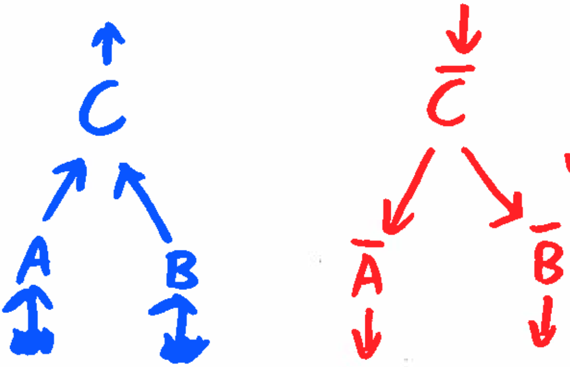
# Reverse mode differentiation

Piece of computation:




Backpropagate derivatives wrt final scalar output

For any $Z$,

$$\bar{Z}_{ij} = \frac{\partial \text{output}}{\partial Z_{ij}}$$

## Use standard rules:

Matlab .*

Python *

$C = \cos A \;\Rightarrow\; \bar{A} = \bar{C} \odot \sin A$

$C = AB \;\Rightarrow\; \bar{A} = \bar{C}B^\top, \quad \bar{B} = A^\top\bar{C}$

$C = A + B \;\Rightarrow\; \bar{A} = \bar{C}, \quad \bar{B} = \bar{C}$

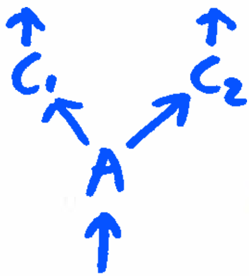
$C = A^\top \;\Rightarrow\; \bar{A} = \bar{C}^\top$

...

Stored in forward pass

Passed in by backprop

## Multiple children




$$\bar{C}_1 \downarrow \quad \bar{C}_2 \downarrow$$

$$\bar{A} = \bar{A}_1 + \bar{A}_2$$

Apply rules separately for children and add

# Matrix multiplication

$$\underset{L\times N}{C} = \underset{L\times M}{A}\;\underset{M\times N}{B} \qquad O(LMN)$$

$$C_{\ell n} = \sum_m A_{\ell m} B_{mn}$$

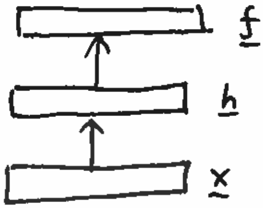
$\ell n$: LN terms; $\sum_m$: cost $O(M)$ each

Square matrix-matrix multiply $O(N^3)$

There are $O(N^2)$ numbers in the matrices

[See also tutorial 5, Q1]

# Autoencoder (Unsupervised)



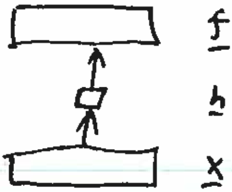

Learning task

$\underline{f} \approx \underline{x}$

not useful

def autoencode(x):
    ~~return x~~
    h = np.dot(I, x)
    f = np.dot(I, h)
    return f

## Dimensionality Reduction




D outputs

K units $K \ll D$

D inputs

$K \times D$

$$\underline{h} = g^{(1)}\left(W^{(1)}\underline{x} + \underline{b}^{(1)}\right)$$

$$\underline{f} = g^{(2)}\left(W^{(2)}\underline{h} + \underline{b}^{(2)}\right) \leftarrow \text{decoder}$$

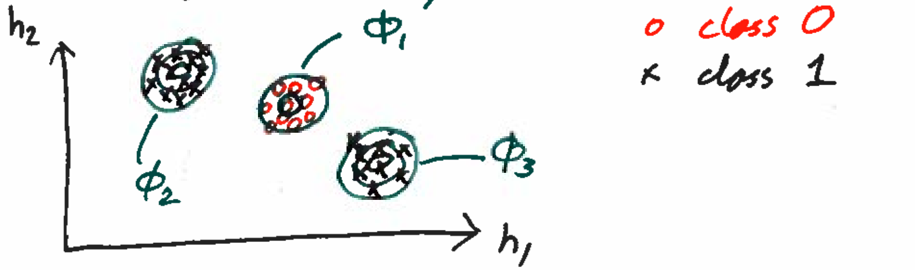
Encoder could be useful

Train this thing on a huge dataset

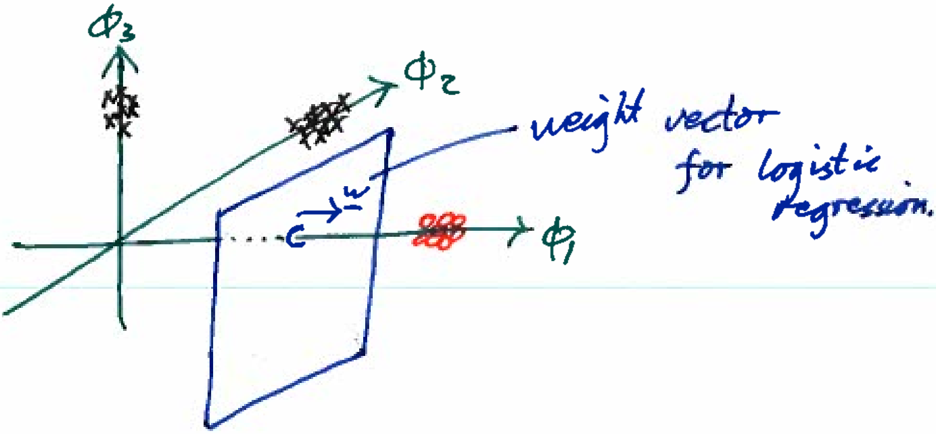
Use $\underline{h}(\underline{x}; W^{(1)}, \underline{b}^{(1)})$ to transform data

# Visualization

Set $k=2$




We might want to increase dim. of data



# Sparse Autoencoder

Encourage most elements of $\underline{h}$ to be close to zero – sparse

# Denoising Autoencoder

While training we mask out (delete) some of the inputs, set some $x_d$ to zero

$\underline{m}$ mask vector, of random 0's and 1's

Cost on an example $\quad \| \underline{f}(\underline{x}^{(n)} \odot \underline{m}) - \underline{x}^{(n)} \|^2$

Cost function $\quad \sum_{\underline{m}} p(\underline{m}) \frac{1}{N} \sum_{n=1}^{N} \| \underline{f}(\underline{x}^{(n)} \odot \underline{m}) - \underline{x}^{(n)} \|^2$

Monte Carlo

$\approx$ pick random $\underline{m}$
random $n$

# Principal Components Analysis (PCA)

Linear auto-encoder $\quad g^{(1)}(a) = g^{(2)}(a) = a$

$$\underline{h} = V^T(\underline{x} - \bar{\underline{x}})$$

Training set mean

$$\underline{f} = V\underline{h} + \bar{\underline{x}}$$

shared $D \times K$ matrix

## PCA advantages

- Fit columns of $V$ to be eigenvectors of the covariance of data

  (No SGD!)

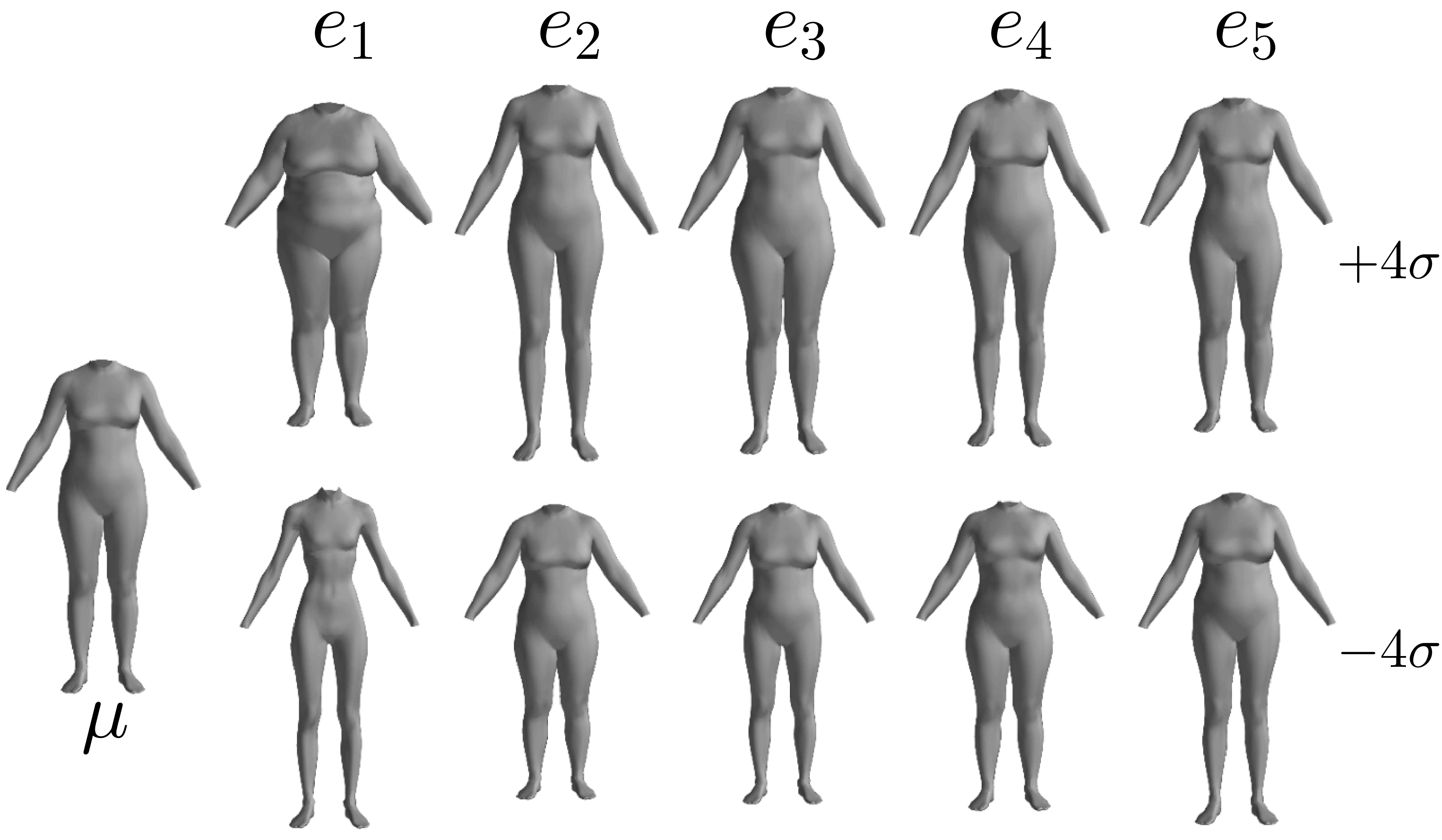
- Same answer every time.

- The solutions for different $K$ they're nested

  $h_1(\underline{x})$ it's the same for all $K$

  $h_2(\underline{x})$ " " " " " $K \geq 2$

# PCA applied to bodies




Freifeld and Black, ECCV 2012

# PCA applied to DNA

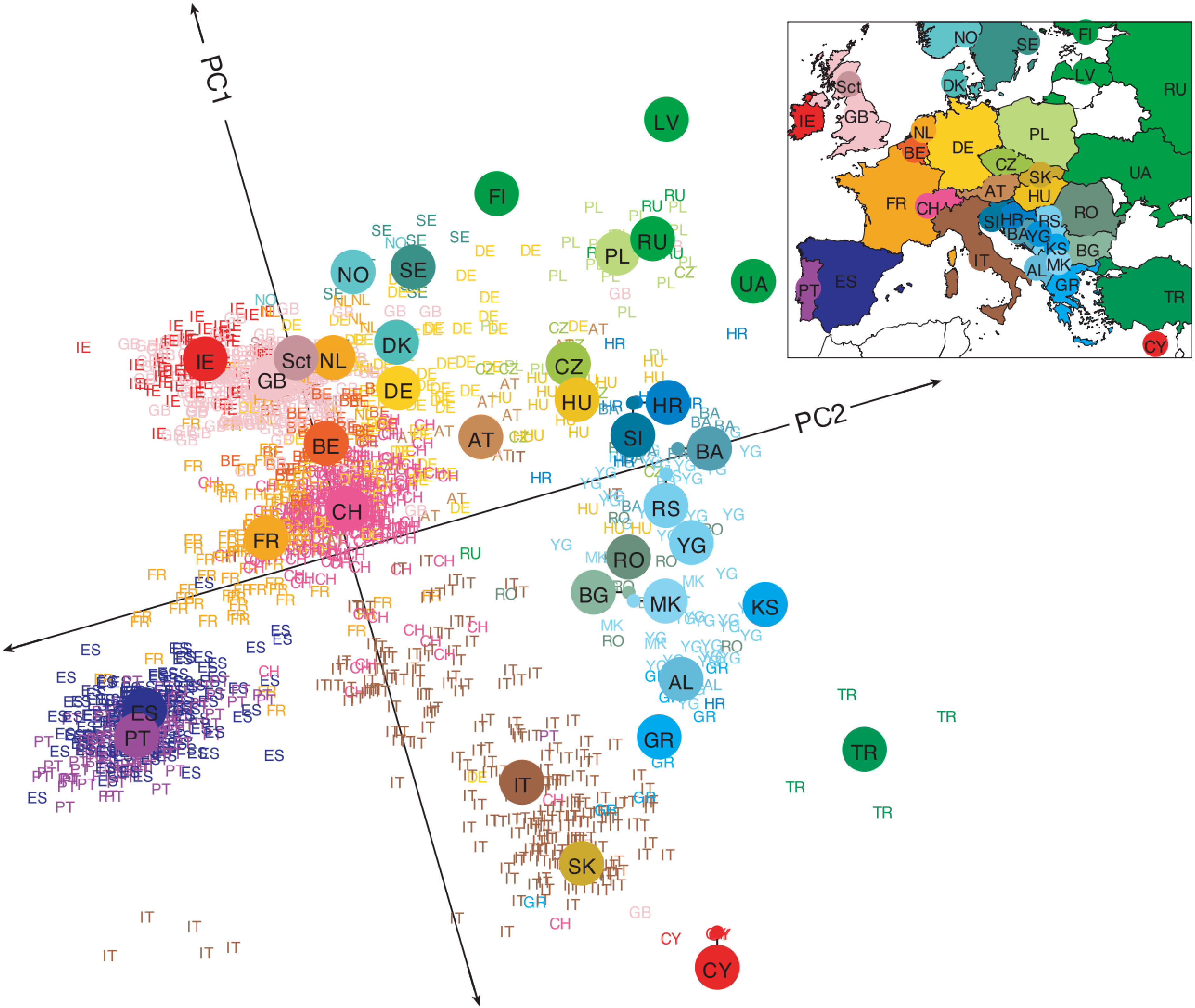
Novembre et al. (2008) — doi:10.1038/nature07331

Carefully selected both individuals and features

1,387 individuals

197,146 single nucleotide polymorphisms (SNPs)

Each person reduced to two(!) numbers with PCA



PC1
PC2
LV
FI
RU
PL
UA
NO
SE
DK
IE
Sct
GB
NL
DE
CZ
HU
HR
SI
BA
BE
AT
CH
FR
RS
YG
RO
BG
MK
KS
AL
GR
TR
ES
PT
IT
SK
CY

# MSc course enrollment data

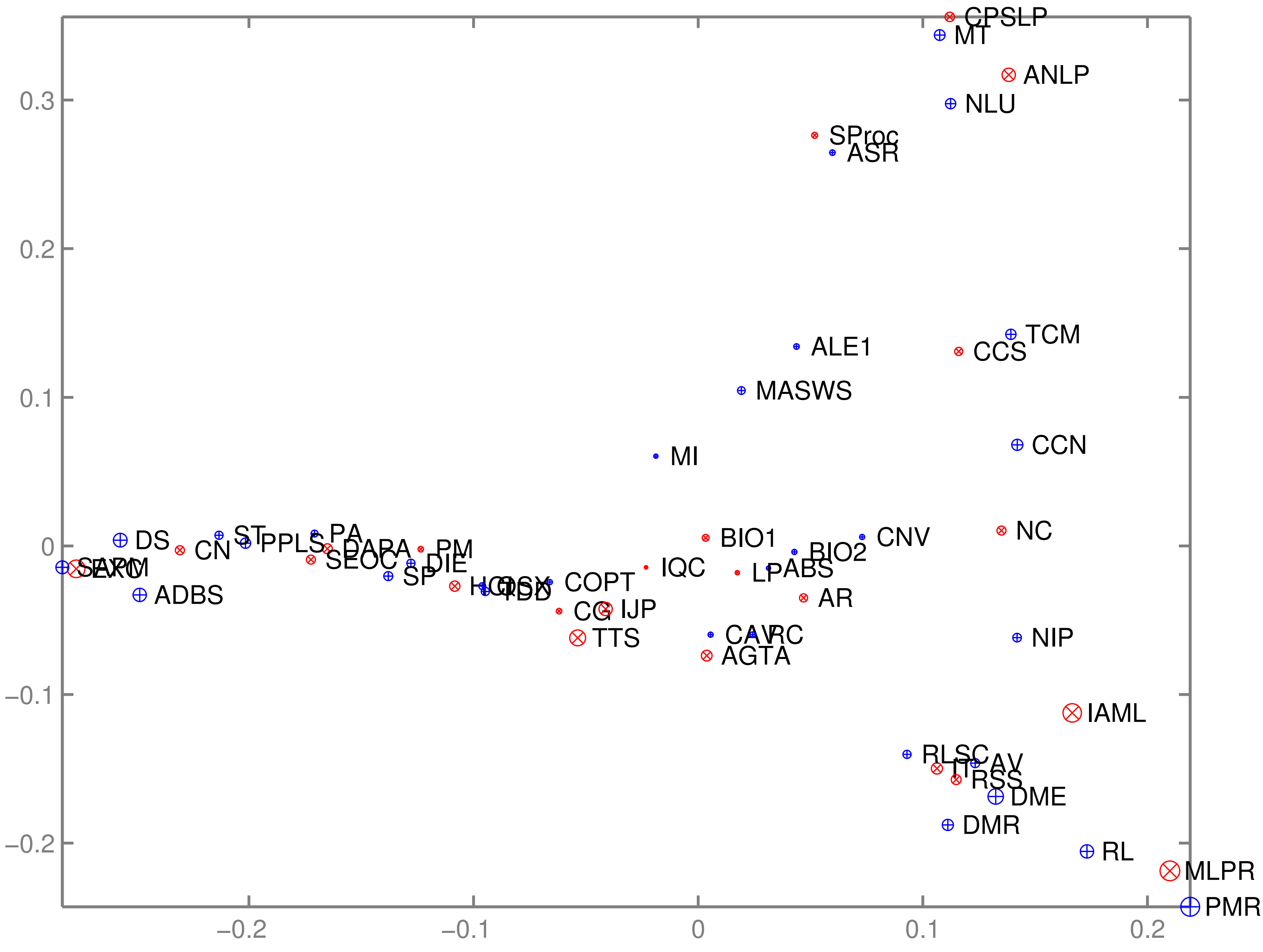
Binary $S \times C$ matrix $M$

$M_{sc} = 1$, if student $s$ taking course $c$

Each course is a length $S$ vector

. . . OR each student is a length $C$ vector

# PCA applied to MSc courses



# PCA applied to MSc students