UCB - CS189
Introduction to Machine Learning
Fall 2015
Lecture 11: Embedded methods for model and feature selection

Isabelle Guyon

ChaLearn

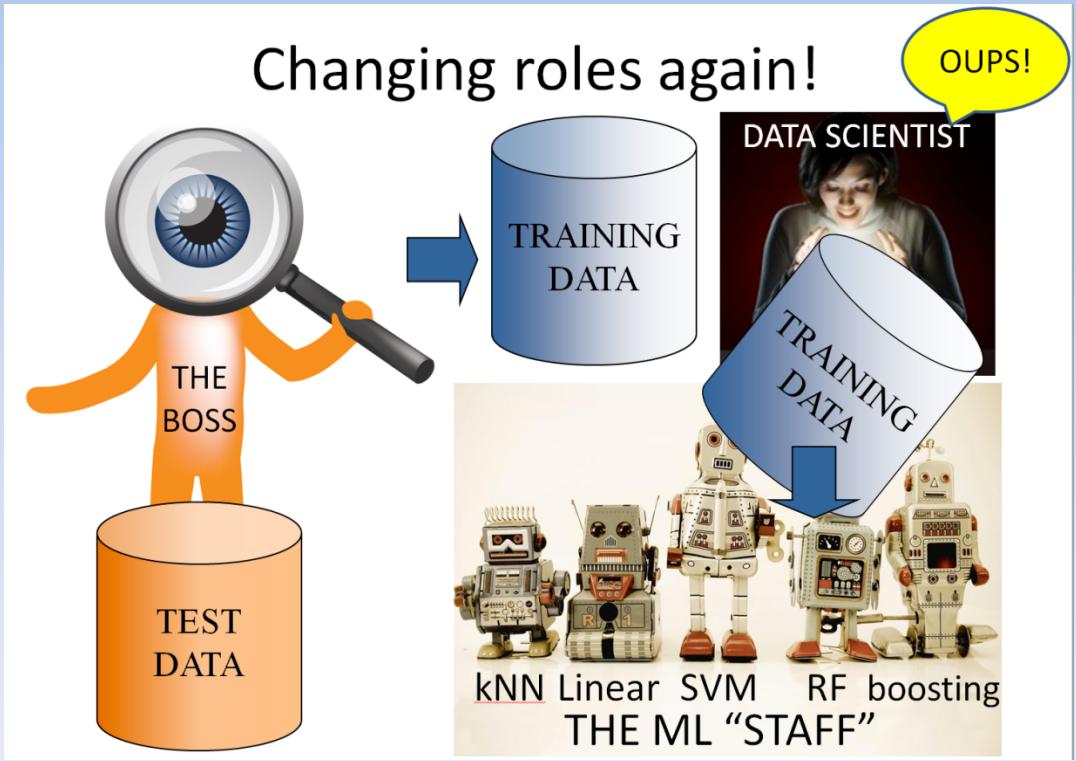
Come to my office hours…

Wed 2:30-4:30 Soda 329

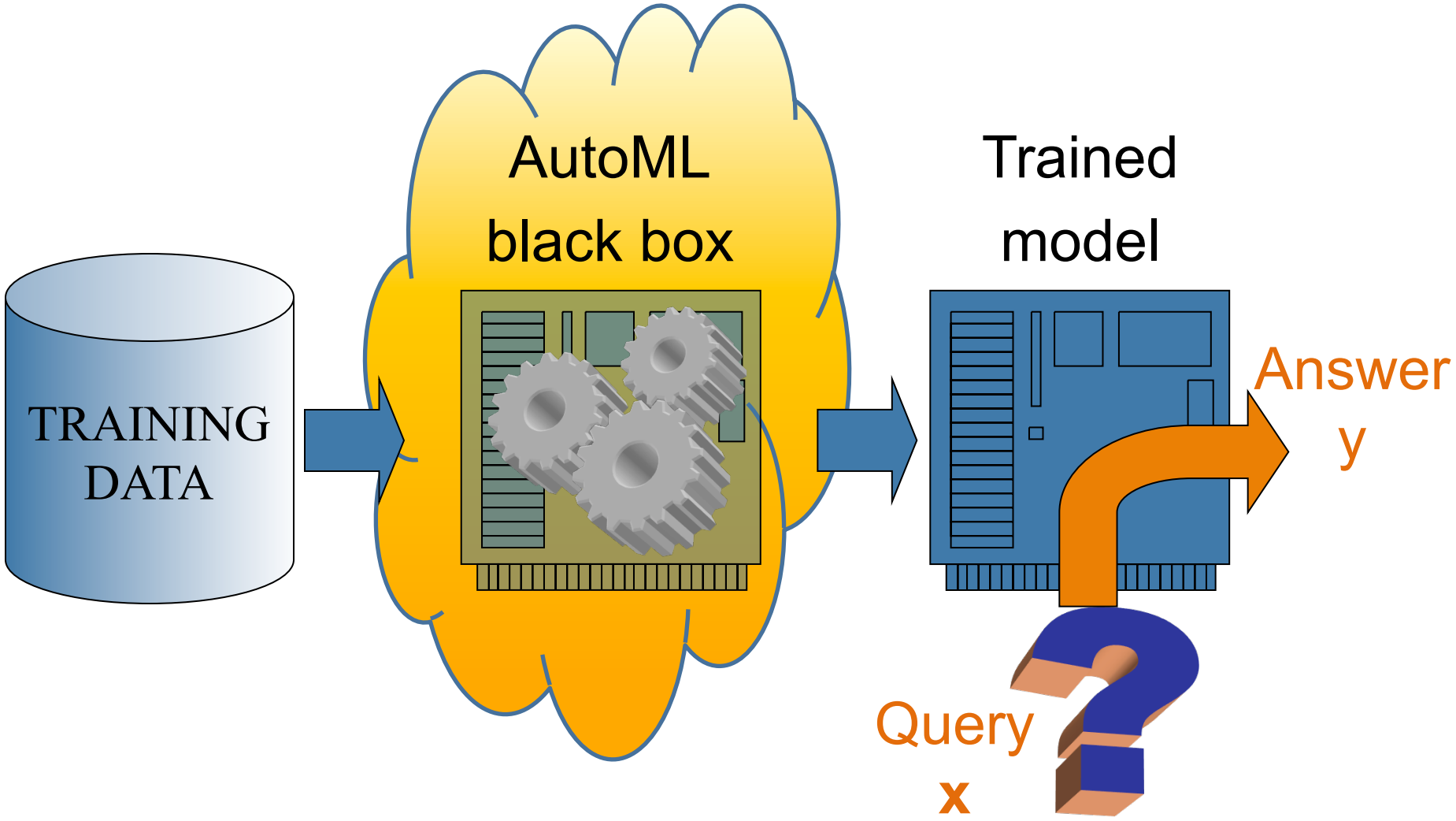
Last time: model search

Come to my office hours…
Wed 2:30-4:30 Soda 329

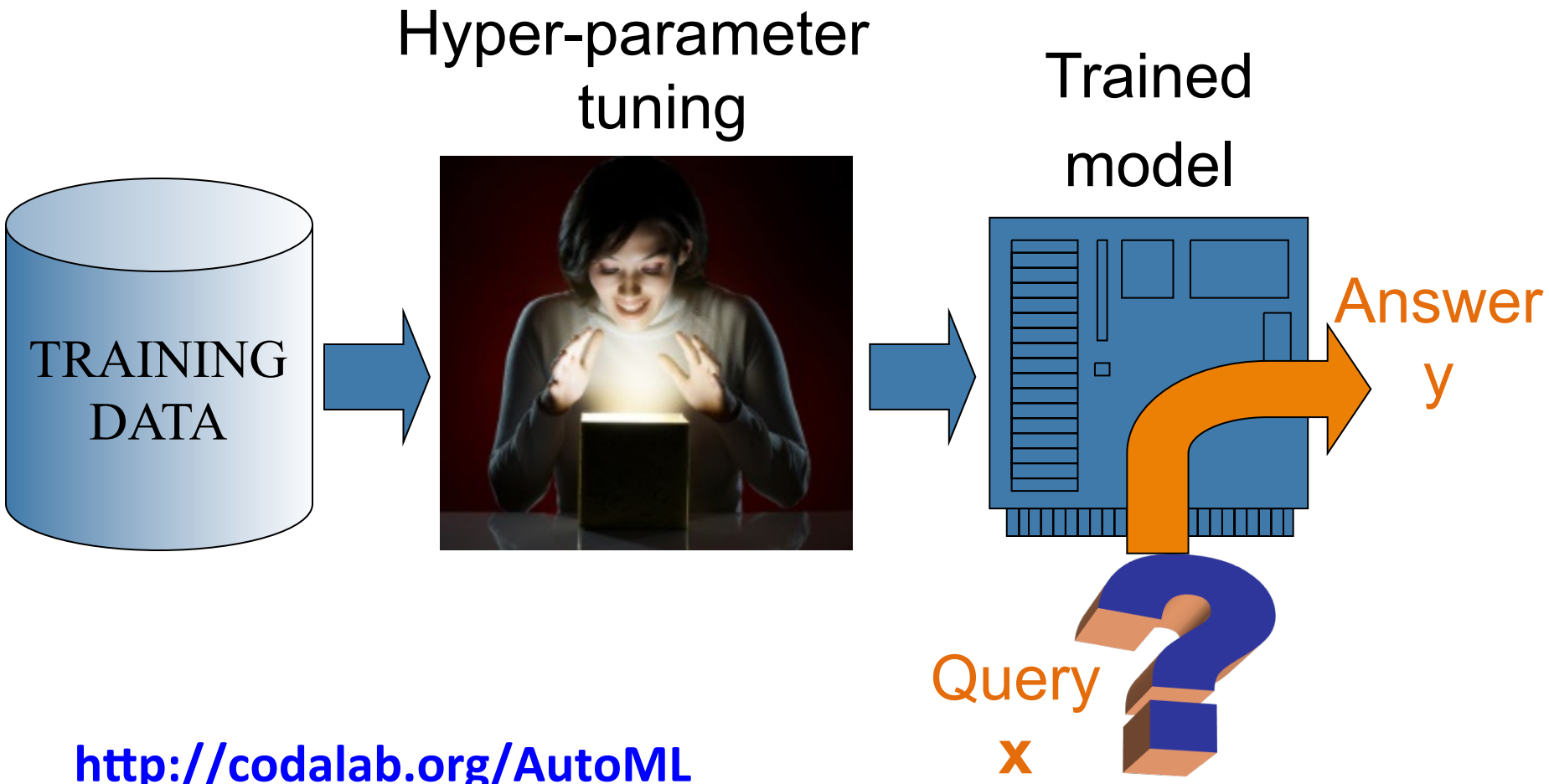
Today



Remember: The DREAM

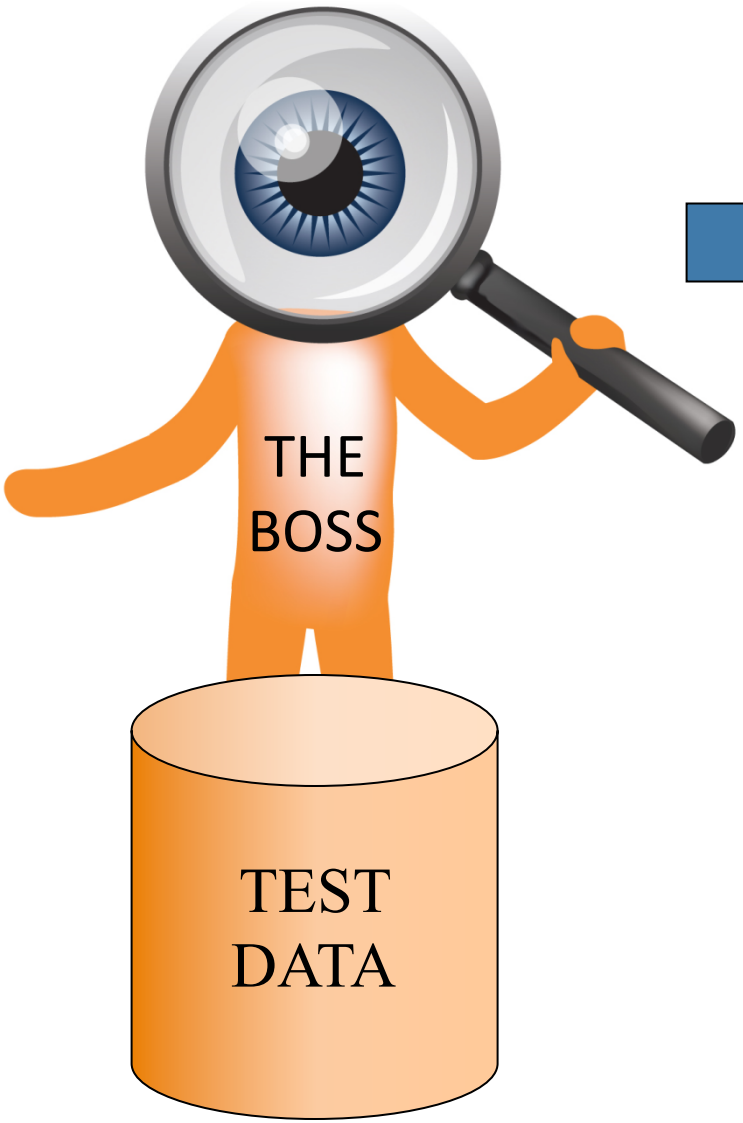


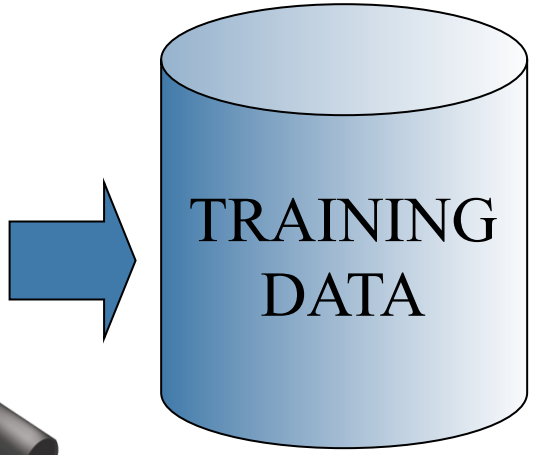
Remember: The REALITY

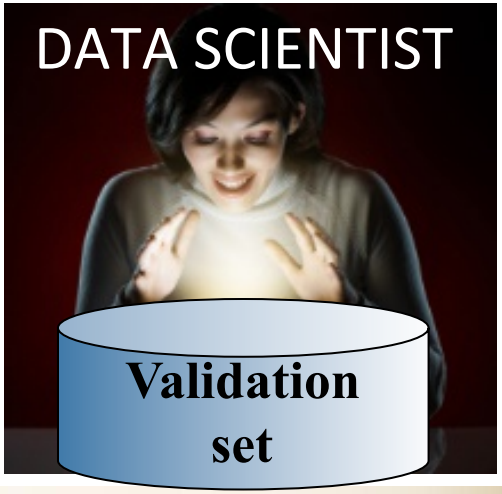


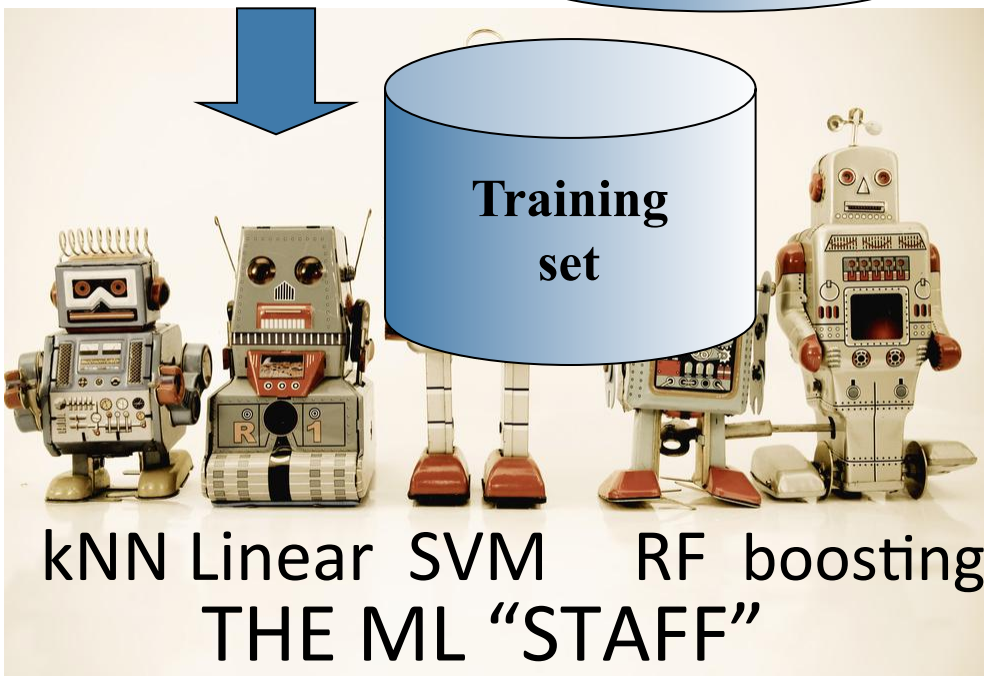
http://codalab.org/AutoML

Last time you were the data scientist

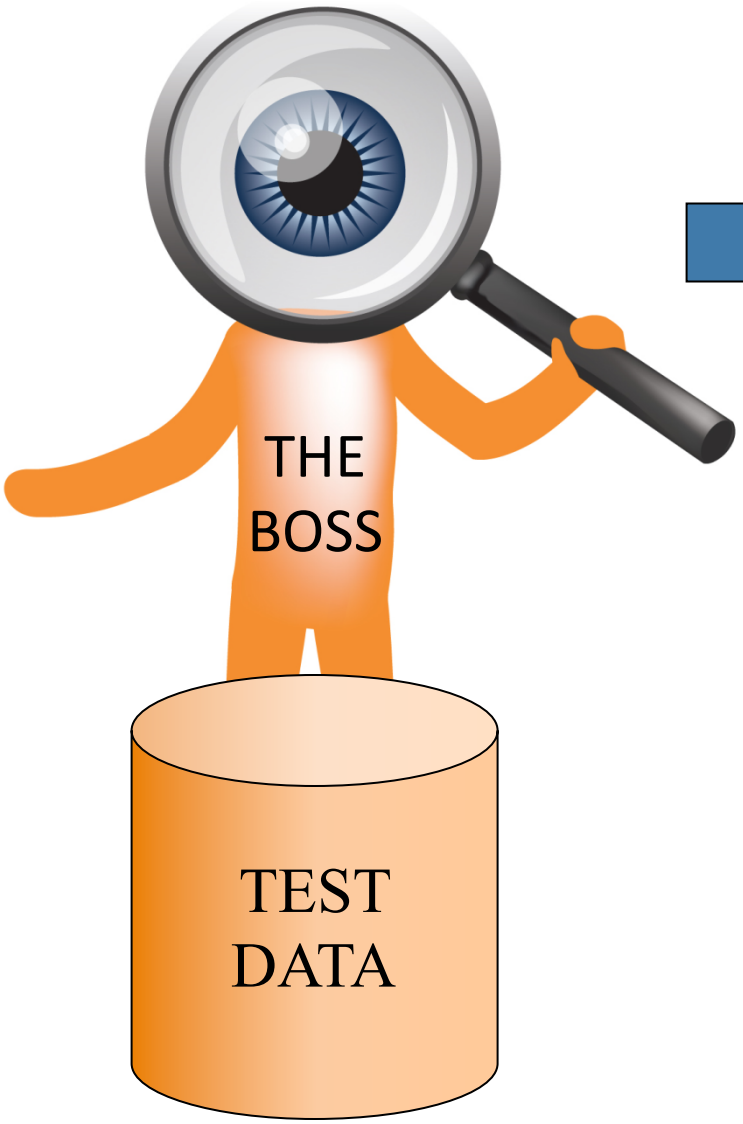


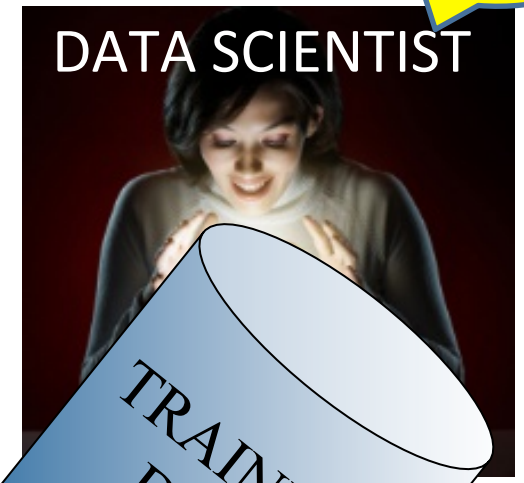






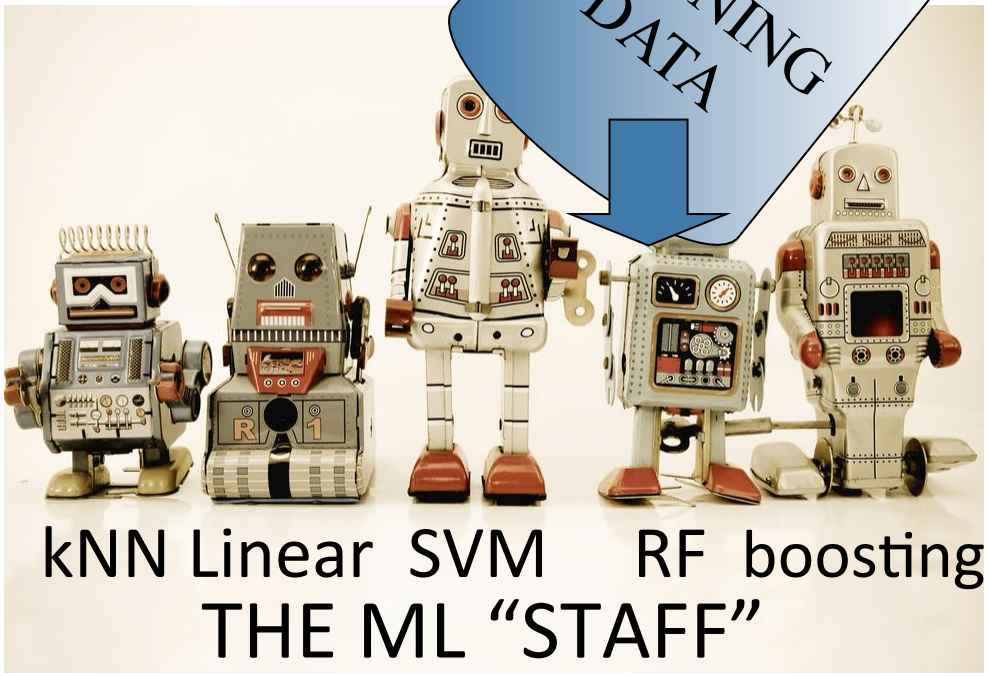
Changing roles again!





OUPS!

TRAINING
DATA



Remember your options

1. **Reduce the number of hyper-parameters.**

2. **Learn how to do “fancy” search.**

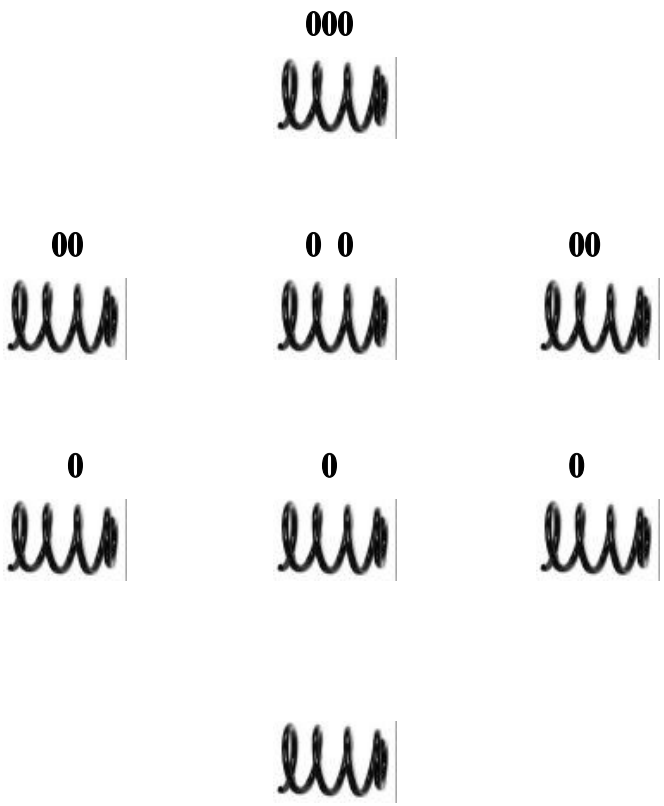
Remember your options

1. **Reduce the number of hyper-parameters.**

2. **Learn how to do “fancy” search.**

3. **Push more hyper-parameters to the “lower level”.**

Embedded methods

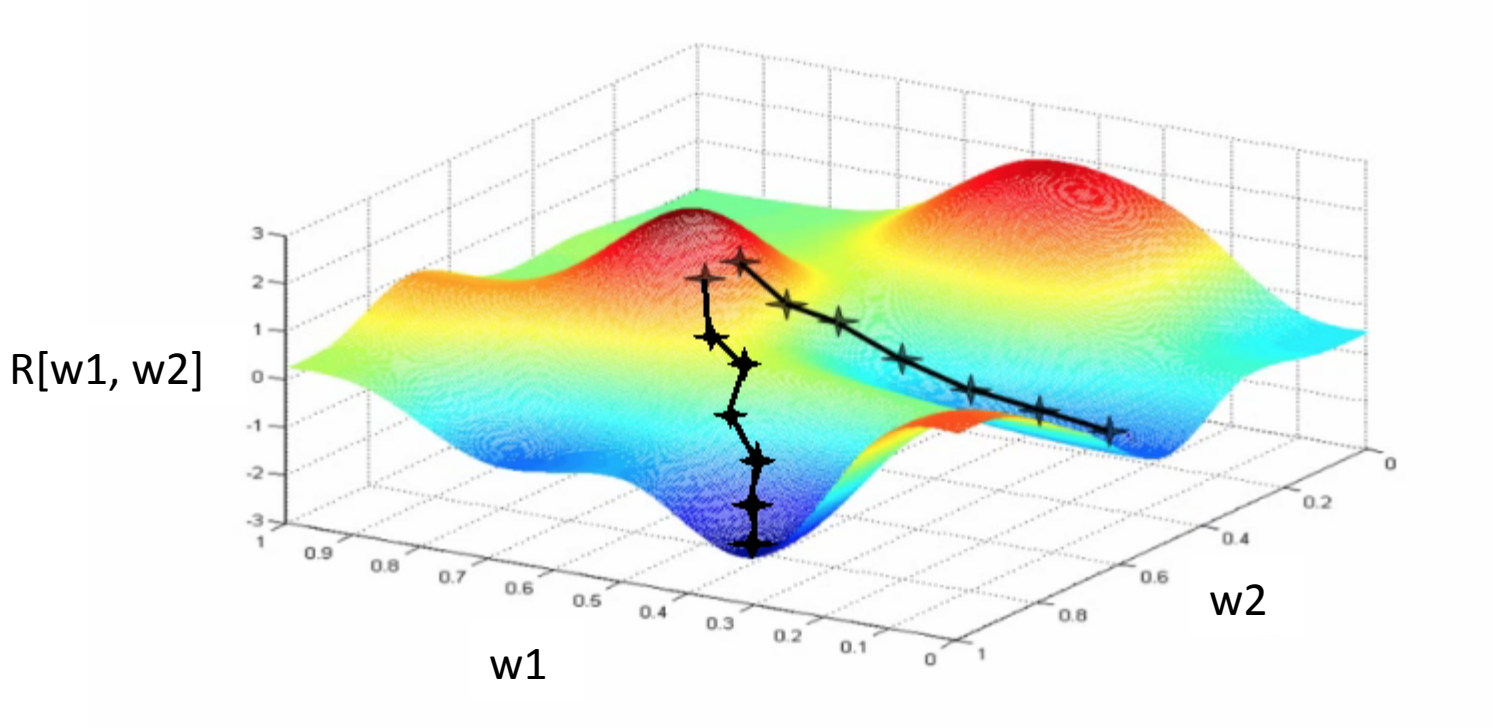


Why do we need HP?

- Could we not just optimize the parameters and hyper-parameters together with the training error?
- No, because:
  - **Computational complexity**: we loose convexity in parameter space.
  - **Statistical complexity**: we get infinite (or zero) capacity.

Complexity: computational

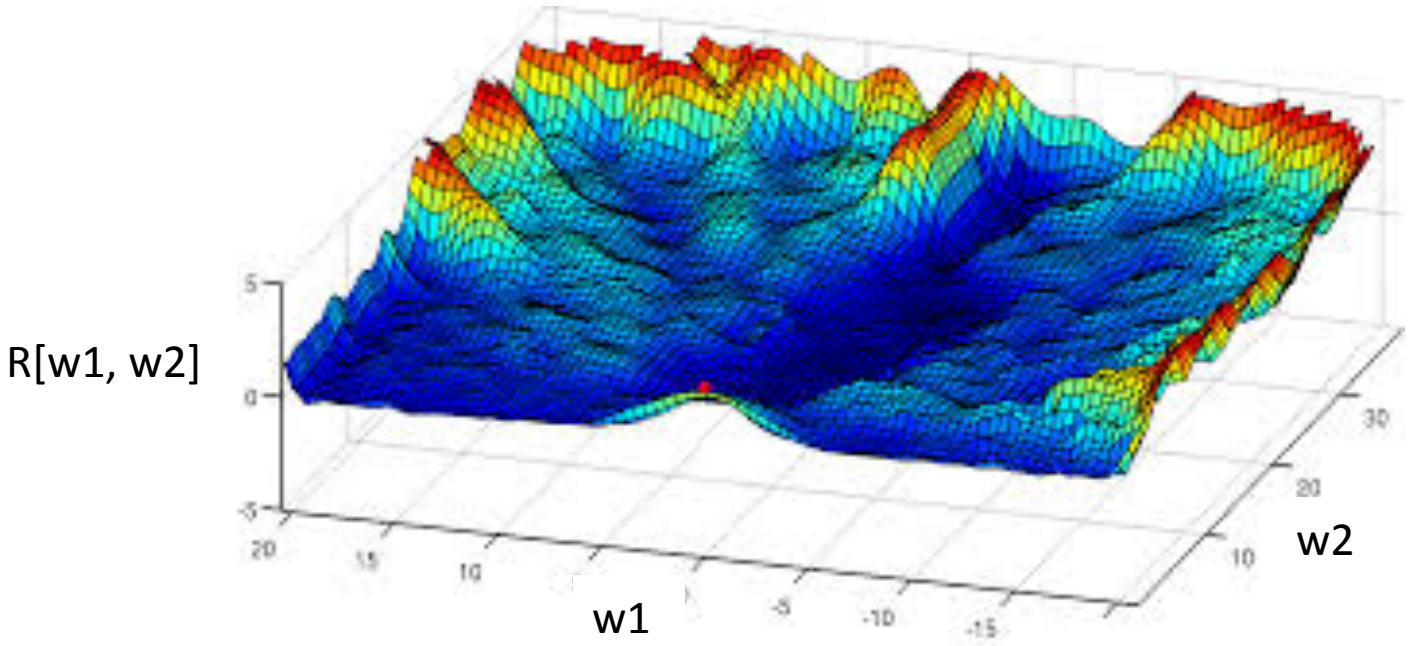
Gradient descent falls into local minima…



η is the learning rate of gradient step.

Complexity: computational

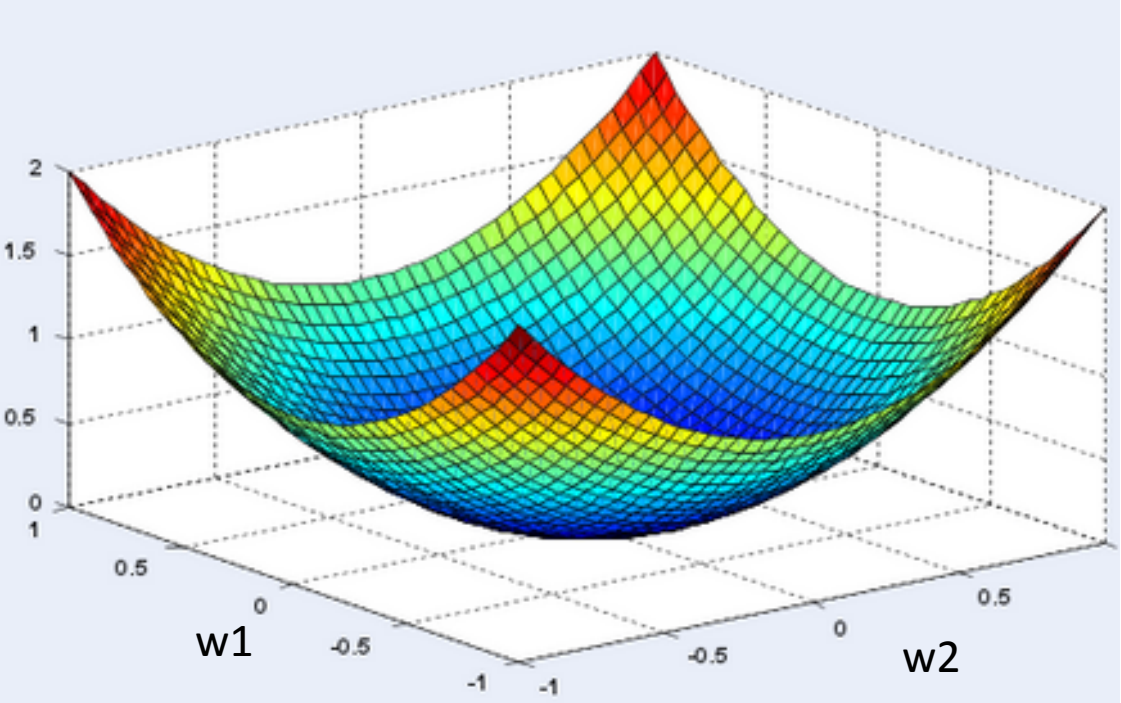
… finding the global optimum can be hard …



Complexity: computational

… except if the risk functional is convex!

R[w1, w2]



Convexity: Hessian matrix is positive semidefinite.

Complexity: computational

Convexity

- $R = RSS = \Sigma_k (\mathbf{x}^k \mathbf{w}^T - y^k)^2$
  $= \| X\mathbf{w}^T - \mathbf{y} \|^2$
  $= \mathbf{w}X^TX\mathbf{w}^T - 2\mathbf{w}X^T\mathbf{y} + \mathbf{y}^T\mathbf{y}$
- Gradient:
  $\nabla_{\mathbf{w}} R = 2 (X^TX\mathbf{w}^T - X^T\mathbf{y})$
- Hessian:
  $H(R) = 2 X^TX$

X^TX is an outer product, therefore it is PSD.

Complexity: computational

Kernel trick

- Hessian:

$$H(R) = 2\,X^T X$$

- In Φ-space:

$$H(R) = 2\,\Phi^T \Phi$$

$$= 2\,K$$

This is why it is useful that the kernel matrix K be PSD.

Complexity: computational

SVM in α space

- $f(\mathbf{x}) = \sum_k \alpha_k\, k(\mathbf{x}^k, \mathbf{x})$ $\quad K = [k(\mathbf{x}^k, \mathbf{x}^h)]$

1) Hinge loss version:

$\min_\alpha \alpha^T \mathbf{y} - (1/2)\, \alpha^T K \alpha$

$0 \le \alpha_k\, y_k \le 1/\lambda$ box constraint

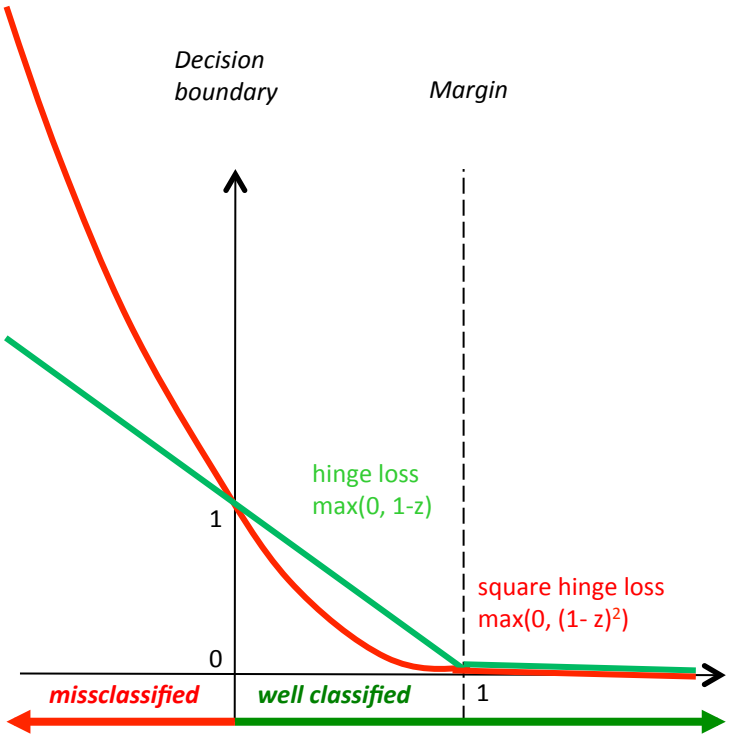
$\alpha^T 1 = 0$ bias constraint

2) Square hinge loss version

$\min_\alpha \alpha^T \mathbf{y} - (1/2)\, \alpha^T (K + \lambda I)\alpha$

$\alpha_k\, y_k \ge 0$ ridge

$\alpha^T 1 = 0$ bias constraint



http://homepages.rpi.edu/~bennek/class/mmld/papers/svn.pdf
http://cs229.stanford.edu/notes/cs229-notes3.pdf

Complexity: computational

Loosing convexity

$$R[\alpha, \theta] = \alpha^T \mathbf{y} - (1/2)\, \alpha^T K(\theta)\, \alpha$$

$$K(\theta) = [\, k(\mathbf{x}^k, \mathbf{x}^h; \theta)\,]_{k=1:N,\, h=1:N}$$

$$k(\mathbf{x}, \mathbf{x}'; \theta) = (a + \mathbf{x} \cdot \mathbf{x}')^b \exp(- c\, \|\mathbf{x} - \mathbf{x}'\|^d)$$

$$\theta = [a, b, c, d]$$

Complexity: Statistical

Avoid overfitting…

$$f(\mathbf{x}) = \left(\sum_k \alpha_k \left[\Phi(\mathbf{x}^k) \right) \bullet \Phi(\mathbf{x}) \right]$$

$\mathbf{w}$ $\quad k(\mathbf{x}^k, \mathbf{x}; \theta)$

Perceptrons

- α_k fitted to R_{train}
- $k(\mathbf{x}^k, \mathbf{x}) = \mathbf{x}^k \bullet \mathbf{x}$
- No θ
- $f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x}$

Hebb's method

- $\alpha_{k=} y_k$
- $k(\mathbf{x}^k, \mathbf{x}) = \mathbf{x}^k \bullet \mathbf{x}$
- No θ
- $f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x}$

Parzen windows

- $\alpha_{k=} y_k$
- Any $k(\mathbf{x}^k, \mathbf{x}; \theta)$
- θ fitted to R_{CV}
- $f(\mathbf{x})=\Sigma_k \alpha_k k(\mathbf{x}^k, \mathbf{x}; \theta)$

Kernel methods

- α_k fitted to R_{train}
- Any $k(\mathbf{x}^k, \mathbf{x}; \theta)$
- θ fitted to R_{CV}
- $f(\mathbf{x})=\Sigma_k \alpha_k k(\mathbf{x}^k, \mathbf{x}; \theta)$

Complexity: Statistical

Nested subsets

Capacity /complexity

Perceptrons

- α_k fitted to R_{train}
- $k(\mathbf{x}^k, \mathbf{x}) = \mathbf{x}^k \bullet \mathbf{x}$
- No θ
- $f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x}$

Hebb's method

- $\alpha_{k} = y_k$
- $k(\mathbf{x}^k, \mathbf{x}) = \mathbf{x}^k \bullet \mathbf{x}$
- No θ
- $f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x}$

Kernel methods

- α_k fitted to R_{train}
- Any $k(\mathbf{x}^k, \mathbf{x}; \theta)$
- θ fitted to R_{CV}
- $f(\mathbf{x})=\Sigma_k \alpha_k k(\mathbf{x}^k, \mathbf{x} ; \theta)$

Complexity: Statistical

Nested subsets

Capacity /complexity

Hebb's method

- $\alpha_{k=} y_k$
- $k(\mathbf{x}^k, \mathbf{x}) = \mathbf{x}^k \bullet \mathbf{x}$
- No θ
- $f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x}$

Parzen windows

- $\alpha_{k=} y_k$
- Any $k(\mathbf{x}^k, \mathbf{x}; \theta)$
- θ fitted to R_{CV}
- $f(\mathbf{x})=\Sigma_k \alpha_k k(\mathbf{x}^k, \mathbf{x} ; \theta)$

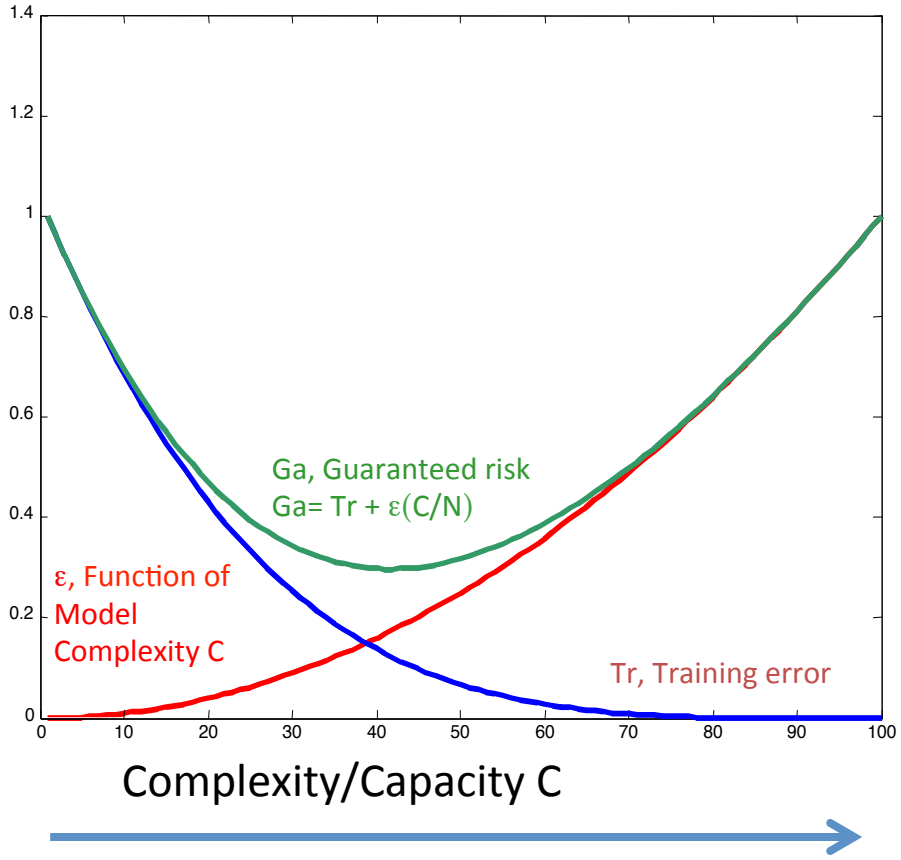
Kernel methods

- α_k fitted to R_{train}
- Any $k(\mathbf{x}^k, \mathbf{x}; \theta)$
- θ fitted to R_{CV}
- $f(\mathbf{x})=\Sigma_k \alpha_k k(\mathbf{x}^k, \mathbf{x} ; \theta)$

Complexity: Statistical

Guaranteed risk

to avoid overfitting



With *high* probability $(1-\delta)$, $0<\delta<<1$

$$R[f] \leq \underbrace{R_{train}[f] + \varepsilon(\delta, C/N)}_{R_{gua}[f]}$$

Complexity: Statistical

Shrinkage: $\min_w R_{train}[w] + \lambda \|w\|^2$

Capacity = $d_{effective}$

Perceptrons

$f(x) = w \bullet x$

Complexity: Statistical

Shrinkage: $\min_w R_{train}[w] + \lambda \|w\|^2$

Capacity = $d_{effective}$

Kernel methods

$f(\mathbf{x})=\Sigma_k \alpha_k k(\mathbf{x}^k, \mathbf{x}; \theta)$

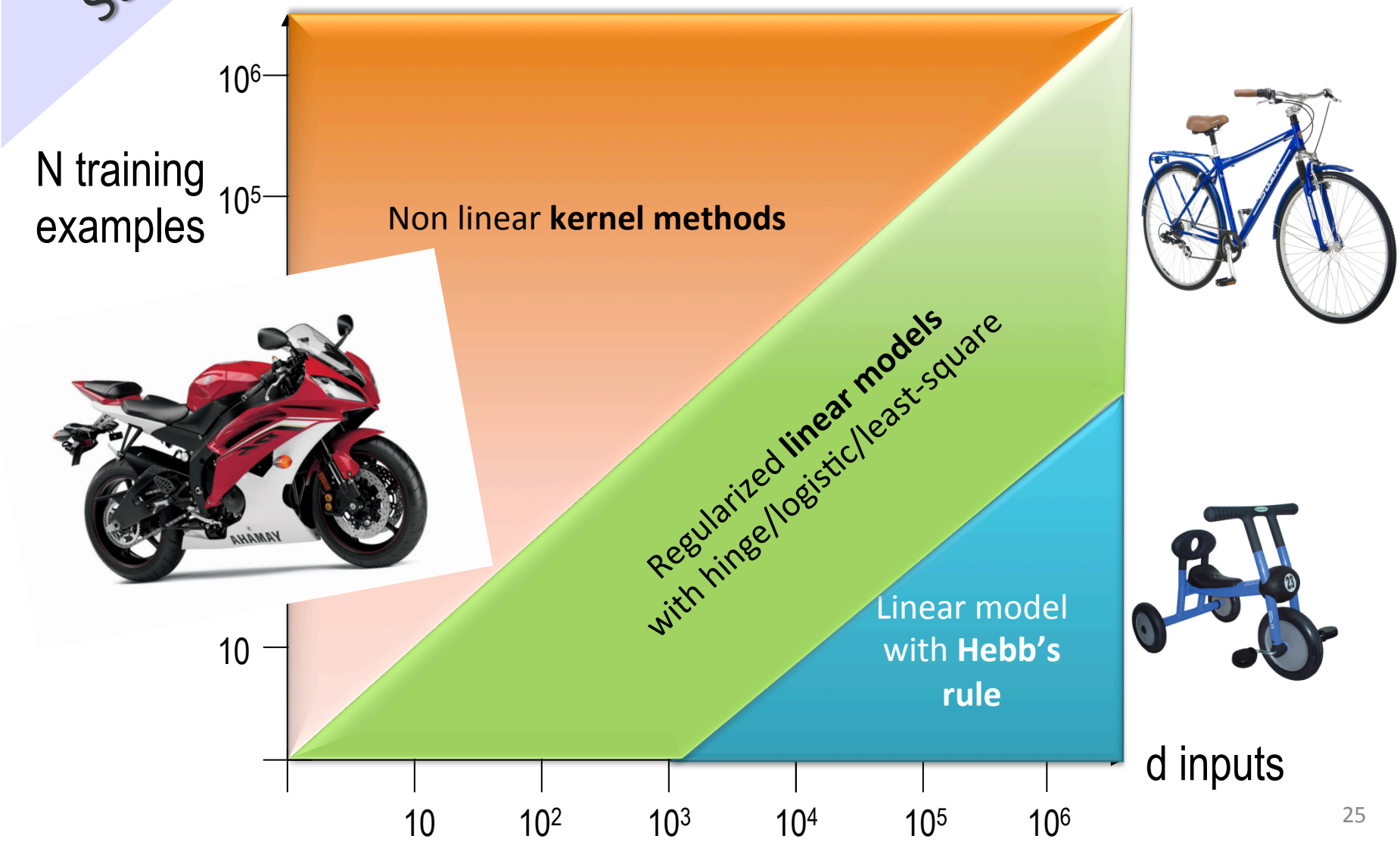
$\|\mathbf{w}\|^2 = \mathbf{w}\mathbf{w}^T$

$= \alpha^T \Phi \Phi^T \alpha$

$= \alpha^T \mathbf{K} \alpha$

Complexity: Statistical

Statistical domains



Complexity: Statistical

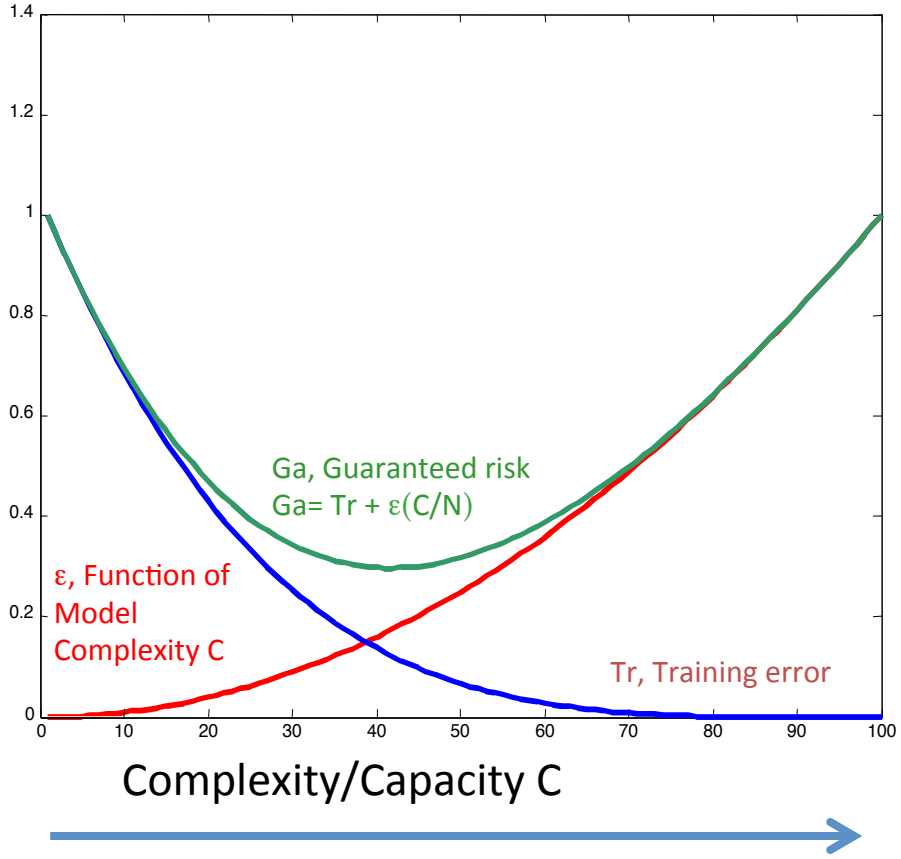
What you want:

Keep C/N (or d_{eff}/N) under control

Complexity: Statistical

Guaranteed risk

to avoid overfitting

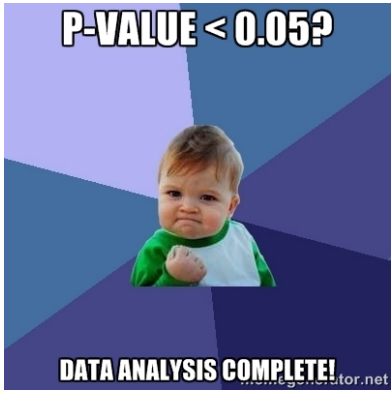


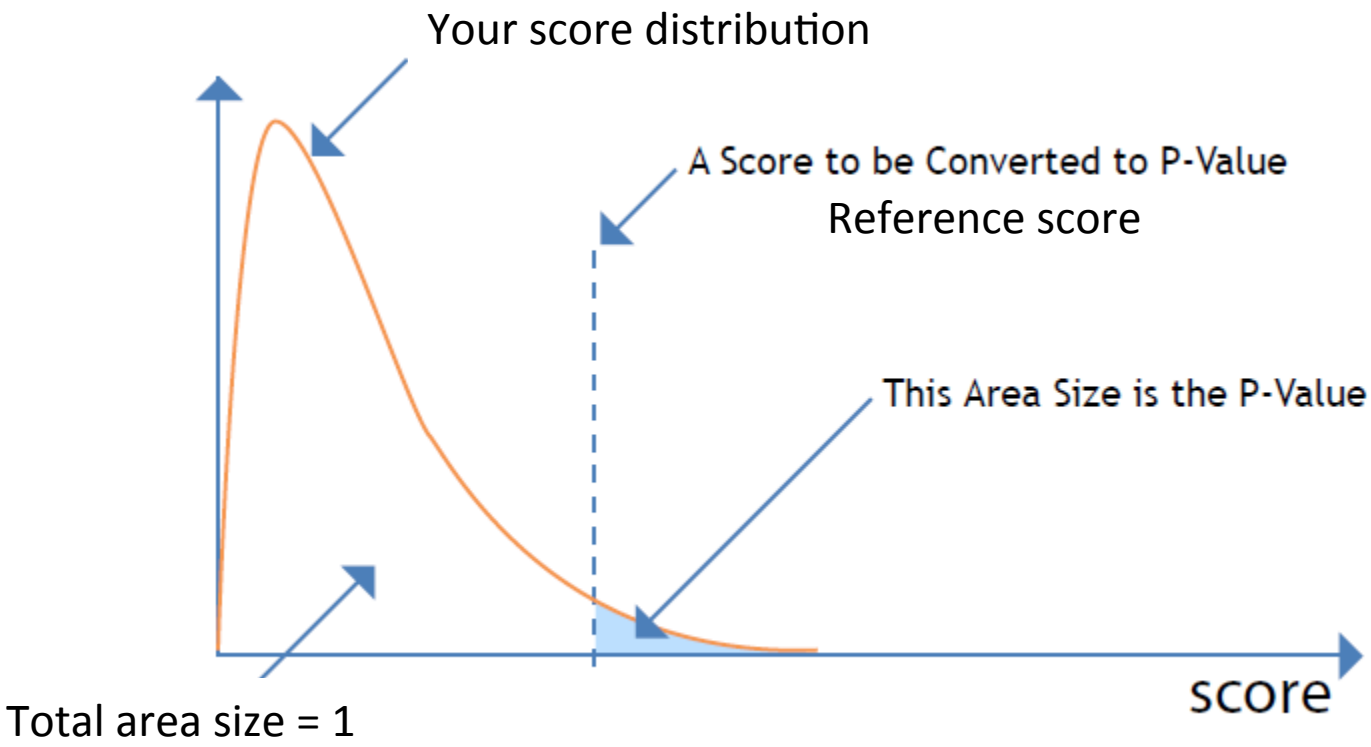
With *high* probability $(1-\delta)$, $0<\delta<<1$

$$R[f] \le \underbrace{R_{train}[f] + \varepsilon(\delta, C/N)}_{R_{gua}[f]}$$

Complexity: Statistical

The intuition: remember pvalues

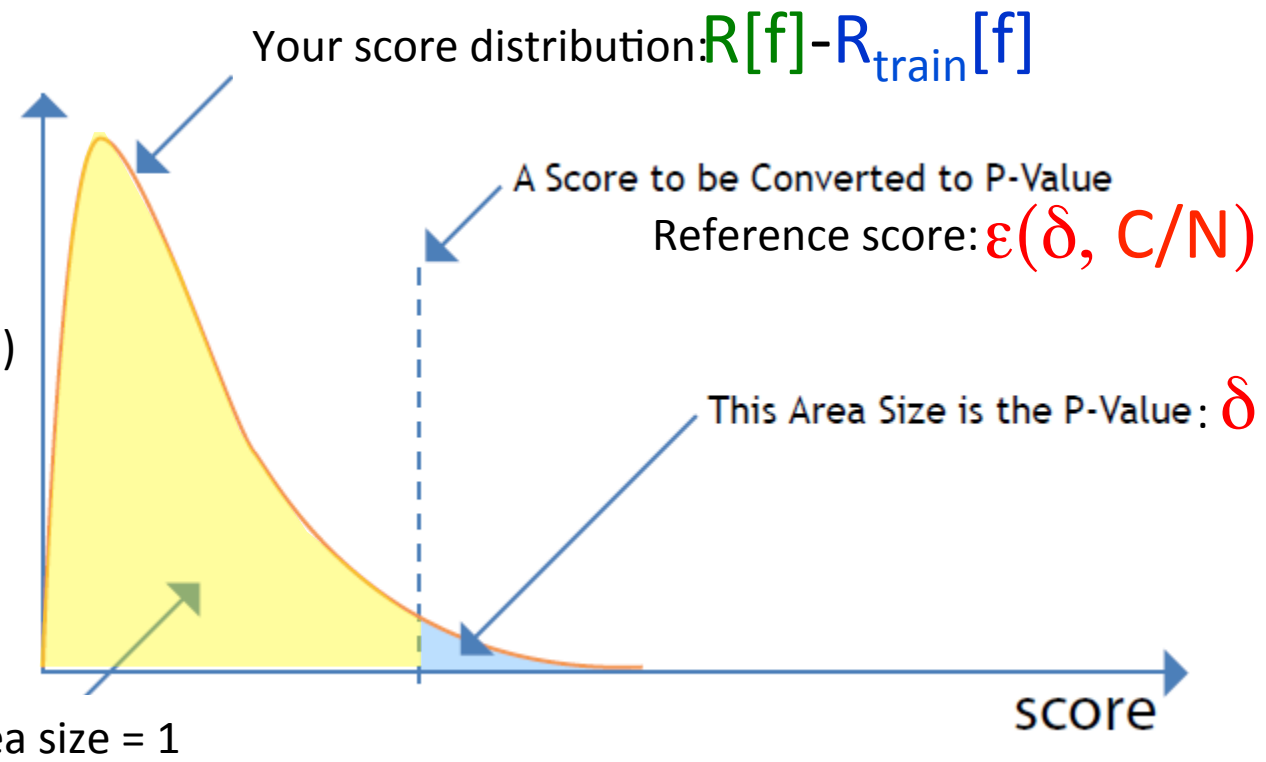




Pvalue = fraction of score values above the reference score.
SMALL pvalues are good: they shed doubt on the "null hypothesis".

Complexity: Statistical

Guaranteed risk and pvalue



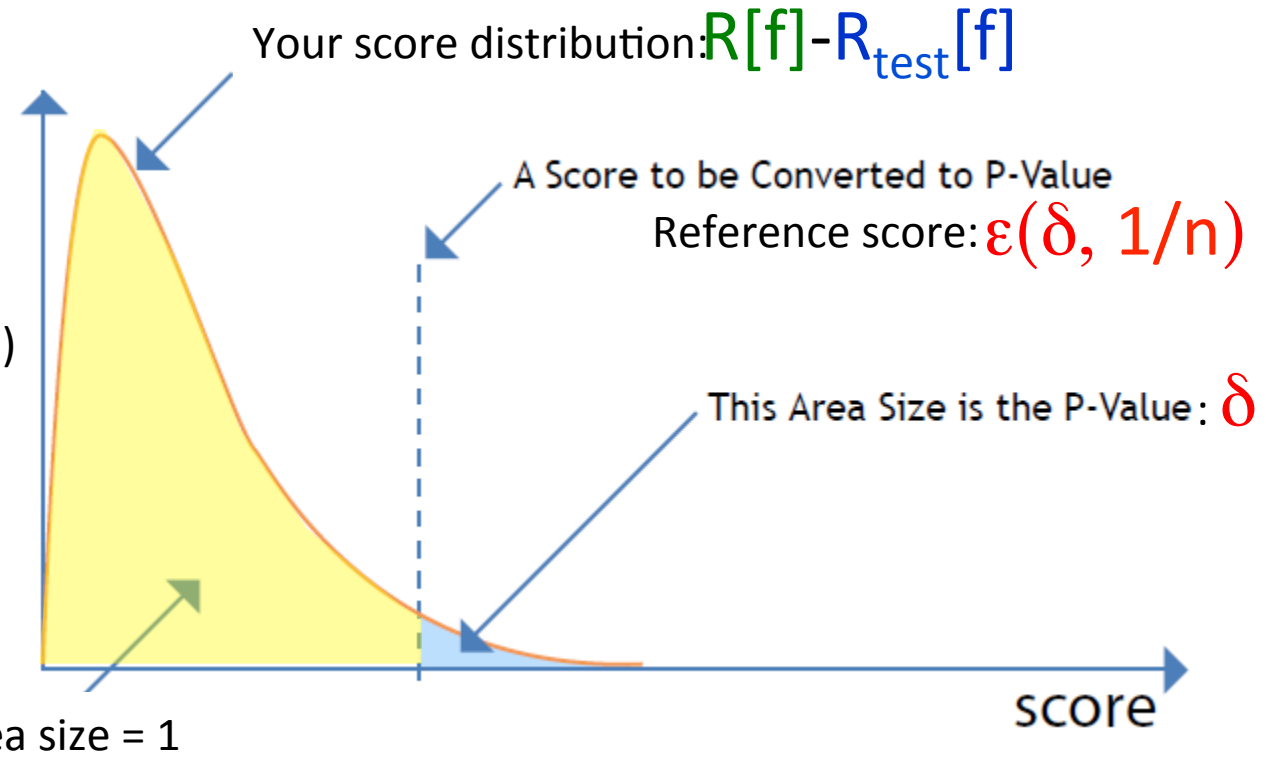
With *high* probability $(1-\delta)$
$0<\delta<<1$

$R[f] \le \underbrace{R_{train}[f] + \varepsilon(\delta, C/N)}_{R_{gua}[f]}$

Pvalue = fraction of score values above the reference score.
SMALL pvalues are good: they shed doubt on the "null hypothesis".

Complexity: Statistical

Test set bounds



With *high* probability (1-δ)
$0<\delta<<1$

$R[f] \le R_{test}[f] + \varepsilon(\delta, 1/n)$

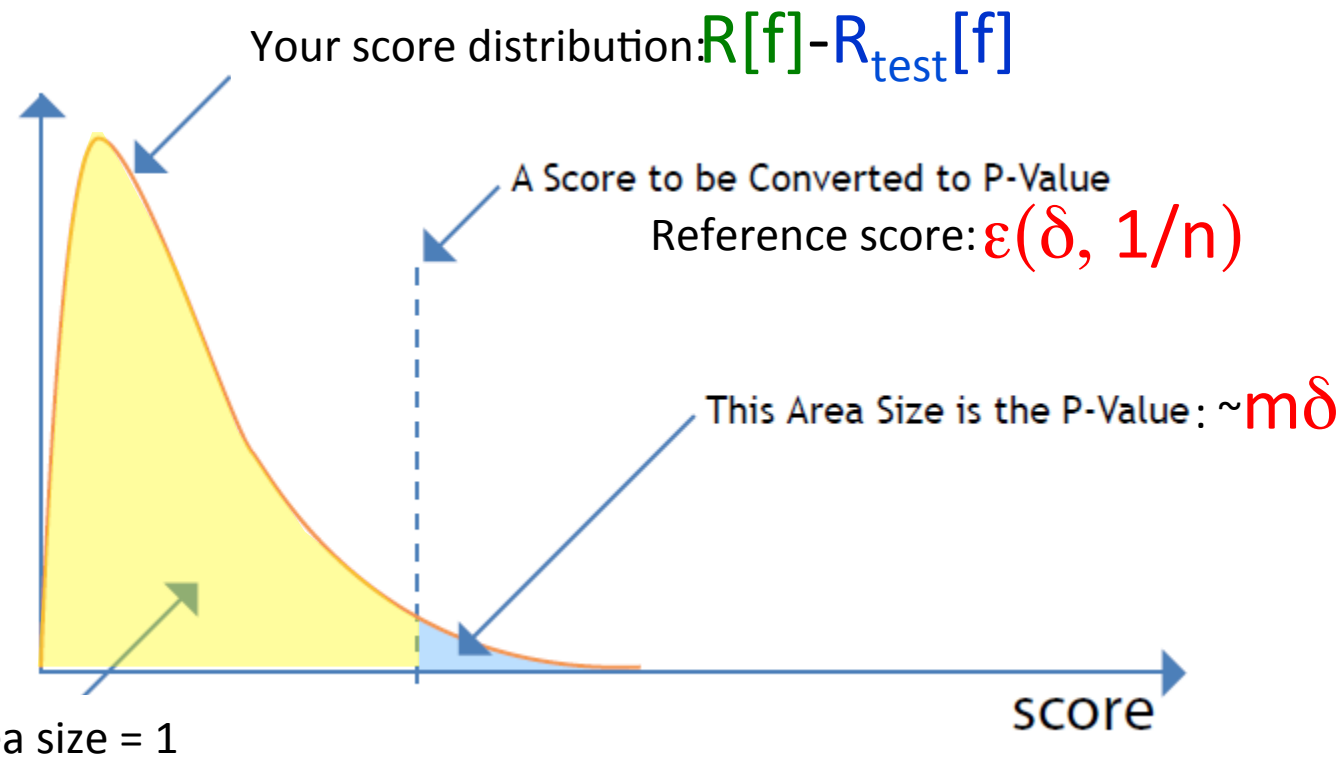
For instance, $R_{test} = E$
$\varepsilon(\delta, 1/n) \sim \sqrt{E(1-E)/n}$

Pvalue = fraction of score values above the reference score.
SMALL pvalues are good: they shed doubt on the "null hypothesis".

Complexity: Statistical

Multiple testing

Bonferroni's correction: pvalue * m



The area is now $(1-\delta)^m$
$0<\delta<<1$

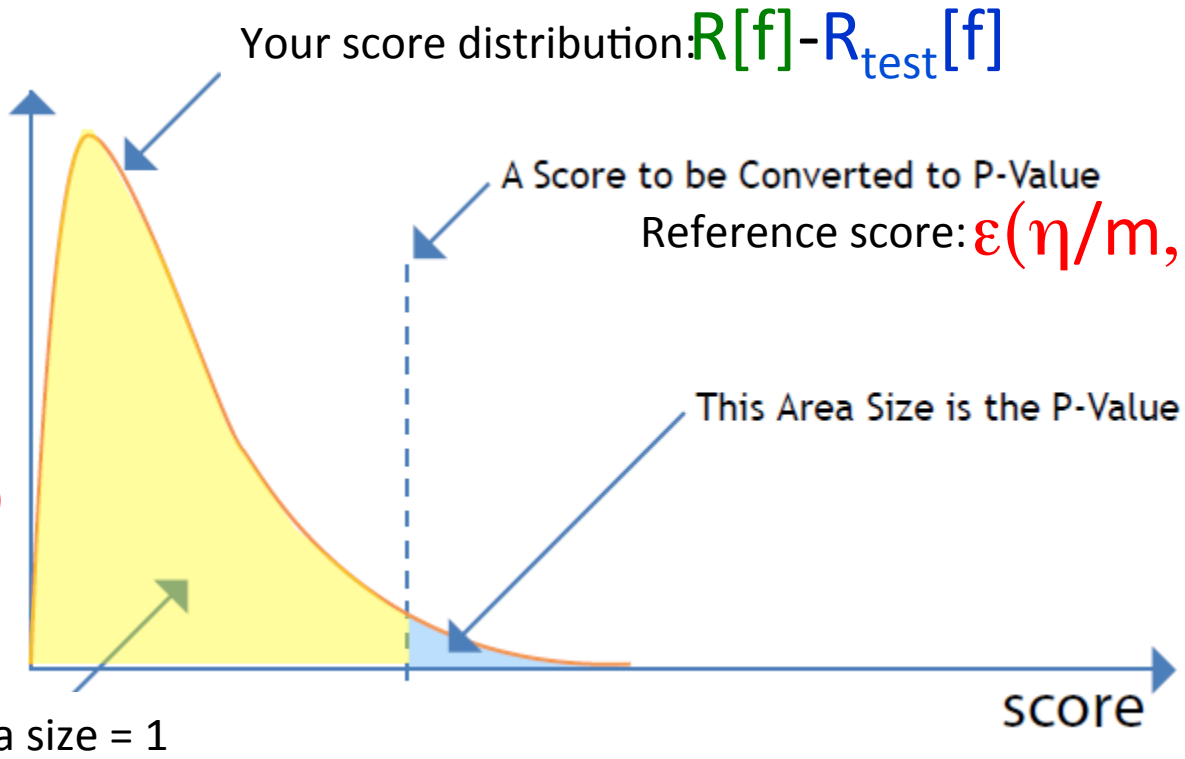
$R[f] \leq R_{test}[f] + \varepsilon(\delta, 1/n)$

Pvalue = fraction of score values above the reference score.
SMALL pvalues are good: they shed doubt on the "null hypothesis".

Complexity: Statistical

Multiple testing

Variable change



The area is now (1-η)
$0<\eta<<1$

$R[f] \le R_{test}[f] + \varepsilon(\eta/m, 1/n)$

Total area size = 1

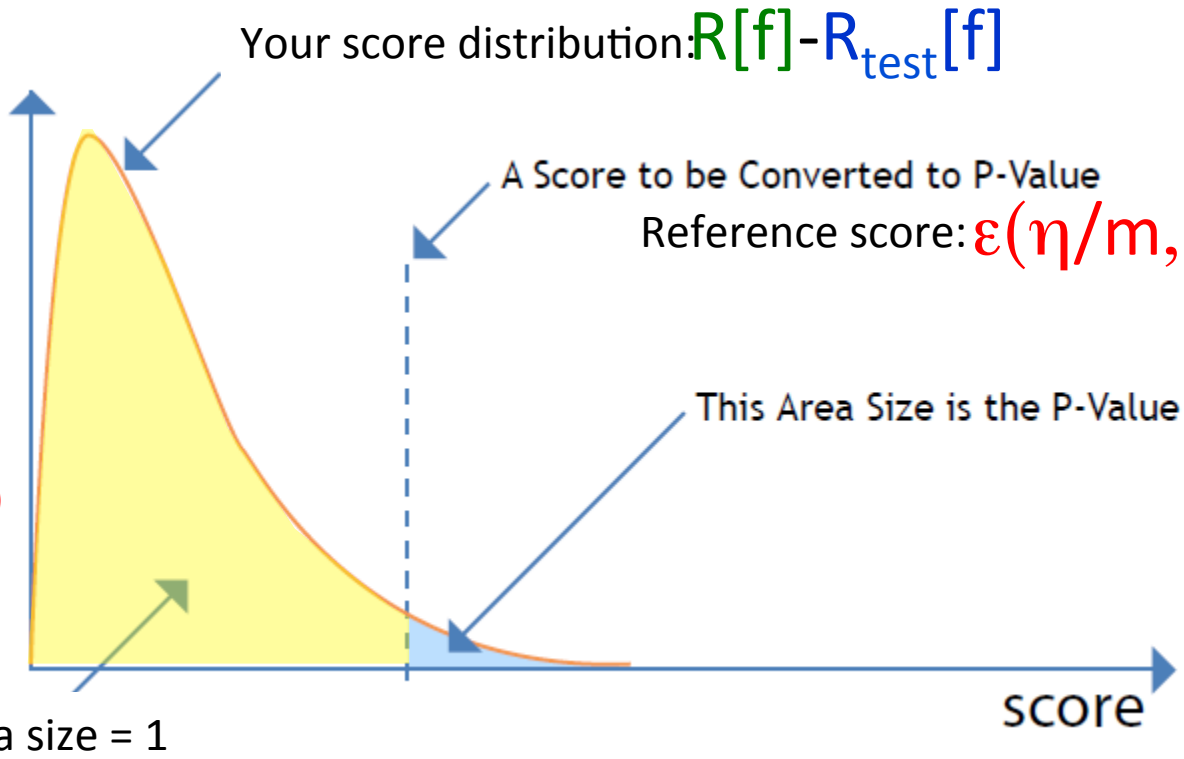
Pvalue = fraction of score values above the reference score.
SMALL pvalues are good: they shed doubt on the "null hypothesis".

Complexity: Statistical

In fact…

log(m) is a complexity/capacity

Vapnik 1974



The area is now (1-η)
0<η<<1

$R[f] \le R_{test}[f] + \varepsilon(\eta/m, 1/n)$

Total area size = 1

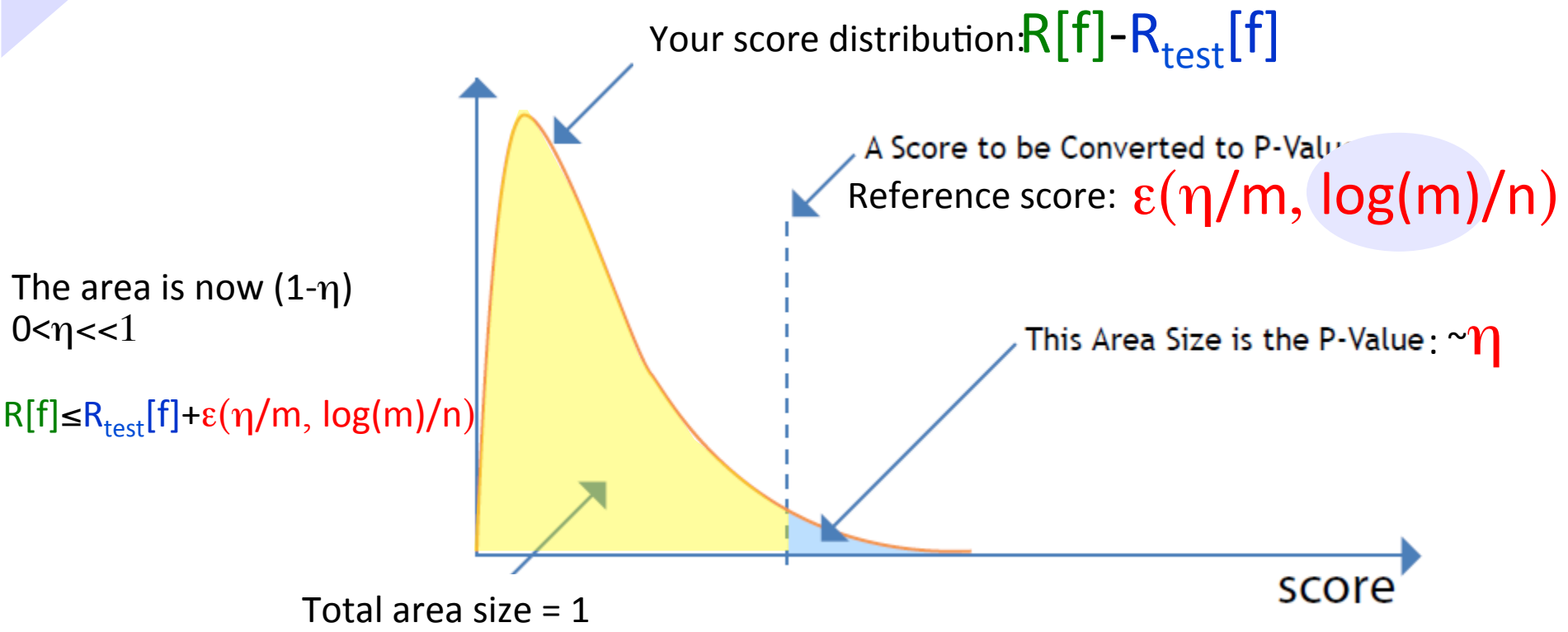
$$\varepsilon(\eta/m, 1/n) = \mathrm{sqrt}(\log(2m/\eta)/2n)$$

Complexity: Statistical

In fact…

log(m) is a complexity/capacity

Vapnik 1974



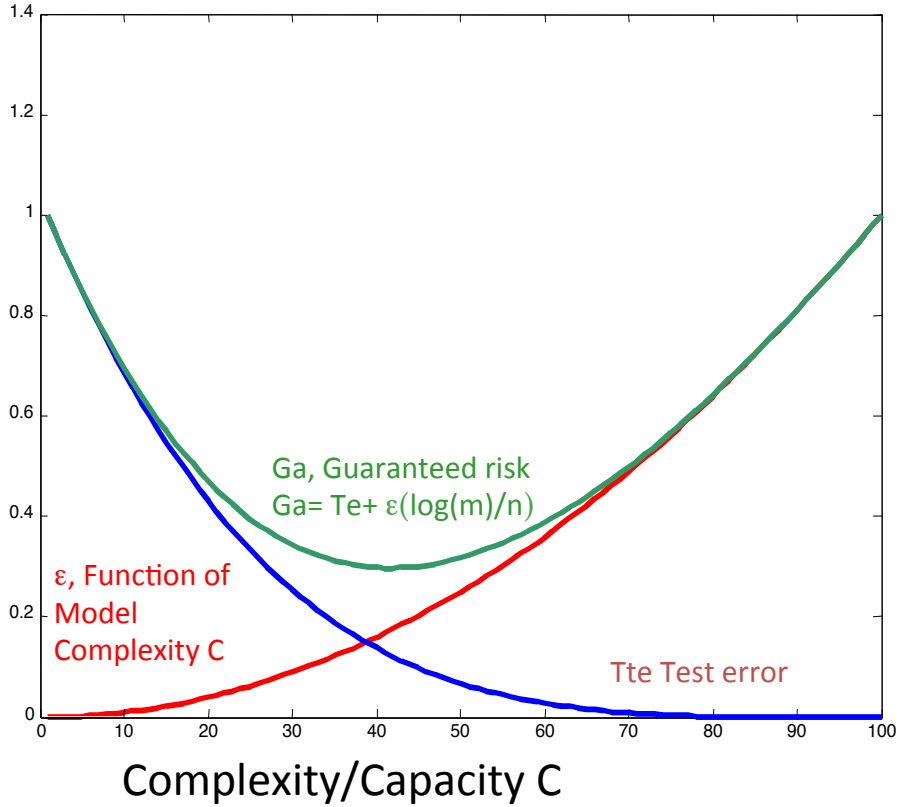
$$\varepsilon(\eta/m, 1/n) = \mathrm{sqrt}(\log(2m/\eta)/2n)$$

Complexity: Statistical

Capacity for "multiple testing"

Vapnik 1974

The "test" set became a training set!!!!



With *high* probability (1-δ),

$0<\delta<<1$

$$R[f] \le \underbrace{R_{test}[f] + \varepsilon(\delta, \log(m)/N)}_{R_{gua}[f]}$$

C

Complexity: Statistical

Capacity

Vapnik-Chervonenkis 1971

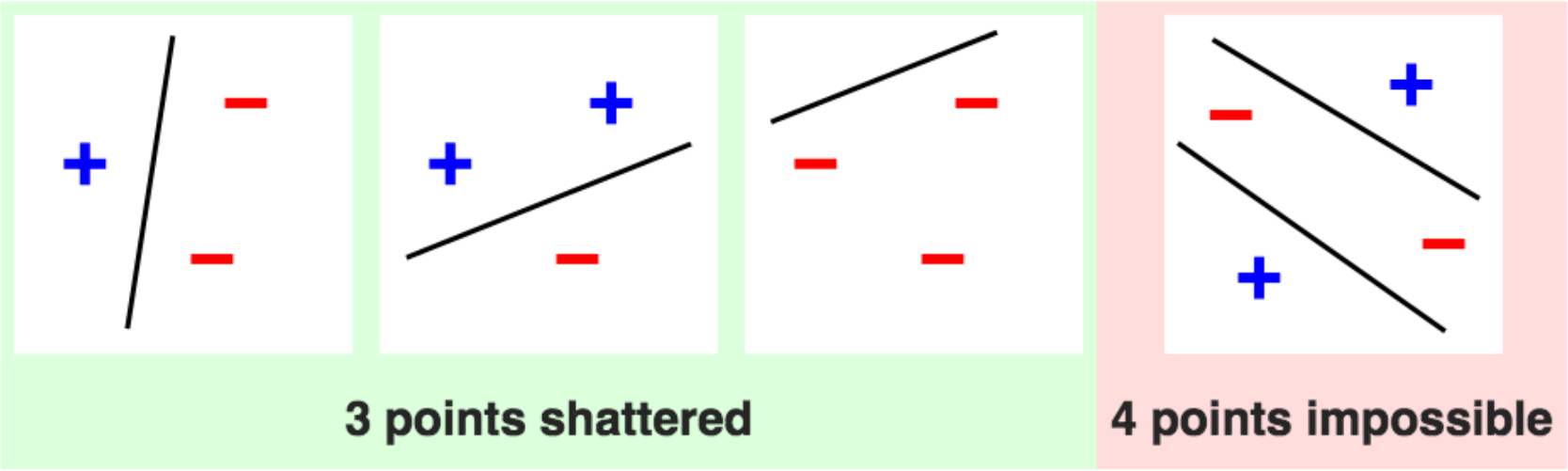
- C = log(m) is the capacity for a FINITE number of hypotheses (m is number of models tried).
- What is C for a continuous hypothesis space (parameterized with **w**, α, q, σ, etc.?)

Complexity: Statistical

VC dimension

Vapnik-Chervonenkis 1971

C = number of training examples that can be separated, *regardless of the assignment of labels*.



Source: Wikipedia

For f(**x**) = **w.x** (linear decision boundary/hyperplane) the VC dimension is C = d (the dimension of input space).

Complexity: Statistical

VC dimension of Perceptrons and kernel methods

$f(\mathbf{x}) = \mathbf{w} \bullet \Phi(\mathbf{x})$, the VC dimension is D (the dimension of FEATURE space).

$\mathbf{w} = \sum_k \alpha_k \Phi(\mathbf{x}^k) \rightarrow$ kernel trick

$f(\mathbf{x}) = \sum_k \alpha_k \Phi(\mathbf{x}^k) \bullet \Phi(\mathbf{x}) = \sum_k \alpha_k k(\mathbf{x}^k, \mathbf{x})$

$k(\mathbf{x}^k, \mathbf{x}) = \Phi(\mathbf{x}^k) \bullet \Phi(\mathbf{x})$

If D is infinite, the VC dimension is infinite!

Complexity: Statistical

VC dimension of kernel methods

Not quite that bad…

Remember the representer theorem:

$f^*(\mathbf{x}) = \text{argmin}_f R_{train}[f] + \lambda\Omega[f]$ admits a representation

$$f^*(\mathbf{x}) = \sum_k \alpha_k \, k(\mathbf{x}, \mathbf{x}^k)$$

We can consider that $k(., \mathbf{x}^k)$ are features $\Phi(.)$ of a Perceptron with N parameters (the α_k). So:

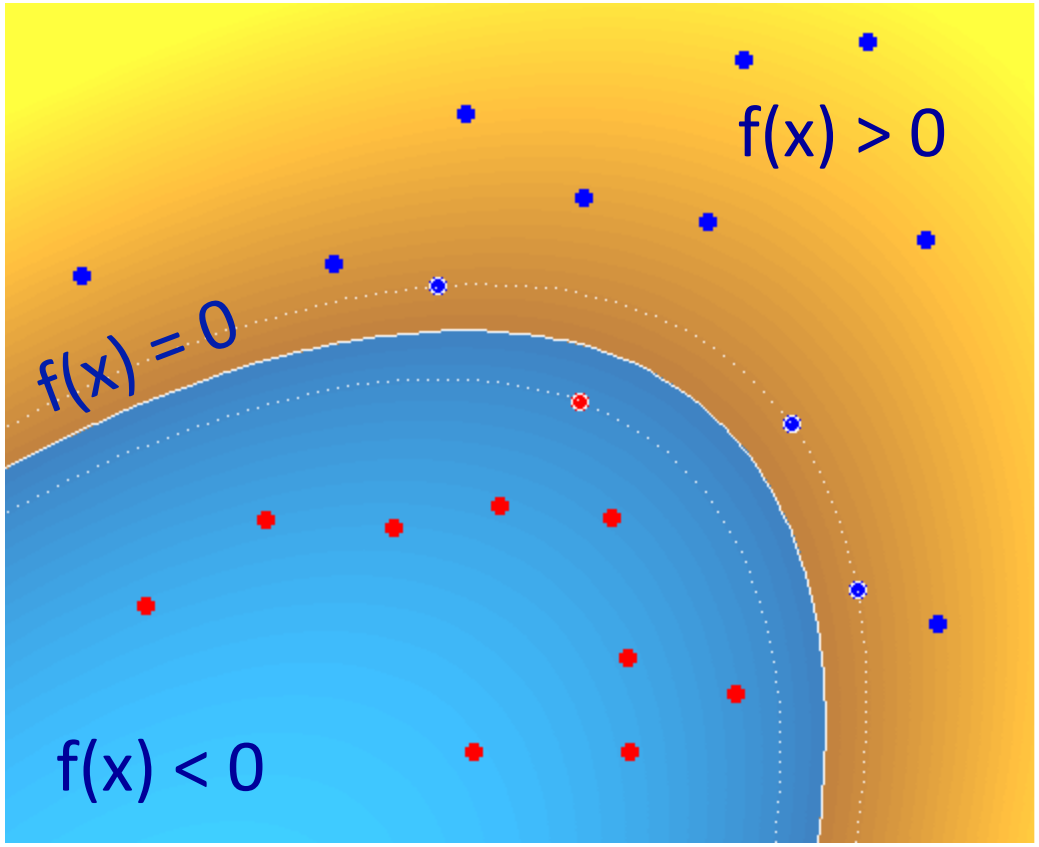
$$C \leq N.$$

For support vector machines the sum runs only over the N_S support vectors, so:

$$C \leq N_S.$$

Complexity: Statistical

SVM leave-one-out error: Bounded by N_{SV}/N



$$f(x) = \sum_k \alpha_k k(x_k, x) + b$$

SVM, Boser-Guyon-Vapnik, 1992

Complexity: Statistical

But… the kernels have HP

$f(\mathbf{x}) = \sum_k \alpha_k \, k(\mathbf{x}, \mathbf{x}^k; \theta)$

They are IN FACT universal approximators:

$C = \infty$

if we also optimize θ with the training set.

So we need to FIX θ to enjoy a finite C when we use the training set. We can then find the best θ with a validation set (a second level learning problem).

Complexity: Statistical

For λ we also need 2 levels

$\min_f R_{reg}[f]$

$R_{reg}[f] = R_{train}[f] + \lambda \| \mathbf{w} \|^2$

$\partial R_{reg}/\partial\lambda = \| \mathbf{w} \|^2$

At the optimum, $\| \mathbf{w} \|^2 = 0$. Not very helpful: C = 0 if we also optimize λ with the training set.

So we need to FIX λ TOO to enjoy a finite C when we use the training set. We can then find the best λ with a validation set (another second level learning problem).

Summary (so far)

- Heuristic HP search (wrapper-style) is computationally expensive, so we want to "push down" some HP to the learning machine.

- But optimizing both parameters and HP with the training error is not a good idea because of:
  - **Computational complexity**: we loose **convexity** in parameter space.
  - **Statistical complexity**: to train we get need **finite capacity** $0<C<\infty$.

- Embedded methods solve a **bi-level optimization problem** by direct optimization of the HP by **cross-validation** using **knowledge about the learning machine** to conduct a more **efficient search**. But how? Grid search => $K * g^d$ learning machine to train. K=number of folds in K-fold, g = number of grid points, d the number of hyper-parameters.

- Idea #1: Approximate the leave-one-out error by "sensitivity analysis" to get a cheap R_{CV}. Gain: replace K-fold by training ONCE.
- Idea #2: Diagonalise X^TX or XX^T (or the kernel matrix): makes inversion "cheap" by varying λ. Gain: replace training g machines by ONE.

Idea #1: Virtual leave-one-out

Support Vector Machines:

Refinements of N_{SV}/N.

Linear regression:

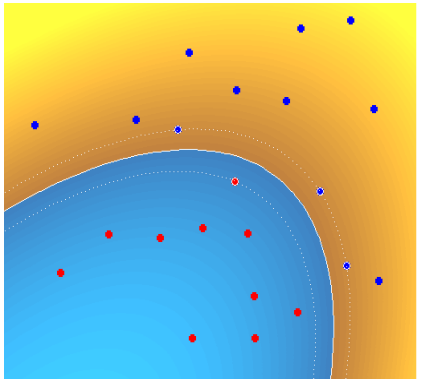
$r_{LOO}^{k} = r_{TRAIN}^{k} / (1 - [XX^{+}]_{kk})$

Kernel methods including SVM, regression, logistic regression:

Efficient approximate leave-one-out cross-validation for kernel logistic regression (Cawley-Talbot, 2008)

Neural Networks:

Local Linear Least Squares: Performing Leave-one-out Without Leaving Anything Out (Monari-Dreyfus, 2002)

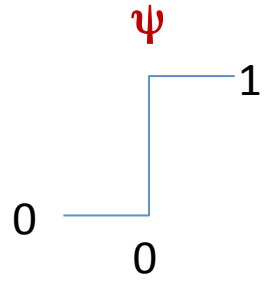


Example 1: SVM (hard margin)

- LOO functional margin

$$R_{LOO} = (1/N) \sum_{k=1:N} \psi(-y_k f^{(-k)}(\mathbf{x}^k))$$

$$R_{LOO} \le (1/N) \sum_{k \in \{SV\}} \psi(U^k - 1)$$

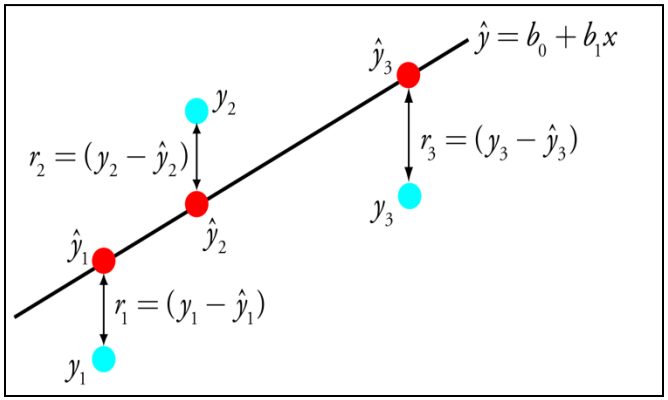


- U^k is a bound on the change in functional margin for SV

$0 \le y_k f^{(0)}(\mathbf{x}^k) - y_k f^{(-k)}(\mathbf{x}^k) \le U^k$

- Simplest bound $U^k = \infty \Rightarrow R_{LOO} = N_{SV}/N$.
- Jaakkola-Haussler (1999): $U^k = \alpha_k \, k(\mathbf{x}^k, \mathbf{x}^k)$
- Opper-Winther (2000): $U^k = \alpha_k / [K_{SV}^{-1}]_{kk}$
- Span bound Vapnik-Chapelle (2000): $U^k = \alpha_k \, S_k^2$

where S_k is the distance of $\phi(\mathbf{x}^k)$ to the span of other SV.



Example 2: Linear regression

$$R_{LOO} = (1/N) \sum_{k=1:N} (r_{LOO}^k)^2$$ MSE

- $r_{LOO}^k = y_k - f^{(-k)}(\mathbf{x}^k)$ residual of example k for $f^{(-k)}$
- $r_{TRAIN}^k = y_k - f(\mathbf{x}^k)$ residual of example k for f
- Efron-Tibshirani, 1993

$$r_{LOO}^k = r_{TRAIN}^k / (1 - [XX^+]_{kk})$$

Idea #2: Matrix factorization

- We need to compute: $(XX^T + \lambda I)^{-1}$ or $(X^TX + \lambda I)^{-1}$, whichever is smallest: dim (N,N) or (d,d).

  So $(K + \lambda I)^{-1}$ in general.
- We want to factorize a symmetric matrix K as:

  $K = U^TDU$, where D is a diagonal matrix

  (of eigenvalues) and U is an orthogonal matrix

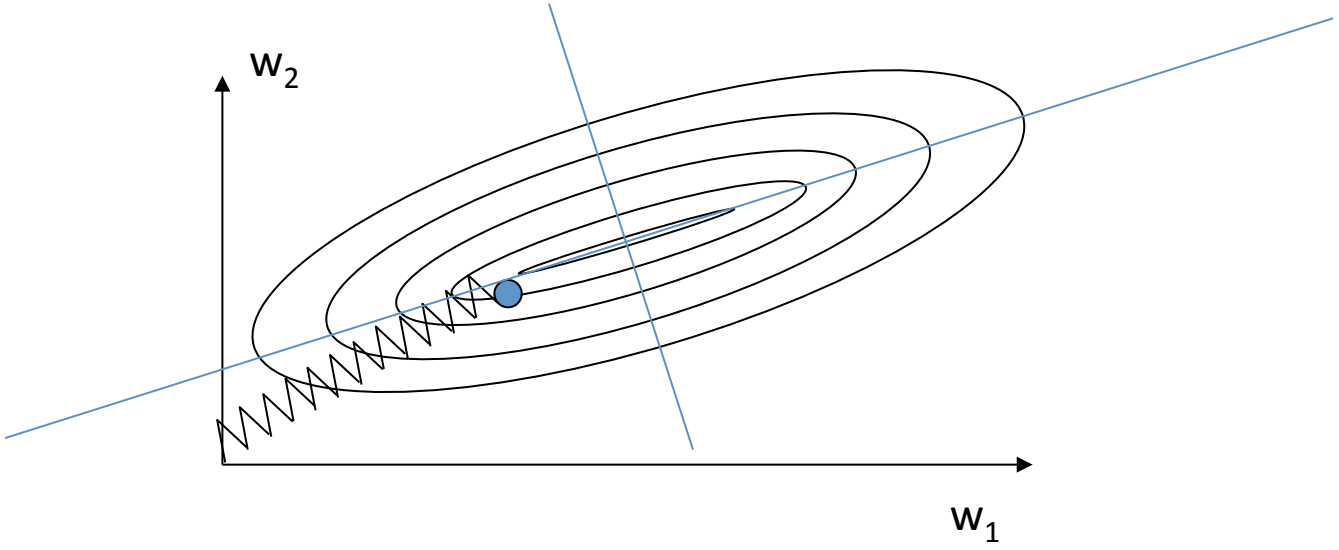
  of eigenvectors: $U^TU = I$ (and $U^T(\lambda I)U = \lambda I$)
- Then $(K + \lambda I)^{-1} = (U^TDU + U^T(\lambda I)U)^{-1}$

$$= (U^T(D+\lambda I)U)^{-1}$$

$$(K + \lambda I)^{-1} = U(D+\lambda I)^{-1}U^T$$

How does this relate to dimensionality reduction?

- Reminder: For f(**x**) = **w.x** the VC dimension is C = d (the dimension of input space).
- $R_{reg} = R_{train} + \lambda \| \mathbf{w} \|^2$



Different regularizers

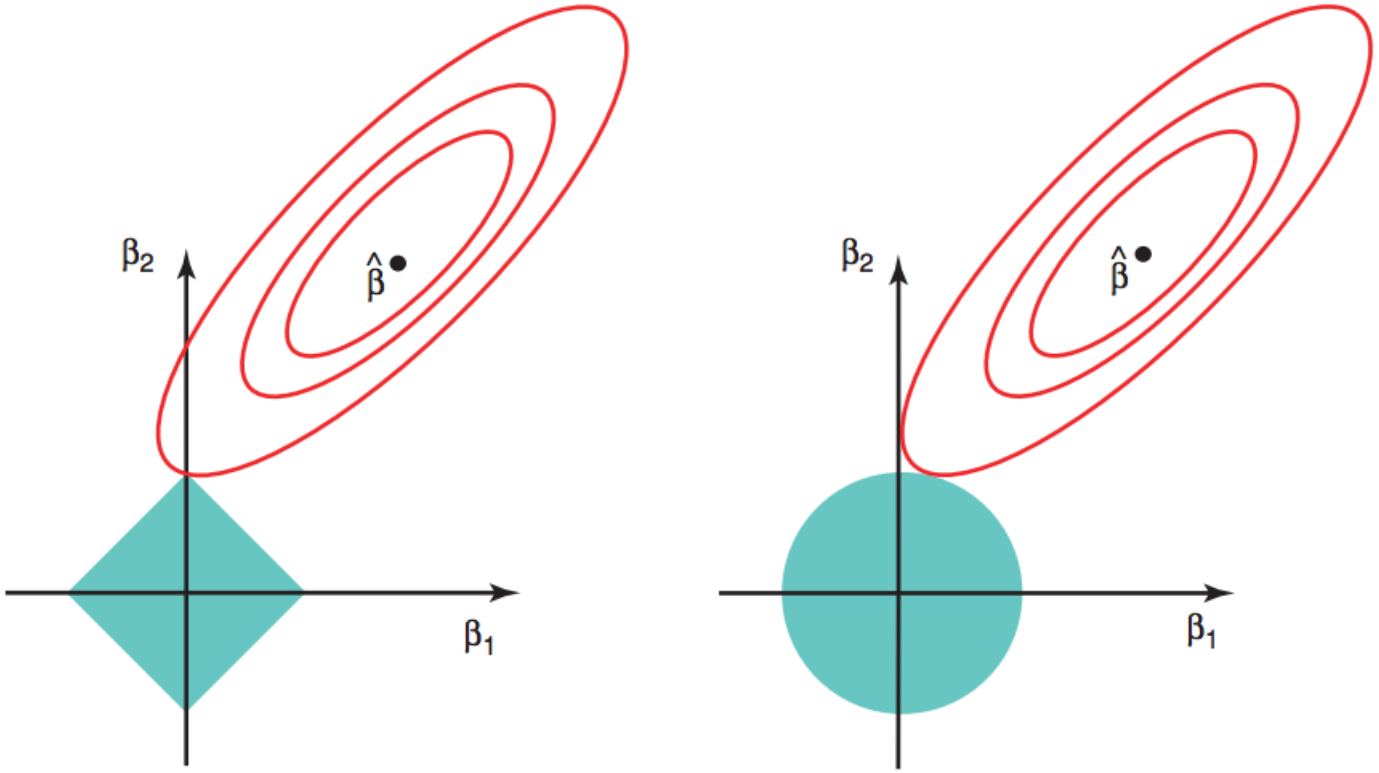
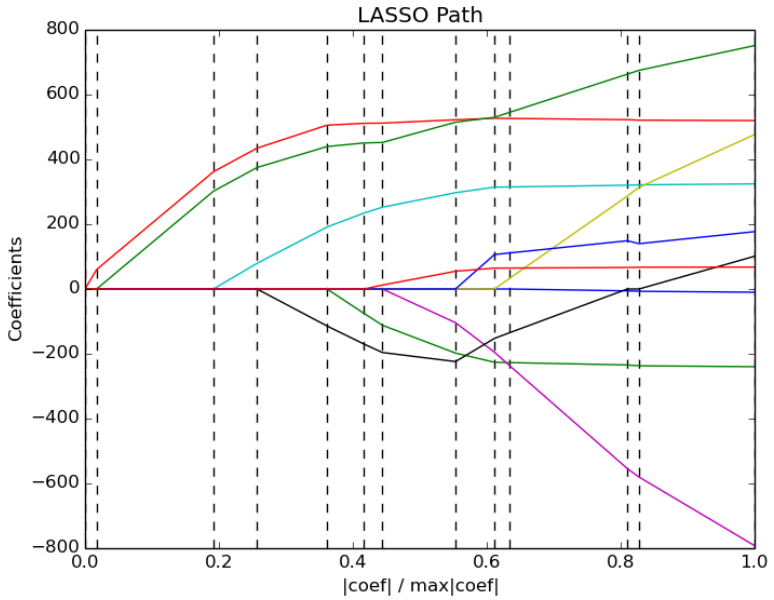


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions,* $|\beta_1| + |\beta_2| \leq s$ *and* $\beta_1^2 + \beta_2^2 \leq s$, *while the red ellipses are the contours of the RSS.*

Source: ISLR http://www-bcf.usc.edu/~gareth/ISL/

Regularization path



http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_lars.html

See also:

- Hastie et al 2004: http://www.jmlr.org/papers/volume5/hastie04a/hastie04a.pdf
- Efron 2004: http://statweb.stanford.edu/~tibs/ftp/lars.pdf
- Friedman et al 2010: http://core.ac.uk/download/pdf/6287975.pdf

Summary

- Heuristic HP search (wrapper-style) is computationally expensive, so we want to "push down" some HP to the learning machine.

- But optimizing both parameters and HP with the training error is not a good idea because of:
  - **Computational complexity**: we loose convexity in parameter space.
  - **Statistical complexity**: we get infinite capacity.

- Embedded methods solve a **bi-level optimization problem** by direct optimization of the HP with the CV error, e.g. by gradient descent.

- Computational tricks for least square regression:
  - Leave-one-out error computed "cheaply" (**virtual LOO**).
  - Matrix factorization to vary the ridge value inexpensively.

- Shrinkage ($\|\mathbf{w}\|_q$ regularizers) perform space dimensionality reduction.

Come to my office hours…
Wed 2:30-4:30 Soda 329

Next time:

Gaussian classifier