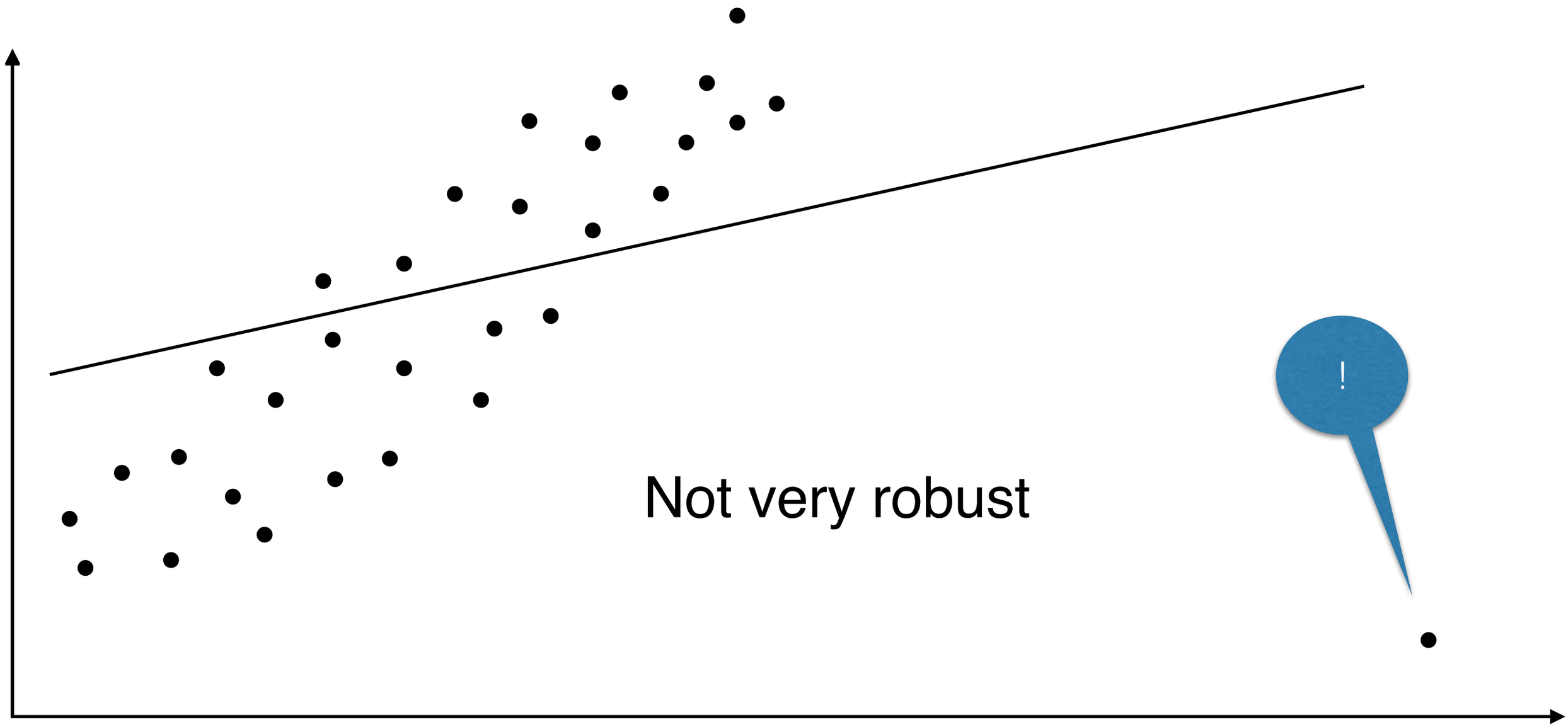


How many principal components?



Heuristics

How robust is PCA to outliers?



Not very robust