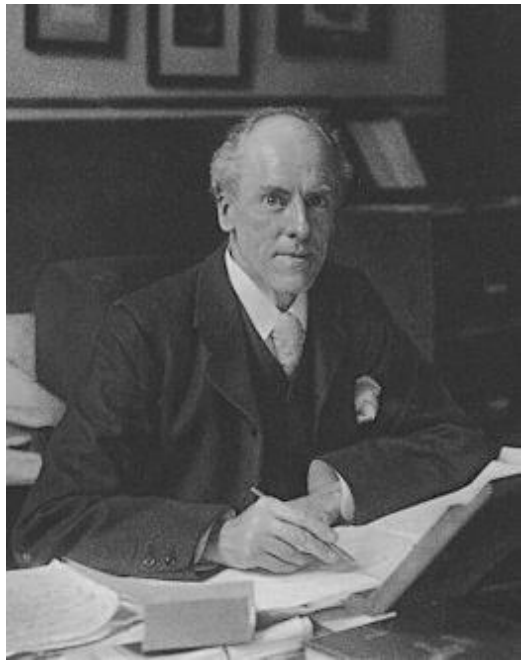
Fundamentals of AI

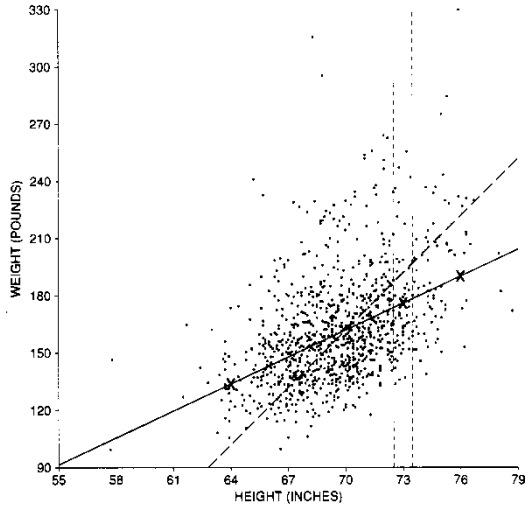
Dimensionality reduction

# Principal Component Analysis (**PCA**):

(really) central method for unsupervised machine learning which is 120 years old



# Pearson (1901): problem of choice of dependent and independent variables




$$\text{Weight} = a_0 + a_1 \text{ Height}$$

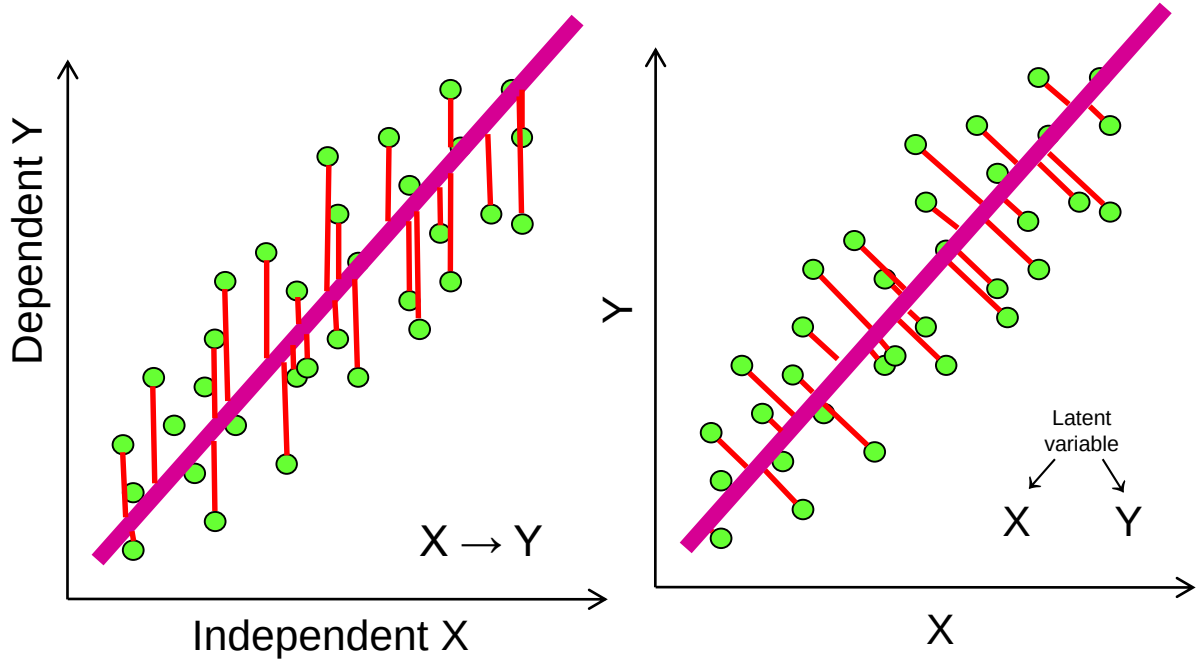
$$\text{Height} = b_0 + b_1 \text{ Weight}$$

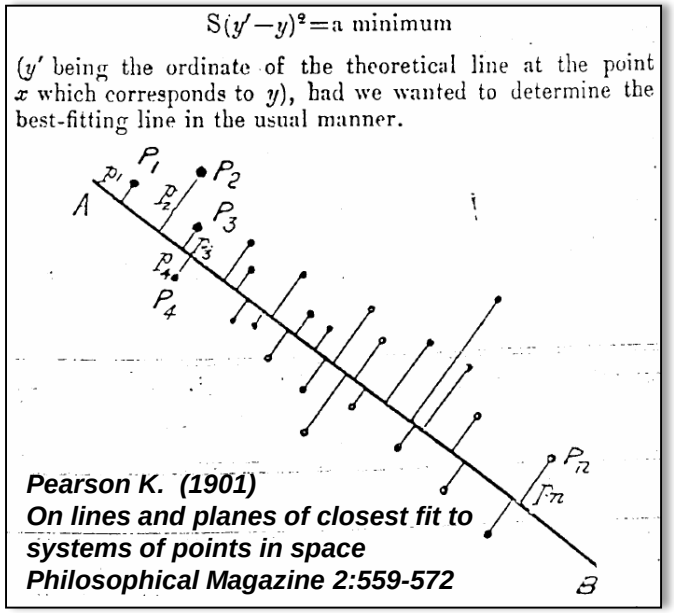
$$a_1 \neq 1/b_1 \text{ !!!}$$

$$\sum_{i=1}^{m} \| \text{———} \|^2 \rightarrow \min$$

Linear regression

Principal component (best fit line)



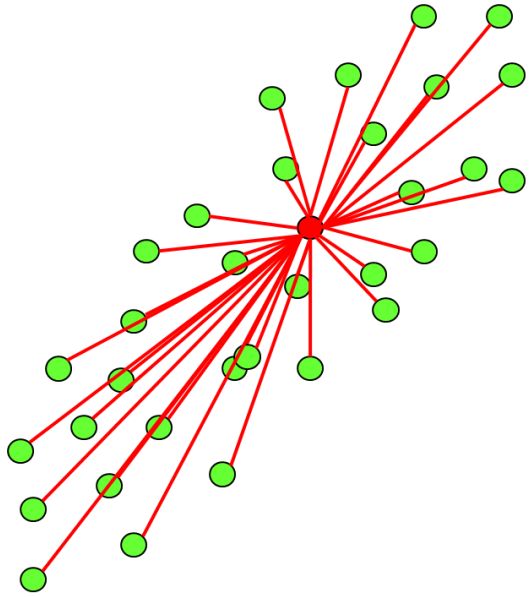


S(y′−y)²=a minimum

(y′ being the ordinate of the theoretical line at the point x which corresponds to y), had we wanted to determine the best-fitting line in the usual manner.

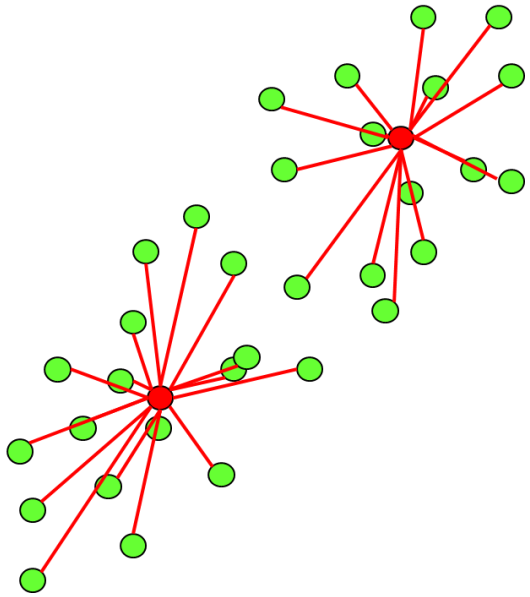
***Pearson K. (1901)***
***On lines and planes of closest fit to systems of points in space***
***Philosophical Magazine 2:559-572***

# Principal line and principal plane

Mean point

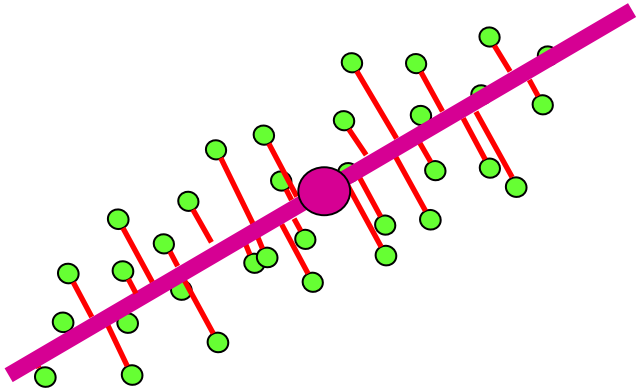


K-Means

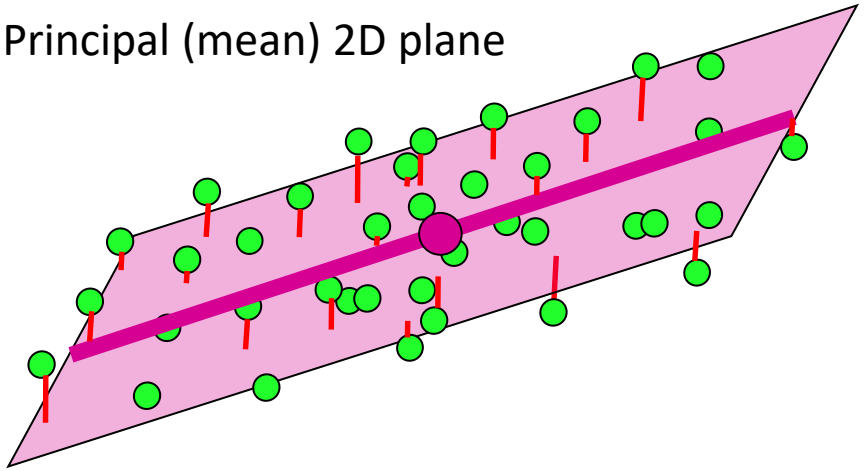


$$\sum_{i=1}^{m} \left\| \text{———} \right\|^2 \to \min$$

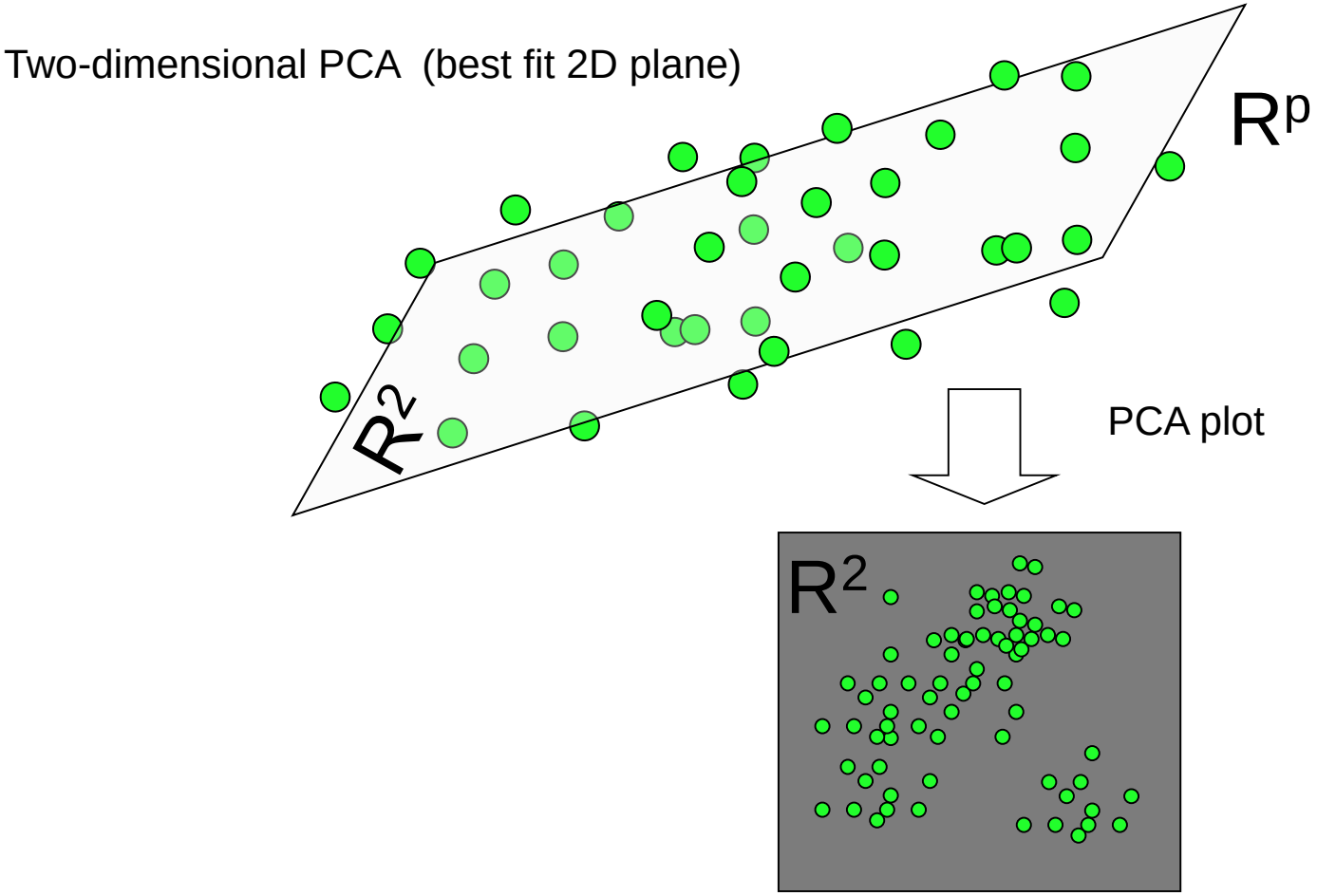
Principal (mean) line



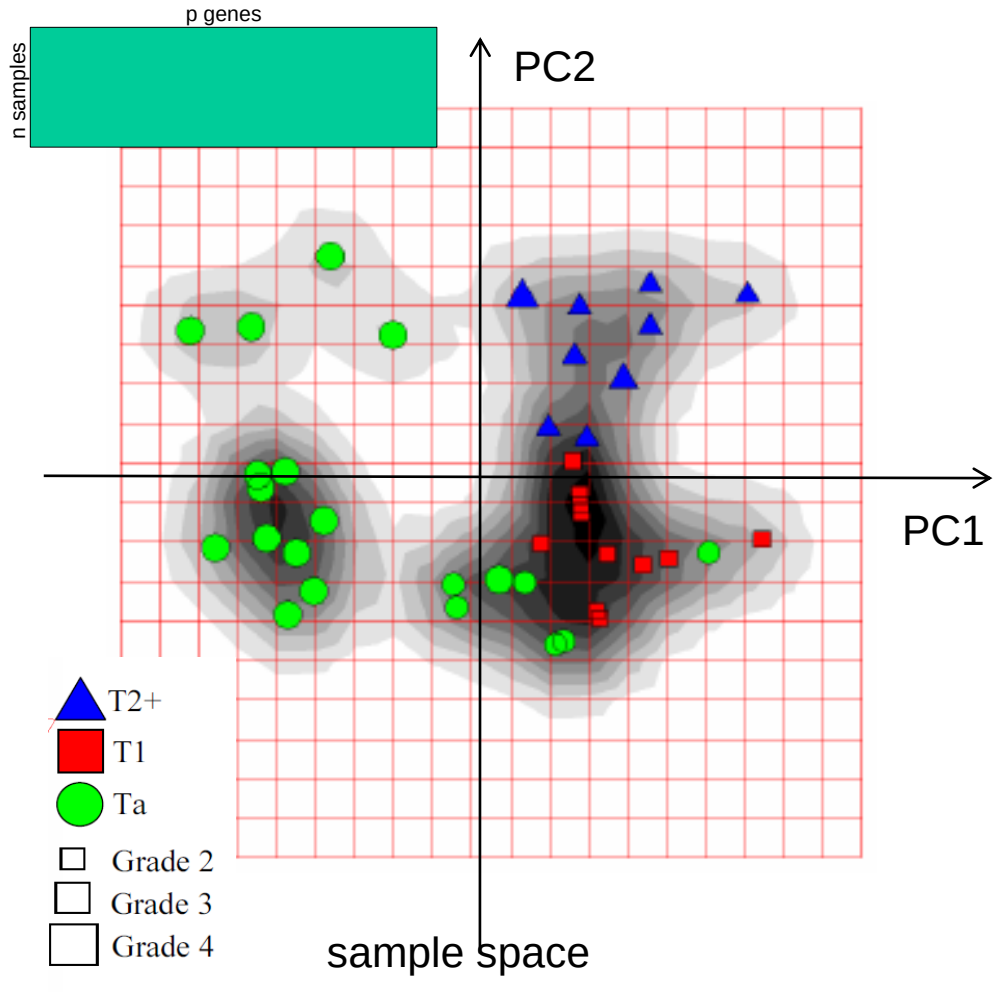
Principal (mean) 2D plane

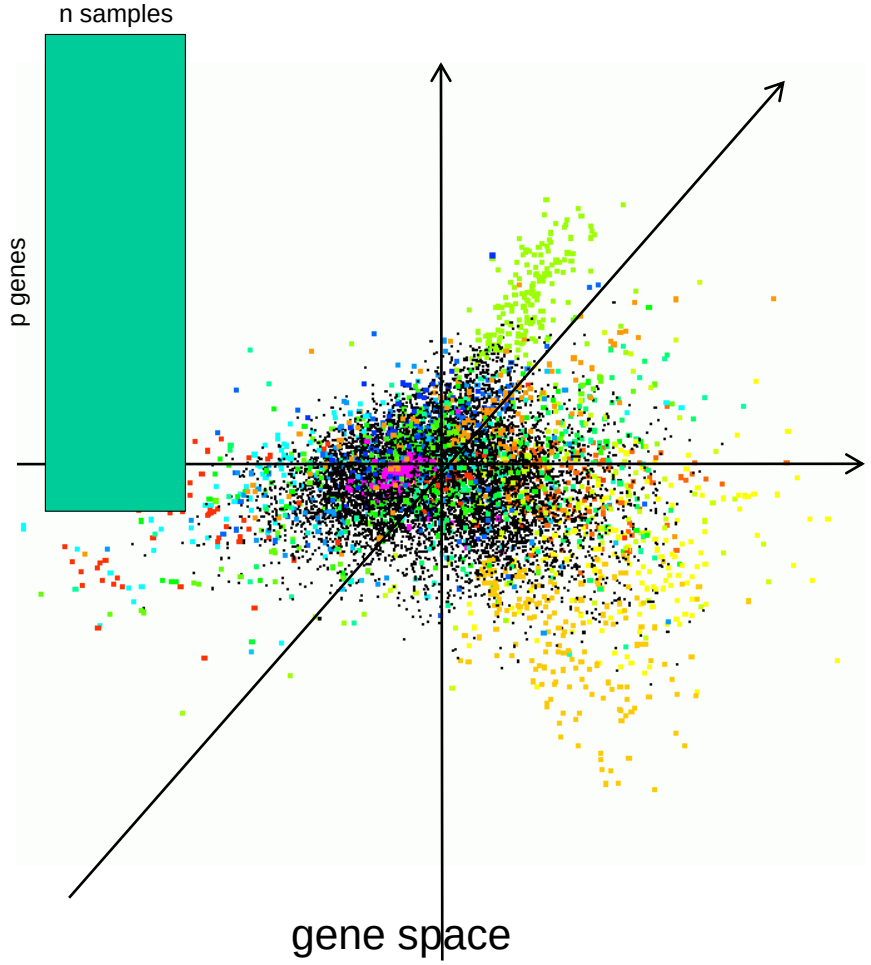


# PCA as data visualization method, based on dimension reduction



# PCA plots of transcriptomic datasets




***Classification, diagnosis, prognosis***

***Identification of molecular mechanisms, Interpretation***

# PCA as simple ‘continuous’ linear autoencoder

Given..

- an encoder function: ENCODE : $R^p \rightarrow u \in R$
- a decoder function: DECODE : $u \rightarrow R^p$
- assume the data is centered : $\sum_{i=1}^{N} \mathbf{x}_i = 0$
- assume linearity: DECODE(u) = $u\mathbf{v}_1$ , $\mathbf{v}_1 \in R^p$ is unit length vector, $(\mathbf{v}_1, \mathbf{v}_1) = 1$

$$\text{Distortion} = \sum_{i=1}^{N} \left(\mathbf{x}_i - \text{DECODE}[\text{ENCODE}(\mathbf{x}_i)]\right)^2$$

Let us minimize the distortion (just as we did in case of k-means)!

# Given DECODE(), let us find optimal linear ENCODE() for a point $\mathbf{x}_i$

DECODE($u_i$) = DECODE(ENCODE($\boldsymbol{x}_i$)) = $u_i \mathbf{v}_1$

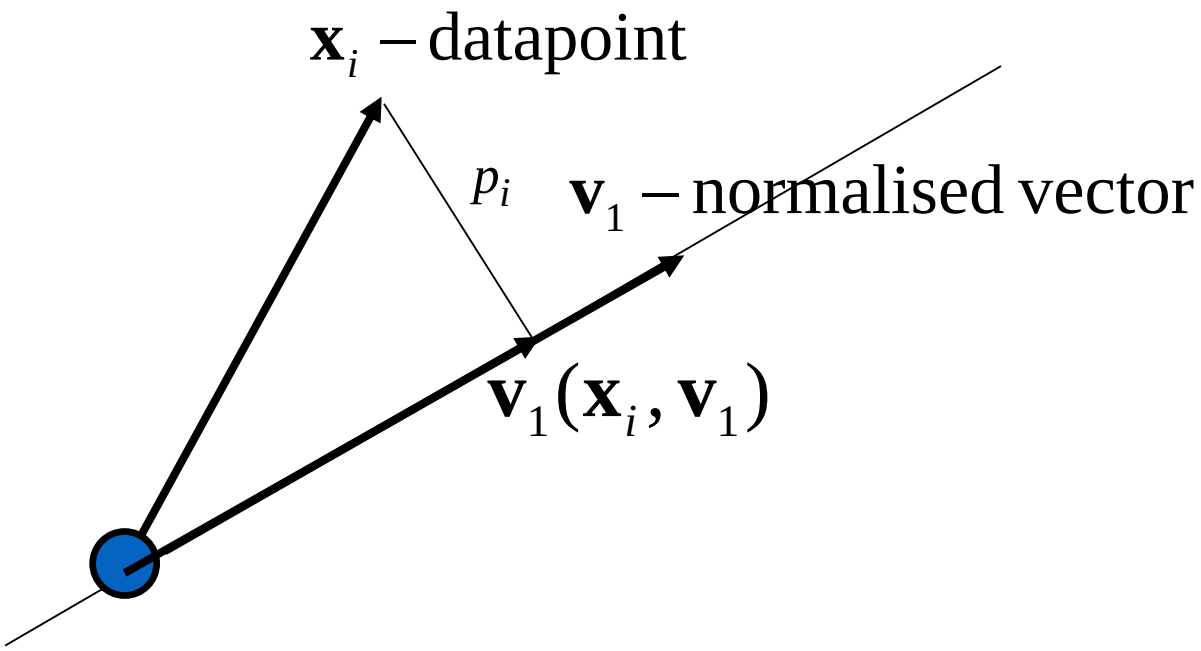
$$\text{Distortion} = \sum_{i=1}^{N} \left(\mathbf{x}_i - \text{DECODE}[\text{ENCODE}(\mathbf{x}_i)]\right)^2 = \sum_{i=1}^{N} \left(\mathbf{x}_i - u_i \mathbf{v}_1\right)^2$$

$$\frac{\partial}{\partial u_i} \text{Distortion} = -2\mathbf{v}_1(\mathbf{x}_i - u_i \mathbf{v}_1) = 0$$

$$u_i = (\mathbf{x}_i, \mathbf{v}_1)$$

Orthogonal (closest) projection!

Exercise: prove that it is the closest point on the line



# Given ENCODE(), let us find optimal DECODE()

$$\text{Distortion} = \sum_{i=1}^{N} \left(\mathbf{x}_i - \text{DECODE}[\text{ENCODE}(\mathbf{x}_i)]\right)^2 = \sum_{i=1}^{N} \left(\mathbf{x}_i - u_i \mathbf{v}_1\right)^2$$

$$\frac{\partial}{\partial \mathbf{v}_1} \text{Distortion} = -2 \sum_{i=1}^{N} u_i (\mathbf{x}_i - u_i \mathbf{v}_1) = 0$$

$$\sum_{i=1}^{N} u_i \mathbf{x}_i - \mathbf{v}_1 \sum_{i=1}^{N} u_i^2 = 0$$

$$\mathbf{v}_1 = \frac{\sum_{i=1}^{N} u_i \mathbf{x}_i}{\sum_{i=1}^{N} u_i^2}$$

1) Given DECODE(), let us find optimal ENCODE()
2) Given ENCODE(), let us find optimal DECODE()
3) Iterate until convergence!
4) At each iteration the Distortion not increasing

# Algorithm for finding the optimal linear autoencoder

1) Choose random $\mathbf{v}_1$
2) Normalize $\mathbf{v}_1 \leftarrow \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|}$
3) Compute $u_i = (\mathbf{x}_i, \mathbf{v}_1)$
4) Compute $\mathbf{v}_1 = \frac{\sum_{i=1}^{N} u_i \mathbf{x}_i}{\sum_{i=1}^{N} u_i^2}$
5) Go to 2, iterate till convergence

Simple and fast iterative algorithm for finding the first principal component (aka the best line approximating the data point cloud)

# How to find next principal component?

1) Subtract the contribution of the first component

$$\mathbf{x}_i \leftarrow \mathbf{x}_i - (\mathbf{x}_i, \mathbf{v}_1)\mathbf{v}_1$$

2) Repeat the algorithm for finding the first component, call it $\mathbf{v}_2$

3) Same for the 3rd, 4th ,…, *m*th components

# Principal Components and Singular Value Decomposition

- The best rank-one matrix approximation

$$\mathbf{x}_i \approx u_i^{(1)}\mathbf{v}_1 = (\mathbf{x}_i, \mathbf{v}_1)\mathbf{v}_1$$

$$x_i^j \approx u_i^{(1)} v_1^j$$

- The best rank-*m* matrix approximation

$$\mathbf{x}_i \approx (\mathbf{x}_i, \mathbf{v}_1)\mathbf{v}_1 + (\mathbf{x}_i, \mathbf{v}_2)\mathbf{v}_2 + \ldots + (\mathbf{x}_i, \mathbf{v}_m)\mathbf{v}_m$$

$$x_i^j \approx u_i^{(1)} v_1^j + u_i^{(2)} v_2^j + \ldots + u_i^{(m)} v_m^j = \sigma^{(1)} \upsilon_i^{(1)} v_1^j + \ldots + \sigma^{(m)} \upsilon_i^{(m)} v_m^j$$

$$(\mathbf{v}_s, \mathbf{v}_t) = 0,\ s \neq t \qquad \sigma^{(k)} = \sum_{i=1}^{N} \left(u_i^{(k)}\right)^2 \ ,\ \sum_{i=1}^{N} \left(\upsilon_i^{(k)}\right)^2 = 1,\ k = 1 \ldots m$$

$$(\mathbf{v}_s, \mathbf{v}_s) = 1$$

# Can we do it in one shot, without iterations?

$$\text{Distortion} = \sum_{i=1}^{N} \left(\mathbf{x}_i - \text{DECODE}[\text{ENCODE}(\mathbf{x}_i)]\right)^2 = \sum_{i=1}^{N} \left(\mathbf{x}_i - u_i \mathbf{v}_1\right)^2$$

$$u_i = (\mathbf{x}_i, \mathbf{v}_1)$$

Quadratic form, non-negative definite, global minimum if not degenerated!

$$\text{Distortion} = \sum_{i=1}^{N} \left(\mathbf{x}_i - (\mathbf{x}_i, \mathbf{v}_1)\mathbf{v}_1\right)^2 =$$

$$= \sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{x}_i) - 2\sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{v}_1)^2 + \sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{v}_1)^2 = \sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{x}_i) - \sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{v}_1)^2$$

$$\text{Distortion} \to \min \qquad \sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{v}_1)^2 = \sum_{i=1}^{N} u_i^{\,2} \to \max$$

$\mathbf{v}_1$ must be the direction of the maximum variance of the orthogonal projections!

# Solving the maximum variance direction problem

$$\sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{v}_1)^2 \to \max; \quad \mathbf{v}_1 = ?$$

$$\sum_{i=1}^{N} (\mathbf{x}_i, \mathbf{v}_1)^2 = \sum_{i=1}^{N} \left( \sum_{j=1}^{p} x_{ij} v_{1j} \right)^2 = \sum_{i=1}^{N} \left( \sum_{j,k=1}^{p} x_{ij} v_{1j} x_{ik} v_{1k} \right) =$$

$$= \sum_{j,k=1}^{p} v_{1j} \left( \sum_{i=1}^{N} x_{ij} x_{ik} \right) v_{1k} = N(\mathbf{v}_1, \mathbf{C(X)}\mathbf{v}_1),$$

where $\mathbf{C(X)}$ – empirical covariance matrix : $\mathbf{C(X)}_{jk} = \frac{1}{N} \sum_{i=1}^{N} x_{ij} x_{ik}$

# Properties of empirical covariance matrix

$$\mathbf{C(X)}_{jk} = \frac{1}{N}\sum_{i=1}^{N} x_{ij}\, x_{ik}$$

1. $\mathbf{C(X)}$ is symmetric: $\mathbf{C(X)}_{jk} = \mathbf{C(X)}_{kj}$;

2. $\mathbf{C(X)}$ is non-negative definite: $(\mathbf{v}, \mathbf{C(X)v}) \geq 0.$

$$\text{Indeed, } (\mathbf{v}, \mathbf{C(X)v}) = 1/N \sum_{i=1}^{N} (\mathbf{X}_i, \mathbf{v})^2 \geq 0$$

Hence, eigenvalues of $\mathbf{C(X)}$ are non-negative real numbers,

$$\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n \geq 0$$

# Principal components are eigenvectors of empirical covariance matrix, proof

$$\mathbf{C(X)}_{jk} = \frac{1}{N}\sum_{i=1}^{N} x_{ij}\, x_{ik}$$

Eigenvalues of $\mathbf{C(X)}$ are non-negative real numbers, $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_p \geq 0$;

$\mathbf{e}_1, \mathbf{e}_2, \ldots \mathbf{e}_p$ are the correspondent orthonormal eigenvectors.

We are looking for $\mathbf{v}_1 = \sum_{i=1}^{p} \varepsilon_{1i}\mathbf{e}_i, \quad \varepsilon_{1i} = (\mathbf{v}_1, \mathbf{e}_i), \quad \sum_{i=1}^{p} \varepsilon_{1i}{}^2 = 1.$

$$\mathbf{C(X)v}_1 = \sum_{i=1}^{p} \varepsilon_{1i}\mathbf{C(X)e}_i = \sum_{i=1}^{p} \varepsilon_{1i}\lambda_i\mathbf{e}_i;$$

$(\mathbf{v}_1, \mathbf{C(X)v}_1) = \sum_{i=1}^{p} \varepsilon_{1i}^2\lambda_i \rightarrow \max$ under condition $\sum_{i=1}^{p} \varepsilon_{1i}{}^2 = 1.$

Let first eigenvalues be different $\lambda_1 > \lambda_2 > \ldots$

In this case, $\varepsilon_{11}^2 = 1, \varepsilon_{1i} = 0\ (i > 1), \quad \mathbf{v}_1 = \pm\mathbf{e}_1$

# We've just learnt: two ways to compute principal components

- Iterative SVD algorithm – simple and fast
- Computing the eigenvectors of the covariance matrix *p* x *p* – can be slow, especially for large *p*
- Let us have a look in MATLAB help:

```
[...] = pca(..., 'PARAM1',val1, 'PARAM2',val2, ...) specifies optional
parameter name/value pairs to control the computation and handling of
special data types. Parameters are:

 'Algorithm' - Algorithm that pca uses to perform the principal
               component analysis. Choices are:
     'svd'   - Singular Value Decomposition of X (the default).
     'eig'   - Eigenvalue Decomposition of the covariance matrix. It
               is faster than SVD when N is greater than P, but less
               accurate because the condition number of the covariance
               is the square of the condition number of X.
```

# Attention, in sklearn, PCA can be non-deterministic!

## sklearn.decomposition.PCA

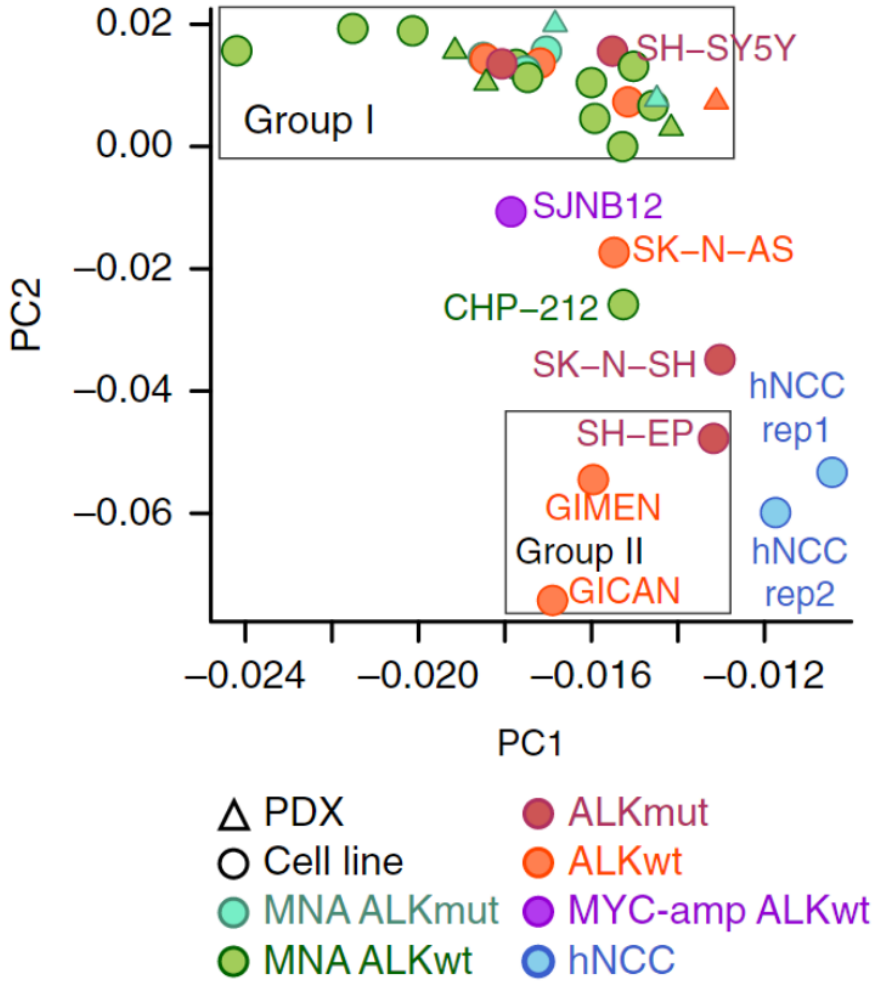
*class* sklearn.decomposition.**PCA**(*n_components=None*, *, *copy=True*, *whiten=False*, *svd_solver='auto'*, *tol=0.0*, *iterated_power='auto'*, *random_state=None*) [source]

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

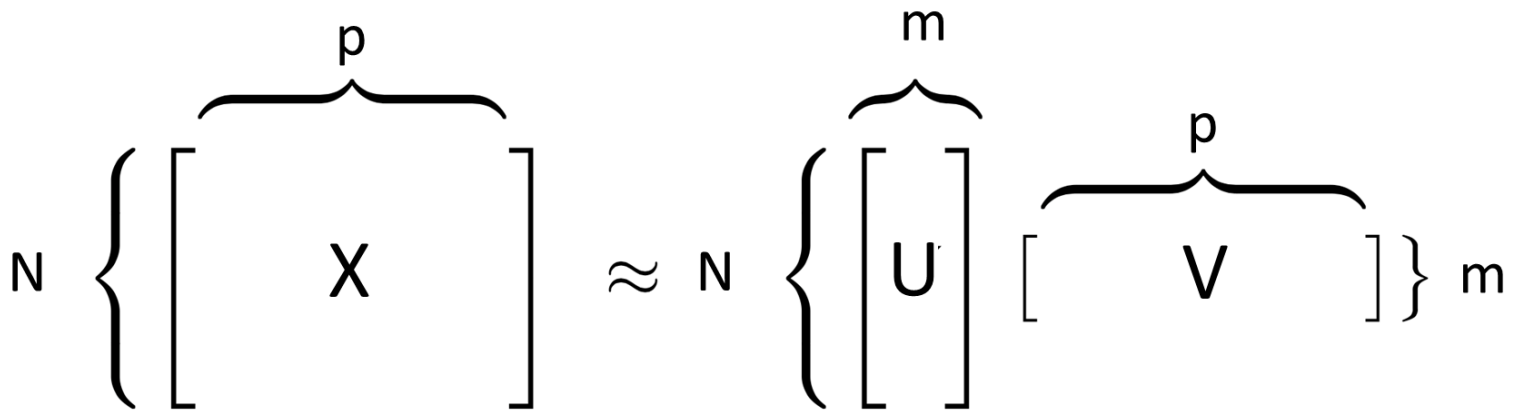
It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

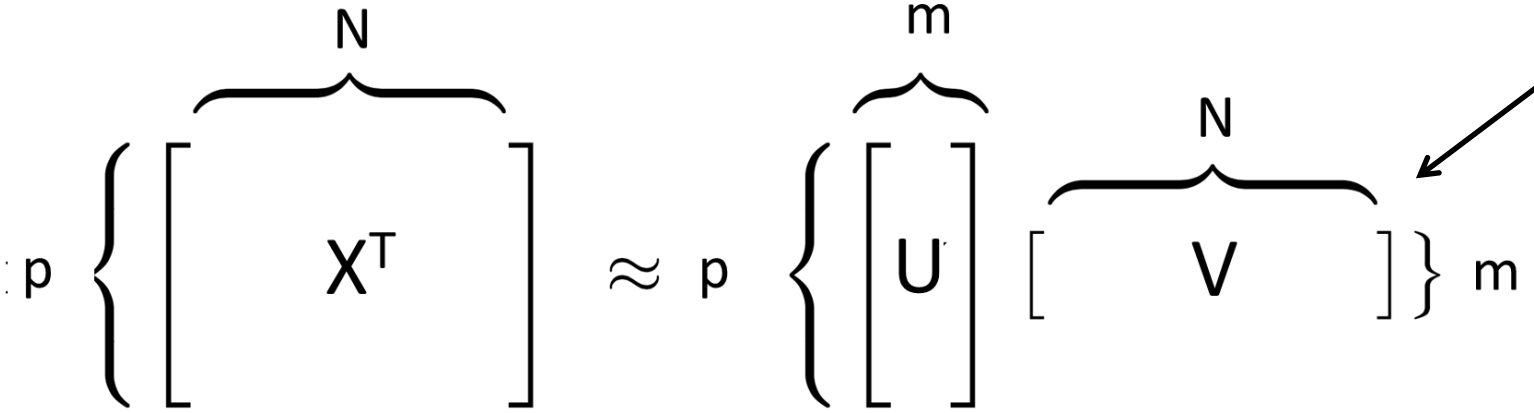
# ‘Impossible’ PCA plots



# ‘Impossible’ PCA plots

U – Object projections
V – Coordinate ‘loadings’




Using loadings for data visualization is incorrect! Because they do not reflect the distances in $R^m$

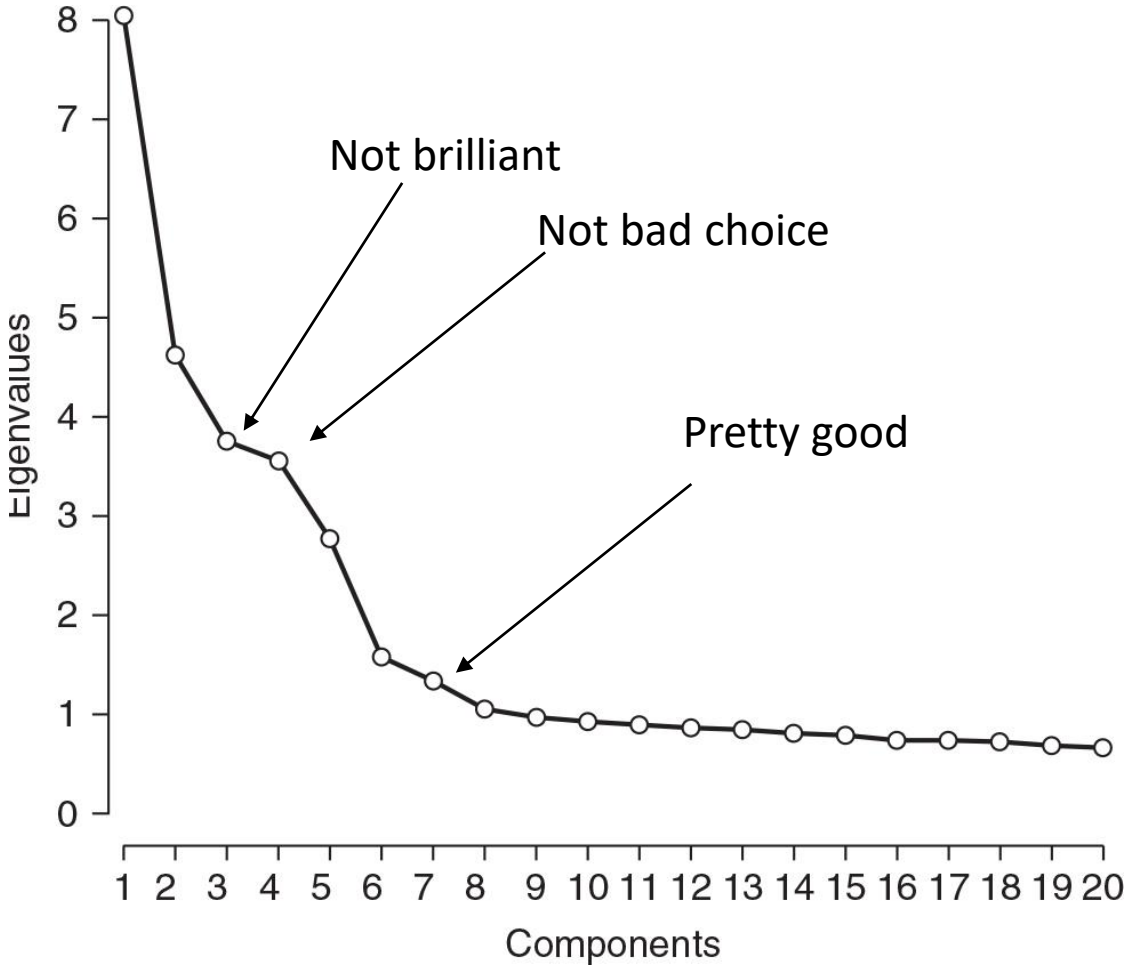
# How many principal components one can compute?

- If $p < N$ then one can compute $p$ components
- If $p \geq N$ then one can compute at maximum N-1 components (this is the maximum rank of the empirical covariance matrix)
- *Exercise: demonstrate this fact in Python*

# How many principal components to retain?

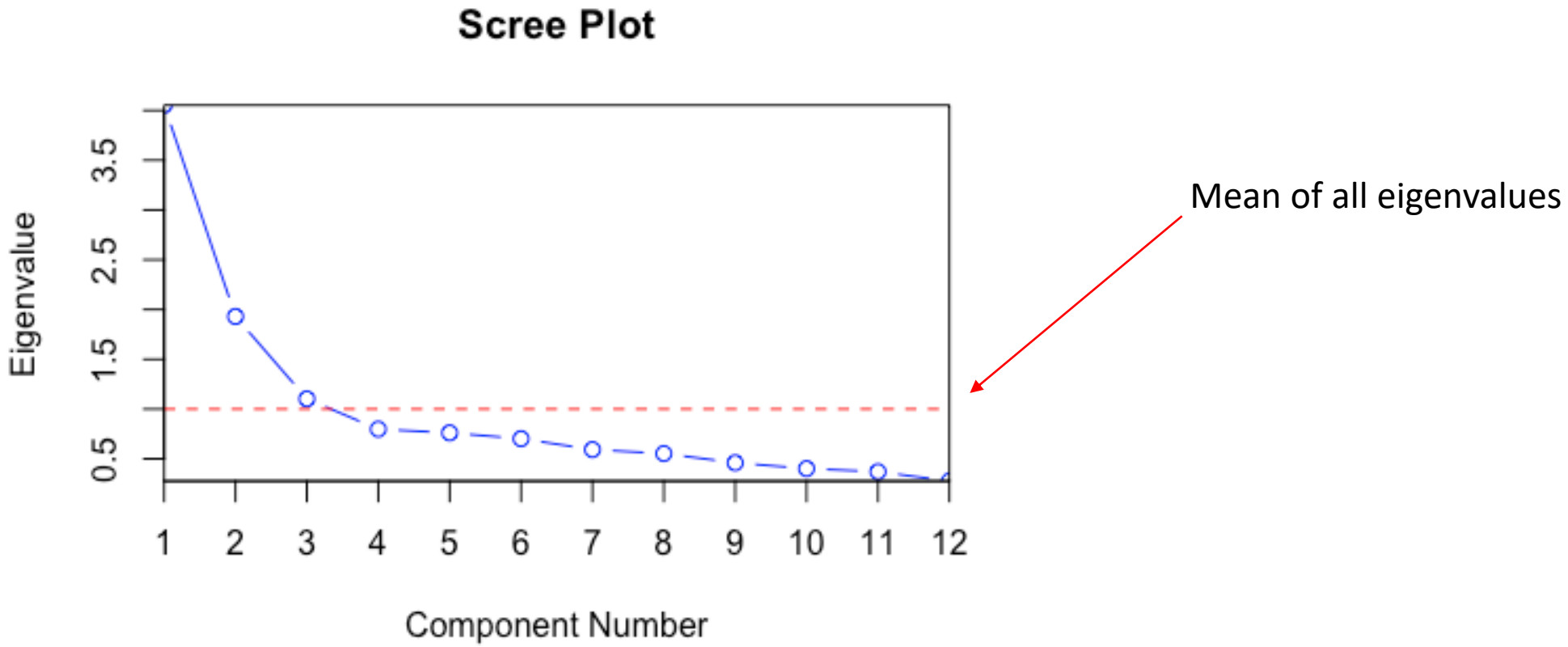
- Scree plot

- If eigenvalues are normalized to unity sum, then the y-axis is called ‘the fraction of variance explained (FVE)’



# How many principal components to retain?

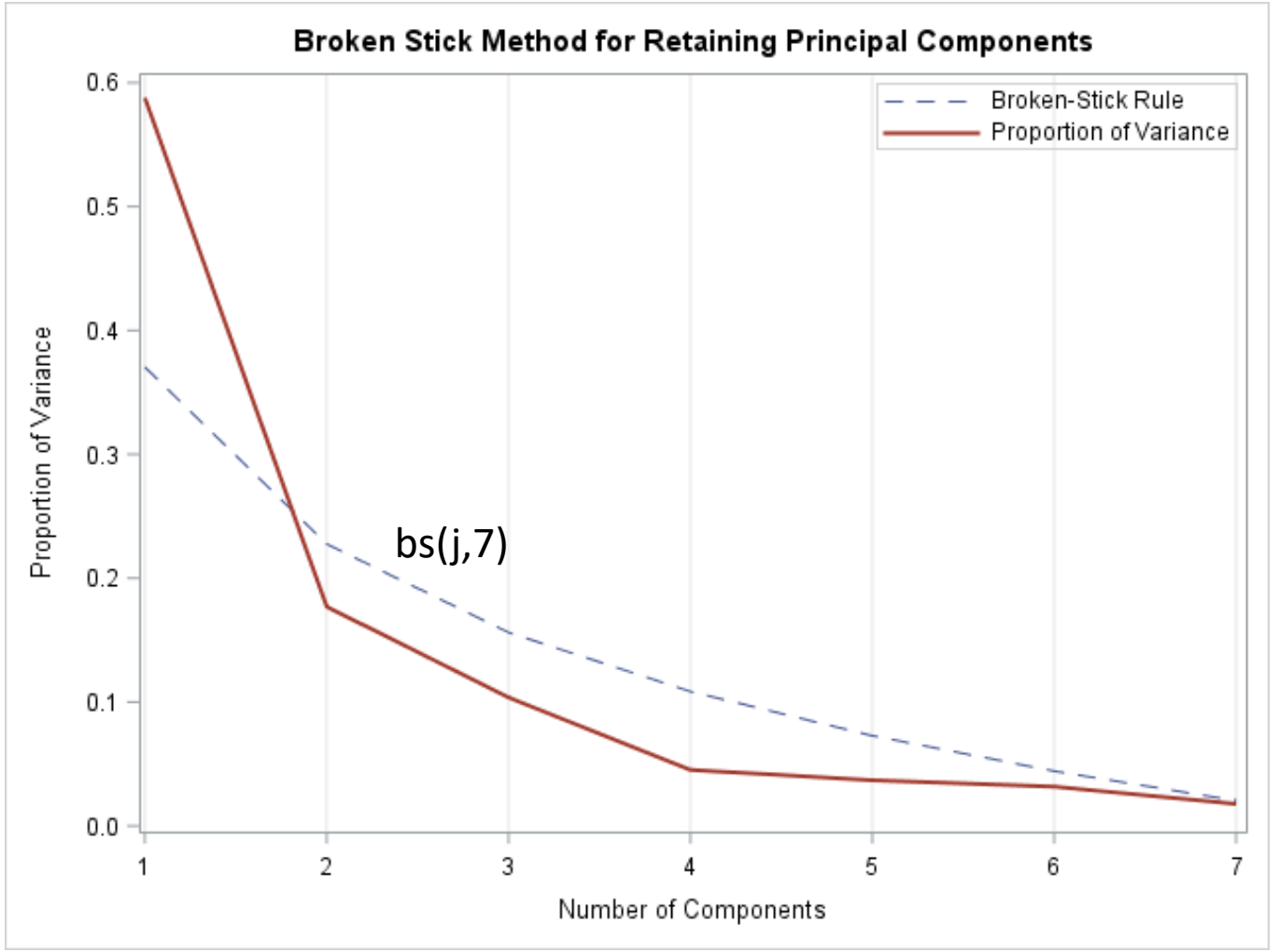
- Kaiser rule



# How many principal components to retain?

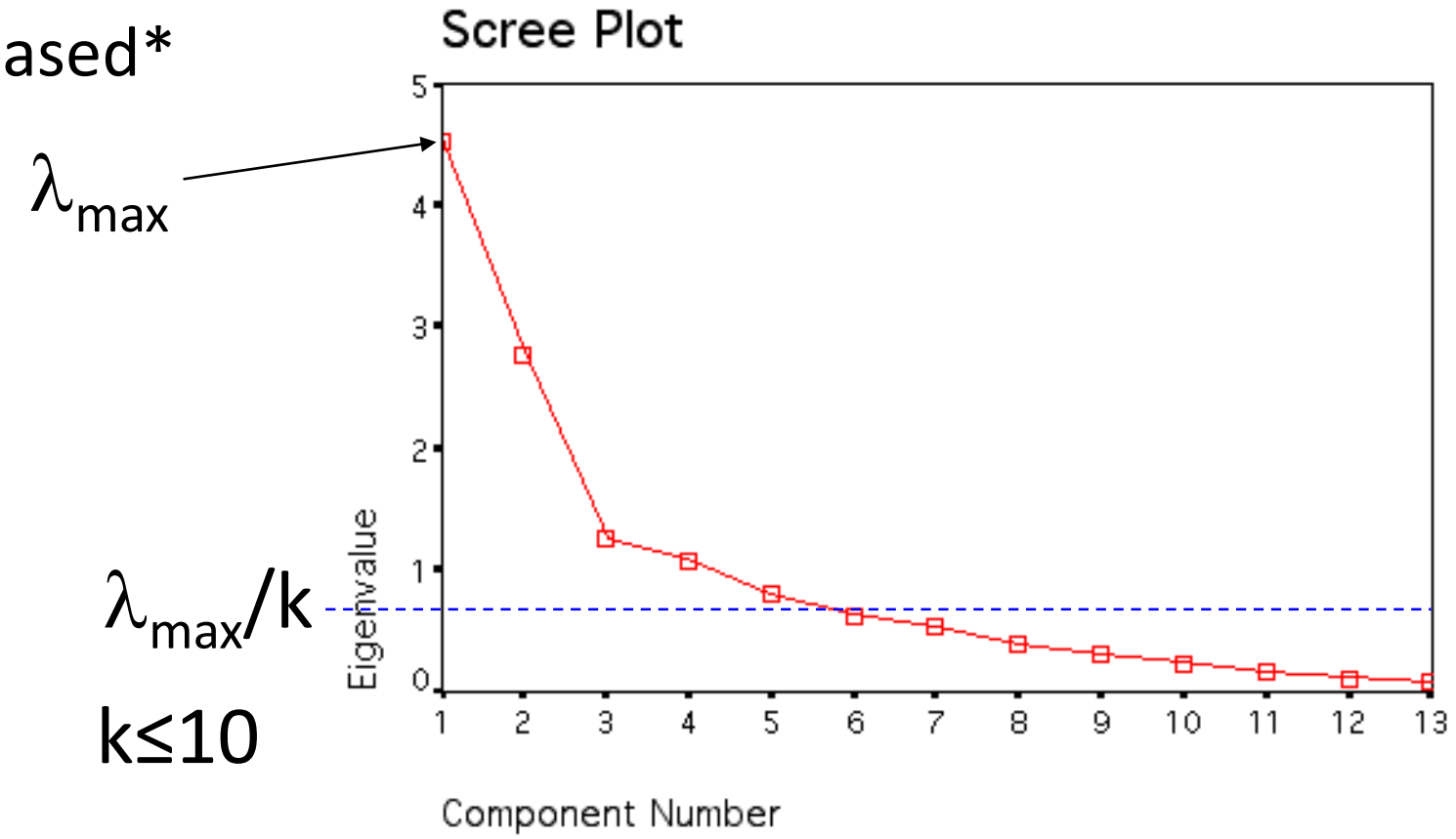
- Broken-stick model

- bs(j,p) is the expected (mean) length of the *j*th longest segment of a unit length stick randomly broken in p segments



# How many principal components to retain?

- Condition number-based*




*Least dependent on the number of redundant features

1700
handwritten
digits
32x32 pixels
$R^{1024}$

# Four definitions of PCA

Geometrical approach

- **Classical**: approximate the data point cloud by linear manifold of small dimension
- **Textbook:** Find such a linear subspace in which the variance of orthogonally projected data points is maximal
- Find such a linear subspace in which the sum of pairwise distances between orthogonally projected data points is maximal

Probabilistic approach

- For a given probability distribution, find such an orthogonal transformation of the coordinate system that correlations between new coordinates will become zero

# Case of REALLY BIG DATA

- Case of N >> p, p ~< 10000 : incrementally compute the covariance matrix
- Case p >> N > 10000 : subsampling or probabilistic approaches such as Probabilistic PCA or Random SVD
- Use on-line learning for PCA (e.g., neural networks!)
- Case of sparse data : Sparse PCA
- Incremental principal component analysis (IPCA) is typically used as a replacement for principal component analysis (PCA) when the dataset to be decomposed is too large to fit in memory

# Much more about PCA…

- PCA with data point weights…
- Robust PCA (e.g., PCA using L1 metrics)
- PCA for data with missing values without pre-imputation or filtering (alternating least squares matrix factorization )
- Eigen-decomposition of the correlation matrix (scaled PCA)
- Performing PCA on a correlation matrix
- Generalized Singular Value Decomposition
- Common Principal Component Analysis
- Tensorial PCA
- Supervised PCA
- Kernel PCA