Data Science

Thierry Artières and QARMA people

Ecole Centrale Marseille - Option DIGITALE

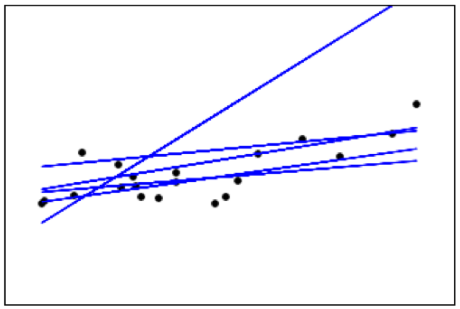
15 octobre 2019

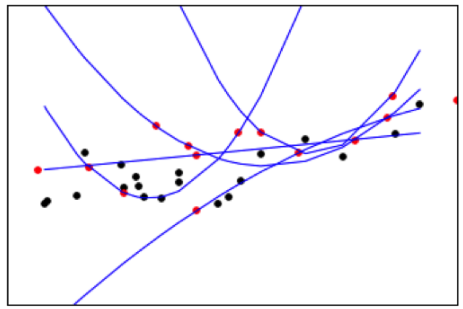
Outline

Bias and Variance

Illustration in regression

- Consider a Train dataset for e.g. regression
- Choose a model family
  - Repeat many times : Take a few training points in the Train set and learn a model
  - Compute the average prediction (over models) for any Test input
  - Compute the variance of the prediction (over models) for any Test input
- The bias measures how close the average predictions are close to true values
- The variance measures how close all predictions are close to this average. It measures how the solution is sensitive to the training set.





Bias Variance decomposition

Notations

- Consider a problem $x \to y = f^*(x) + \epsilon$ where ϵ is a noise (e.g. regression from $\mathbb{R}$ to $\mathbb{R}$). We want to learn $f*$ from data using a model family $\mathcal{F}$.
- Consider a training dataset D of a given size N. We may learn from this dataset and obtain a model noted $f_D \in \mathcal{F}$ that produces predictions $f_D(x)$
- Taking expectations over D we may consider
  - The average prediction for any x, $\mathbb{E}_D[f_D(x)]$
  - Their variance around this average (for any x), $\mathbb{E}_D[(f_D(x) - y)^2]$

Bias Variance decomposition

- The bias is defined as : $\mathbb{E}_{D,x,y}[(\mathbb{E}_D[f_D(x)] - y)^2]$
- The variance is defined as : $\mathbb{E}_{D,x}[(\mathbb{E}_D[f_D(x)] - f_D(x))^2]$
- One may show that : $\mathbb{E}_{D,x,y}[(f_D(x) - y)^2] = Bias + Variance + Var(\epsilon)$

Bias Variance decomposition

In words

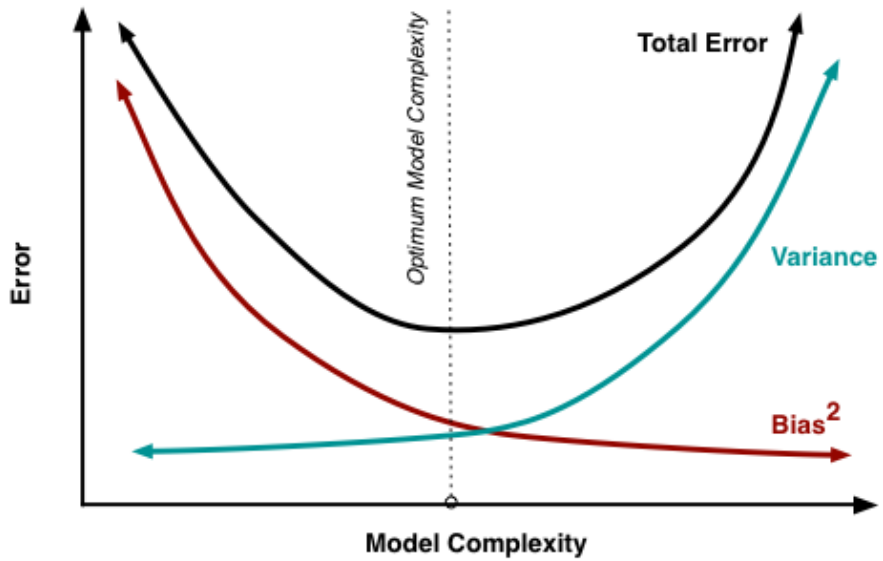
- The expectation $\mathbb{E}_{D,x,y}[(f_D(x) - y)^2]$ measures how well one can expect to learn the right dependency by learning a model from $\mathcal{F}$ from a dataset of a given size.
- The bias measures how well a model may be found in $\mathcal{F}$ that approximates well f^* on average. A non null bias means that you will never get close to the true function within $\mathcal{F}$, which is a too low capacity family.
- The variance measures how well the solution found by learning on a dataset Ddepends on D. It equals 0 for a model family that ignores the data. A high variance means your result will be much dependent on the choice of the training data, the capacity is probably too large and you may get very different results from one try to another.

Proof of the decomposition : see TD

Bias Variance decomposition

Back to the example

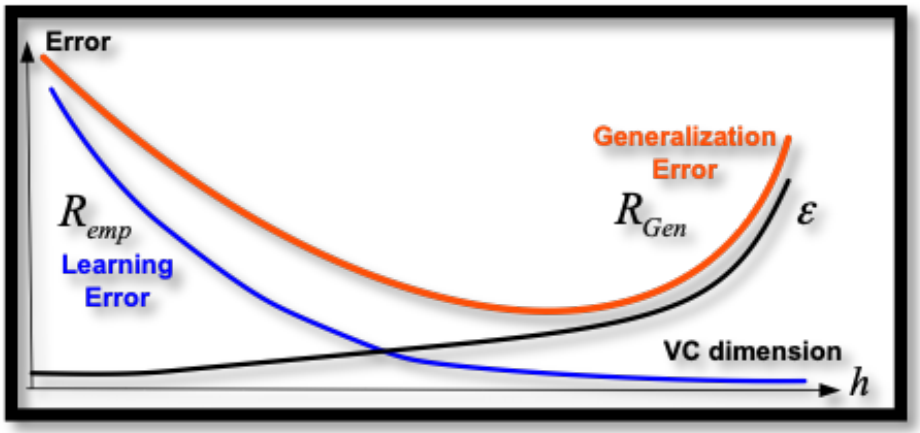
- The higher the capacity the lower the bias : high degree polynomials may learn complicated functions and in average learn the good one.
- The higher the capacity the higher the variance : high degree polynomials may perfectly learn any training set but at the price of doing very weird things elsewhere. High sensitivity to training



Generalization and overfitting

Schematic behaviour

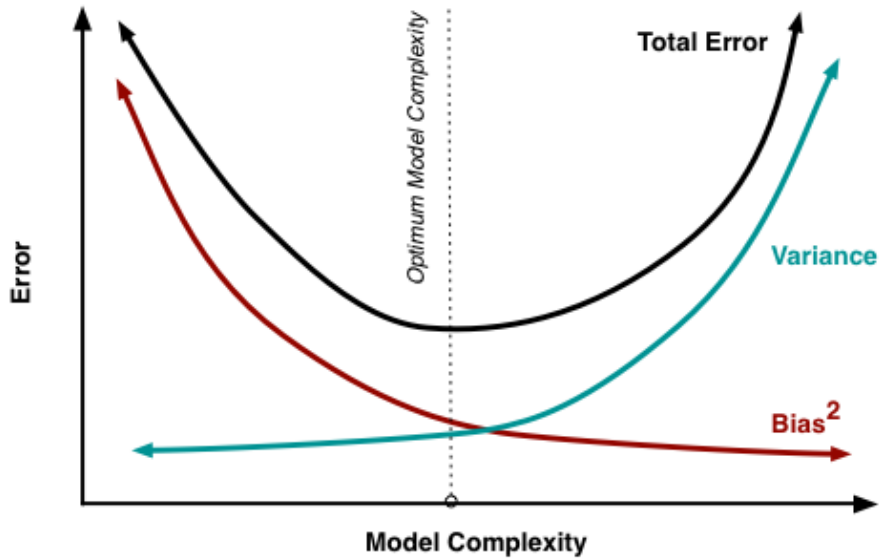
- Training error decreases with the model family capacity (not necessarily reaching 0) : undertraibing regime
- Generalization error decreases with the training error up to some complexity when the error starts increasing : overtraining regime
- Where : model family complexity = measure of the number/complexity of functions that may be implemented within the family
  - For instance : Linear regression models has lower capacity than Quadratic regression models (since the former is included in the latter)



Generalization and overfitting

Schematic behaviour

- Training error decreases with the model family capacity (not necessarily reaching 0) : undertraibing regime
- Generalization error decreases with the training error up to some complexity when the error starts increasing : overtraining regime
- Where : model family complexity = measure of the number/complexity of functions that may be implemented within the family
  - For instance : Linear regression models has lower capacity than Quadratic regression models (since the former is included in the latter)



Capacity

Vapnik Chervonenkis dimension : A measure of the capacity of a model family

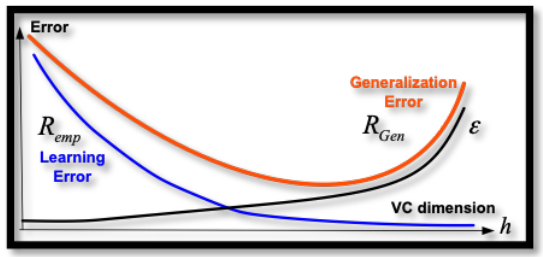
- Preliminary
  - Consider a function a family function $\mathcal{F} = \{f_w, w \in \mathbb{R}^{|w|}\}$ with $f_w(x) \in \{-1, 1\}$
  - Any set of N samples may be labeled in 2^N ways
  - A particular set of N samples is shattered by $\mathcal{F}$ if for any of its labeling there exists a function in $\mathcal{F}$ that implements it
- The VC dimension of $\mathcal{F}$ is the maximum number of training points that can be shattered by $\mathcal{F}$

Generalization bound

- Whatever ν, with probability $1 - \nu$

$$R(w) \leq R_{emp}(w) + \sqrt{\frac{h(\log(2N/h) + 1) - \log(\nu/4)}{N}}$$

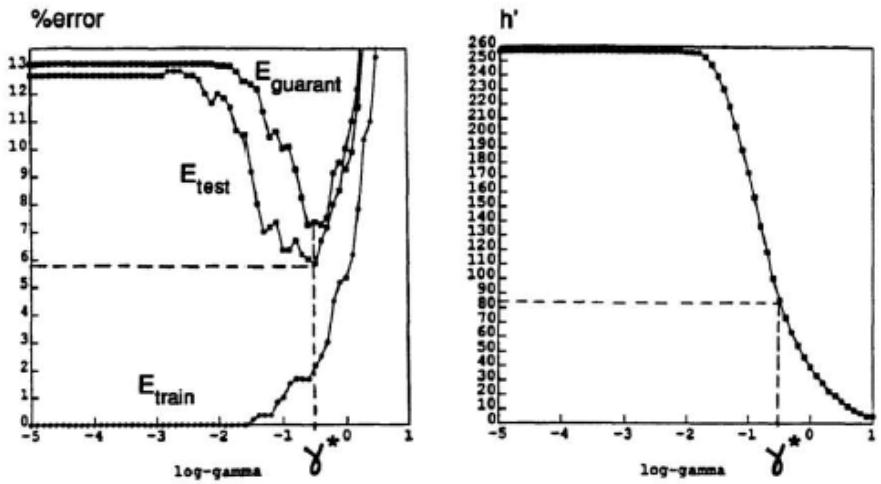
- with
  - N = Number of training samples
  - h = V.C. dimension of the model family



Overtraining

From Vapnik's bound

- Vapnik results show how the excess of generalization error on training error, bounded by above ϵ (with some probability depending on ϵ) evolves with model familiy capacity h
- Actually ϵ depends on h and of the number of training samples
- How to detect overtraining
  - Curves like below
  - Same curves as a function of the number of iteration in the optimization
  - Large gap between training error and generalization error
- Model family complexity examples : Value of regularization constant, Number of hidden cells in a MLP, dimension of the input data for linear classifiers...

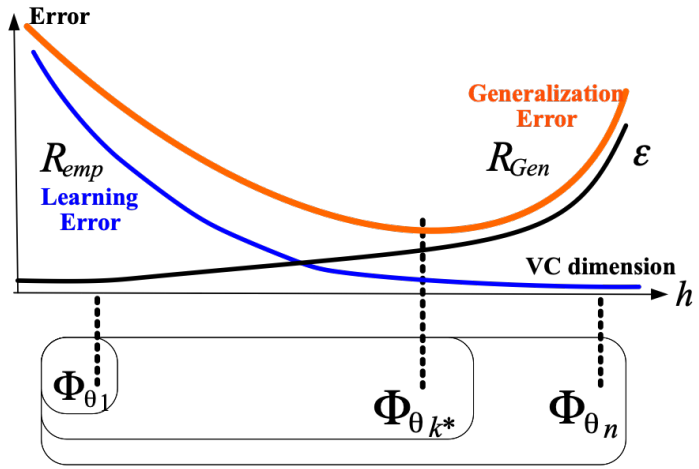


[Vapnik et al., 1992]

Structural Risk Minimization (SRM)

Principle

- Implementation of minimizing the generalization bound = train error + $\epsilon(N, h, ...)$
- If the bound is tight the minimum should be reached more or less at the same optimal model family capacity



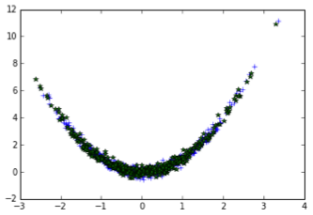
Implementation

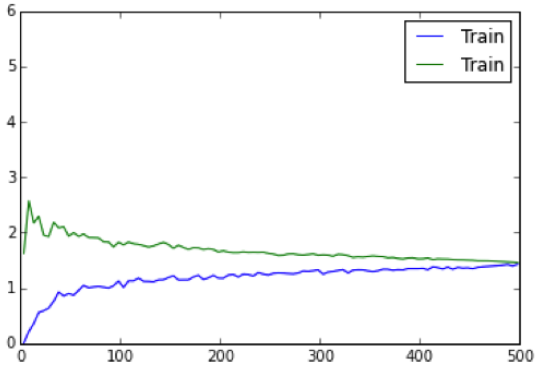
- Consider a cascade of nested families of models (each family includes all previous families) $\Rightarrow$ model families of increasing h. Within each family h is constant.
- Within each family consider a number of models and learn them (modifying hyper-parameters, starting optimization from different random initial solutions...)
- Find the best model within each family : i.e. select the one with lowest bound on generalization error $\Leftrightarrow$ select the one with lowest training error
- Select the model with lowest generalization error amongst all the winer models of each family.

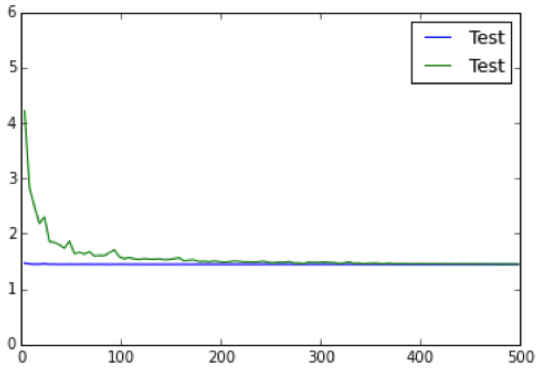
Confidence Intervals

Consequence of previous ideas

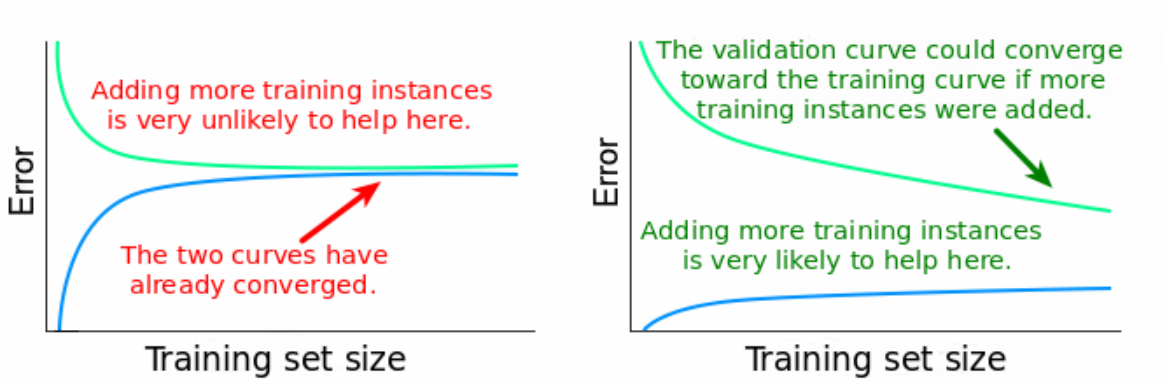
- Illustration on a dataset of train error and test error distribution (y-axis) when learning models as a function of the training set size (x-axis)
  - Dataset (up)
  - Train error distribution (middle) (95% confidence intervals)
  - Test error distribution (bottom) (95% confidence intervals)
- Interesting facts
  - Train and Test intervals width decrease with training set size
  - Train and Test intervals width converge towards same values



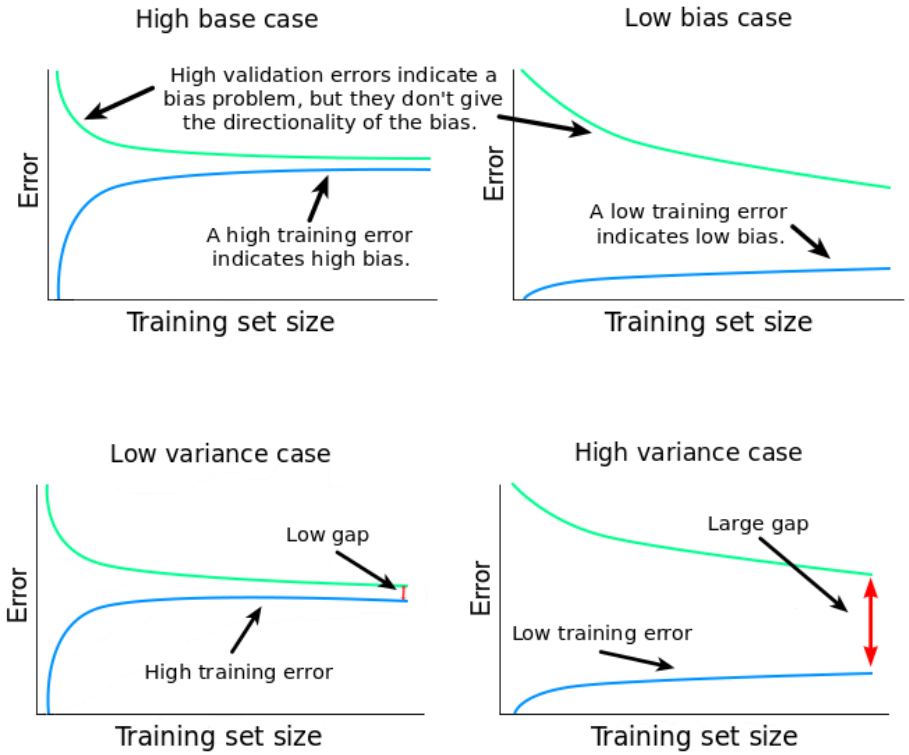




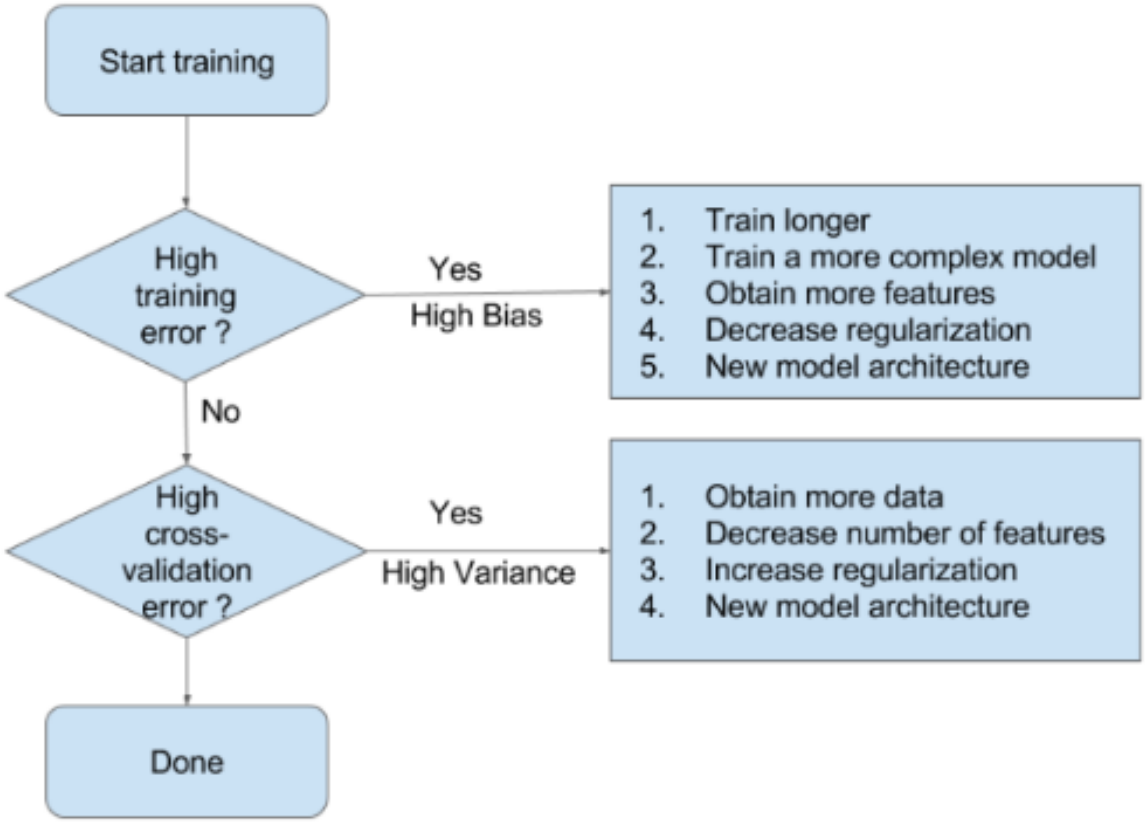
Detecting problems



Detecting problems



Methodology



Regularization

Motivation

- Penalize complexity ($\Leftrightarrow$ reducing variance)
- Perform it while training because setting the right family complexity beforehand is not always easy or even doable (e.g. what's between linear and quadratic regression ?)
- The reciprocal (starting from a simple family and increasing its capacity incrementally during the training is much less popular)

How

- Early stopping
  - Stopping before convergence prevents from learning training data by heart
  - Look at the decision boundary learned by a nonlinear model as learning progresses : It starts as a linear boundary then its s nonlinearly deformed
- Additive term in the cost function : L1, L2, ...
  - Standard tools for estimation problems (e.g. inverse problems etc)
  - Penalizing the norm of the weights makes useless weights go towards 0 $\Rightarrow$

Regularization

Additional cost term

- Objective function : $C(w) = R_{emp}(w) + \lambda\Omega(w)$
- where :
  - $R_{emp}(w)$ is empiric risk
  - $\Omega(w)$ is a penalty term on the complexity of the model with parameters w
  - λ is a positive constant (e.g. set by cross validation) that defines the trade-off between optimizing $R_{emp}(w)$ and $\Omega(w)$

L0 regularizer

- $\Omega(w) = \|w\|_0 =$ Number of non null weights in w
- In a linear model : stands for the number of inputs used (related to Vapnik dimension)
- In a non linear model (e.g. Neural Network) : Number of non null weights related to the capacity of the model
- But : Cannot not be optimized though gradient descent or standard optimization tools used in ML

Regularization

L1 regularizer

- $\Omega(w) = \|w\|_1 =$ Sum of absolute values of weights in w
- In a linear model : the bigger $|w_i|$ the most important the i^{th} feature is
- Gradient : $\frac{\partial C(w)}{\partial w_i} = \frac{\partial R_{emp}(w)}{\partial w_i} + \delta_{sign(w_i)}$ where $\delta_{sign(w_i)}$ is 1 if w_i is positive, -1 otherwise.
- $\Rightarrow$ pushes useless weights towards 0

L2 regularizer

- $\Omega(w) = \|w\|_2 =$ Sum of squarred weights in w
- $\|w\|_2 + Card(w) =$ Relaxation of $\|w\|_0$ (upper bound)
- Smooth and differentiable $\Rightarrow$ may be optimized with gradient
- Gradient : $\frac{\partial C(w)}{\partial w_i} = \frac{\partial R_{emp}(w)}{\partial w_i} + 2 \times w_i$
- $\Rightarrow$ pushes useless weights towards 0 (but slower and slower as it comes close to 0)

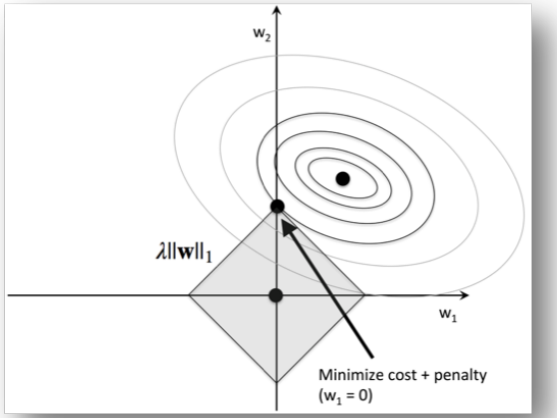
Regularization

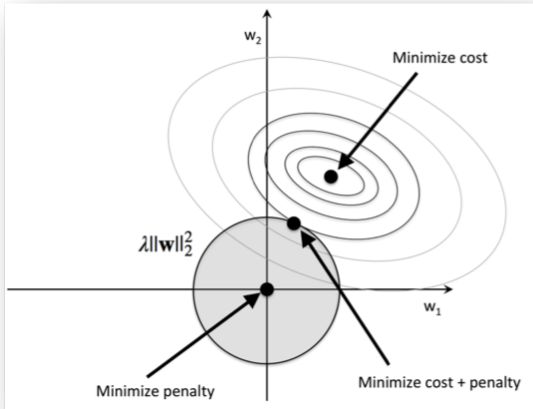
Link between optimization of regularized risk and parameter space constraints

- From Lagrange optimization one may show that the constrained minimization of...

$$\arg\min_{w} C(w) + \lambda \|w\|^2$$

- ... is equivalent to the minimization $\arg\min_w C(w)$ under the constraint that $\|w\|^2 \leq K_\lambda$
- Moreover $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N \Rightarrow K_{\lambda_1} \geq K_{\lambda_2} \geq ... \geq K_{\lambda_N}$



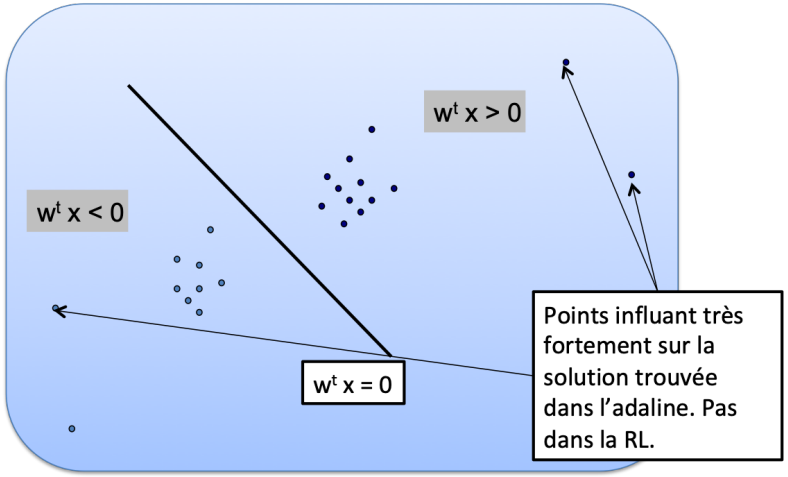


Outline

Logistic Regression (LR)

Problems with linear discriminative models (e.g. perceptron)

Without nonlinearity on the output, the samples that have much influence on reestimating the boundary are the ones that are far from it !



LR's Principle

- Binary classification model
- Model : $P(C = 1|x) = \frac{1}{1+e^{-w^T x}}$
- Maximum likelihood, i.e. : $\hat{w} = \arg\max_w \prod_{i=1}^{N} P(C^i|x^i)$

Logistic Regression (LR)

Maximum Likelihood

- Optimization criterion for probabilistic models

$$L = \log \left(\prod_{i=1}^{N} P(C^i|x^i) \right) = \sum_{i=1}^{N} \log P(C^i|x^i)$$

- with : $P(C^i|x^i) = P(C^i|x^i)^{C^i} \left[1 - P(C^i|x^i)\right]^{1-C^i}$
- Hence

$$L = \sum_{i=1}^{N} C^i \log P(C^i|x^i) + (1 - C^i) \log \left[1 - P(C^i|x^i)\right]$$

- $\Rightarrow$ Gradient descent optimization (Question : what is the expression of this gradient ?)

Outline

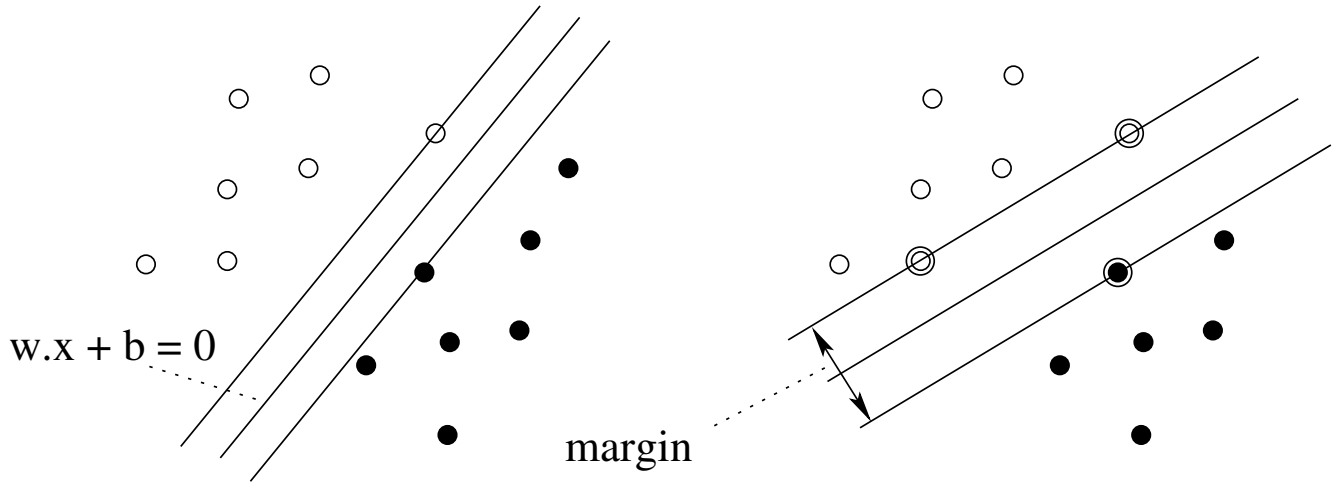
Hard margin linear SVM

Setting

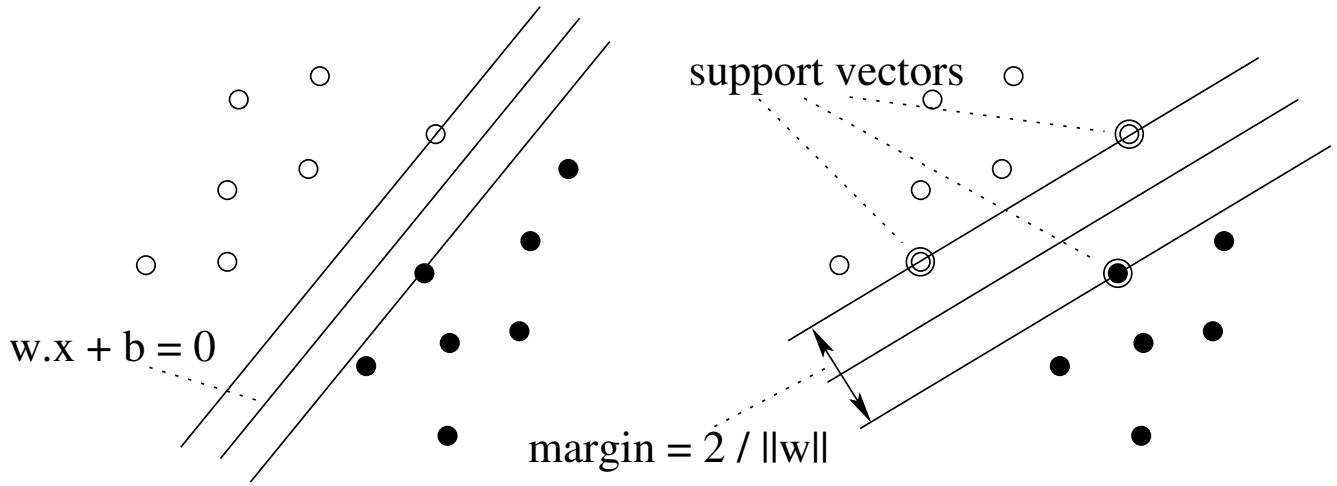
- Input space $\mathcal{X}$ with dot product $\cdot$
- Target space $\mathcal{Y} = \{-1, +1\}$
- $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ linearly separable training set

Optimal hyperplane

Find the separating hyperplane with maximum *margin*, i.e. the one with maximal distance with the closest data



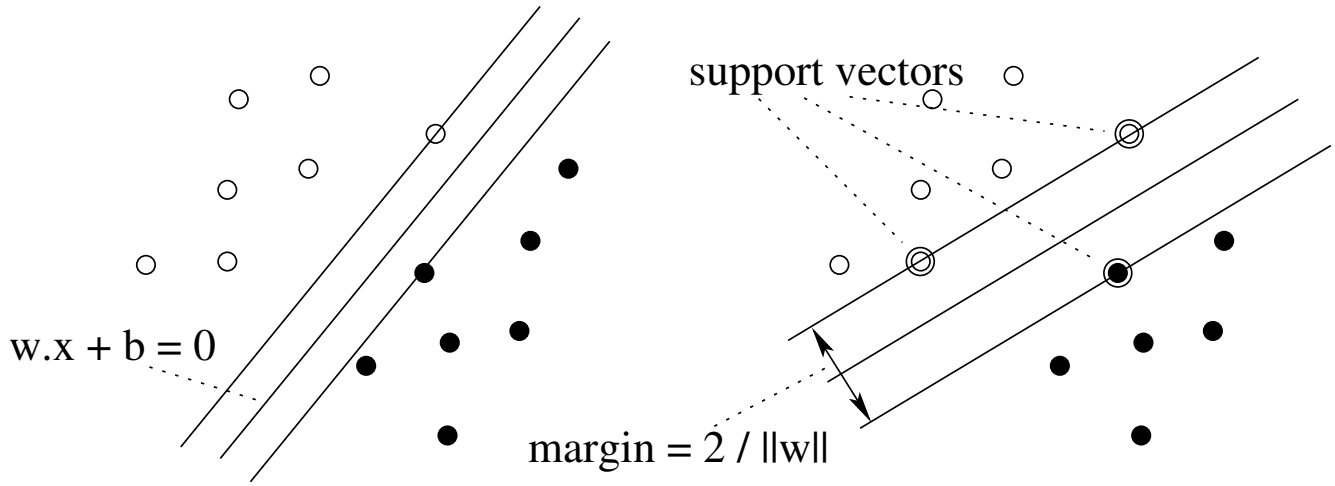
Hard margin linear SVM and convex optimization



Definition (Canonical hyperplane wrt S)

A hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ is canonical wrt S if $\min_{\mathbf{x}_i \in S} |\mathbf{w} \cdot \mathbf{x}_i + b| = 1$.
(High school class :) the margin for a canonical separating hyperplane is equal to $2/\|\mathbf{w}\|$

Hard margin linear SVM and convex optimization

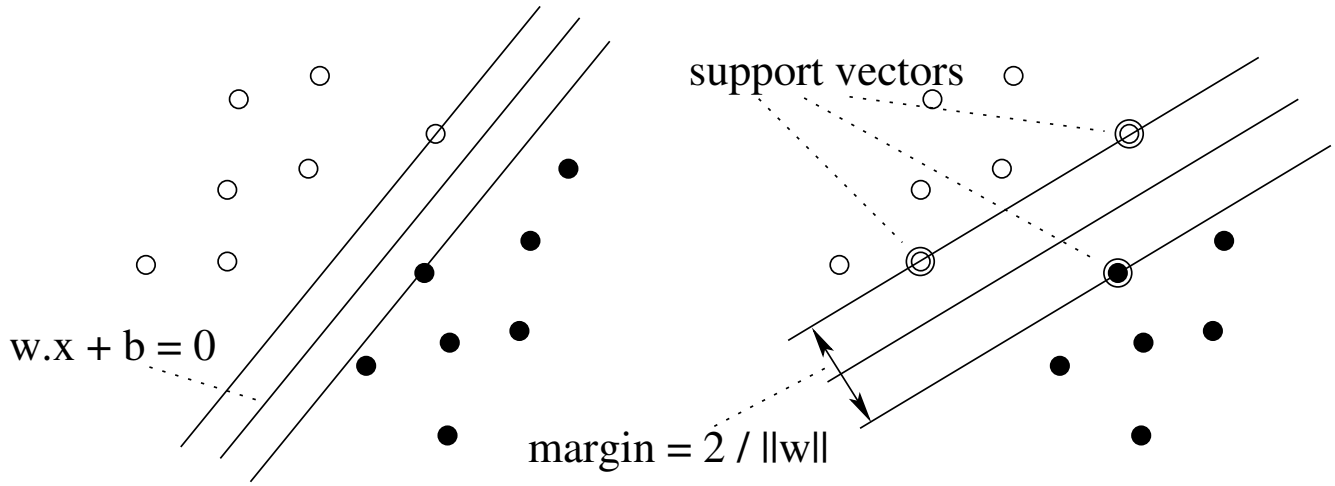


Primal problem (recall that margin=$2/\|\mathbf{w}\|$)

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \begin{cases} \mathbf{w}\cdot\mathbf{x}_i + b \geq +1 & \text{if } y_i = 1 \\ \mathbf{w}\cdot\mathbf{x}_i + b \leq -1 & \text{otherwise} \end{cases}$$

Hard margin linear SVM and convex optimization



Primal problem (recall that margin=$2/\|\mathbf{w}\|$)

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \begin{cases} \mathbf{w}\cdot\mathbf{x}_i + b \geq +1 & \text{if } y_i = 1 \\ \mathbf{w}\cdot\mathbf{x}_i + b \leq -1 & \text{otherwise} \end{cases}$$

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y_i\,[\mathbf{w}\cdot\mathbf{x}_i + b] \geq +1$$

Hard margin linear SVM and convex optimization

Introducing Lagrange multipliers

The solution $\mathbf{w}, b$ can be found be solving the following problem

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha} \geq 0} L(\mathbf{w}, b, \boldsymbol{\alpha})$$

with

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) := \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) - 1\right]$$

The α_i's (≥ 0) are *Lagrange mulipliers*, one per constraint

Another formulation of the constrained optimization problem

- if a constraint is violated, i.e., $y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) - 1 < 0$ for some i, then the value of the objective function $\max_{\boldsymbol{\alpha} \geq 0} L(\mathbf{w}, b, \boldsymbol{\alpha})$ is $+\infty$ (for $\alpha_i \to +\infty$)
  optimal $\mathbf{w}$ and b necessarily verify $y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) - 1 \geq 0$
- also, if $y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) - 1 > 0$ for some i, then $\alpha_i = 0$: again, just look at the function $\max_{\boldsymbol{\alpha} \geq 0} L(\mathbf{w}, b, \boldsymbol{\alpha}) \to$ at the solution $\alpha_i \left[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) - 1\right] = 0$ (KKT conditions)

Hard margin linear SVM and convex optimization

Switching the min and the max

A theorem of convex optimization (see, e.g. **?**) exploiting the fact that L is convex wrt $\mathbf{w}$ and b and concave wrt $\boldsymbol{\alpha}$ gives

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha} \geq 0} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha} \geq 0} \min_{\mathbf{w},b} L(\mathbf{w}, b, \boldsymbol{\alpha})$$

with the same optimal points

Making the gradient be $\mathbf{0}$

$$\min_{\mathbf{w},b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) - 1\right]$$

is an unconstrained strictly convex (and coercive) optimization problem. It suffices to have the gradient of the functional to be $\mathbf{0}$ to get the solution.

- $\nabla_{\mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = \mathbf{0} \Rightarrow \quad \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
- $\nabla_b L(\mathbf{w}, b, \boldsymbol{\alpha}) = \sum_i y_i \alpha_i = 0 \Rightarrow \quad \sum_i y_i \alpha_i = 0$

Hard margin linear SVM and convex optimization

Switching the min and the max

A theorem of convex optimization (see, e.g. **?**) exploiting the fact that L is convex wrt $\mathbf{w}$ and b and concave wrt $\boldsymbol{\alpha}$ gives

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha} \geq 0} L(\mathbf{w}, b, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha} \geq 0} \min_{\mathbf{w},b} L(\mathbf{w}, b, \boldsymbol{\alpha})$$

with the same optimal points

A dual quadratic program

Plugging in the value of $\mathbf{w}$ and the constraint provides the following problem

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \boldsymbol{\alpha} \geq \mathbf{0}$$

On the dual formulation and support vectors

The QP

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \boldsymbol{\alpha} \geq \mathbf{0}$$

Remark

- As many optimization variables as the number of training data
- Convex quadratic program
- Only dot products appear (the kernel trick will strike soon)
- b^* is found through the KKT conditions
- $\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i$,

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i \cdot \mathbf{x} + b^*$$

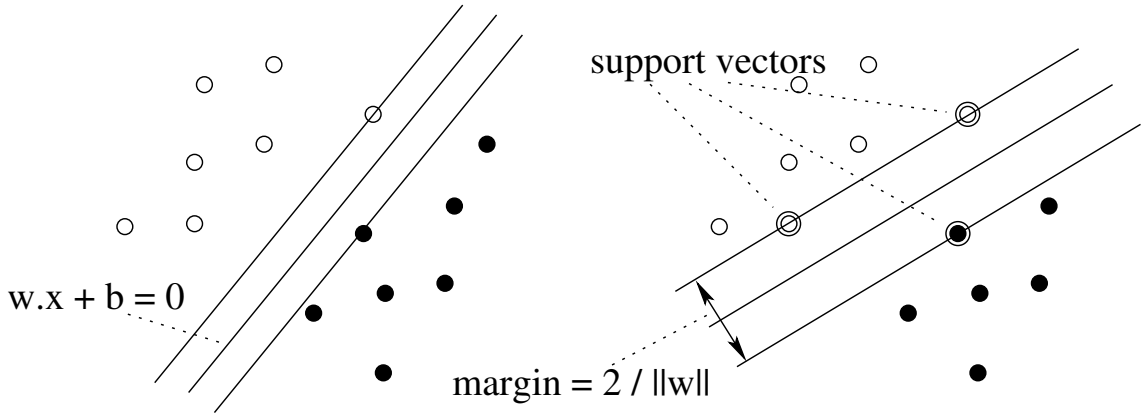
On the dual formulation and support vectors

The QP

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \boldsymbol{\alpha} \geq \mathbf{0}$$

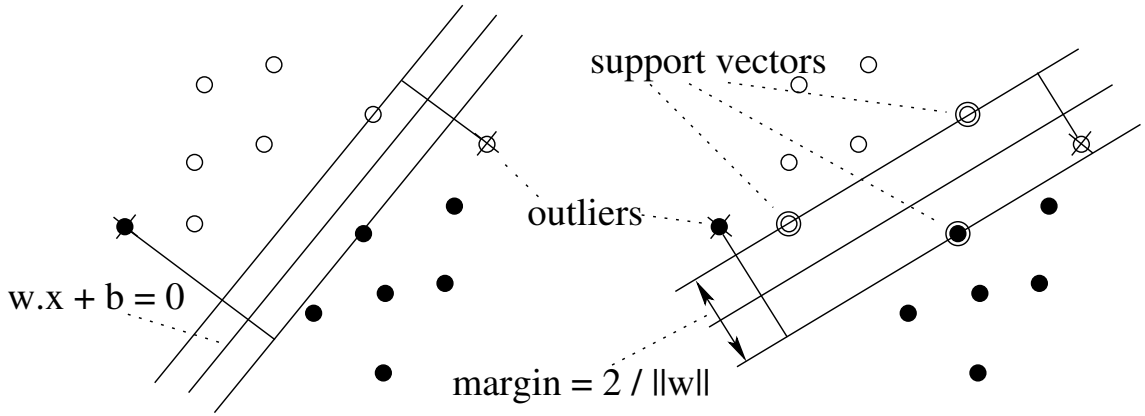
Support vectors



The support vectors are those points for which $\alpha_i^* > 0$, they "support" the margin.

Soft margin linear SVM

Presence of outliers



Slack variables - 1–norm

$$\min_{\mathbf{w},b,\boldsymbol{\xi}\geq 0} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i\left[\mathbf{w}\cdot\mathbf{x}_i + b\right] \geq 1-\xi_i$$

with $C > 0$

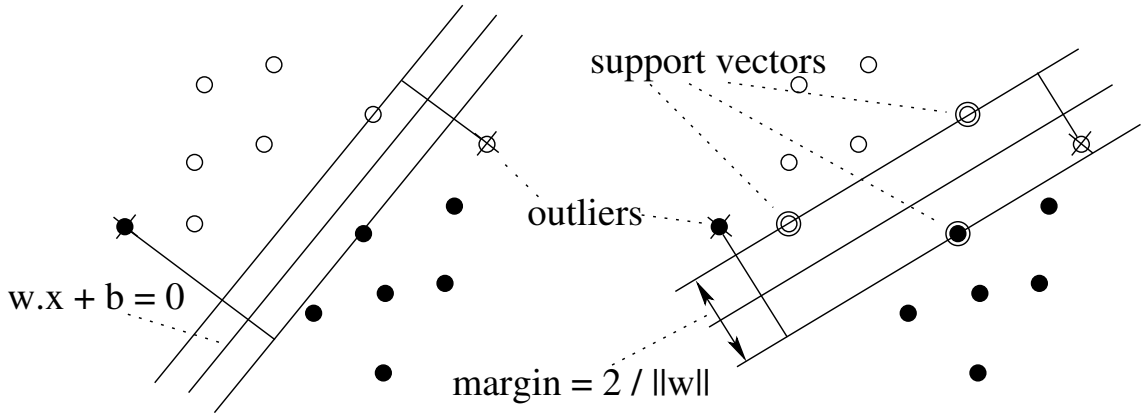
$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } \mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{C}$$

(using the same machinery as before)

Soft margin linear SVM

Presence of outliers



Slack variables - 2–norm

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i^2$$

$$\text{s.t.} \quad y_i\left[\mathbf{w}\cdot\mathbf{x}_i + b\right] \geq 1 - \xi_i$$

with $C > 0$

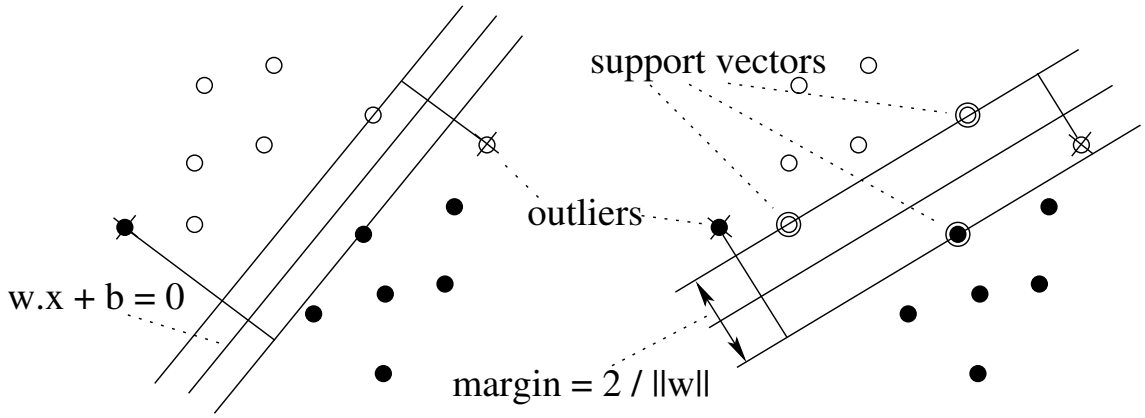
$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \left(\mathbf{x}_i\cdot\mathbf{x}_j + \frac{\delta_{ij}}{C}\right)$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0 \text{ and } \boldsymbol{\alpha} \geq \mathbf{0}$$

(using the same machinery as before)

Soft margin linear SVM

Presence of outliers



Unconstrained 1-norm primal form

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} |1 - y_i\,(\mathbf{w}\cdot\mathbf{x}_i + b)|_+\,, \qquad \text{where } |\theta|_+ = \max(\theta, 0)$$

convex, non-differentiable

Unconstrained 2-norm primal form

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} |1 - y_i\,(\mathbf{w}\cdot\mathbf{x}_i + b)|_+^2$$

convex, differentiable

Nonlinear SVM : tricking SVMs with kernels

1-norm and 2-norm soft-margin SVM

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \qquad \max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \left(k(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C} \right)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{C} \qquad \text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \boldsymbol{\alpha} \geq \mathbf{0}$$

Classifier output $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i k(\mathbf{x}_i, \mathbf{x}) + b^*$

Remark

- The sizes of the problems scale with the number of data
- Efficient methods to solve these problems (that exploit the sparsity of the solution)
- k and C are two hyperparameters that need be chosen adequately

Partial conclusion

SVM

- margins
- kernels
- convex optimization problem, duality
- effectiveness
- ...

Not covered here

- algorithmic details (chunking, decomposition, active set, stochastic gradient)
- learning in the primal (see practical session)
- theory (be patient)
- ...

The kernel trick

Nonlinearly separable dataset $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

Idea to learn a nonlinear classifier

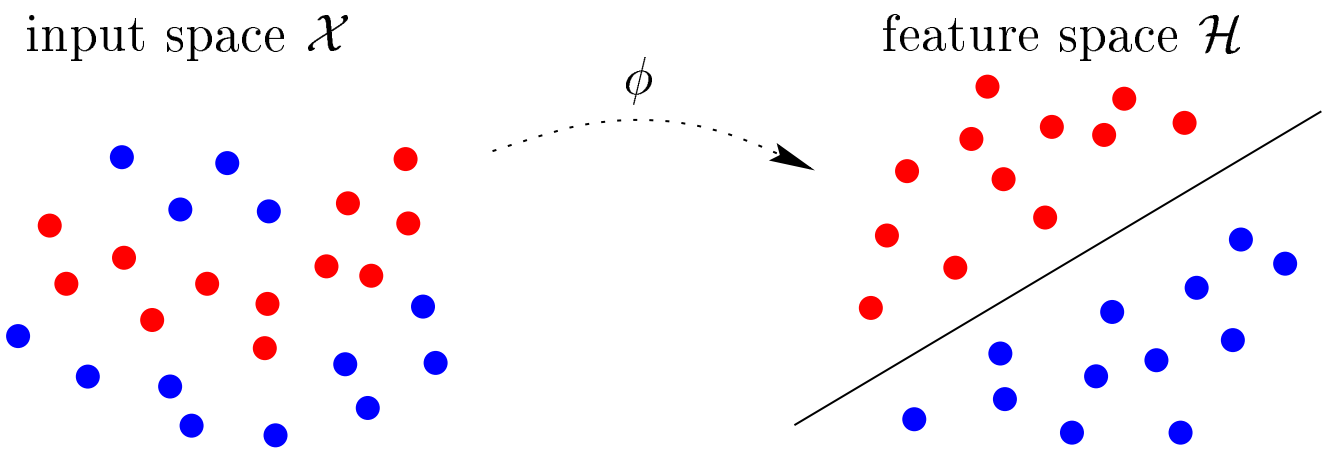
- choose a (nonlinear) mapping ϕ

$$\begin{array}{rccc} \phi: & \mathcal{X} & \to & \mathcal{H} \\ & \mathbf{x} & \mapsto & \phi(\mathbf{x}) \end{array}$$

  where $\mathcal{H}$ is an inner product space (inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$), called *feature space*
- find a linear classifier (i.e. a separating hyperplane) in $\mathcal{H}$ to classify

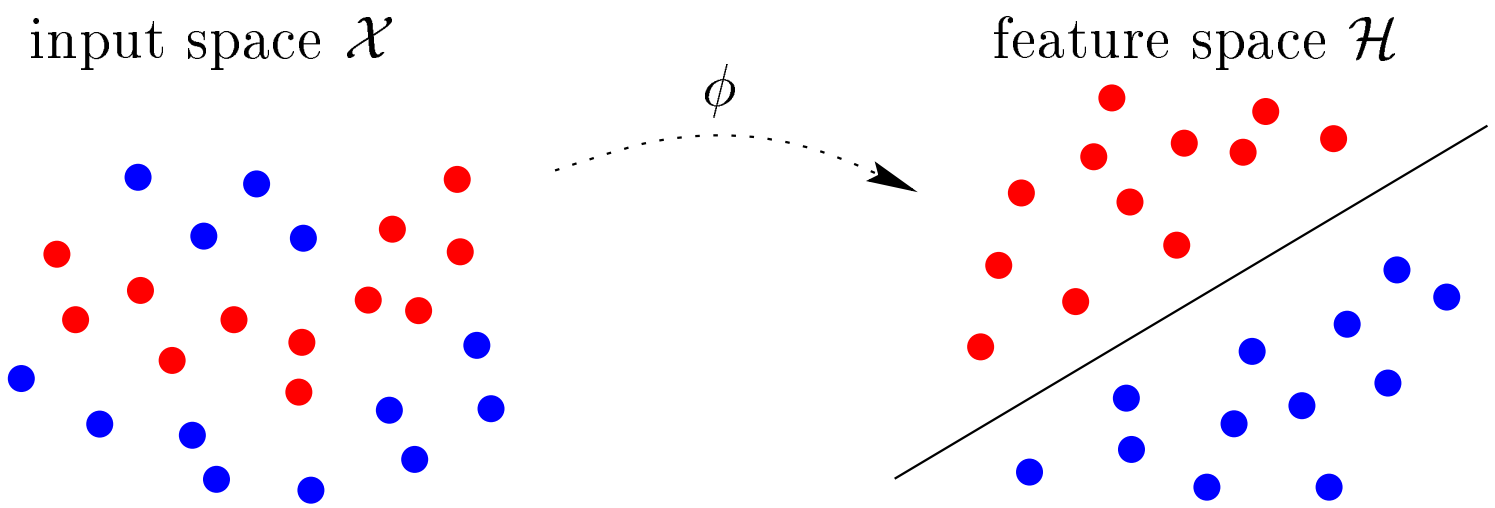
$$\{(\phi(\mathbf{x}_1), y_1), \ldots, (\phi(\mathbf{x}_n), y_n)\}$$

- to classify test point $\mathbf{x}$, consider $\phi(\mathbf{x})$



The kernel trick

Linearly classifying in *feature space*



Taking the previous linear algorithm and implementing it in $\mathcal{H}$:

$$h(\mathbf{x}) = \sum_{i=1,\ldots,n} \alpha_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle_{\mathcal{H}} + b$$

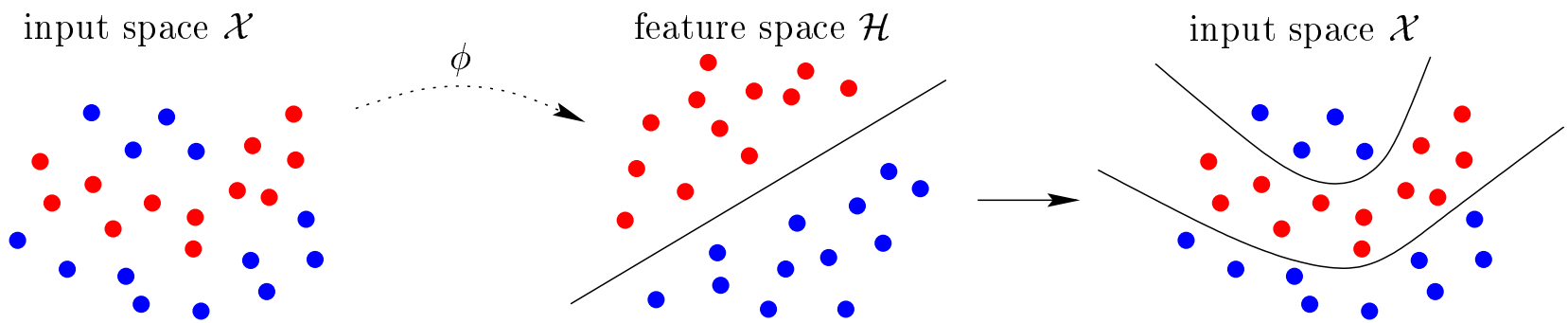
The kernel trick

Mercer Kernels

The kernel trick can be applied if there is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that :
$k(\mathbf{u}, \mathbf{v}) = \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle_{\mathcal{H}}$
If so, all occurrences of $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle_{\mathcal{H}}$ are syntactically replaced by $k(\mathbf{x}_i, \mathbf{x})$

- Keypoint : emphasis is sometimes (often) more on k than on ϕ
- Kernels must verify Mercer's property to be valid kernels
  - ensures that there exist a space $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \to \mathcal{H}$ such that $k(\mathbf{u}, \mathbf{v}) = \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle_{\mathcal{H}}$
  - however non valid kernels have been used with success
  - and, research is in progress on using non semi-definite kernels
- k might be viewed as a similarity measure

The kernel trick



no separating hyperplane | one possible separating hyperplane | separating surface

Kernel trick recipe

- choose a linear classification algorithm (expr. in terms $\langle \cdot, \cdot \rangle$)
- replace all occurrences of $\langle \cdot, \cdot \rangle$ by a kernel $k(\cdot, \cdot)$

Obtained classifier :

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1,\ldots,n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b \right)$$

Common kernels

Gaussian/RBF kernel

- $k(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right), \quad \sigma^2 > 0$
- the corresponding $\mathcal{H}$ is of infinite dimension

Polynomial kernel

- $k(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u}, \mathbf{v} \rangle + c)^d, \quad c \in \mathbb{R}, d \in \mathbb{N}$
- a corresponding analytic ϕ may be constructed (see below)

Tangent kernel (it is *not* a Mercer kernel)

- $k(\mathbf{u}, \mathbf{v}) = \tanh(a\langle \mathbf{u}, \mathbf{v} \rangle + c), \quad a, c \in \mathbb{R}$

Common kernels

Polynomial kernel $k = \langle \mathbf{u}, \mathbf{v} \rangle^2_{\mathbb{R}^2}$

Polynomial kernel with $c = 0$ and $d = 2$, defined on $\mathbb{R}^2 \times \mathbb{R}^2$

- Consider the mapping :

$$\begin{array}{rccc} \phi : & \mathbb{R}^2 & \to & \mathbb{R}^3 \\ & \mathbf{x} = [x_1, x_2]^\top & \mapsto & \phi(\mathbf{x}) = \left[x_1^2, \sqrt{2}x_1x_2, x_2^2\right]^\top \end{array}$$

- We have, for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$:

$$\begin{aligned} \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle_{\mathbb{R}^3} &= \langle [u_1^2, \sqrt{2}u_1u_2, u_2^2]^\top, [v_1^2, \sqrt{2}v_1v_2, v_2^2]^\top \rangle \\ &= (u_1v_1 + u_2v_2)^2 \\ &= \langle \mathbf{u}, \mathbf{v} \rangle^2_{\mathbb{R}^2} \\ &= k(\mathbf{u}, \mathbf{v}) \end{aligned}$$

(Kernel) Gram matrices

Gram matrix

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel. For a set of patterns $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$

$$K_S = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_1, \mathbf{x}_2) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_n) \\ & \cdots & \cdots & \\ k(\mathbf{x}_1, \mathbf{x}_n) & k(\mathbf{x}_2, \mathbf{x}_n) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

is the Gram matrix of k with respect to S

Property (Mercer's property)

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric function.
k is a Mercer kernel $\Leftrightarrow$

$$\forall S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \mathcal{X}, \mathbf{v} K_S \mathbf{v} \geq 0, \forall \mathbf{v} \in \mathbb{R}^n$$

(and, therefore there exists ϕ such that $k(\mathbf{u}, \mathbf{v}) = \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle$)

Remark

This means that for any Mercer kernel k and any set of patterns S, the Gram matrix K_S has only nonnegative eigenvalues

(Kernel) Gram matrices

Gram matrix

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel. For a set of patterns $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$

$$K_S = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_1, \mathbf{x}_2) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_n) \\ & \cdots & \cdots & \\ k(\mathbf{x}_1, \mathbf{x}_n) & k(\mathbf{x}_2, \mathbf{x}_n) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

is the Gram matrix of k with respect to S

Property (Mercer's property)

Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric function.
k is a Mercer kernel $\Leftrightarrow$

$$\forall S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \mathcal{X}, \mathbf{v} K_S \mathbf{v} \geq 0, \forall \mathbf{v} \in \mathbb{R}^n$$

(and, therefore there exists ϕ such that $k(\mathbf{u}, \mathbf{v}) = \langle \phi(\mathbf{u}), \phi(\mathbf{v}) \rangle$)

Property (Structural results)

k_1 and k_2 being Mercer kernels, we have

- k_1^p, $p \in \mathbb{N}$ *is a Mercer kernel*
- $\lambda k_1 + \gamma k_2$, $\lambda, \gamma > 0$ *is a Mercer kernel*
- $k_1 k_2$ *is a Mercer kernel*

Partial conclusion

Kernel trick

- Pick your favorite linear classifier and *kernelize it* : you have a nonlinear classifier
- Mercer's condition on kernels (positive definite kernels)
- A few, well-known existing kernels (RBF kernels, polynomial kernels)
- Dealing with structured data is easy as soon as a kernel is available

Questions

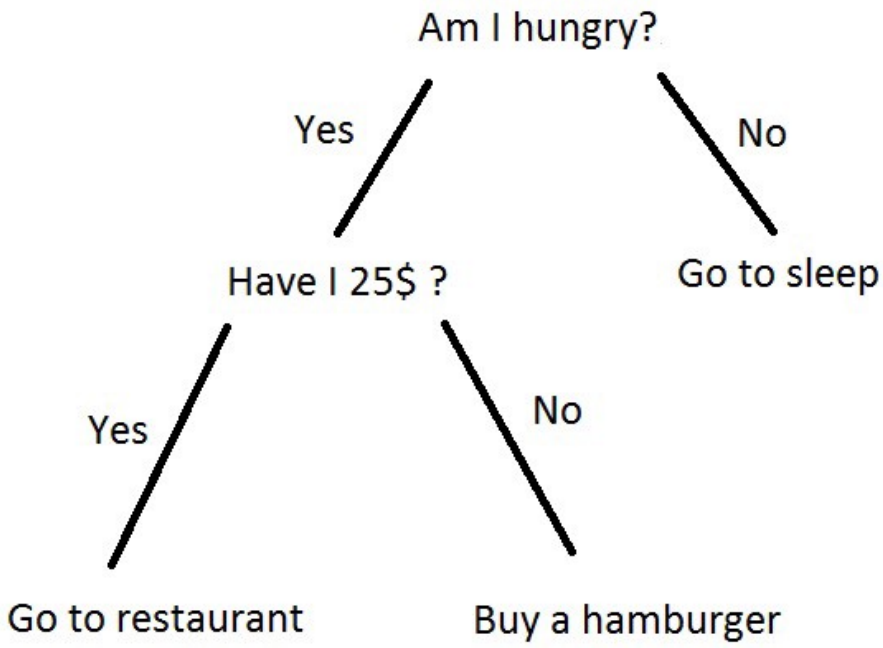
- Choosing the right kernel ?
- Combining kernels ?
- Building kernels for structured data
- Kernel and large scale learning
- Kernelizable algorithms ? (So many : Perceptron, PCA...)

Outline

Decision Trees

An old classification model !

- Lots of variants (CART, ID3...)
- Basically the idea of a decision tree is to classify a sample by examining sequentially specific features of the sample and progressing in the tree down to a leaf, where leaves are labeled with classes.



Decision Trees

An old classification model !

- Lots of variants (CART, ID3...)
- Basically the idea of a decision tree is to classify a sample by examining sequentially specific features of the sample and progressing in the tree down to a leaf, where leaves are labeled with classes.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Trees

Learning a DT

- Incremental learning
- Starting from the root, find the single best feature to separate data of different labels in the leaves according to this feature's values
- Proceed recursively
- Many measures of interest of a feature investigated for feature selection (entropy, information gain...)

Outline

Motivations

No-free-lunch theorem

No machine learning algorithm better than others ion all settings

Core idea : combine multiple models (*base learners*)

- various algorithms
- various sub sampling of the data
- various hyperparameter settings

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8642	9.27	2009-07-15 14:53:22
14	Gravity	0.8643	9.26	2009-04-22 18:31:32
15	Ces	0.8651	9.18	2009-06-21 19:24:53
Progress Prize 2007 - RMSE = 0.8723 - Winning Team: Korbell				
Cinematch score - RMSE = 0.9525				

“Our final solution (RMSE=0.8712) consists of blending 107 individual results. “

“Predictive accuracy is substantially improved when blending multiple predictors. Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique. “

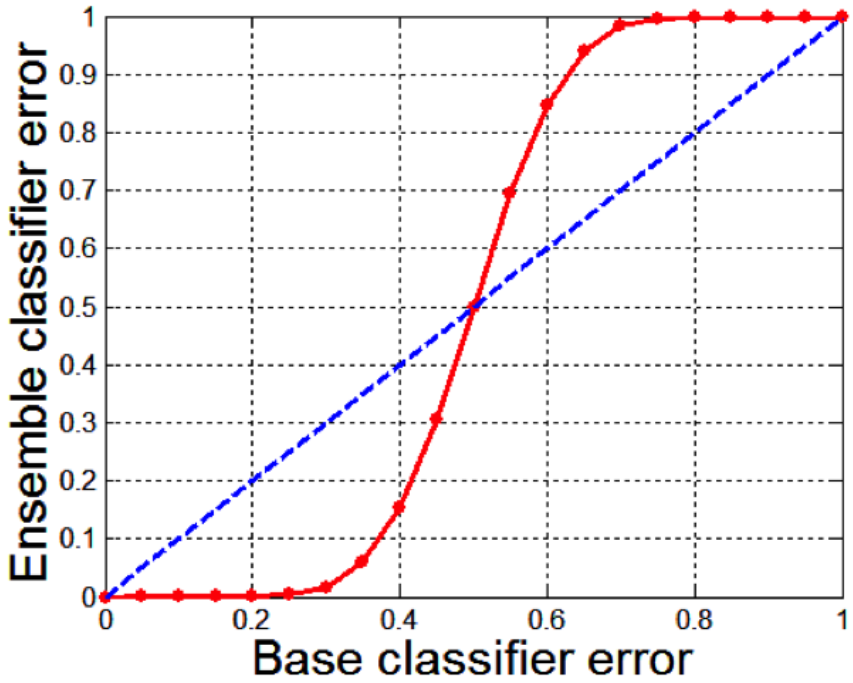
Motivations

No-free-lunch theorem

No machine learning algorithm better than others ion all settings

Core idea : combine multiple models (*base learners*)

- various algorithms
- various sub sampling of the data
- various hyperparameter settings



Motivations

Main approaches

- Sequential methods : Boosting, Stacking
- Parallel methods : Bagging, Majority voting...

Ideas

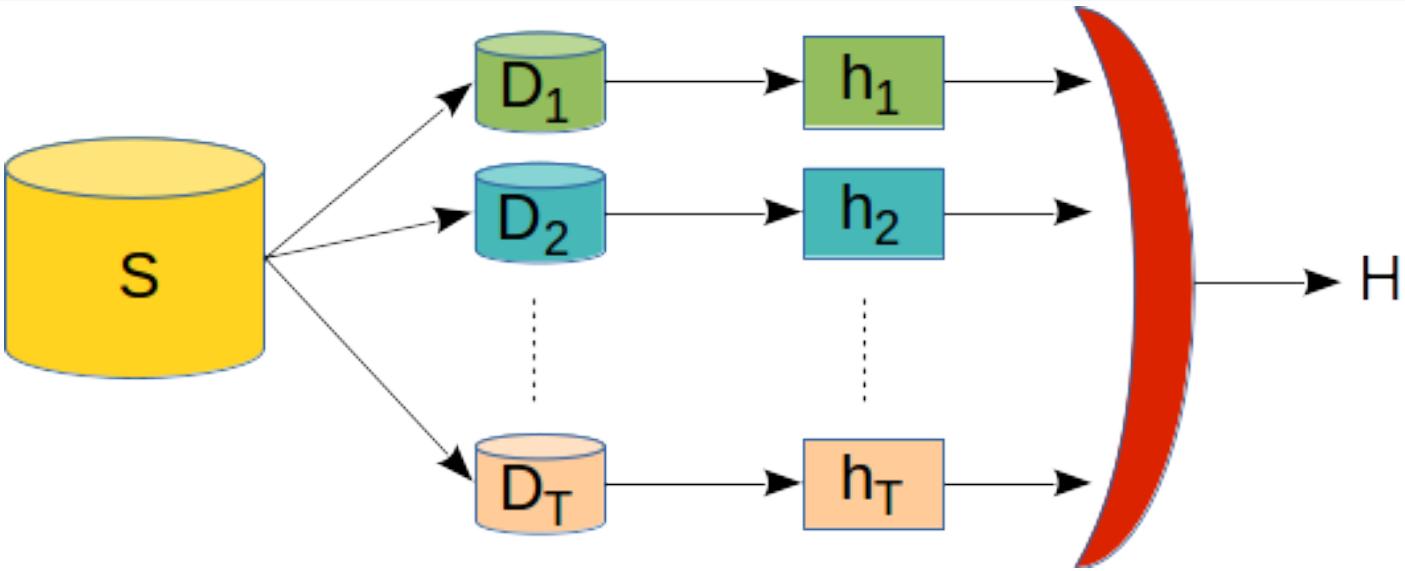
- Sequential
  - Learn a classifier and exploit its behaviour to build another one (e.g. focusing on difficult samples) etc
- Parallel
  - Learn many classifiers enforcing diversity
  - Make them vote and expect individual errors will be corrected by the number

Bagging = Boostrap AGGregatING

Combine weak models

Each of the models is learned on a subsampling of training data set S

- Ramdomly sample T subsets (with replacement)
- On each S_t, learn an hypothesis h_t with whatever classifier ou like
- Combine hypotheses through majority voting (sum, wieighted sum...)



Bagging : details

Why is is useful

- Every hypothesis is unstable (large bias and low variance), but the combination is stable (uncorrelated errors)
- Works better with Decision Trees or NNs as base learners.
- Works less with lore stable learners (SVM, k-NN...)

In practice

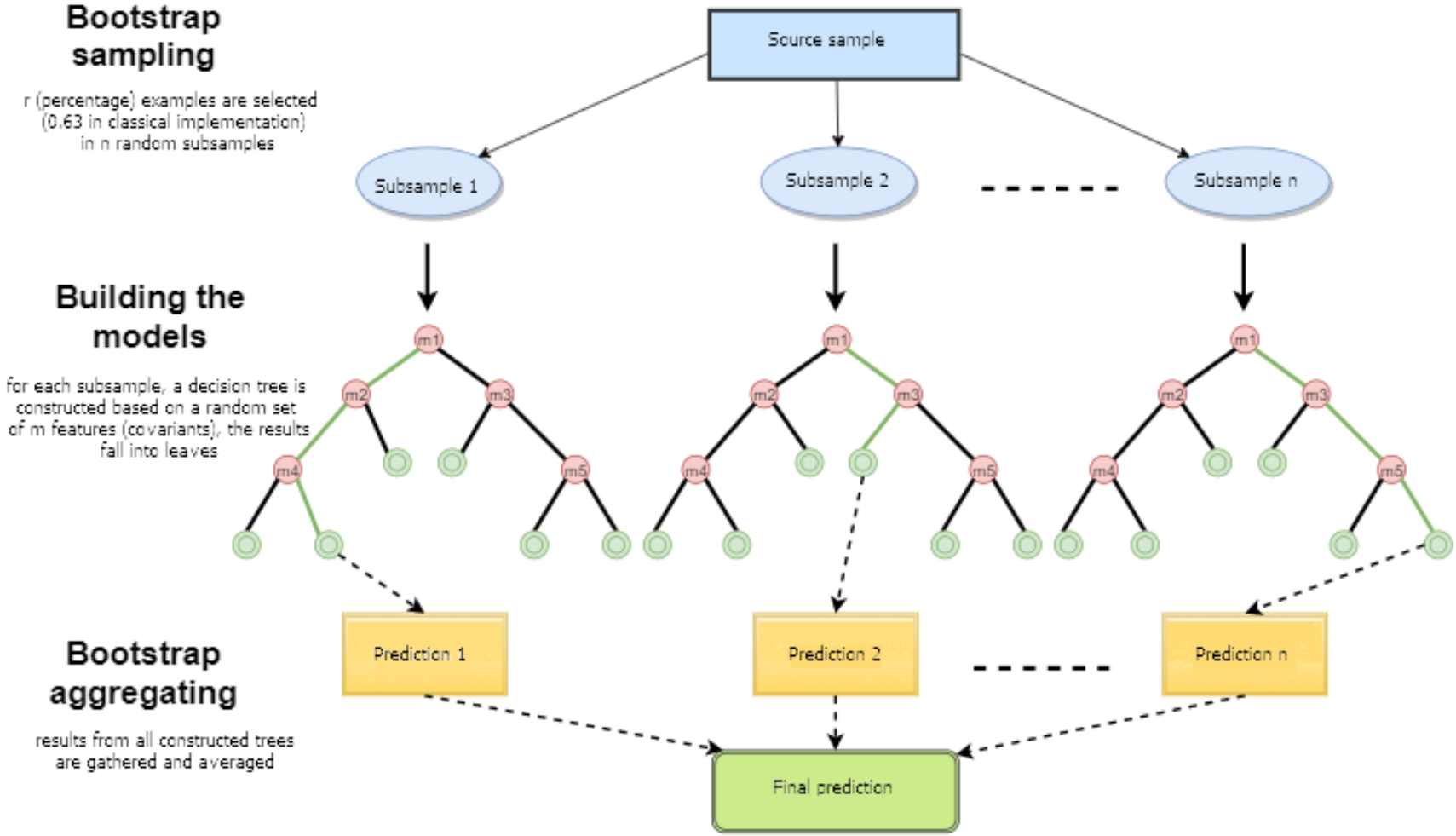
- If subset size = data set size $\Rightarrow$ S_t includes $\approx 60\%$ of unique samples from S
- Many variants
- Bagging reduces variance (while eventually slightly increasing bias) because averaging reduces variance ! Hence mostly interesting for high variance models

Random forests : Bagging and Trees, and more

Principle : uncorrelate decisions trees in the forest to uncorrelate their errors

- Double sampling strategy to bring more diversity
- Each tree is learned on a subset sampled with replacement, with same size as T
- Each tree is learned with a subset of k features which is randomly selected from the d feature of the original representation space of the data $\mathcal{X}$ (e.g. $k = \sqrt{d}$)
- Final prediction by voting

Schema of Random Forests

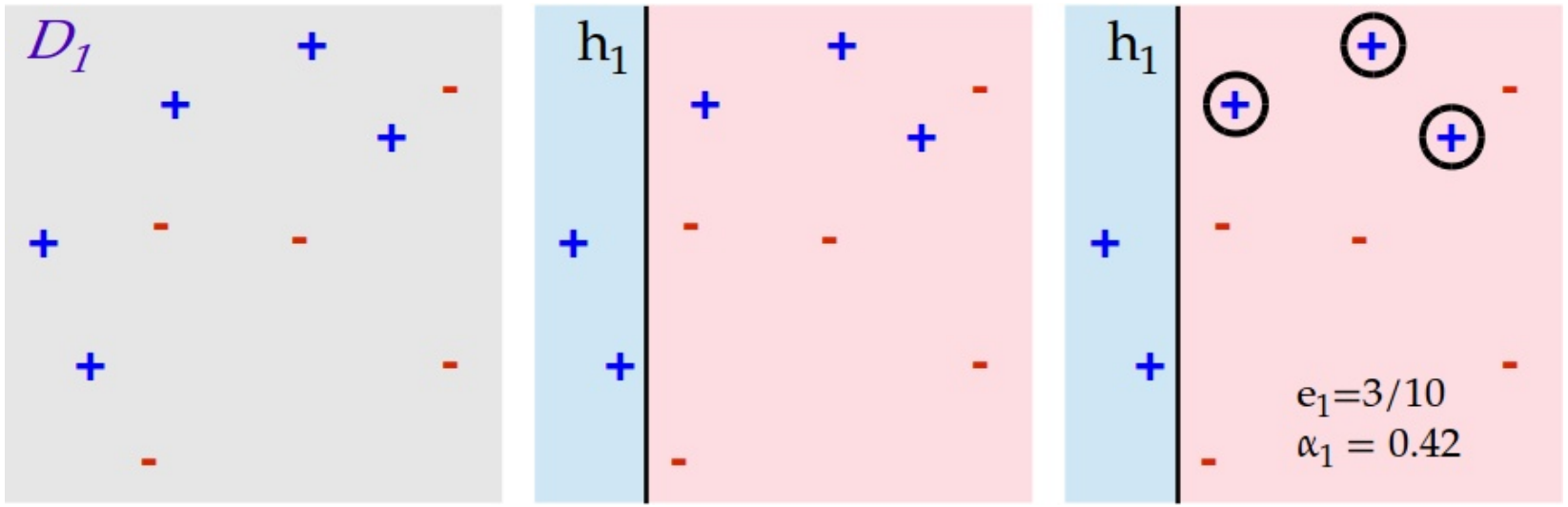


Sequential method : Boosting : AdaBoost [Freund *et.al*, 1996]

Principle

- Learn a decision rule from a weighted vote of weak learners, whose are supposed to be expert on some areas of the input space.
- Iterative algorithm that learn a classifier at each iteration on original data samples but with weights. The distribution is change along iterations to enforce the next learners to focus on difficult samples
- Proof : see C. Capponi slides M1IAAA

Illustration

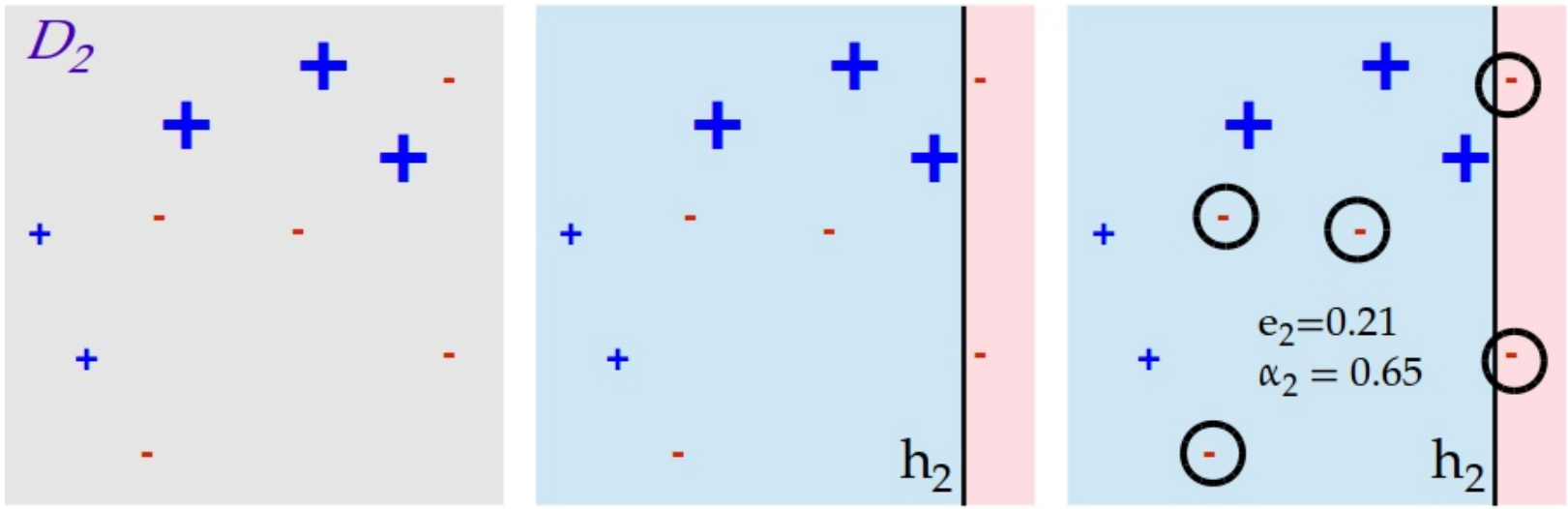


Sequential method : Boosting : AdaBoost [Freund *et.al*, 1996]

Principle

- Learn a decision rule from a weighted vote of weak learners, whose are supposed to be expert on some areas of the input space.
- Iterative algorithm that learn a classifier at each iteration on original data samples but with weights. The distribution is change along iterations to enforce the next learners to focus on difficult samples
- Proof : see C. Capponi slides M1IAAA

Illustration

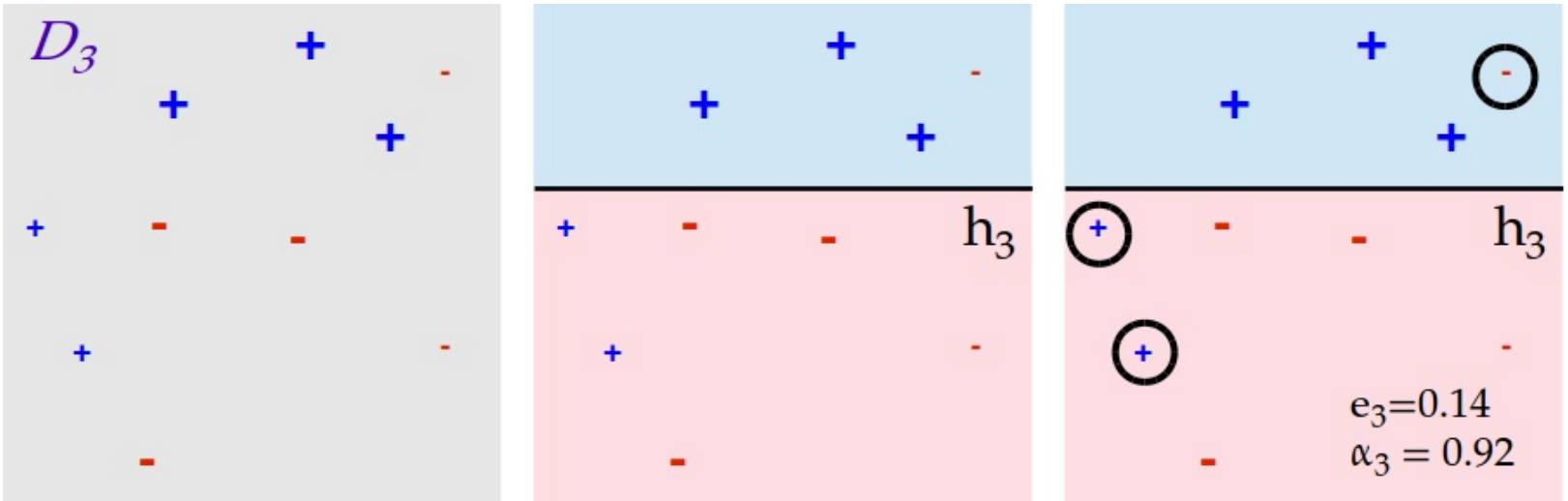


Sequential method : Boosting : AdaBoost [Freund *et.al*, 1996]

Principle

- Learn a decision rule from a weighted vote of weak learners, whose are supposed to be expert on some areas of the input space.
- Iterative algorithm that learn a classifier at each iteration on original data samples but with weights. The distribution is change along iterations to enforce the next learners to focus on difficult samples
- Proof : see C. Capponi slides M1IAAA

Illustration

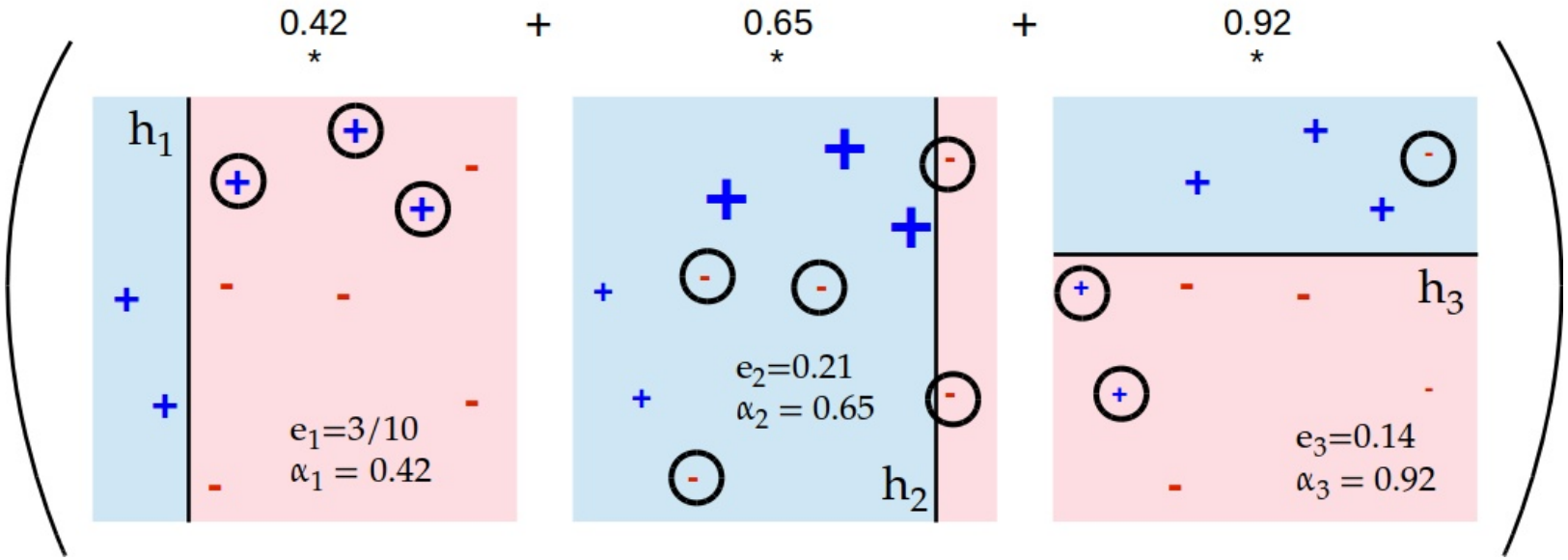


Sequential method : Boosting : AdaBoost [Freund *et.al*, 1996]

Principle

- Learn a decision rule from a weighted vote of weak learners, whose are supposed to be expert on some areas of the input space.
- Iterative algorithm that learn a classifier at each iteration on original data samples but with weights. The distribution is change along iterations to enforce the next learners to focus on difficult samples
- Proof : see C. Capponi slides M1IAAA

Illustration

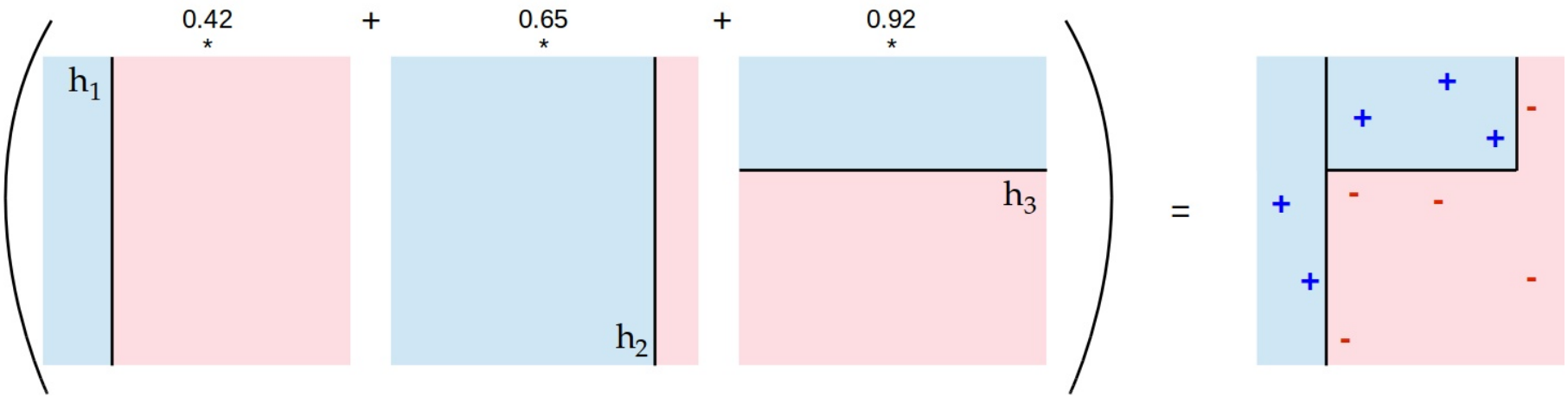


Sequential method : Boosting : AdaBoost [Freund *et.al*, 1996]

Principle

- Learn a decision rule from a weighted vote of weak learners, whose are supposed to be expert on some areas of the input space.
- Iterative algorithm that learn a classifier at each iteration on original data samples but with weights. The distribution is change along iterations to enforce the next learners to focus on difficult samples
- Proof : see C. Capponi slides M1IAAA

Illustration



AdaBoost : Supervised case

AdaBoost [Freund *et al.* 1996]

Données : $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ ou $\mathbf{x}_i \in \mathcal{X}, y_i \in \{-1, +1\}$
Initialiser $D_1(i) = 1/n$
Pour $t = 1...T$:

1. Extraire alatoirement de S un sous-chantillon S_t i.i.d. selon D_t
2. *Apprendre* $h_t : \mathcal{X} \to \{-1, +1\}$ *sur* S_t, D_t *avec erreur*

$$\epsilon_t = \sum_{i=1}^{n} D_t(i) \mathbb{I}\left[(\right] h_t(\mathbf{x}_i \neq y_i)$$

3. *Choisir* $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
4. *Mise à jour de la distribution* :

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_t h_t(\mathbf{x}_i))}{Z_t}$$

où Z_t est un coefficient de normalisation

Sortie : $H(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

References I