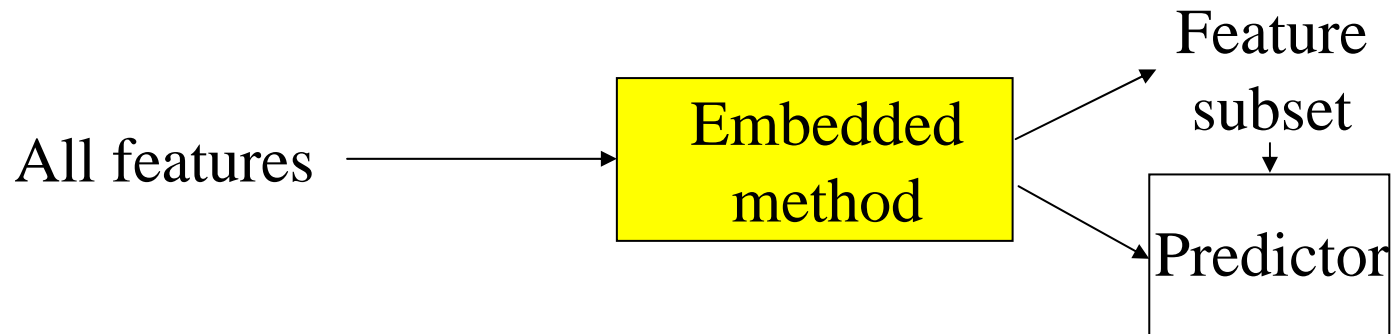
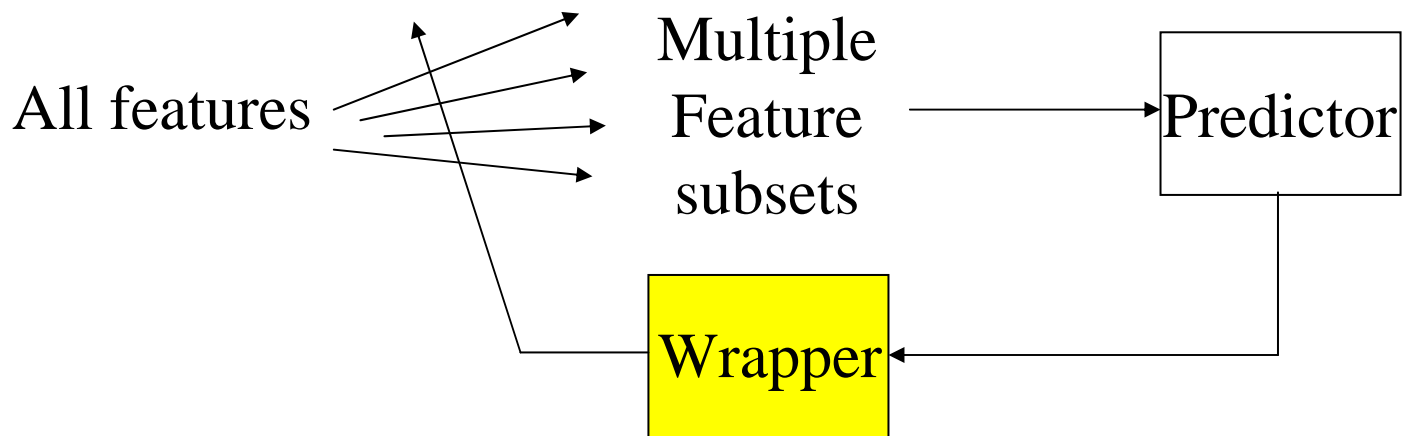
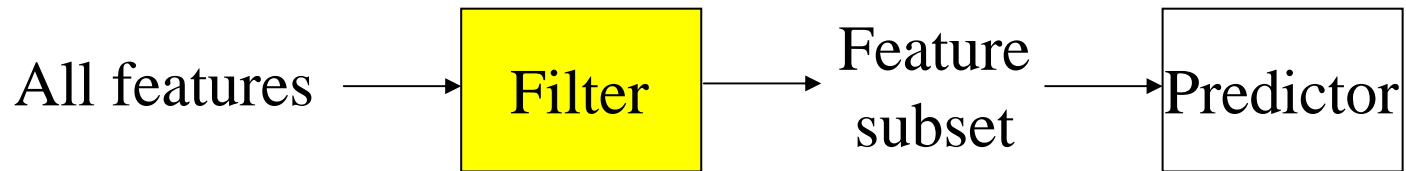


# *Lecture 3:* *Embedded methods*

Isabelle Guyon

[isabelle@clopinet.com](mailto:isabelle@clopinet.com)

# *Filters, Wrappers, and Embedded methods*



# *Filters*

---

## Methods:

- Criterion: Measure feature/feature subset “relevance”
- Search: Usually order features (individual feature ranking or nested subsets of features)
- Assessment: Use statistical tests

## Results:

- Are (relatively) robust against overfitting
- May fail to select the most “useful” features

# *Wrappers*

---

## Methods:

- Criterion: Measure feature subset “usefulness”
- Search: Search the space of all feature subsets
- Assessment: Use cross-validation

## Results:

- Can in principle find the most “useful” features, but
- Are prone to overfitting

# *Embedded Methods*

---

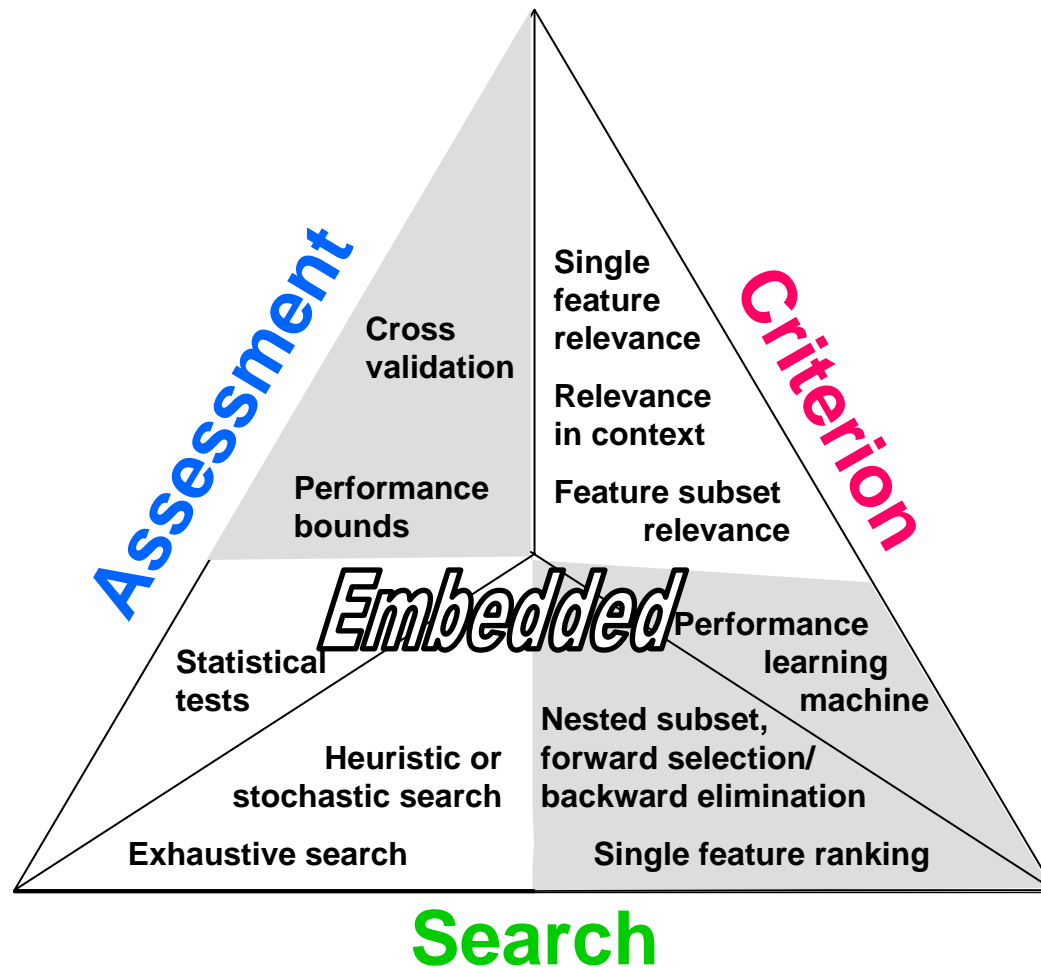
## Methods:

- Criterion: Measure feature subset “usefulness”
- Search: **Search guided by the learning process**
- Assessment: Use cross-validation

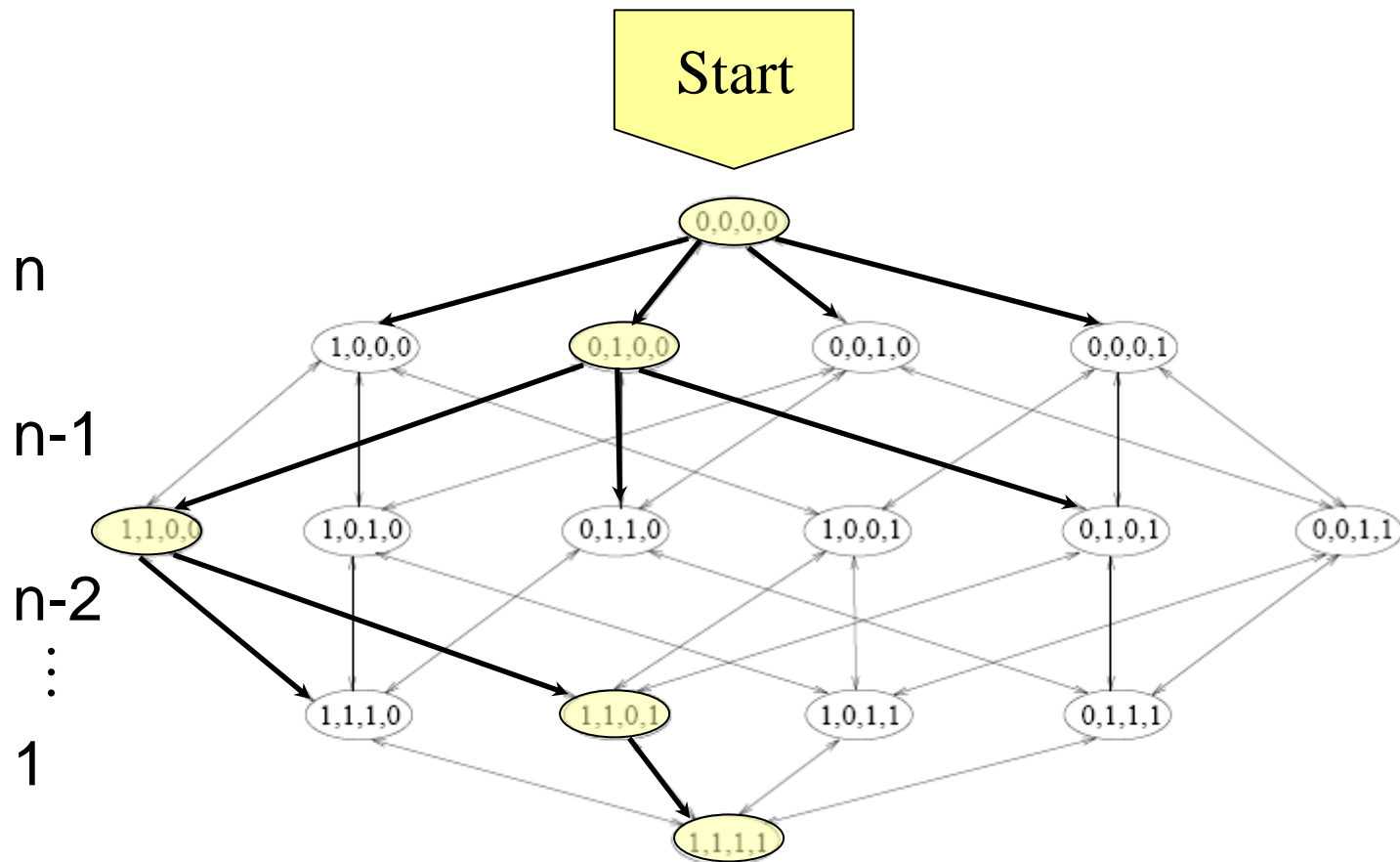
## Results:

- Similar to wrappers, but
- Less computationally expensive
- Less prone to overfitting

# Three “Ingredients”

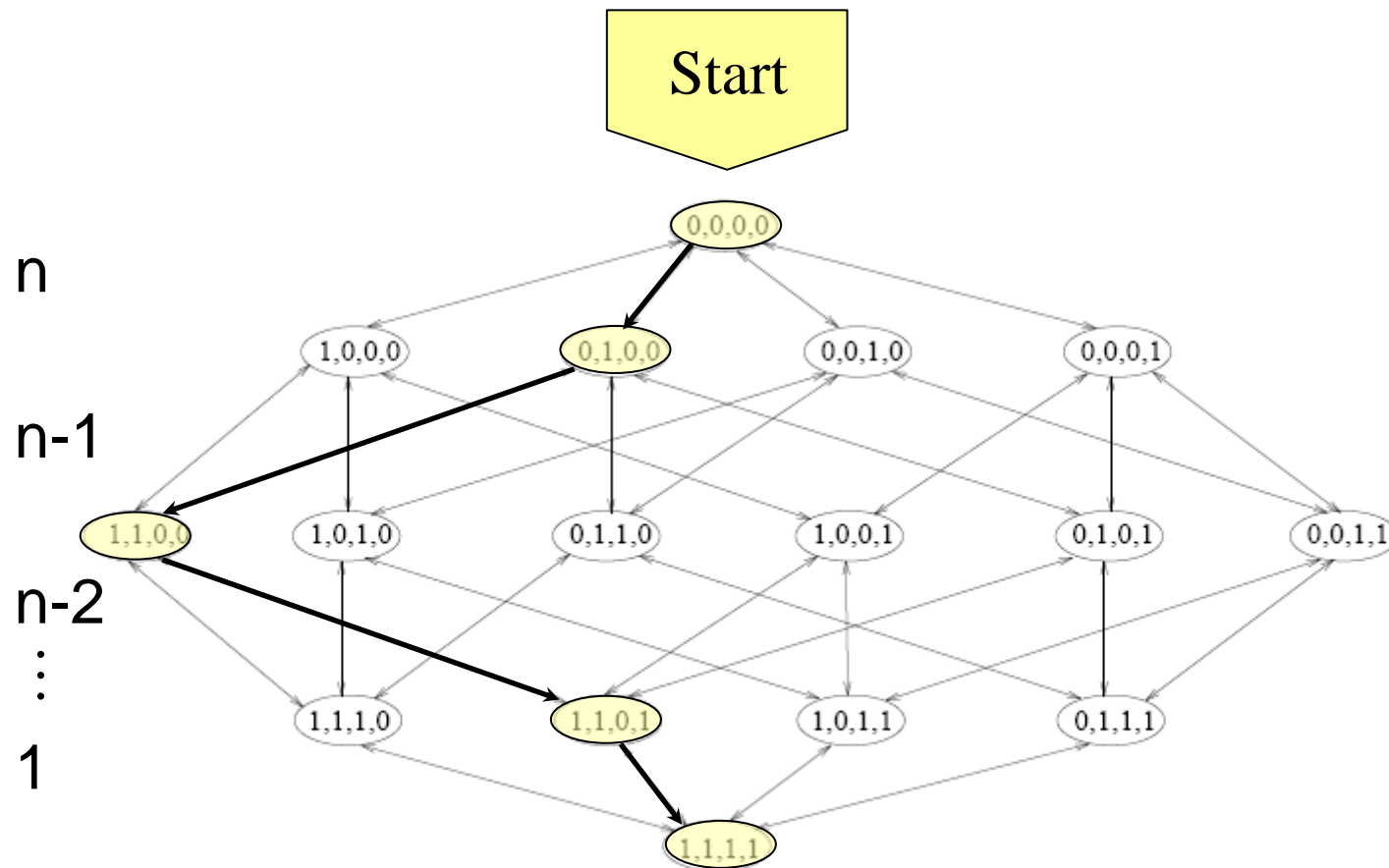


# *Forward Selection (wrapper)*



Also referred to as SFS: Sequential Forward Selection

# *Forward Selection (embedded)*




Guided search: we do not consider alternative paths.



# *Forward Selection with GS*

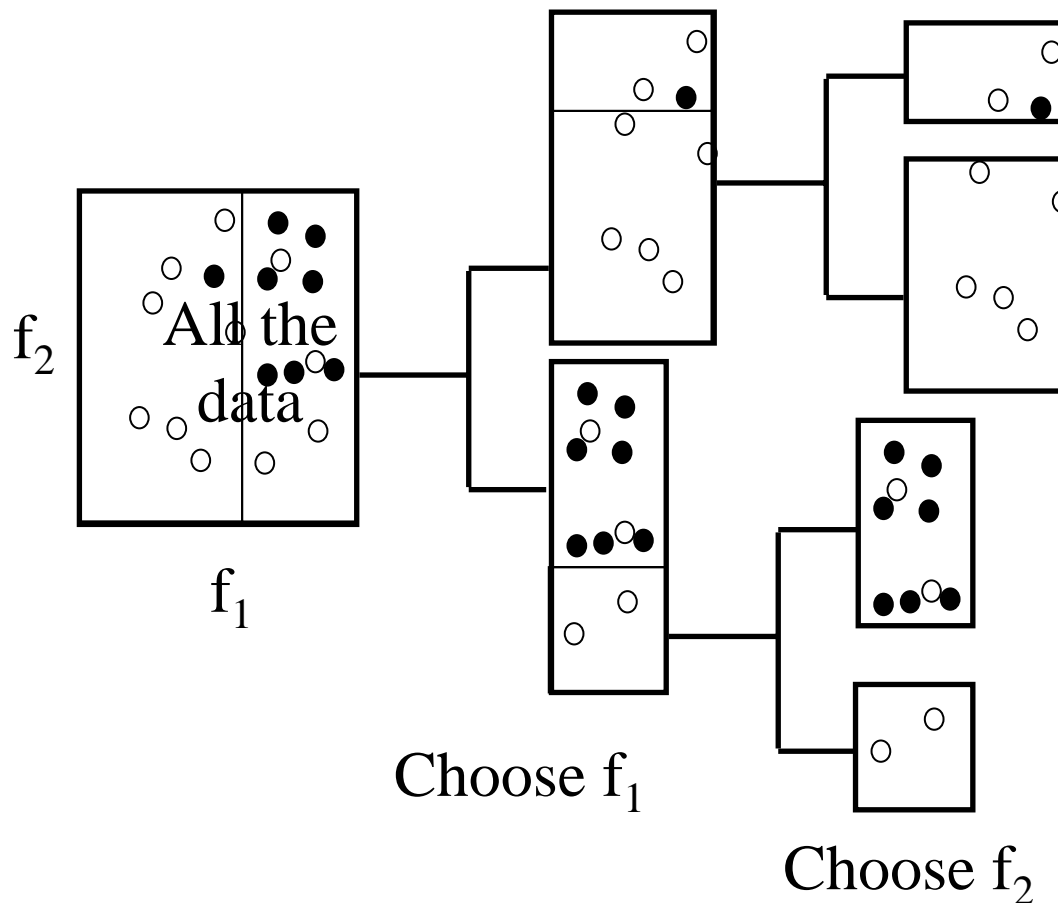
*Stoppiglia, 2002. Gram-Schmidt orthogonalization.*

- 
- Select a first feature  $X_{v(1)}$  with maximum cosine with the target  $\cos(\mathbf{x}_i, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y} / \|\mathbf{x}\| \|\mathbf{y}\|$
  - For each remaining feature  $X_i$ 
    - Project  $X_i$  and the target  $Y$  on the null space of the features already selected
    - Compute the cosine of  $X_i$  with the target in the projection
  - Select the feature  $X_{v(k)}$  with maximum cosine with the target in the projection.

Embedded method for the linear least square predictor

# Forward Selection w. Trees

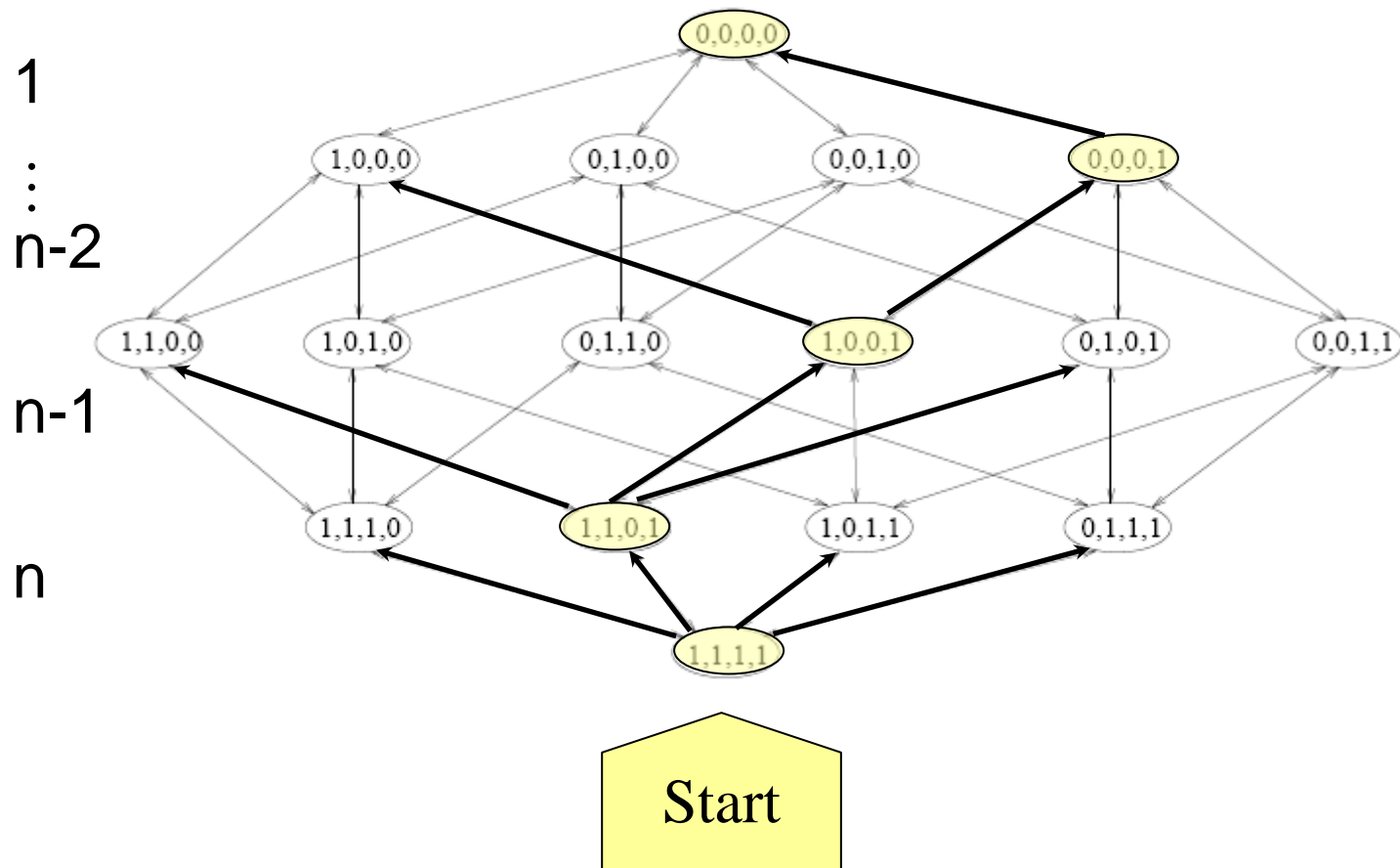
- Tree classifiers,  
like *CART* (Breiman, 1984) or *C4.5* (Quinlan, 1993)



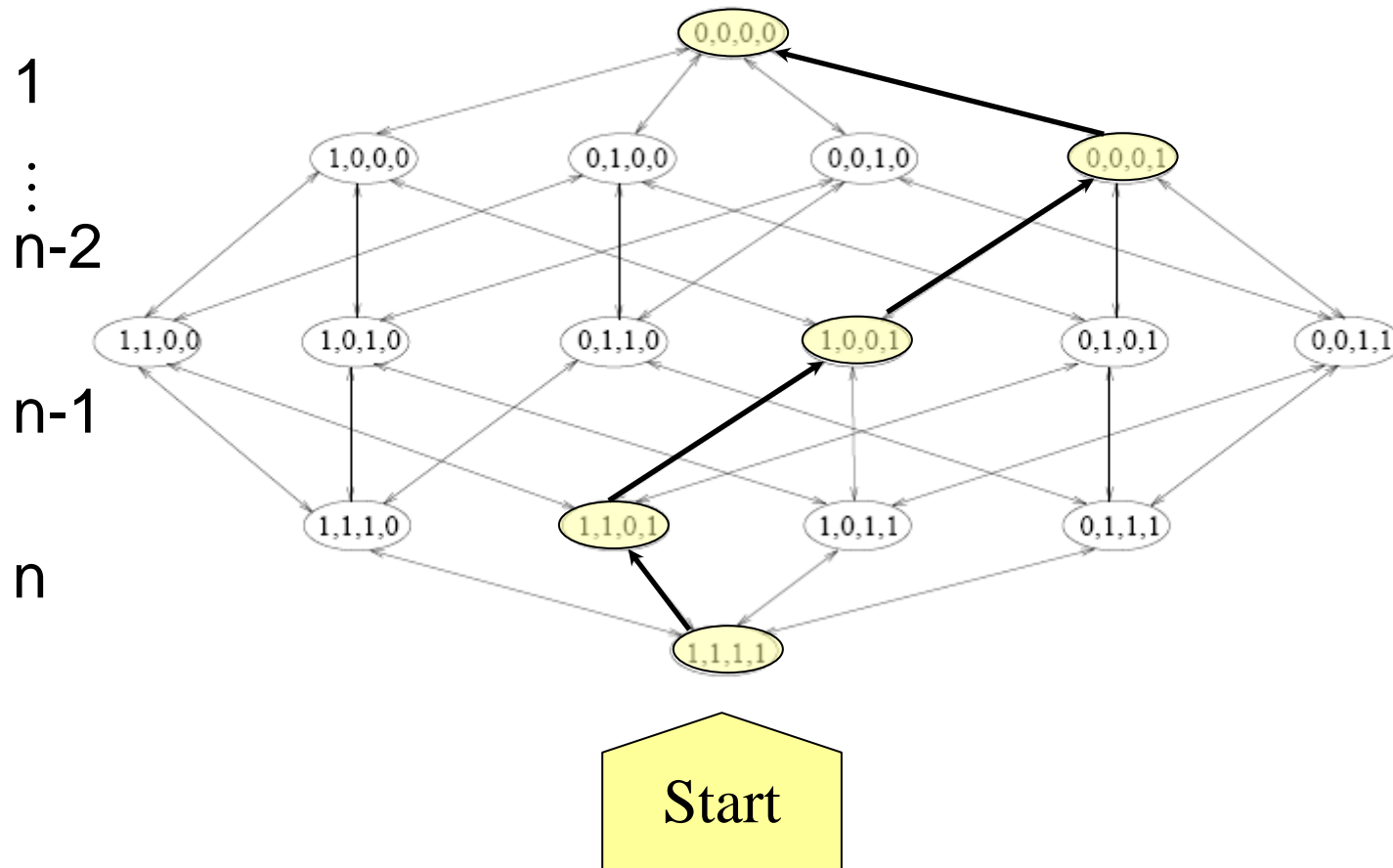
At each step,  
choose the  
feature that  
“reduces entropy”  
most. Work  
towards “node  
purity”.

# *Backward Elimination* (wrapper)

Also referred to as SBS: Sequential Backward Selection




# *Backward Elimination (embedded)*



# *Backward Elimination: RFE*

*RFE-SVM, Guyon, Weston, et al, 2002*

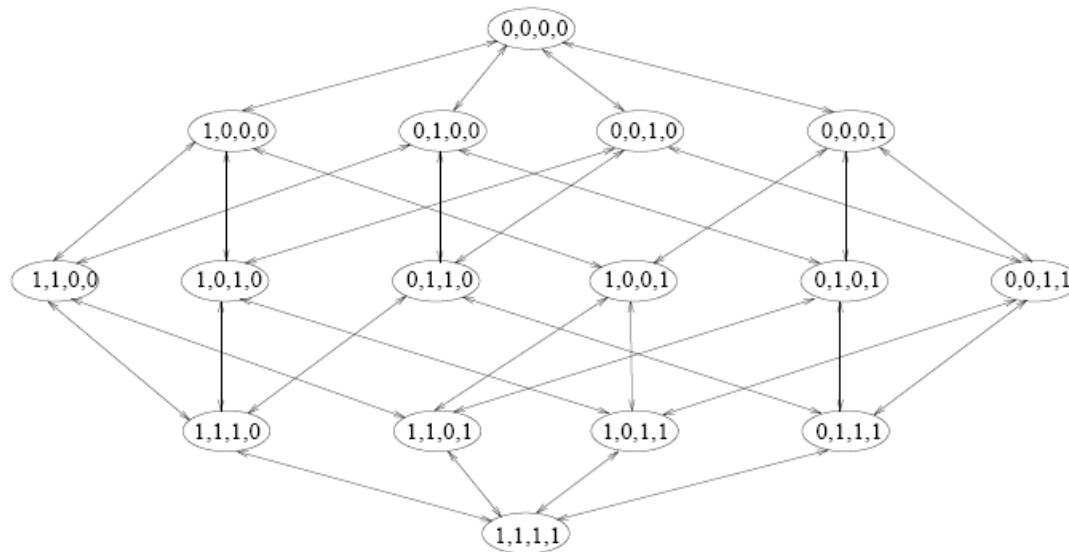
Start with all the features.

- 
- Train a learning machine  $f$  on the current subset of features by minimizing a risk functional  $J[f]$ .
  - For each (remaining) feature  $X_i$ , estimate, without retraining  $f$ , the change in  $J[f]$  resulting from the removal of  $X_i$ .
  - Remove the feature  $X_{v(k)}$  that results in improving or least degrading  $J$ .

Embedded method for SVM, kernel methods, neural nets.

# Scaling Factors

**Idea:** Transform a discrete space into a continuous space.



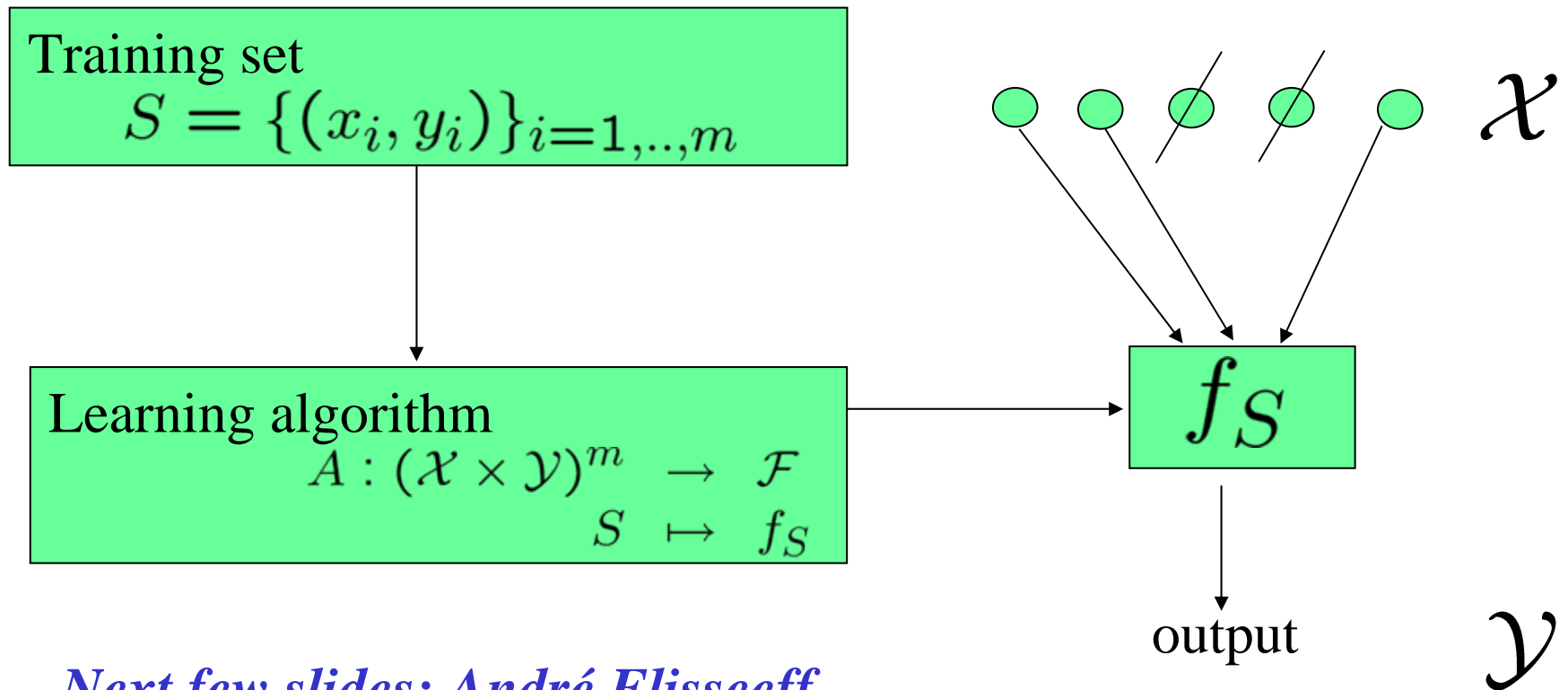
$$\sigma = [\sigma_1, \sigma_2, \sigma_3, \sigma_4]$$

- Discrete indicators of feature presence:  $\sigma_i \in \{0, 1\}$
- Continuous scaling factors:  $\sigma_i \in \mathbb{R}$

Now we can do gradient descent!

# Formalism ( chap. 5)

- Definition: an embedded feature selection method is a *machine learning algorithm* that returns a model using a limited number of features.

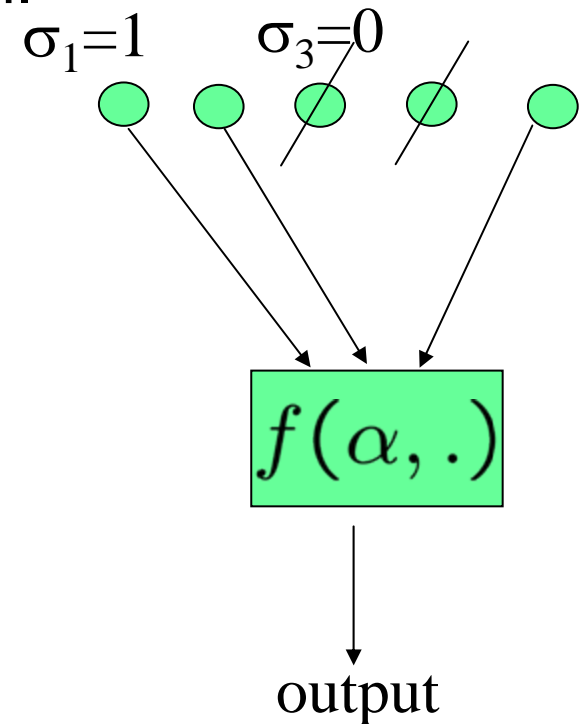


# Feature selection as model selection - 1

- Let us consider the following set of functions parameterized by  $\alpha$  and where  $\sigma \in \{0,1\}^n$  represents the use ( $\sigma_i=1$ ) or rejection of feature  $i$ .

$$\sigma \circ x = (\sigma_1 x_1, \dots, \sigma_n x_n)$$

$$\begin{aligned} f : \Lambda \times \mathbb{R}^n &\rightarrow \mathbb{R} \\ (\alpha, \sigma \circ x) &\mapsto f(\alpha, \sigma \circ x) \end{aligned}$$



Example (linear systems,  $\alpha=w$ ):

$$w \cdot x + b = \sum_{i=1}^n w_i x_i \sigma_i = \sum_{\sigma_i \neq 0} w_i x_i$$



# *Feature selection as model selection - 2*

---

- We are interested in finding  $\alpha$  and  $\sigma$  such that the generalization error is minimized:

$$\min_{\sigma, \alpha} R(\alpha, \sigma)$$

where

$$R(\alpha, \sigma) = \int L(f(\alpha, \sigma \circ x), y) dP(x, y)$$

Sometimes we add a constraint: # non zero  $\sigma_i$ 's  $\cdot s_0$

Problem: the generalization error is not known...

# *Feature selection as model selection - 3*

- The generalization error is not known directly but bounds can be used.
  - Most embedded methods minimize those bounds using different optimization strategies:
    - Add and remove features
    - Relaxation methods and gradient descent
    - Relaxation methods and regularization
- 

Example of bounds (linear systems):

Non separable 
$$R(w, \sigma) \leq \frac{1}{m} \sum_{k=1}^m L(w \cdot x_k + b, y_k) + O\left(\frac{r \|w\|}{\sqrt{m}}\right)$$

Linearly separable 
$$R(w, \sigma) \leq O\left(\frac{r^2 \|w\|^2}{m}\right)$$

# *Feature selection as model selection -4*

- How to minimize  $\min_{\sigma, \alpha} R(\alpha, \sigma)$ ?

Most approaches use the following method:

1. Set  $\sigma = (1, \dots, 1)$
2. Compute  $\alpha^* = \arg \min_{\alpha} \hat{R}(\alpha, \sigma)$
3. Compute  $\sigma^* = \arg \min_{\sigma} \hat{R}(\alpha^*, \sigma)$
4. Set  $\sigma \leftarrow \sigma^*$  and go back to 2.

This optimization is often done by relaxing the constraint  $\sigma \in \{0, 1\}^n$  as  $\sigma \in [0, 1]^n$

# Add/Remove features 1

- Many learning algorithms are cast into a minimization of some regularized functional:

$$\underbrace{\min_{\alpha} \hat{R}(\alpha, \sigma)}_{G(\sigma)} = \min_{\alpha} \sum_{k=1}^m L(f(\alpha, \sigma \circ x_k), y_k) + \Omega(\alpha)$$

Diagram illustrating the components of the regularized functional minimization:

- $G(\sigma)$  (under the minimization over  $\alpha$ )
- Empirical error (under the sum  $\sum_{k=1}^m$ )
- Regularization capacity control (under  $\Omega(\alpha)$ )

- What does  $G(\sigma)$  become if one feature is removed?
- Sometimes,  $G$  can only increase... (e.g. SVM)

# *Add/Remove features 2*

- It can be shown (under some conditions) that the removal of one feature will induce a change in  $G$  proportional to:

$$\sum_{k=1}^m \left( \frac{\partial f}{\partial x^i} \right)^2 (\alpha, x_k)$$

Gradient of  $f$  wrt.  $i^{\text{th}}$   
feature at point  $x_k$

- Examples: SVMs  $\longrightarrow \frac{\partial f}{\partial x^i} \propto w_i$   
! RFE ( $\Omega(\alpha) = \Omega(w) = \sum_i w_i^2$ )

# *Add/Remove features - RFE*

- Recursive Feature Elimination

1. Set  $F = \{1, \dots, n\}$

2. Get  $w^*$  as the solution on a SVM on the data set restricted to features in  $F$

Minimize  
estimate of  
 $R(\alpha, \sigma)$   
wrt.  $\alpha$

3. Select top features as ranked by the  $|w_i^*|$ 's

Minimize the  
estimate  $R(\alpha, \sigma)$   
wrt.  $\sigma$  and under  
a constraint that  
only limited  
number of  
features must be  
selected

4. Back to 2.

# *Add/Remove feature summary*

---

- Many algorithms can be turned into embedded methods for feature selections by using the following approach:
  1. Choose an objective function that measure how well the model returned by the algorithm performs
  2. “Differentiate” (or sensitivity analysis) this objective function according to the  $\sigma$  parameter (i.e. how does the value of this function change when one feature is removed and the algorithm is rerun)
  3. Select the features whose removal (resp. addition) induces the desired change in the objective function (i.e. minimize error estimate, maximize alignment with target, etc.)

What makes this method an ‘embedded method’ is the use of the structure of the learning algorithm to compute the gradient and to search/weight relevant features.

# Gradient descent - 1

- How to minimize  $\min_{\sigma, \alpha} R(\alpha, \sigma)$ ?

Most approaches use the following method:

1. Set  $\sigma = (1, \dots, 1)$

2. Compute  $\alpha^* = \arg \min_{\alpha} R(\alpha, \sigma)$

Would it make sense to perform just a gradient step here too?

3. Compute  $\sigma^* = \sigma - \lambda \nabla_{\sigma} R(\alpha^*, \sigma)$

Gradient step in  $[0, 1]^n$ .

4. Set  $\sigma \leftarrow \sigma^*$  and go back to 2.



# *Gradient descent 2*

---

## Advantage of this approach:

- can be done for non-linear systems (e.g. SVM with Gaussian kernels)
- can mix the search for features with the search for an optimal regularization parameters and/or other kernel parameters.

## Drawback:

- heavy computations
- back to gradient based machine algorithms (early stopping, initialization, etc.)

# *Gradient descent*

## *summary*

---

- Many algorithms can be turned into embedded methods for feature selections by using the following approach:
  1. Choose an objective function that measure how well the model returned by the algorithm performs
  2. Differentiate this objective function according to the  $\sigma$  parameter
  3. Performs a gradient descent on  $\sigma$ . At each iteration, rerun the initial learning algorithm to compute its solution on the new scaled feature space.
  4. Stop when no more changes (or early stopping, etc.)
  5. Threshold values to get list of features and retrain algorithm on the subset of features.

Difference from add/remove approach is the search strategy. It still uses the inner structure of the learning model but it scales features rather than selecting them.

# *Design strategies revisited*

- Model selection strategy: find the subset of features such that the model is the best.
- Alternative strategy: Directly minimize the number of features that an algorithm uses (focus on feature selection directly and forget generalization error).
- In the case of linear system, feature selection can be expressed as:

$$\min_w \sum_{i=1}^n 1_{w_i \neq 0}$$

$$\text{Subject to } y_k (w \cdot x_k + b) \geq 0$$

# *Feature selection for linear system is NP hard*

---

- Amaldi and Kann (1998) showed that the minimization problem related to feature selection for linear systems is NP hard.
- Is feature selection hopeless?
- How can we approximate this minimization?

# *Minimization of a sparsity function*

---

- Replace  $\sum_{i=1}^n 1_{w_i \neq 0}$  by another objective function:
  - $l_1$  norm:  $\longrightarrow \|w\|_1 = \sum_{i=1}^n |w_i|$
  - Differentiable function:  $\longrightarrow \sum_{i=1}^n (1 - \exp^{-\alpha|w_i|})$
- Do the optimization directly!

# *The $l_1$ SVM*

---

- The version of the SVM where  $\|w\|^2$  is replaced by the  $l_1$  norm  $\sum_i |w_i|$  can be considered as an embedded method:
  - Only a limited number of weights will be non zero (tend to remove redundant features)
  - Difference from the regular SVM where redundant features are all included (non zero weights)

# *Effect of the regularizer*

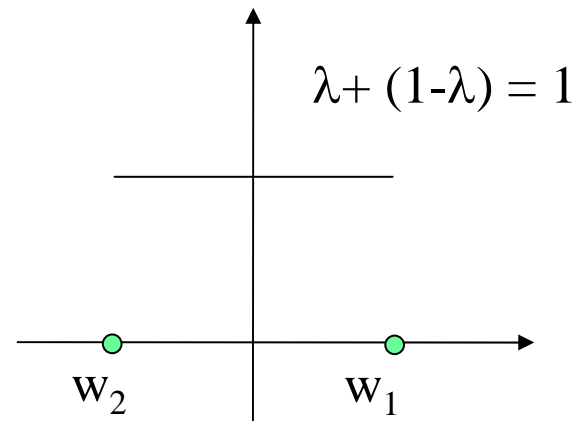
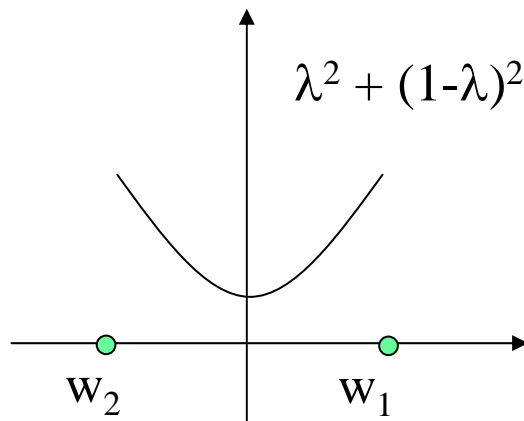
- Changing the regularization term has a strong impact on the generalization behavior...
- Let  $w_1=(1,0)$ ,  $w_2=(0,1)$  and  $w_\lambda=(1-\lambda)w_1+\lambda w_2$  for  $\lambda \in [0,1]$ , we have:

- $\|w_\lambda\|^2 = (1-\lambda)^2 + \lambda^2$

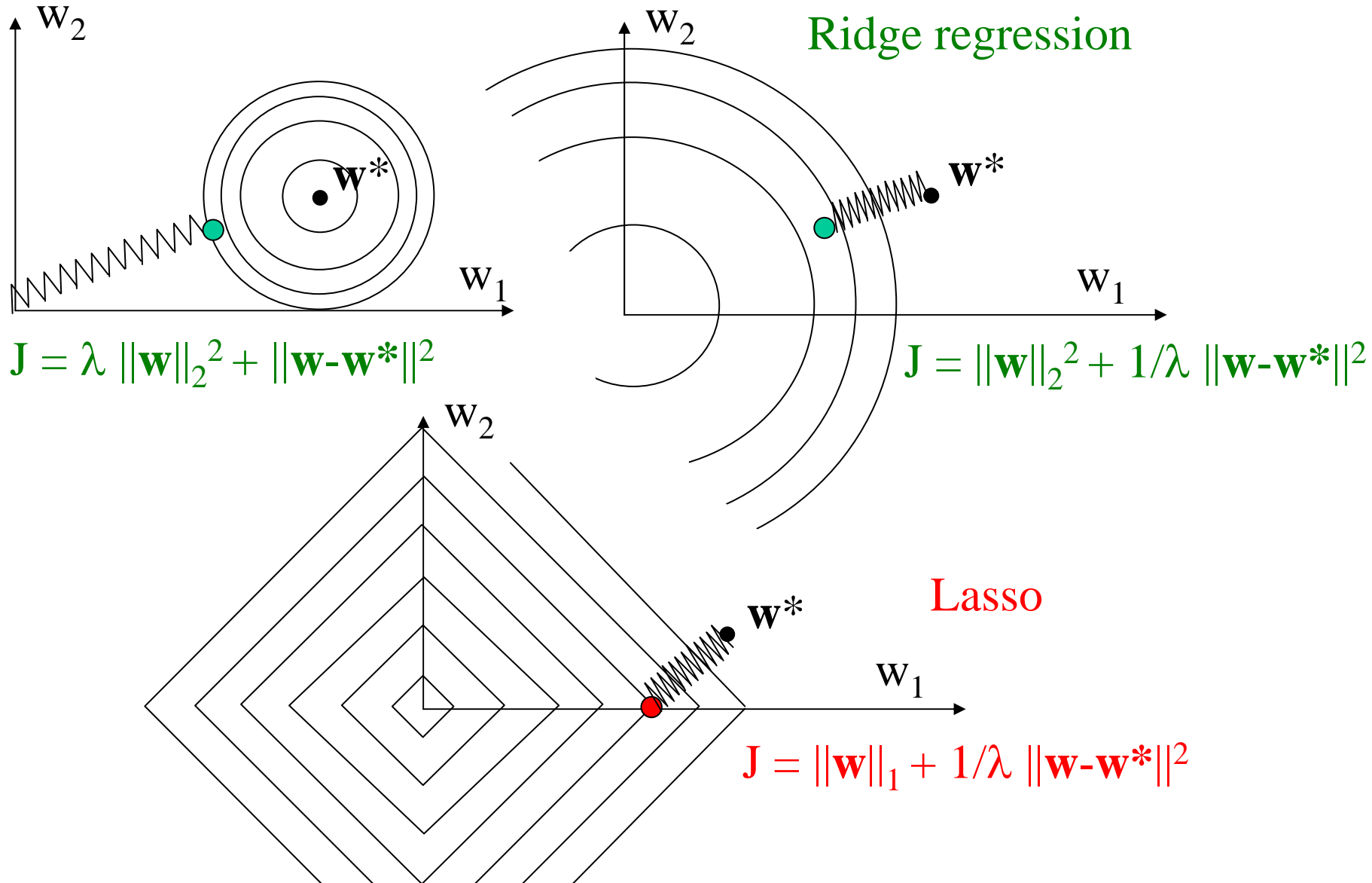
minimum for  $\lambda = 1/2$

- $|w_\lambda|_1 = (1-\lambda) + \lambda$

constant



# *Mechanical interpretation*





# *The $l_0$ SVM*

- Replace the regularizer  $\|w\|^2$  by the  $l_0$  norm  $\sum_{i=1}^n 1_{w_i \neq 0}$
- Further replace  $\sum_{i=1}^n 1_{w_i \neq 0}$  by  $\sum_i \log(\varepsilon + |w_i|)$
- Boils down to the following multiplicative update algorithm:

1. Set  $\sigma = (1, \dots, 1)$
2. Get  $w^*$  solution of an SVM on data set where each input is scaled by  $\sigma$ .
3. Set  $\sigma = w^* \circ \sigma$
4. back to 2.

# *Embedded method - summary*

---

- Embedded methods are a good inspiration to design new feature selection techniques for your own algorithms:
  - Find a functional that represents your prior knowledge about what a good model is.
  - Add the  $\sigma$  weights into the functional and make sure it's either differentiable or you can perform a sensitivity analysis efficiently
  - Optimize alternatively according to  $\alpha$  and  $\sigma$
  - Use early stopping (validation set) or your own stopping criterion to stop and select the subset of features
- Embedded methods are therefore not too far from wrapper techniques and can be extended to multiclass, regression, etc...

# *Book of the NIPS 2003 challenge*

---



## **Feature Extraction, Foundations and Applications**

I. Guyon et al, Eds.

Springer, 2006.

<http://clopinet.com/fextract-book>