npj | Digital Medicine

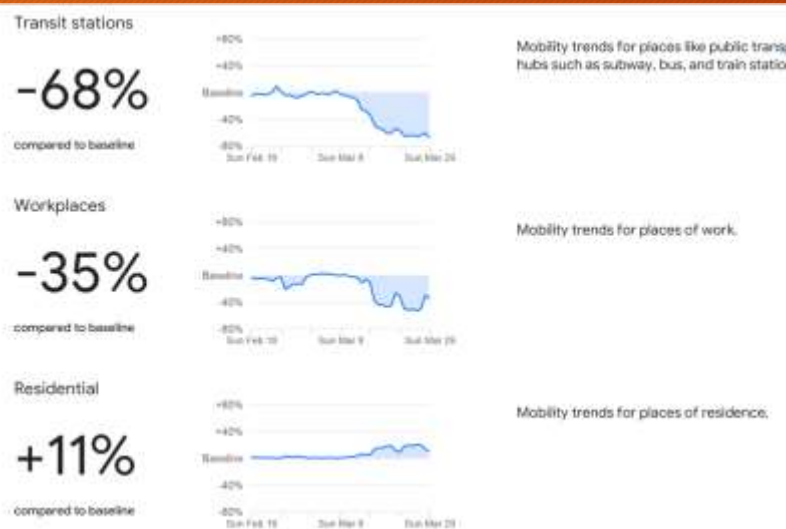www.nature.com/npjdigitalmed

ARTICLE   OPEN

Machine learning-based prediction of COVID-19 diagnosis based on symptoms

Yazeed Zoabi[1], Shira Deri-Rozov[1] and Noam Shomron[1]

Supervised and unsupervised learning

ADS2

Gedi Lukšys

April 15, 2024

Learning objectives

Today's plan:

- main elements & techniques of supervised learning

- a simple example of supervised learning and gradient ascent approach

- an overview of different supervised and unsupervised learning techniques

- relevant considerations and applications of supervised and unsupervised learning

Supervised learning: what is it for?

Learn to predict relations between 2 sets of data: „inputs" ➔ „outputs"

Apt. size: 50, 75, 150, 67, 34, 80m^2

No of rooms: 2, 4, 5, 3, 2, 3

input → black-box ? → output

Rent:
$700, 950, 1400, 1000, 550, 1200

0, 1, 2, …, 9

Spam / no Spam

„Abu Malik:
CONFIDENTIAL TRUST
REPLY URGENTLY"

„bahn.de: Vielen Dank
für Ihren Fahrkartenkauf!
(Auftrag ABCXYZ)"

Input typically multidimensional
(sometimes very much so…)

Output often unidimensional, **discrete**
(classes) or **continuous**

Formally speaking: classification vs. regression

- Supervised learning deals with learning a function f(X, W) = Y, based on a number of available data points $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, … , $(x^{(N)}, y^{(N)})$, also called **examples**

  - where $x^{(i)}$ is usually a vector $(x_1^{(i)}, x_2^{(i)}, … , x_M^{(i)})$

  - $y^{(i)}$ is either a number (=> **regression**)

  a class {0, 1} or a vector of classes (=> **classification**)

  W – model parameters (also a vector), M – dimensionality

- Most common example: linear regression / classifier

Why is it called „supervised"?

- Suppose our model simulates a function y = f(x)

- We can apply it to inputs (x-s) from existing examples to calculate outputs, i.e. $f(x^{(1)}), f(x^{(2)}), \ldots$

- In the beginning the model has not learned to predict the function correctly yet, but as we have the corresponding example („teacher") outputs, we can measure the mean squared error (MSE):

$$MSE = ((y^{(1)} - f(x^{(1)}))^2 + \ldots + (y^{(n)} - f(x^{(n)}))^2)/n$$

- Sometimes root mean square error (RMSE) is used instead.

This is called the **error** or **objective** function. The goal is to minimize it.

A simple example – linear model

- Model: y = ax + b

Objective function:
$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - ax^{(i)} - b \right)^2$$
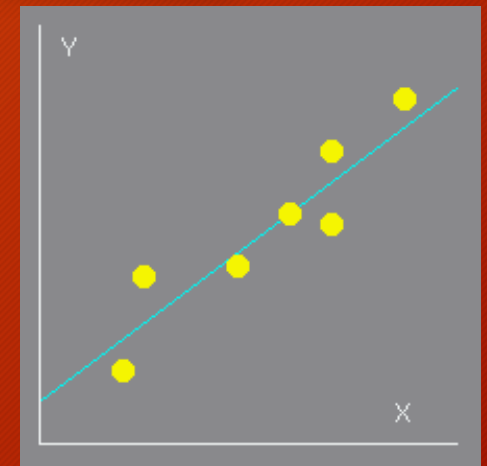
How can we make our model minimize the error?

We have 2 **parameters**: a & b

- We have to find their values that minimize the error!

How?

1. **Guess!** (need to start somewhere)

2. **Adjust guesses**, e.g. try neighboring values, go in the direction that decreases the error most => hill climbing
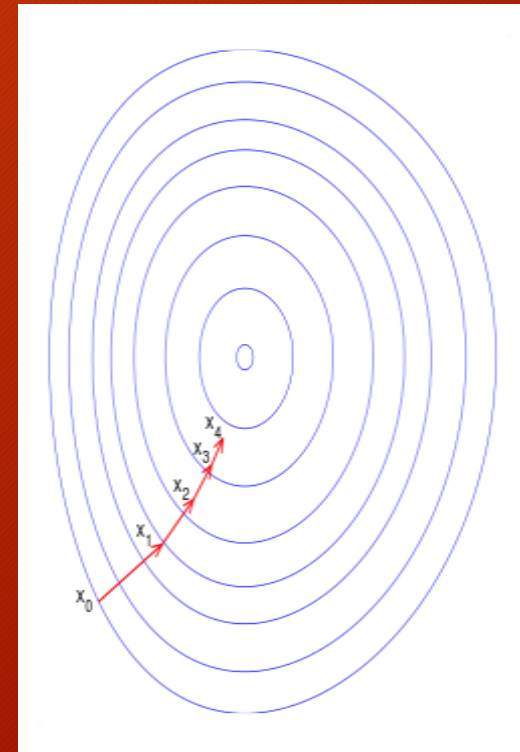
Parameter initialization
Typically random

A standard method: **gradient descent**

- Mean squared error (MSE) is a function of parameters (a and b)

- Derivatives δMSE / δa and δMSE / δb should be 0 at the minimum of MSE.

- How to get there fastest? (analytical solution aside ☺)

- Go along the steepest direction:

  Δa = – α δMSE / δa

  Δb = – α δMSE / δb

- Update of parameters is called a **learning rule**, and **α** is the learning rate.

http://en.wikipedia.org/wiki/File:Gradient_descent.png

Let's try!

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - ax^{(i)} - b\right)^2$$

- Learning rules:

$$\Delta a = \frac{-\alpha}{N}\sum_{i=1}^{N}2\left(y^{(i)} - ax^{(i)} - b\right)\left(-x^{(i)}\right)$$

$$\Delta b = \frac{-\alpha}{N}\sum_{i=1}^{N}2\left(y^{(i)} - ax^{(i)} - b\right)(-1)$$

- Examples: (0,1), (1,3), (4,6), (2,4), (3,4)
- Try different learning rates

α = 0.03
#1: a=0.56, b=0.22
#2: a=0.9, b=0.35
#3: a=1.1, b=0.44
#4: a=1.21, b=0.5
#5: a=1.28, b=0.54
#10: a=1.36, b=0.64
#20: a=1.32, b=0.77
#50: a=1.23, b=1.03
#100: a=1.15, b=1.25
#200: a=1.11, b=1.38

Too slow

α = 0.08
#1: a=1.5, b=0.58
#2: a=1.38, b=0.58
#3: a=1.37, b=0.62
#4: a=1.36, b=0.66
#5: a=1.35, b=0.69
#10: a=1.29, b=0.85
#20: a=1.22, b=1.06
#50: a=1.13, b=1.32
#100: a=1.1, b=1.39
#200: a=1.1, b=1.4

α = 0.13
#1: a=2.44, b=0.94
#2: a=0.59, b=0.36
#3: a=1.93, b=0.89
#4: a=0.9, b=0.6
#5: a=1.63, b=0.91
#10: a=1.17, b=0.97
#20: a=1.16, b=1.22
#50: a=1.11, b=1.38
#100: a=1.1, b=1.4
#200: a=1.1, b=1.4

α = 0.16
#1: a=3.01, b=1.15
#2: a=-0.5, b=0.01
#3: a=3.46, b=1.48
#4: a=-1.12, b=-0.06
#5: a=4.07, b=1.83
#10: a=-4.25, b=-0.85
#20: a=-19.92, b=-6.11
#50: a=-1206.95, b=-422.37
#100: a=-1030083.04, b=-361332.61
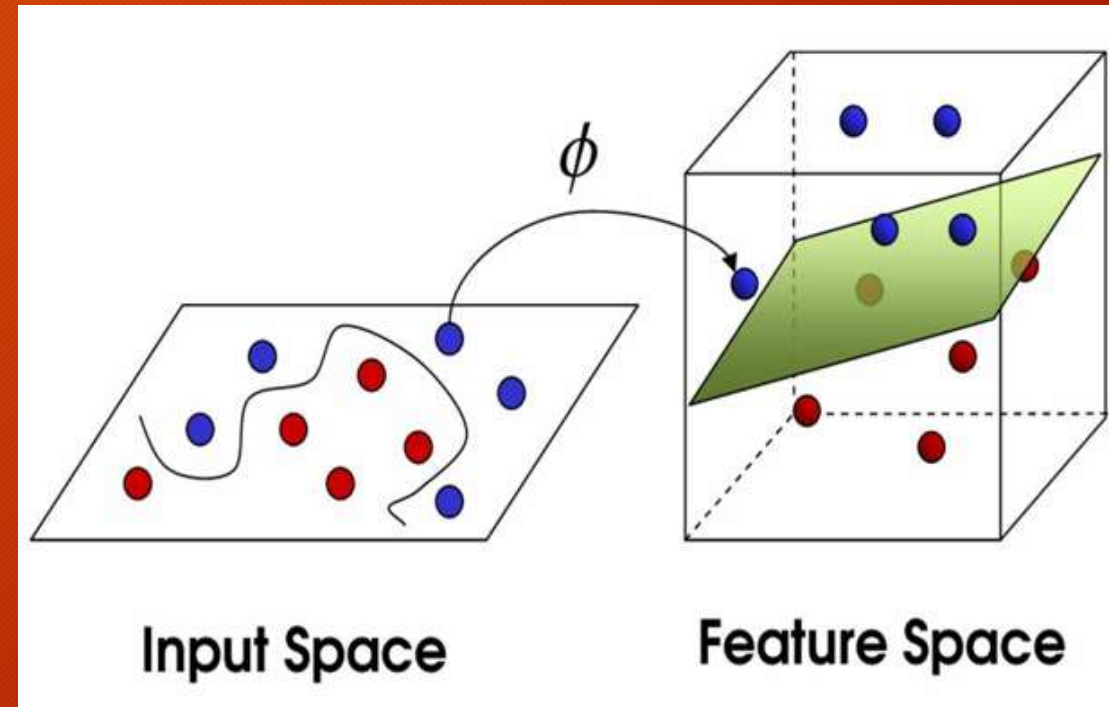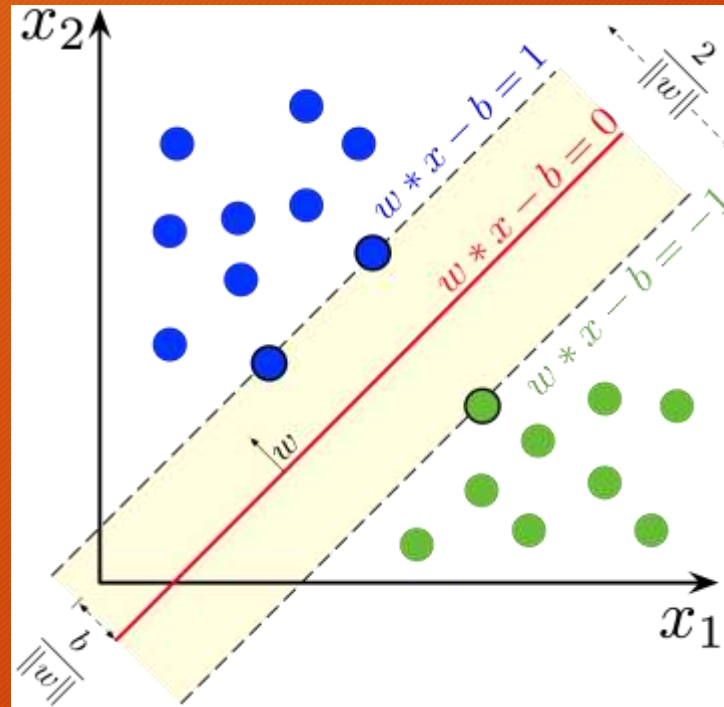#200: a=-748941284697.65, b=-262714417242.33

Artificial neural networks (multi-layer perceptrons)

- Inspired by early ideas on neural networks in the brain, particularly the hierarchically organized visual system („the perceptron")

- Trained using *backpropagation*

- Were shown to have *universal approximation property*

- Still very popular, although recently eclipsed by deep learning (which is a special case)
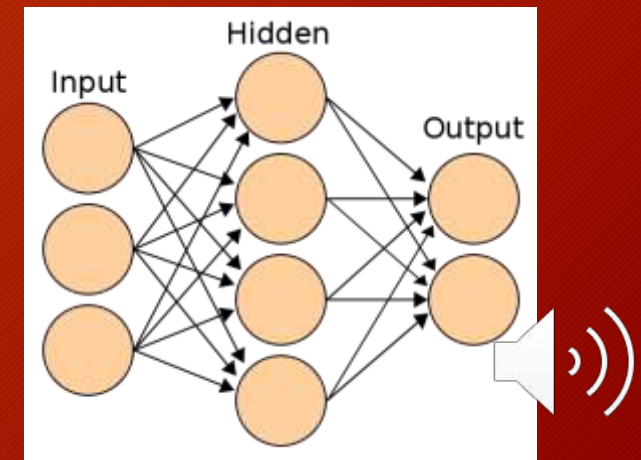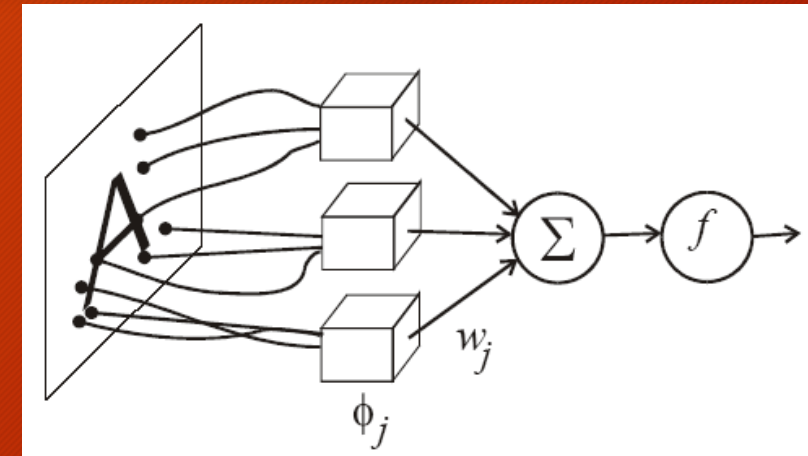
Support vector machines (SVMs) and kernels

- Finding boundaries with largest margins between classes

- Kernels: projecting to higher dimensions

http://www.imtech.res.in/raghava/rbpred/svm.jpg

Shooting fireworks into the sky or gene -> protein parallels ☺

https://en.wikipedia.org/wiki/Support-vector_machine

Unsupervised learning: a clustering example

- Can be solved with K-means
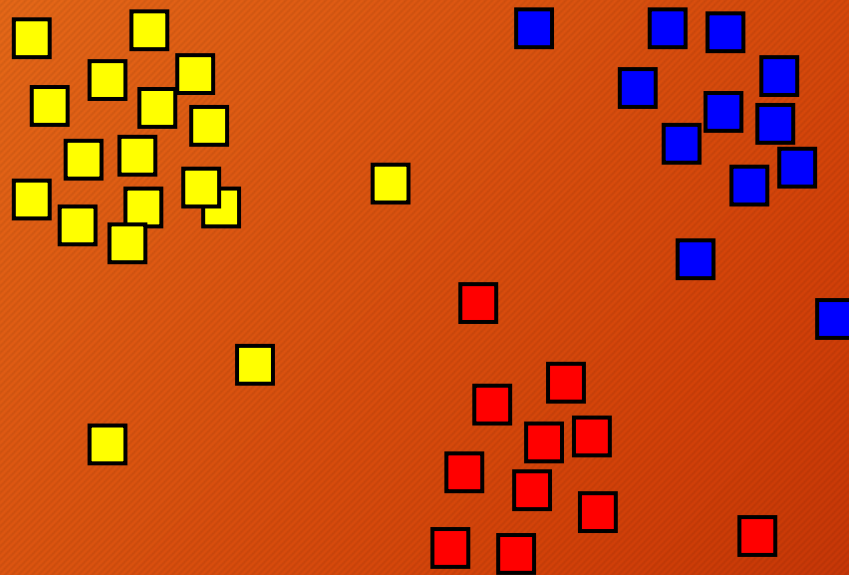
Is it really simple?

How many clusters?

How to treat borderline points?

https://en.wikipedia.org/wiki/Cluster_analysis

More on unsupervised learning: Gaussian mixtures

- Clusters vary in sizes & shapes, modeled by Gaussians
- Calculated using "expectation-maximization" algorithm

https://en.wikipedia.org/wiki/Cluster_analysis
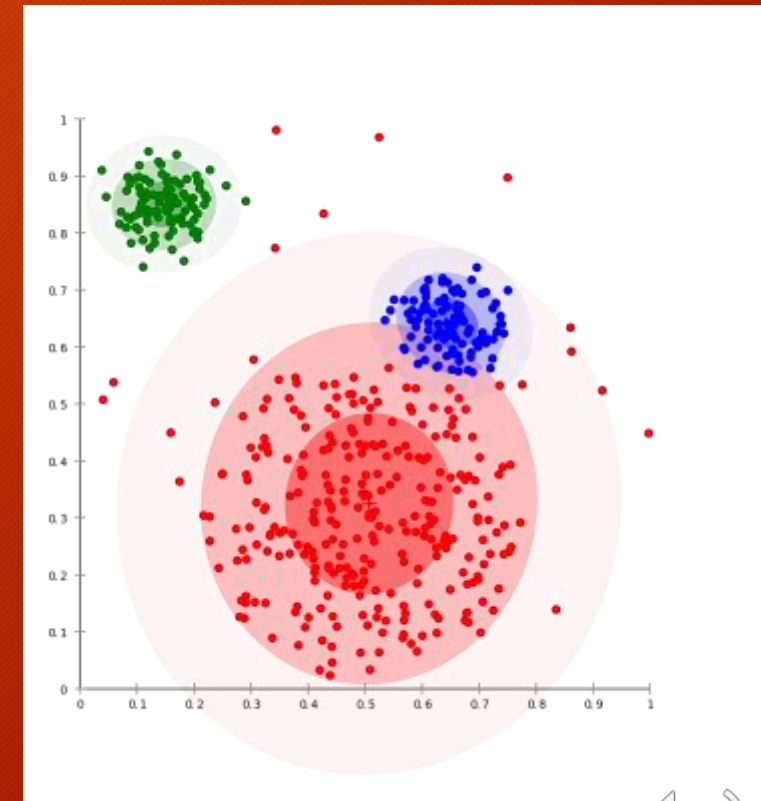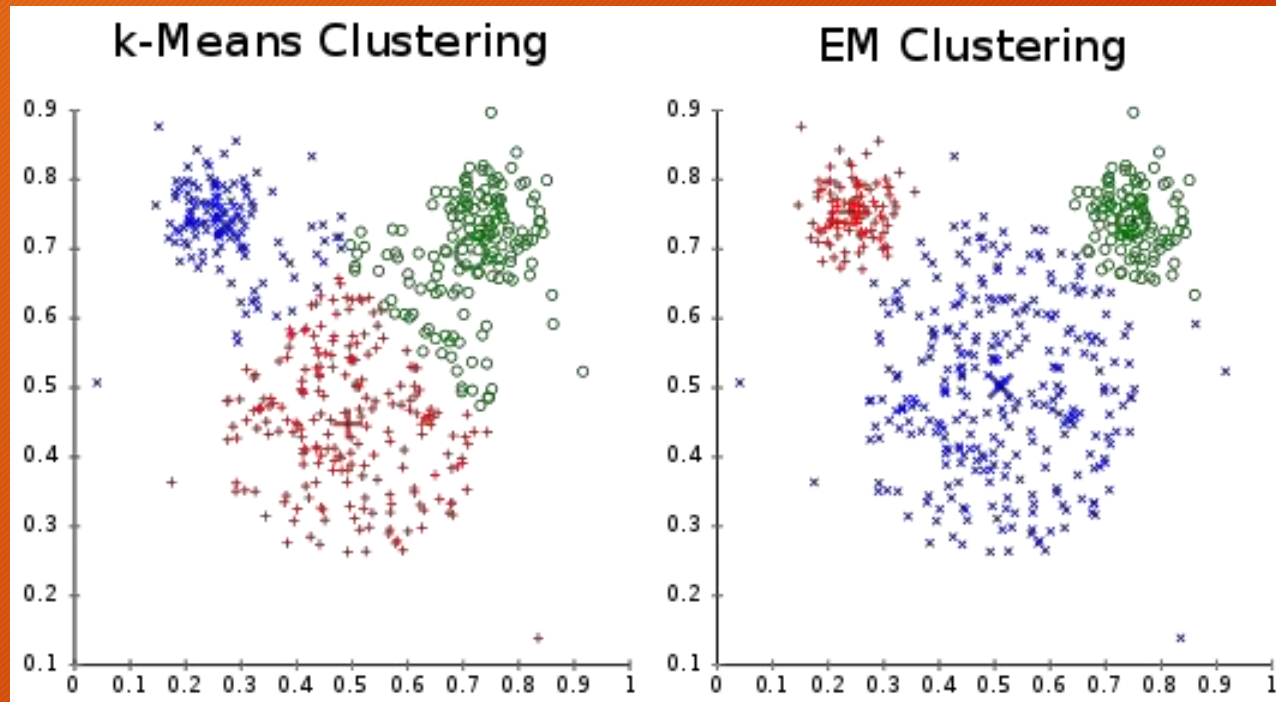
Gaussian mixtures for supervised learning

- Classes have complex distributions and may consist of multiple Gaussian clusters

A.W.Moore's bioassay example
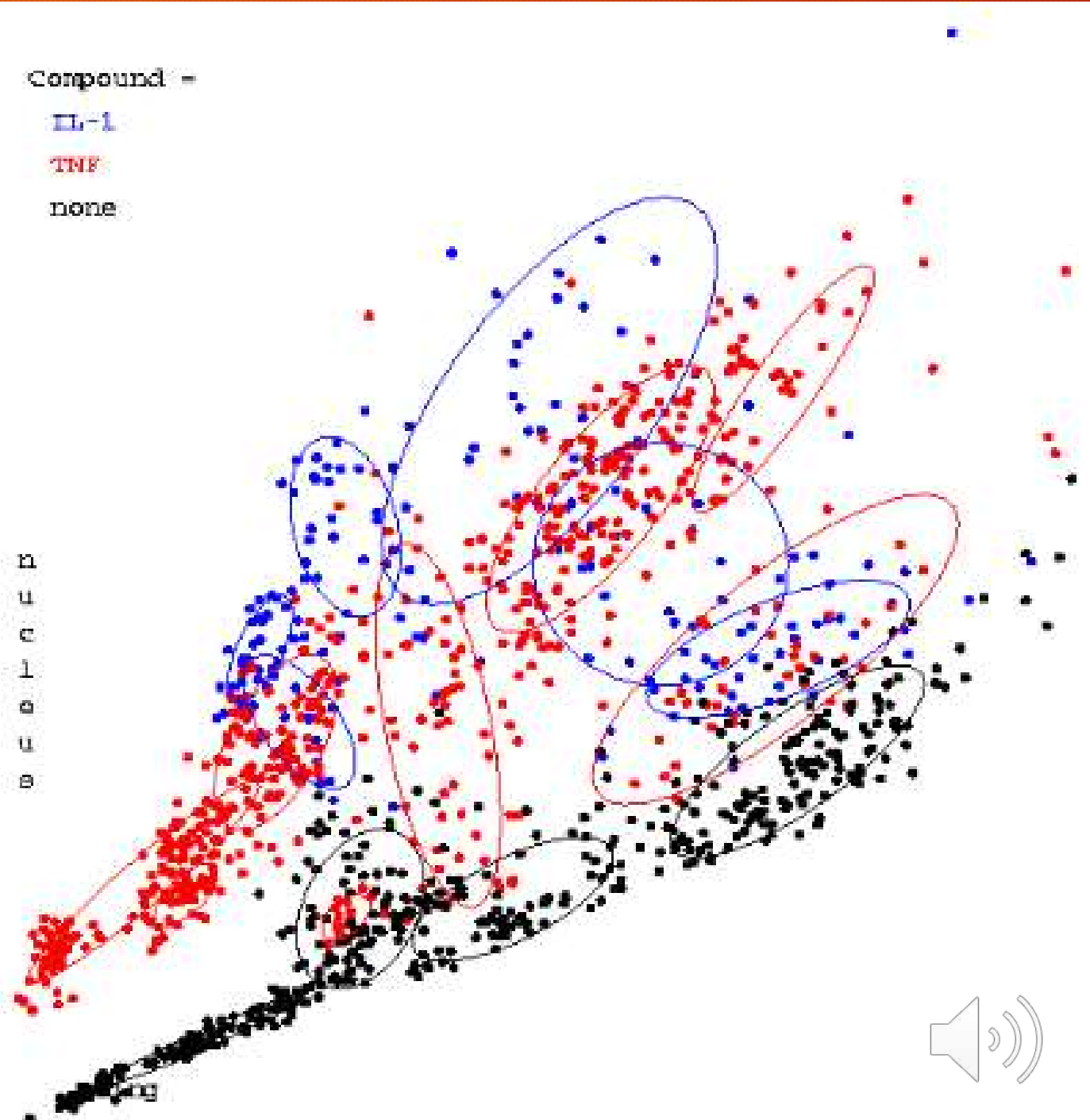
# Clustering vs. dimensionality reduction

- Clustering reduces the number of data points (into much fewer clusters)

  - *e.g. K-means, Gaussian mixtures*

- Dimensionality reduction reduces the number of dimensions per data point

  - *e.g. Principal/Independent Component Analyses*

- Use similar principles but in different direction / for different purpose (leading to either fewer data or fewer dimensions)

- In supervised learning dimensionality reduction often used before prediction – WHY?

Principal component analysis – rotate data to maximize variance
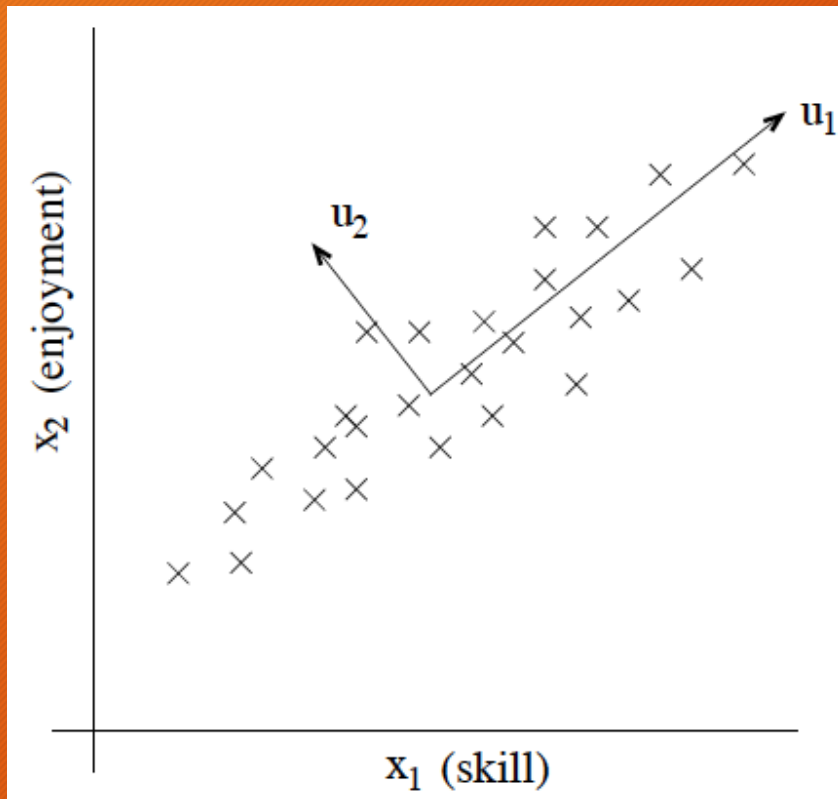
Helicopter driving – Andrew Ng
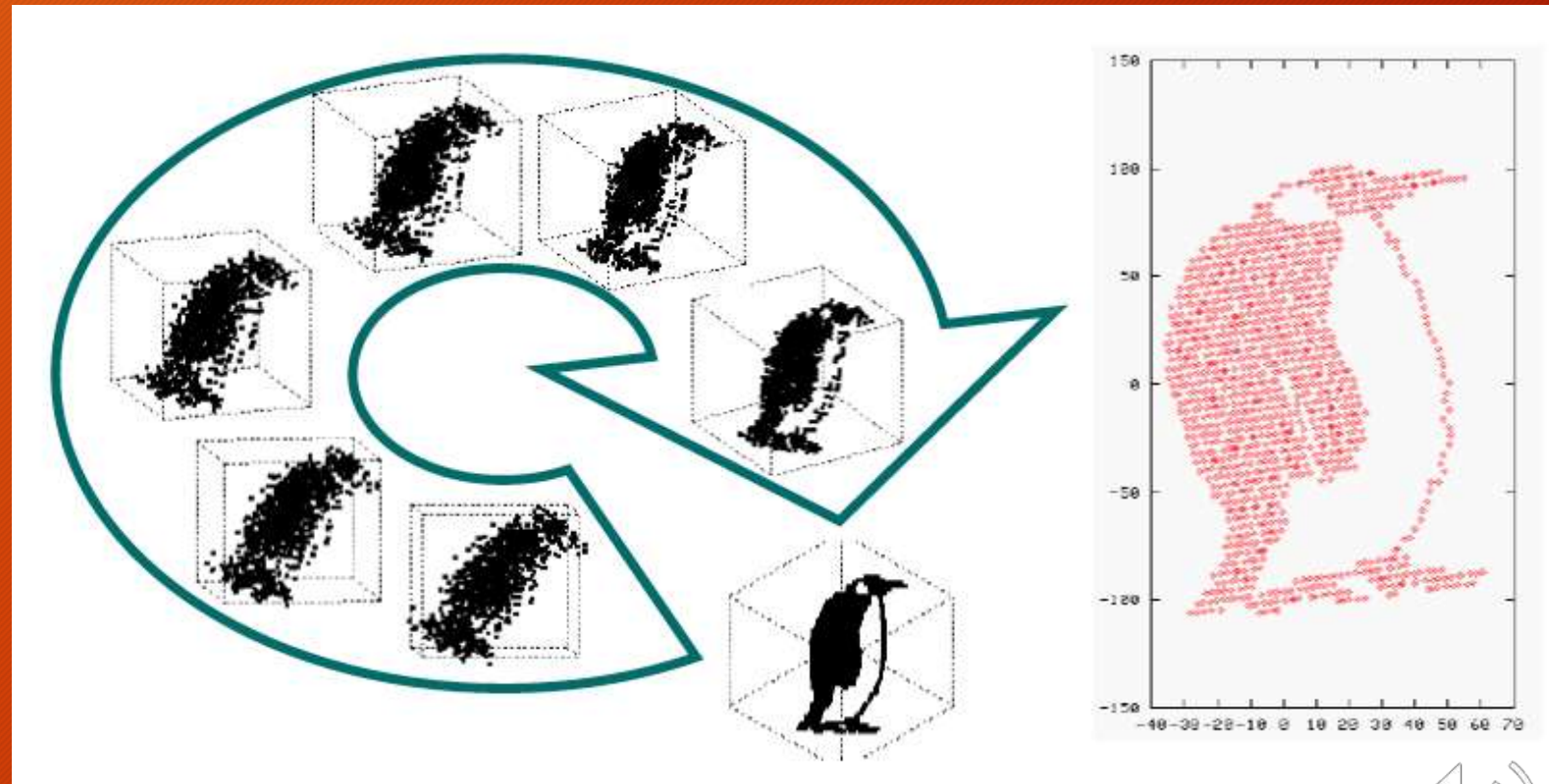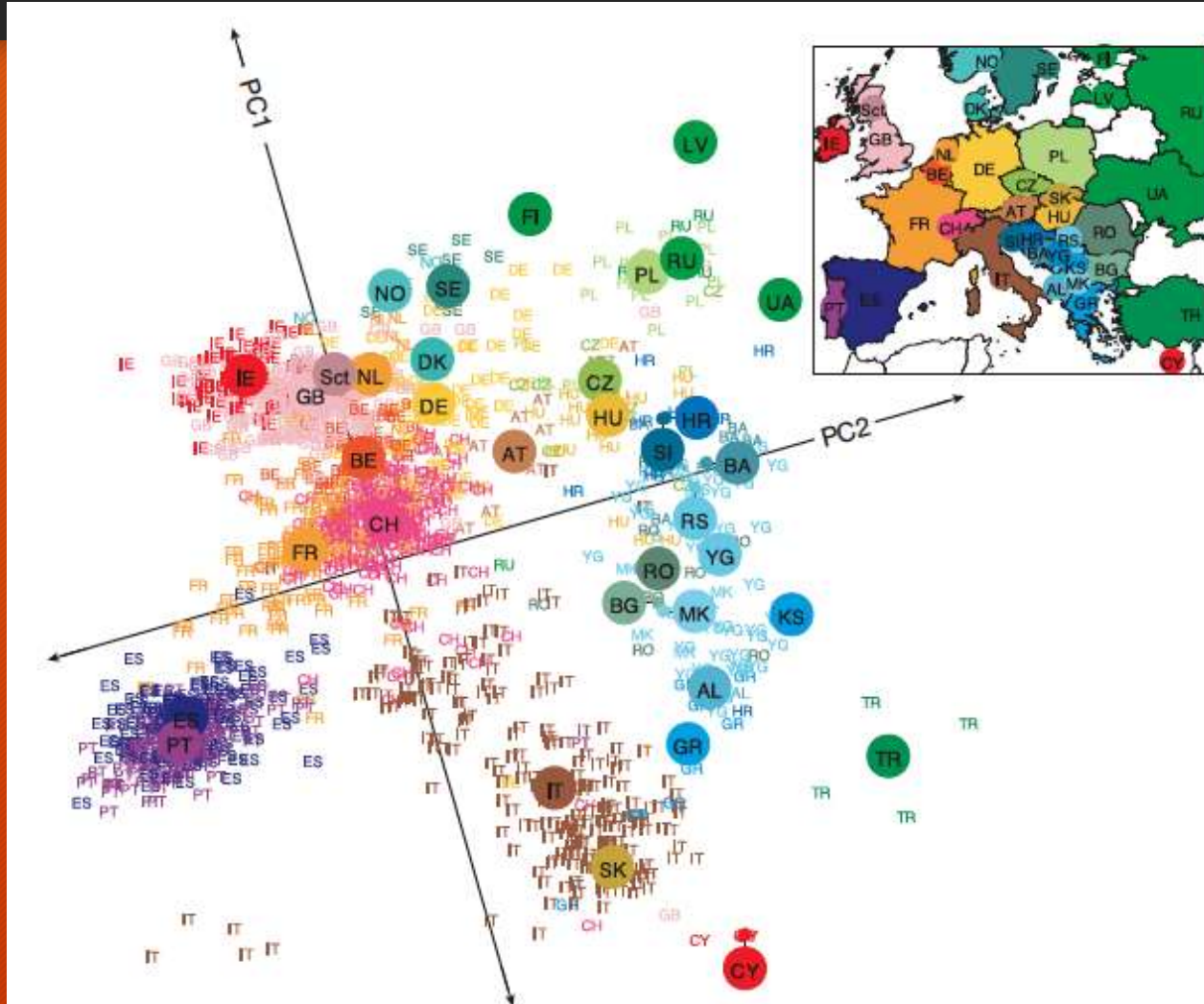
Example by R.Gutierrez-Osuna

PCA can help finding good features

- Principal components are orthogonal / decorrelated, but not always easily interpretable

Where they are:

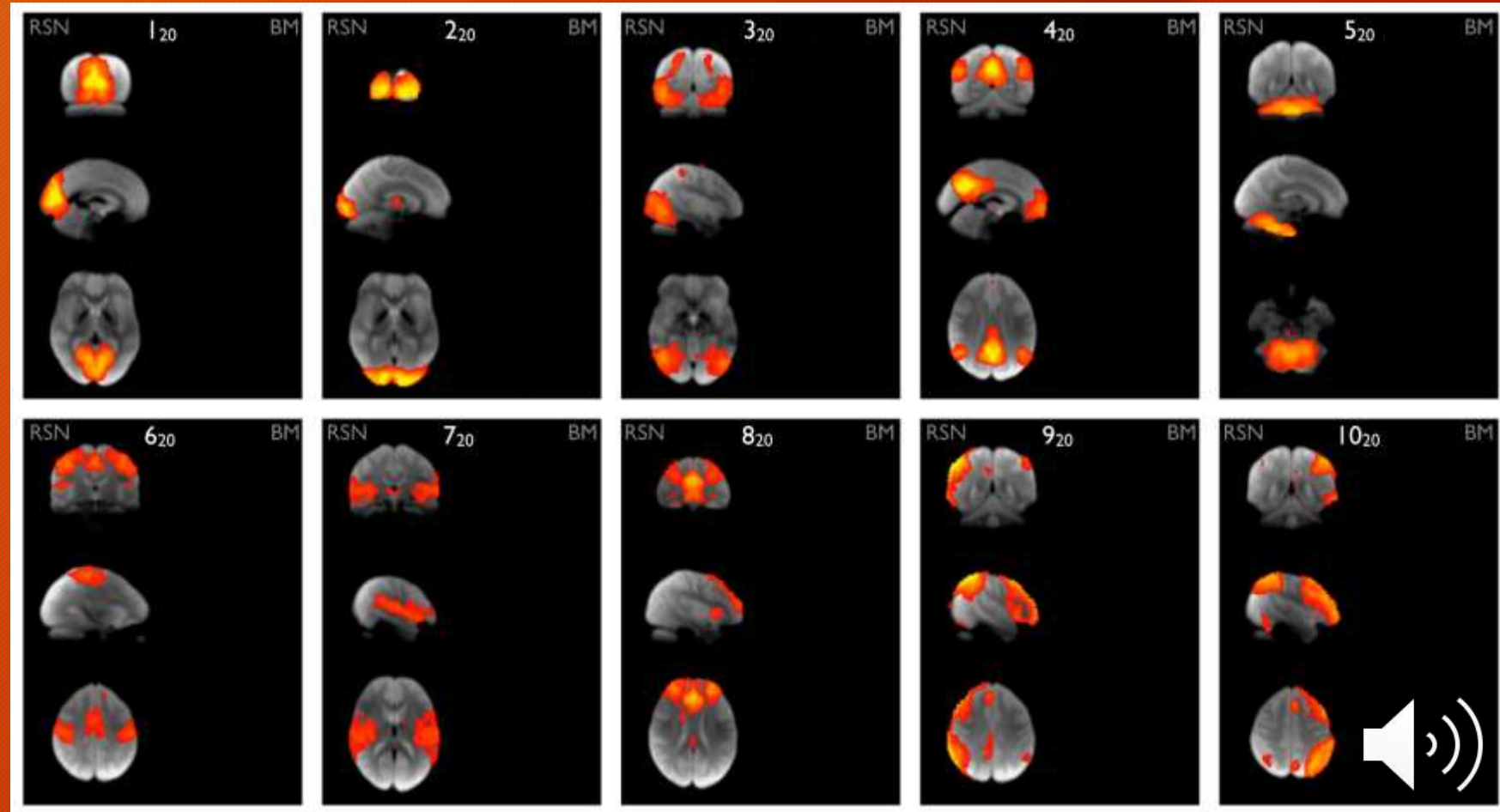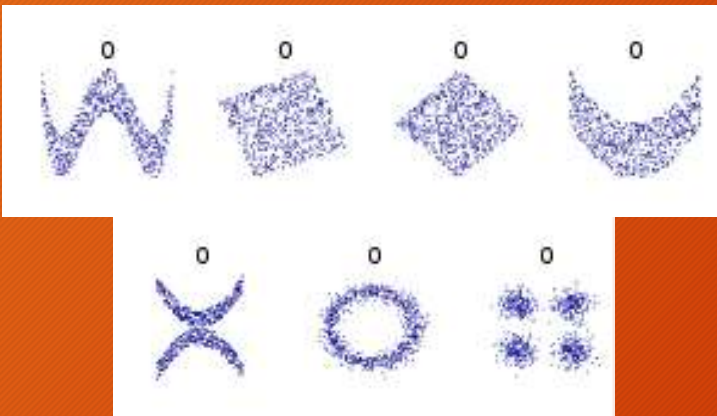- Principal component analysis of genetic variation in Europe

Novembre et al., Nature 2008

Identification of brain's networks using fMRI + Independent Component Analysis during the resting state
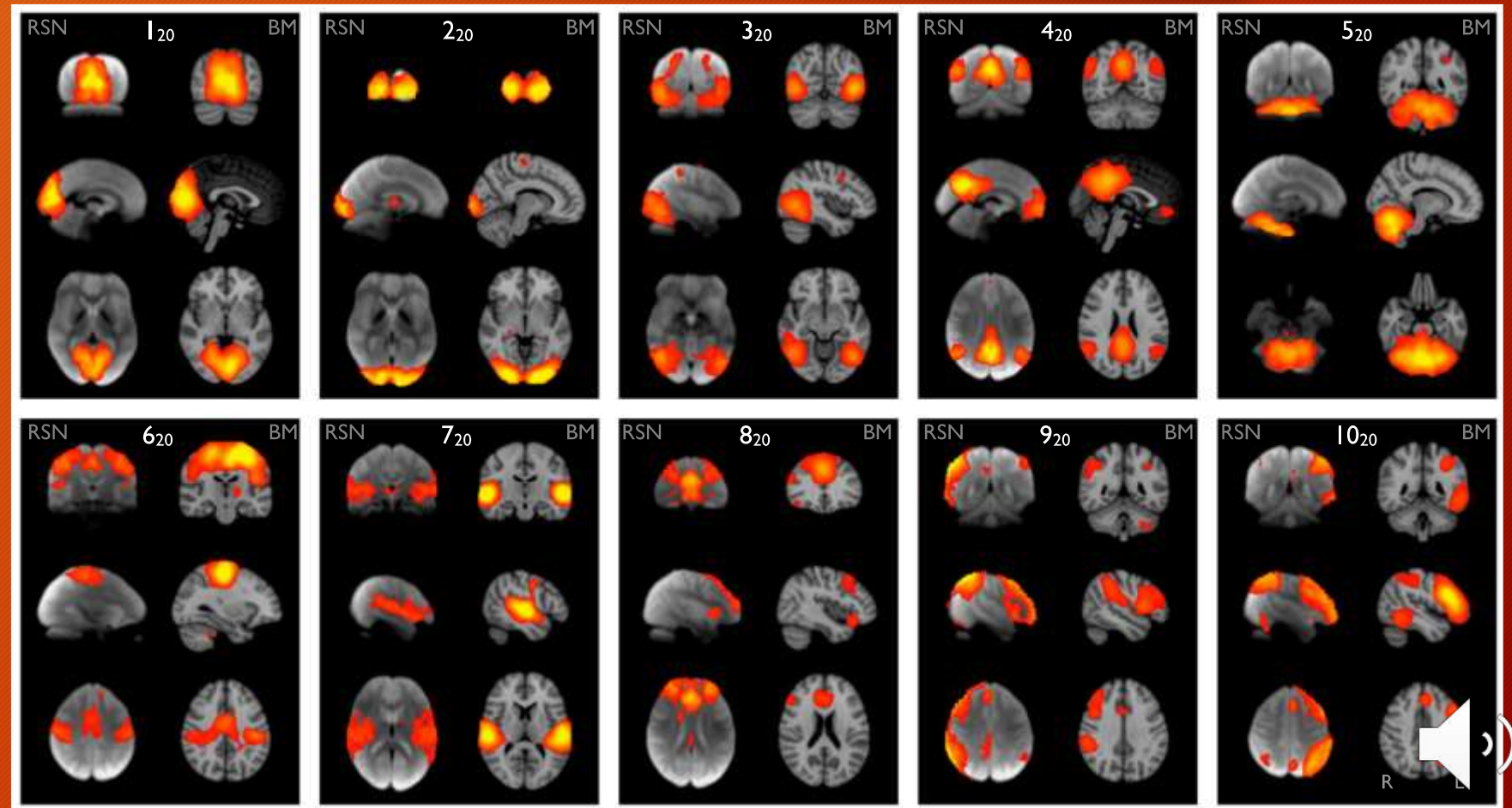
Smith et al., PNAS 2009

Independent Component analysis – extract components that are not just uncorrelated but also independent!

Identification of brain's networks using fMRI + ICA during the resting state & task performance

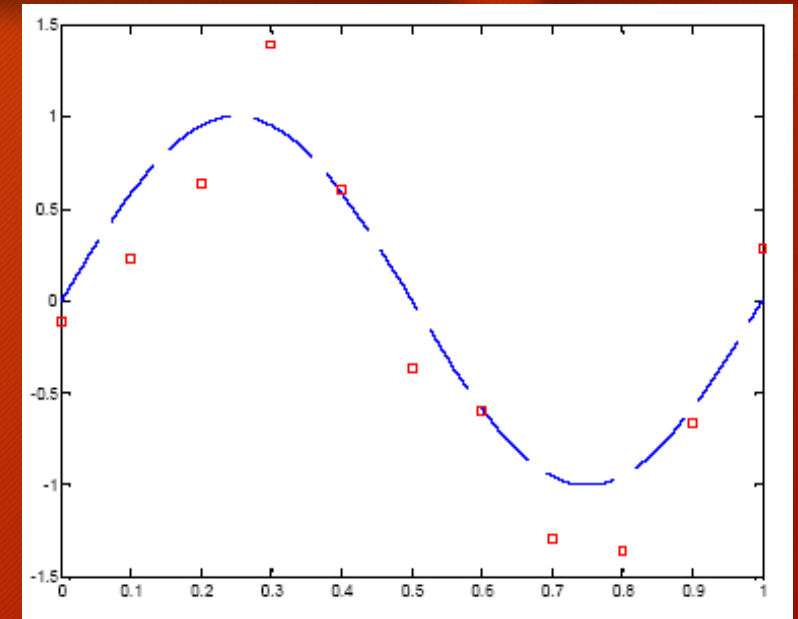Extracted components in the resting state and during a variety of task performances are similar!

Smith et al., PNAS 2009

Potential problem in supervised learning: overfitting

- Imagine a model:

$$p(x) = w_0 + w_1 x + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j.$$

- No. of parameters = M+1

- Goal: minimize error

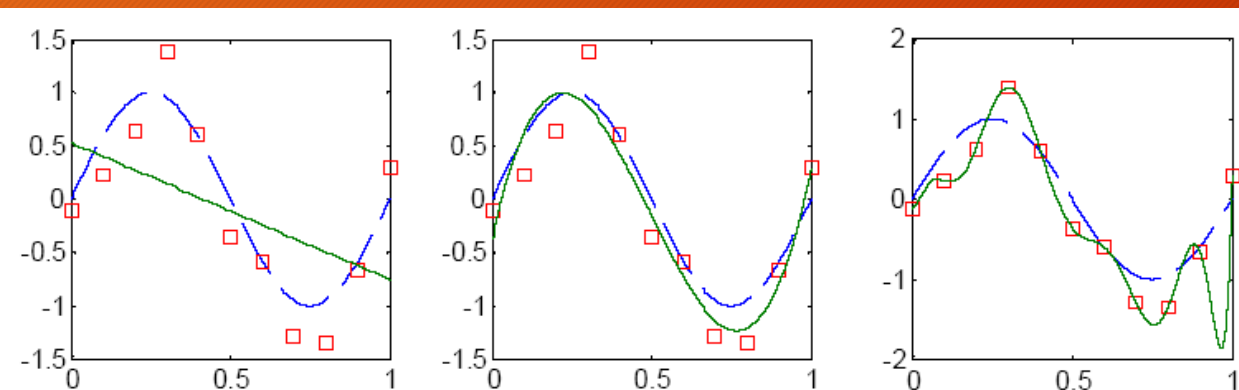$$MSE_{train} = \frac{1}{N} \sum_{i=1}^{N} \left(p(x^{(i)}) - y^{(i)} \right)^2$$

Herbert Jaeger, '*Machine Learning*', Bremen, 2003

M=1, MSE=0.4852
M=3, MSE=0.0703
M=10, MSE = 0!

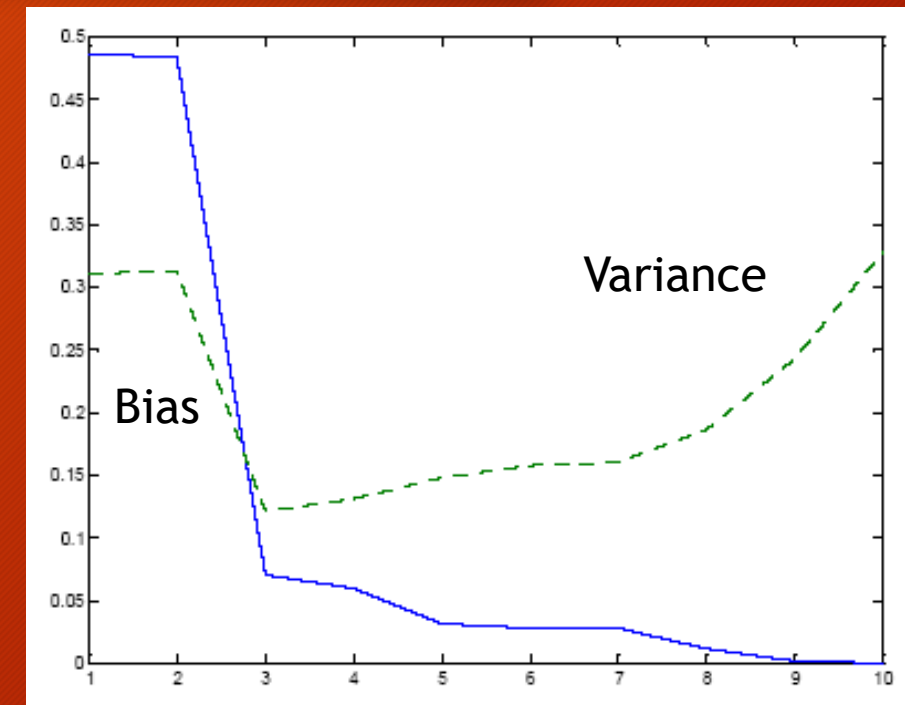**Really?**

Bias-variance dilemma

- Why does *M*=3 look better than *M*=10?
  - It better **generalizes** the function! But...
  - We typically don't know what the function is...
- Use the *testing error*!

$$MSE_{test} = \frac{1}{K} \overset{K}{\underset{j=1}{\mathring{a}}} \left(p\left(x^{(j)}\right) - y^{(j)} \right)^2$$

So *M*=3 is really the best ☺

Why „bias-variance"?

- Bias – fails to account for relevant data structure
- Variance – varies too much depending on training data

Variance

Bias

H.Jaeger, '*Machine Learning*', Bremen, 2003

Dealing with bias-variance dilemma

Finding a right balance (i.e. avoiding overfitting and underfitting) depends on two critical quantities:

- **Model size** (the number of parameters M)

- The **number of samples** N

If M >= N, there will certainly be a substantial overfitting (system of linear equations analogy)

- Normally it's recommended that M would be **much smaller** than N, e.g. 50 or 100 parameters for 1000 samples

How to evaluate over-/underfitting?

- Use **cross-validation**
  - Divide your data into **training** and **validation** sets
  - Use the training set for training the model, and the validation set for choosing the number of parameters
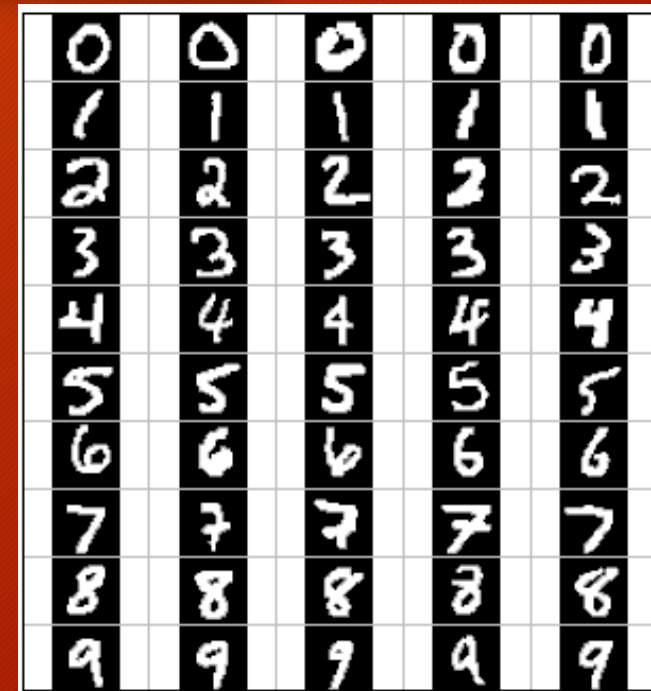 - If have too little data, can use „leave-one-out" cross validation

http://w3.impa.br/~lhf/sib2003/p023/img2.gif
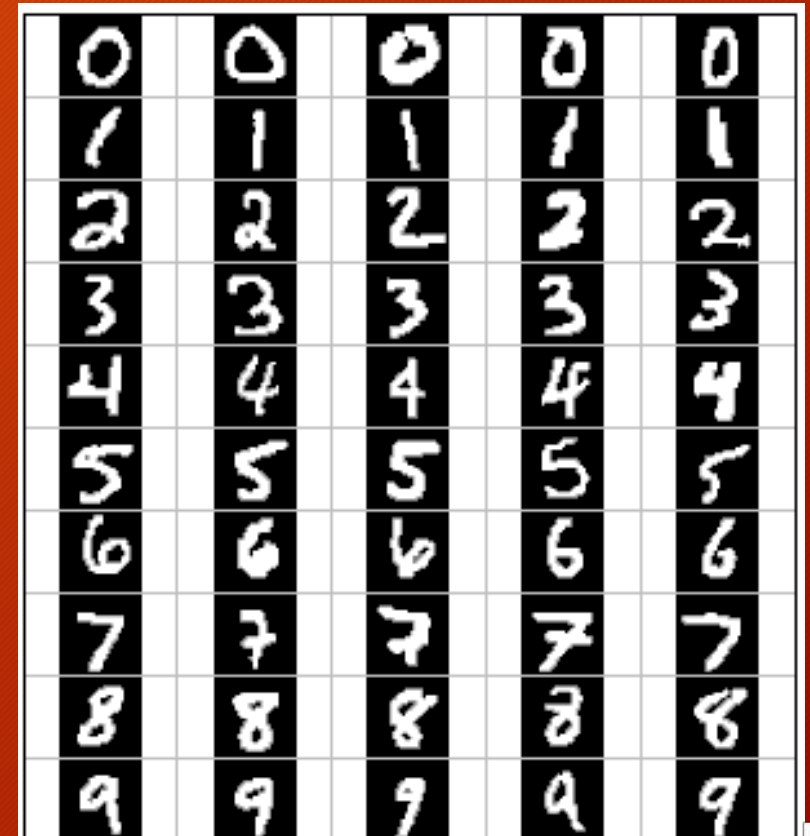
Dimensionality reduction: features

- So our model size is limited by the number of available samples. What's the problem with this?

- Model size is also constrained *(unfortunately from the opposite side)* by the dimensionality of data – usually for each input dimension there is at least 1 parameter
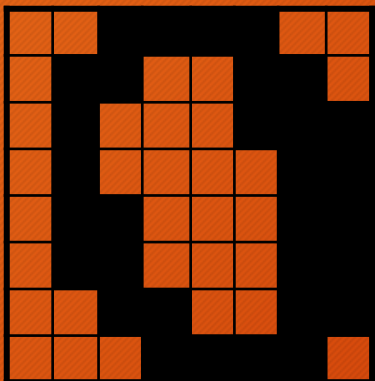
How to resolve this?

- Use not the raw data, but **features**!

- Sometimes this can be done using task-dependent heuristics

http://w3.impa.br/~lhf/ sib2003/p023/img2.gif