Boosting SVMs to understand the SVMs decision function

Gunnar Rätsch

Friedrich Miescher Laboratory of the Max Planck Society

http://www.tuebingen.mpg.de/~raetsch

joint work with Sören Sonnenburg and Christin Schäfer

Overview

- Motivation
- Leveraging as Greedy Optimization
- Boosting and Large Margins
- Boosting SVMs
- Application to Sequence Analysis

Topics:

- Connection between Boosting and Linear Programming
- Application to solve the Multiple Kernel Learning Problem

Electronic copy of the slides will be available from the workshop page.

Understanding the Optimization Objective

Machine learning principle:

- minimize regularized empirical loss

$$L_R[f] = \frac{1}{N}\sum_{n=1}^{N} l(f(\mathbf{x}_n), y_n) + \mathbf{P}[f]$$

- then learning theory can give guarantees:

$$\int l(f(\mathbf{x}), y) dp(\mathbf{x}, y) \leq \frac{1}{N}\sum_{n=1}^{N} l(f(\mathbf{x}_n), y_n) + \mathcal{O}(\mathbf{P}[f])$$

$\Rightarrow$ important what loss function is minimized!

Without knowing the loss function, it is hard to understand what the algorithm actually does.

Understanding and Modifying Boosting

- What loss function is minimized by AdaBoost?
- How can the algorithm be changed to minimize other loss functions?
- How fast does it converge?
- What is the regularizer **P**?
- How is AdaBoost connected to maximal margins & linear Programming?
- How can we design new algorithms?
- How does Boosting compare to Support Vector Machines?

2. Boosting as Greedy Optimization



Hill Inlet, Whitsunday Island

Main Sources:

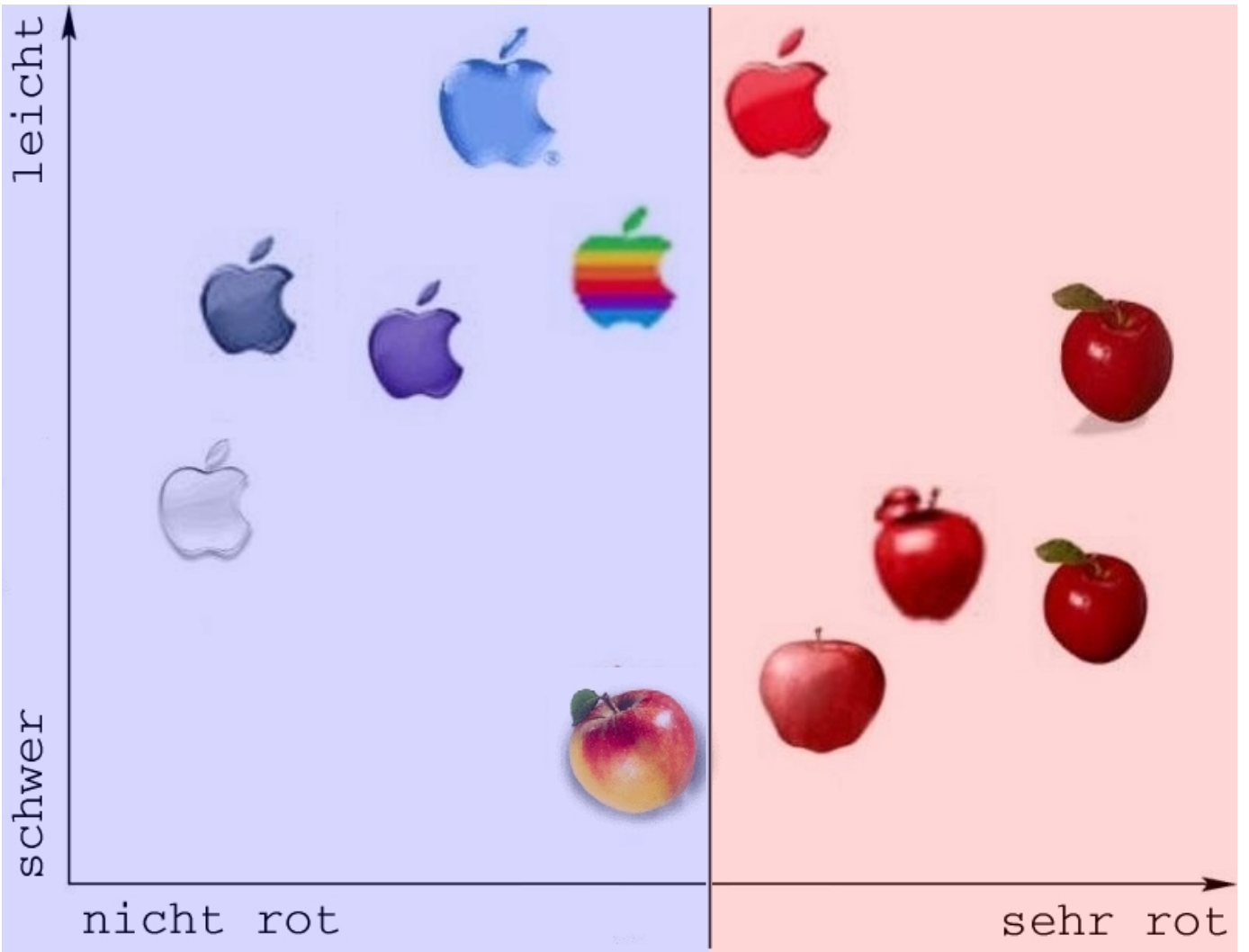
- Kivinen & Warmuth 1999, Lafferty 1999
- Collins, Schapire & Singer 2002
- Rätsch, Mika & Warmuth 2001, Rätsch 2001

AdaBoost (Freund & Schapire 1996)

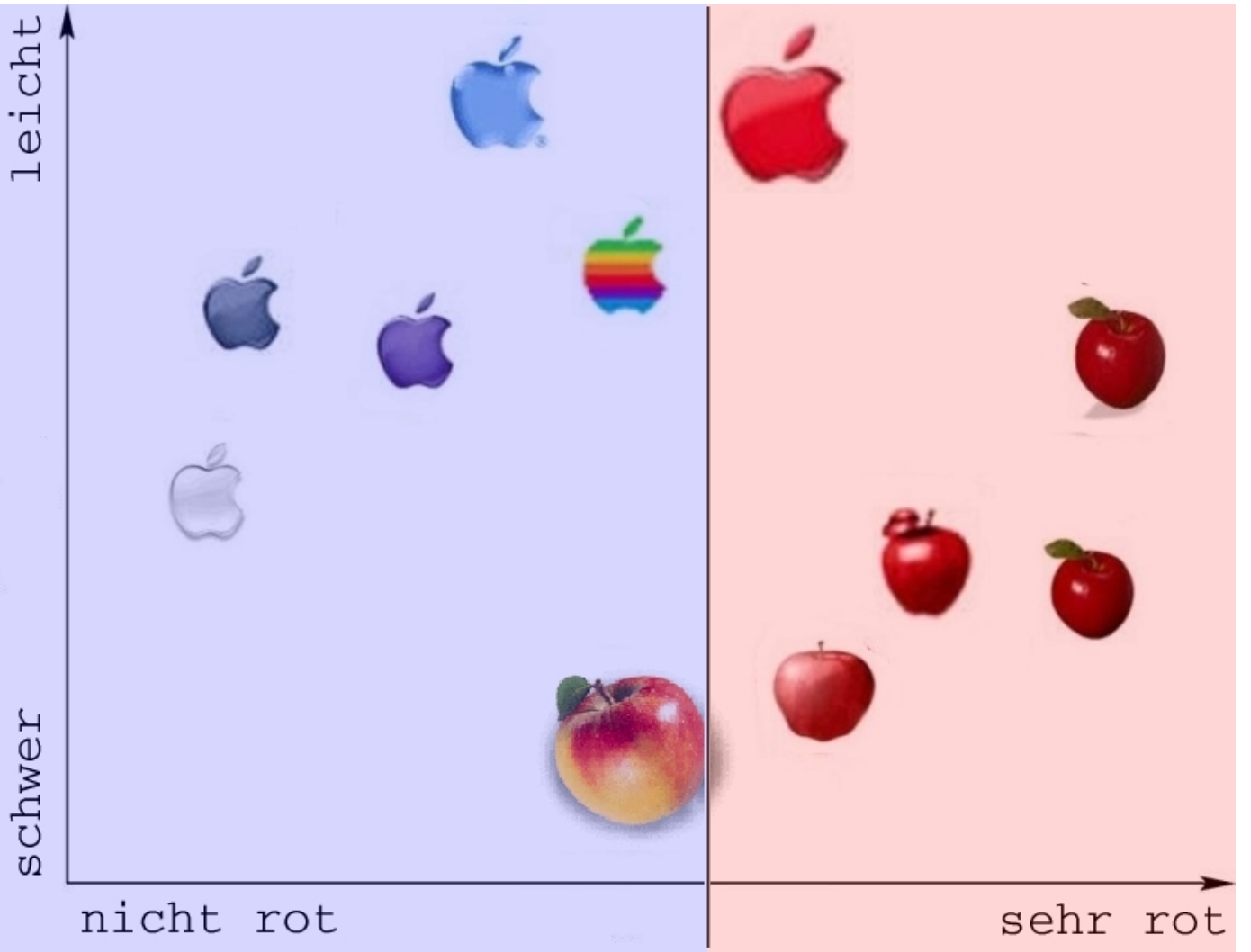
- **Idea:**
  - Simple hypotheses are not perfect!
  - Hypotheses combination $\Rightarrow$ increased accuracy
- Problems:
  - How to generate different hypotheses?
  - How to combine them?
- **Method:**
  - Compute distribution $d_1, \ldots, d_N$ on examples
  - Find hypothesis on the weighted training sample $(\mathbf{x}_1, y_1, d_1), \ldots, (\mathbf{x}_N, y_N, d_N)$
  - Combine hypotheses $h_1, h_2, \ldots$ linearly:

$$f = \sum_{t=1}^{T} \alpha_t h_t$$

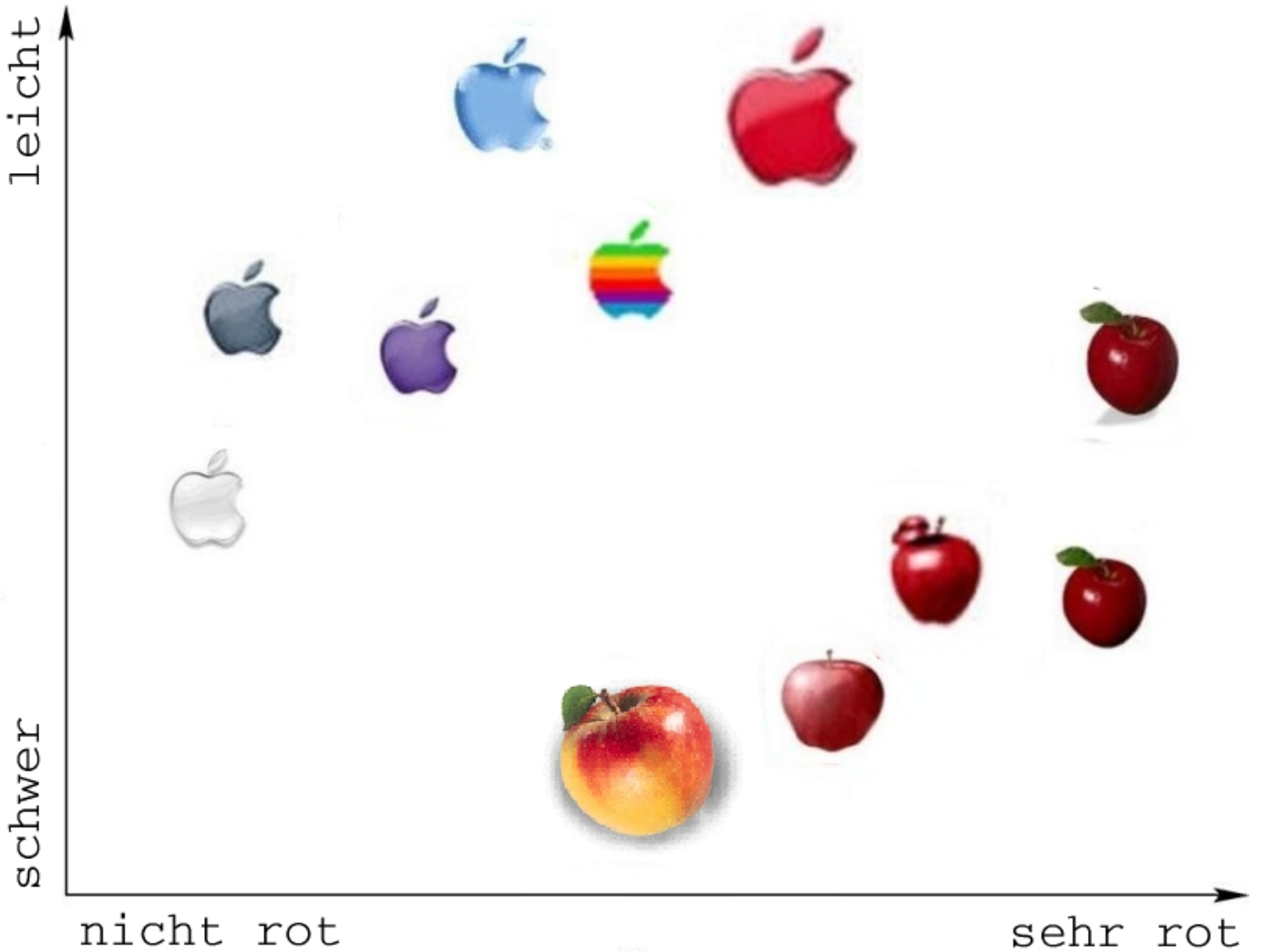
Boosting: 1st Iteration (simple hypothesis)



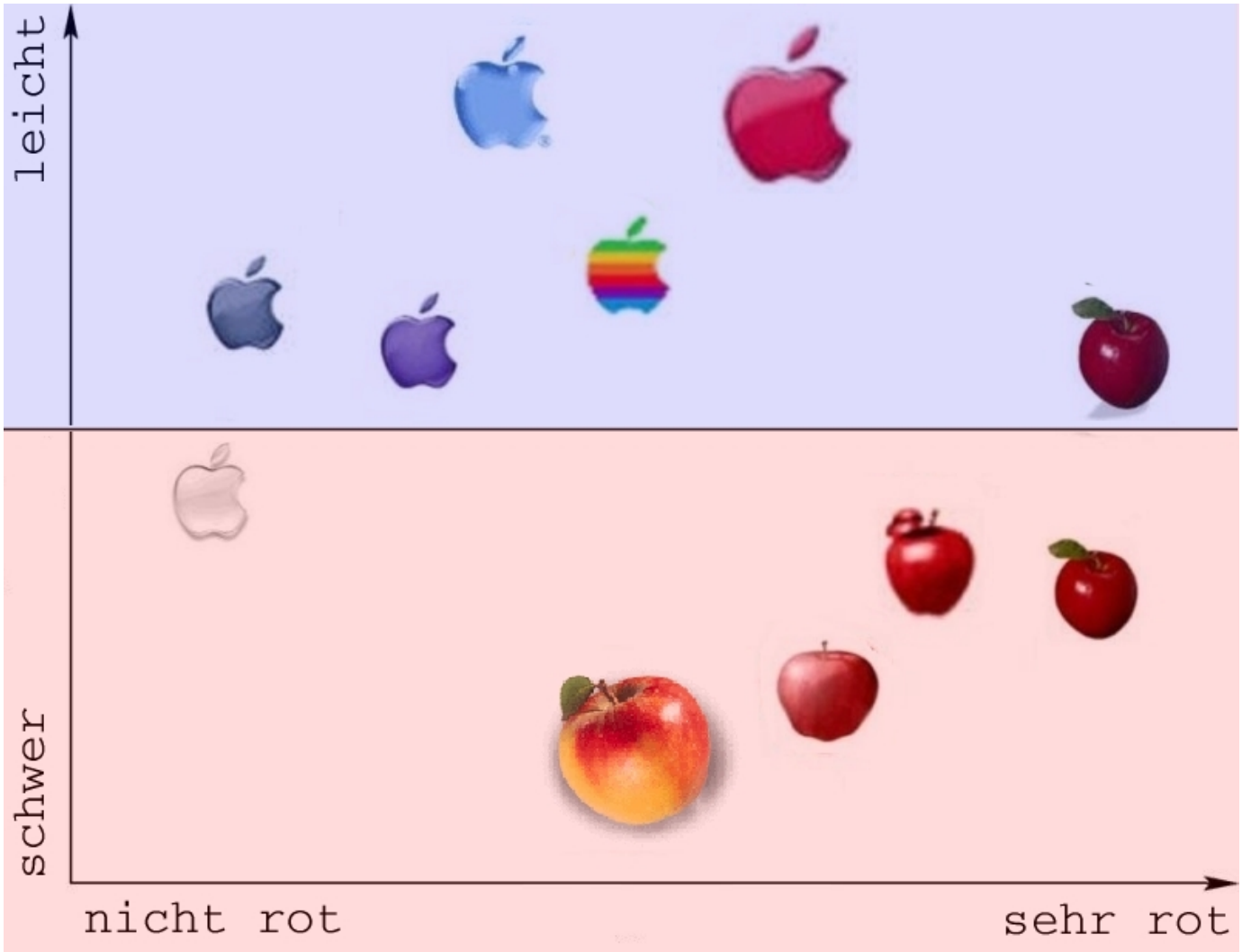
Boosting: Recompute weighting



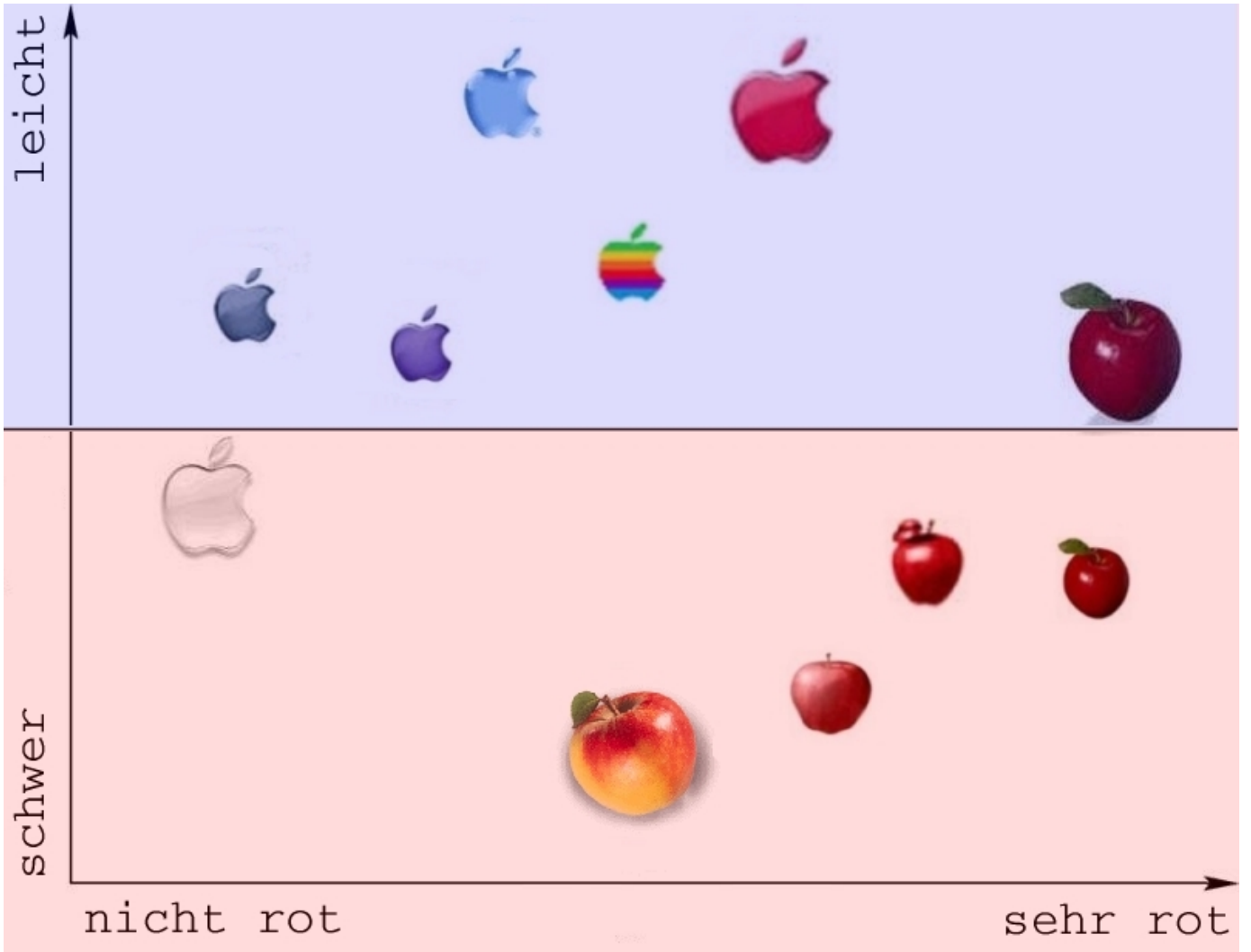
Boosting: 2nd Iteration



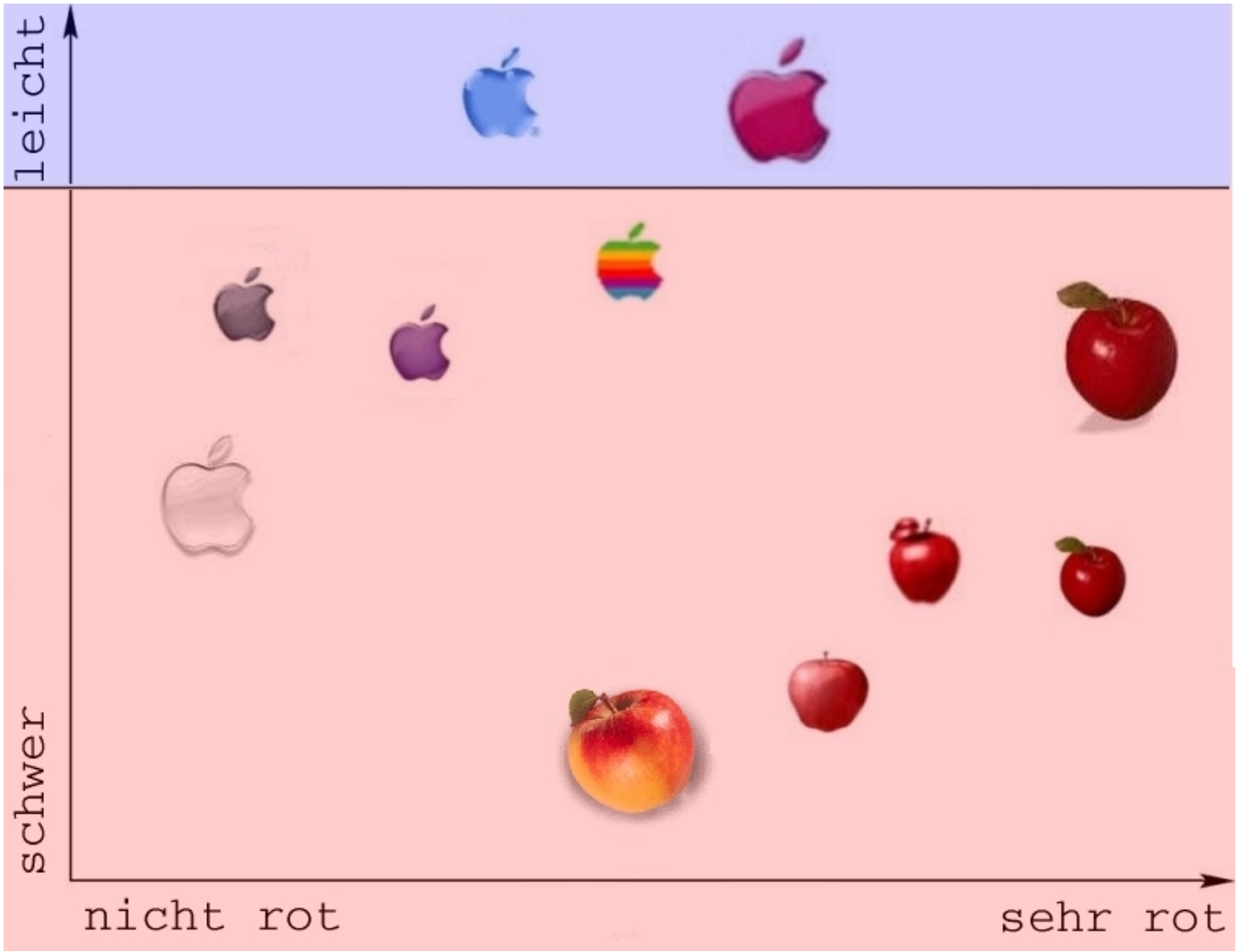
Boosting: 2nd Hypothesis



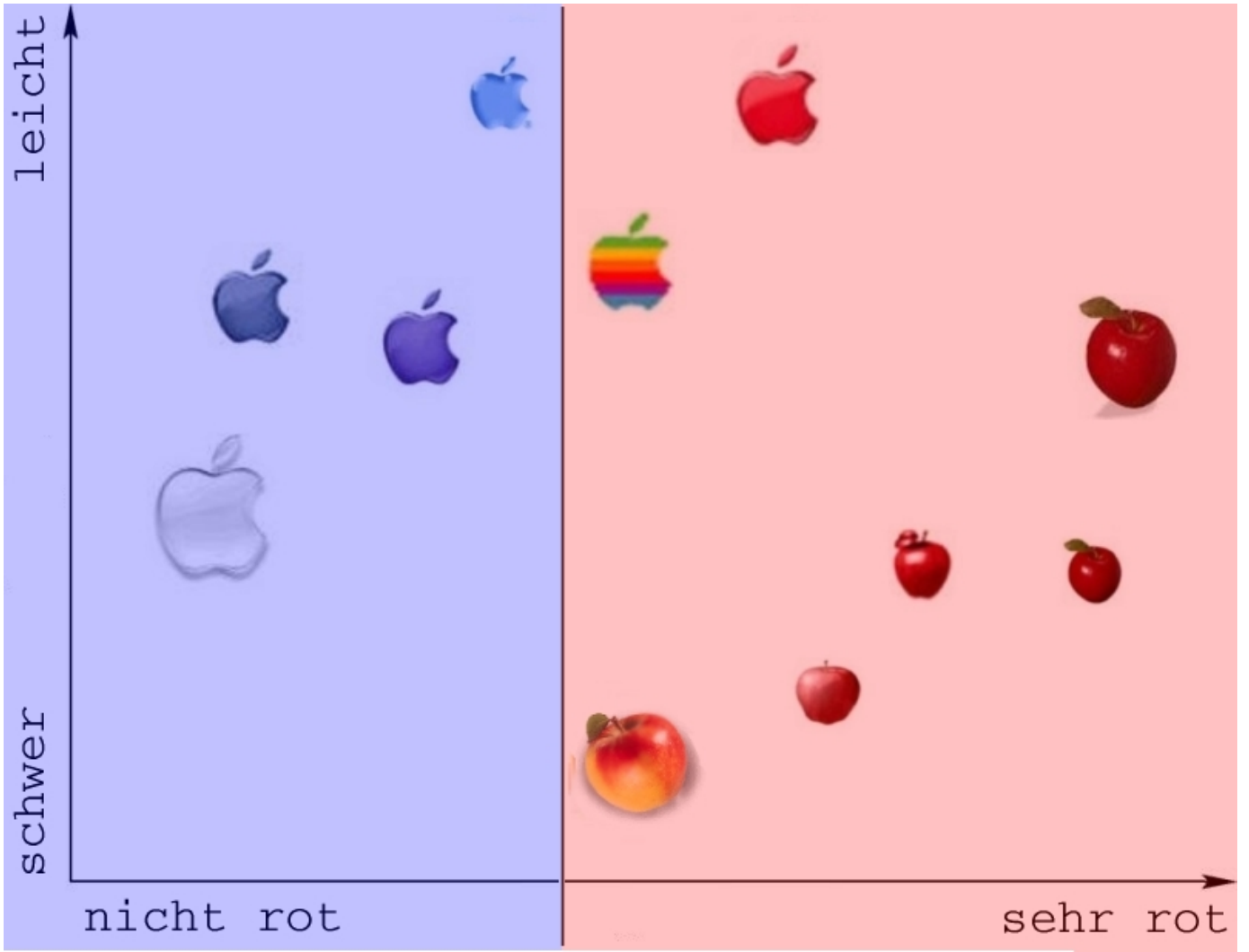
Boosting: Recompute weighting



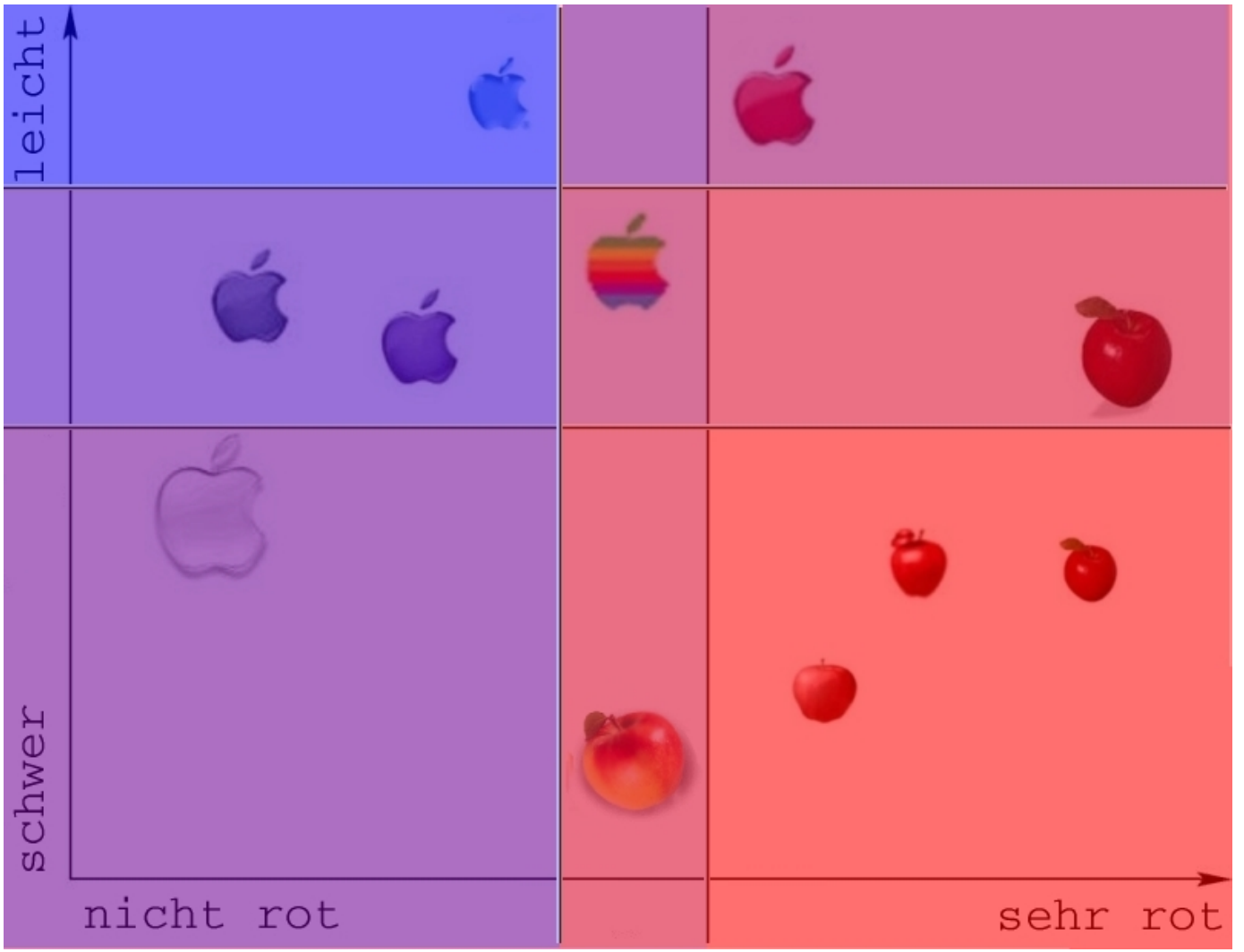
Boosting: 3nd Hypothesis



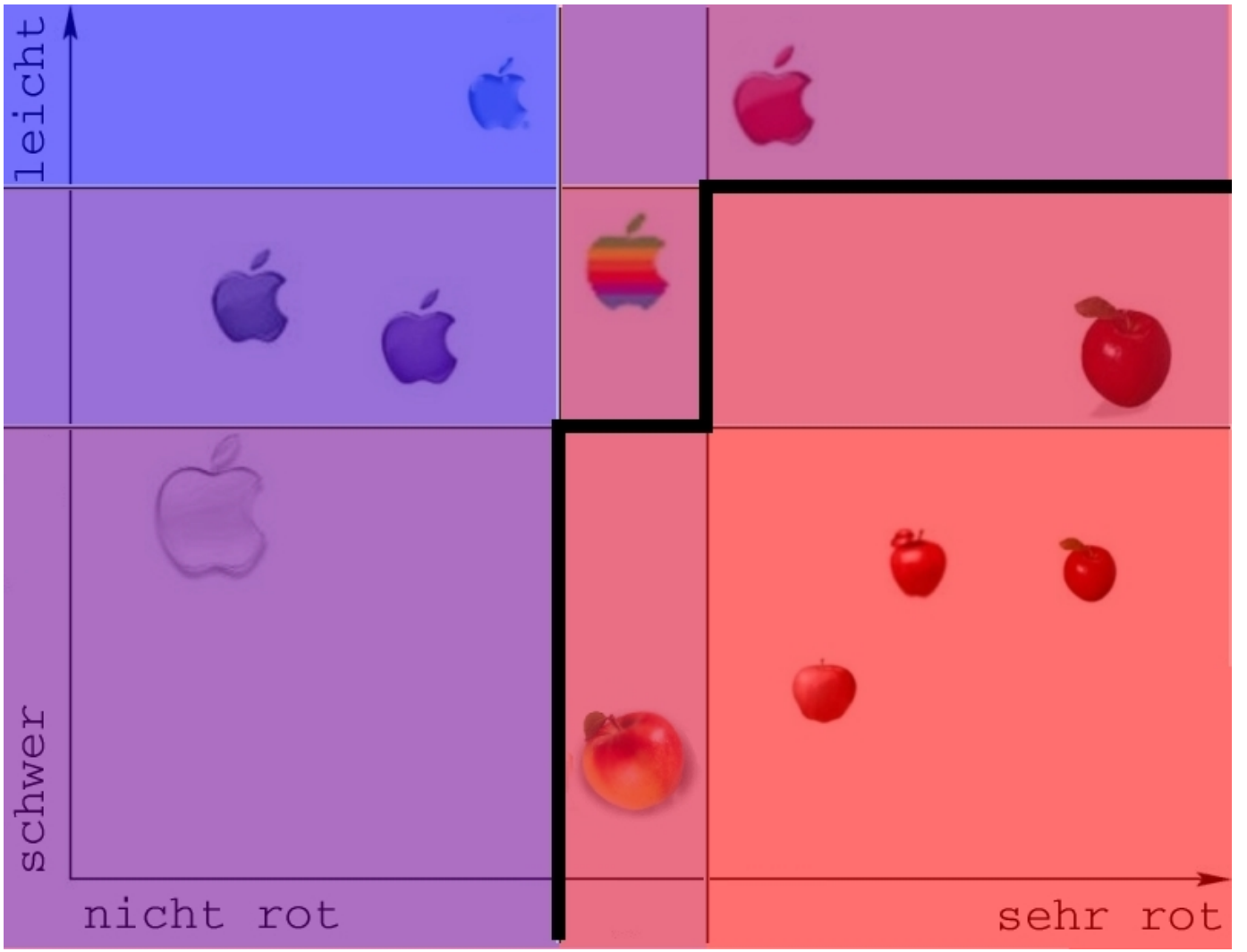
Boosting: 4th Hypothesis



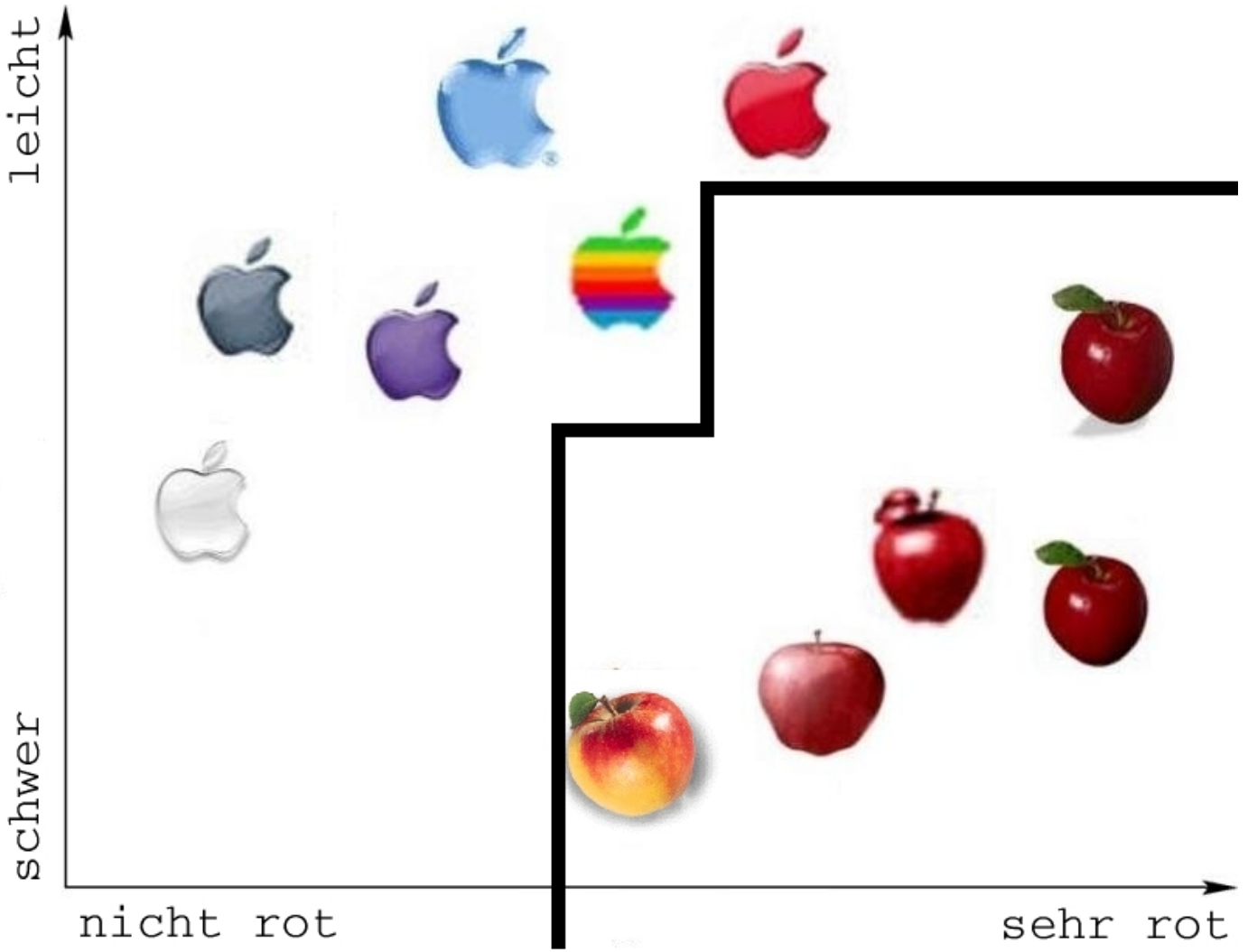
Combination of Hypotheses



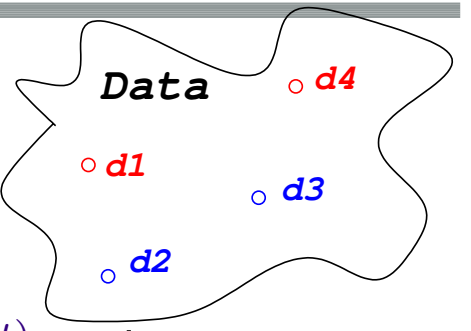
Decision



Decision



AdaBoost Algorithm



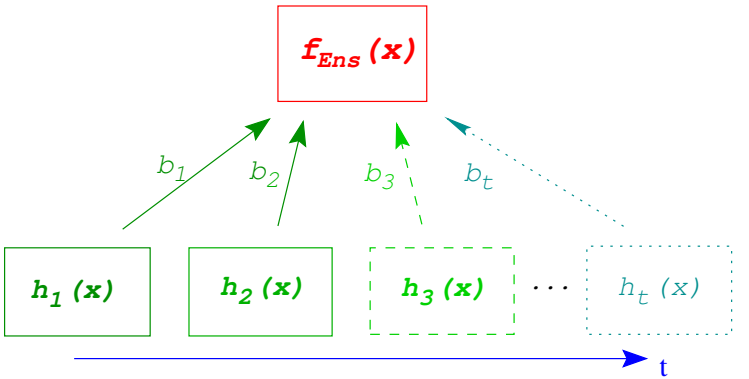
Input: N examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

Initialize: $d_n^{(1)} = 1/N$ $(n = 1 \ldots N)$

Do for $t = 1, \ldots, T$,

1. Train base learner according to example distribution $\mathbf{d}^{(t)}$ and obtain hypothesis $h_t : \mathbf{x} \mapsto \{\pm 1\}$.
2. compute weighted training error $\epsilon_t = \sum_{n=1}^{N} d_n^{(t)} \mathrm{I}(y_n \neq h_t(\mathbf{x}_n))$
3. compute hypothesis weight $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
4. update example distribution

$$d_n^{(t+1)} = \exp\left(-y_n \sum_{r=1}^{t} \alpha_r h_r(\mathbf{x}_n)\right)$$



Output: final hypothesis $f_{\mathrm{Ens}}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

Desirable Properties of the Weak Learner

- generates "simple" hypotheses

  $\Rightarrow$ Uses small base hypothesis set $\mathcal{H}$

- Should minimize or approximately minimize weighted error

$$h_t = \operatorname*{argmin}_{h \in \mathcal{H}} \epsilon_t(h)$$

  where

$$\epsilon_t(h) = \sum_{n=1}^{N} d_n^{(t)} \mathbf{I}(y_n \neq h(\mathbf{x}_n))$$

- Often resampling techniques are used
  - Randomly draw subsets according to distribution $\mathbf{d}^{(t)}$
  - Run weak learner on the subset

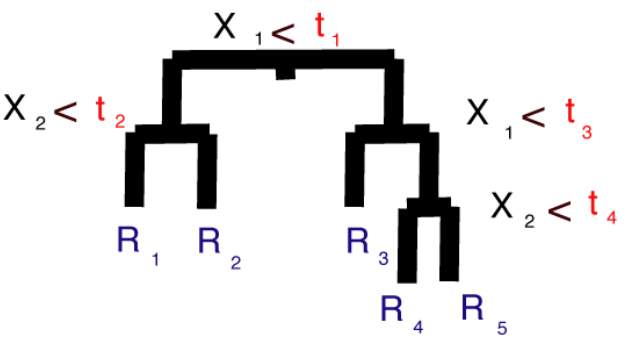
$\Rightarrow$ Allows usage of arbitrary weak learner

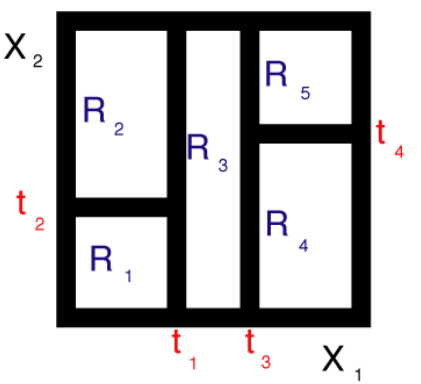
Weak Learners used with Boosting

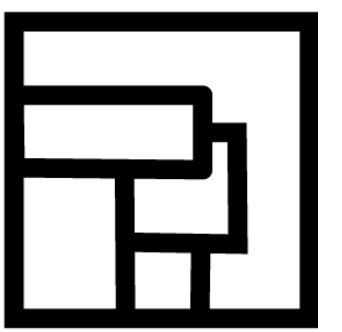
- Decision stumps (axis parallel splits)
- Decision trees (e.g. C4.5 by Quinlan 1996)
- Multi-layer Neural networks (e.g. for OCR)
- Radial basis function networks (e.g. UCI benchmarks, etc)

Decision trees:

- Hierarchical and recursive partitioning on the input space
- Many approaches, usually axis parallel splits







Loss Function for AdaBoost

- AdaBoost stepwise minimizes a function

$$\mathrm{G}[f_{\boldsymbol{\alpha}}] = \sum_{n=1}^{N} \exp\left\{-y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n)\right\}$$

- The gradient of $\mathrm{G}[f_{\boldsymbol{\alpha}^{(t)}}]$ gives the example weights used for AdaBoost:

$$\frac{\partial\, \mathrm{G}[f_{\boldsymbol{\alpha}^{(t)}}]}{\partial f(\mathbf{x}_n)} = \exp\left\{-y_n f_{\boldsymbol{\alpha}^{(t)}}(\mathbf{x}_n)\right\} =: d_n^{(t+1)}$$

- The hypothesis coefficient α_t is chosen, such that $\mathrm{G}[f_{\boldsymbol{\alpha}^{(t)}}] = \mathrm{G}[f_{\boldsymbol{\alpha}^{(t-1)}} + \alpha_t h_t]$ is minimized:

$$\alpha_t = \operatorname*{argmin}_{\alpha_t \in \mathbf{R}} \mathrm{G}[f_{\boldsymbol{\alpha}^{(t)}}] = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$$

The Generic Leveraging Algorithm

Input: loss function $\mathrm{G} : \mathbf{R}^N \to \mathbf{R}$, N examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$
Initialize: $f_0 \equiv 0$, $\mathbf{d}^{(1)} = \nabla \mathrm{G}[f_0]$ (often $\sim 1/N$)
Do for $t = 1, \ldots, T$,

1. Run weak learner according to distribution $\mathbf{d}^{(t)}$ and obtain hypothesis h_t.
2. Compute hypothesis weight α_t such that $\mathrm{G}[f_{t-1} + \alpha_t h_t]$ is minimized
3. Update the combined hypothesis: $f_t = f_{t-1} + \alpha_t h_t$
4. Update example distribution: $\mathbf{d}^{t+1} = \nabla \mathrm{G}[f_t]$

Output: final hypothesis $f_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

Leveraging = Boosting without PAC Boosting property

Examples

- AdaBoost (Freund & Schapire, 94) & Confidence Rated Boosting

$$\mathrm{G}[f_{\boldsymbol{\alpha}}] = \sum_{n=1}^{N} \exp\left\{-y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n)\right\} \qquad d_n = \exp(-y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n))$$

- Logistic Regression Boosting (Friedman et al., Collins et al., 2000)

$$\mathrm{G}[f_{\boldsymbol{\alpha}}] = \sum_{n=1}^{N} \log\left(1 + \exp\left\{-y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n)\right\}\right) \qquad d_n = \frac{\exp(-y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n))}{1 + \exp(-y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n))}$$

- Least Square Boosting (Friedman, 1999)

$$\mathrm{G}[f_{\boldsymbol{\alpha}}] = \sum_{n=1}^{N} (y_n - f_{\boldsymbol{\alpha}}(\mathbf{x}_n))^2 \qquad d_n = y_n - f_{\boldsymbol{\alpha}}(\mathbf{x}_n)$$

- Boosting with Soft Margins (Rätsch et al., 2001a, 2001b)

$$\mathrm{G}[f_{\boldsymbol{\alpha}}] = \sum_{n=1}^{N} \log\left(1 + \exp\left\{1 - y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n)\right\}\right) \qquad d_n = \frac{\exp(1 - y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n))}{1 + \exp(1 - y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n))}$$

3. Boosting and Large Margins



Hill Inlet, Whitsunday Island

Main Sources:

- Schapire, Freund, Bartlett & Lee 1998
- Breiman 1999
- Rätsch 2001, Rätsch & Warmuth 2002
- Bennett, Demiriz & Shawe Taylor 2000

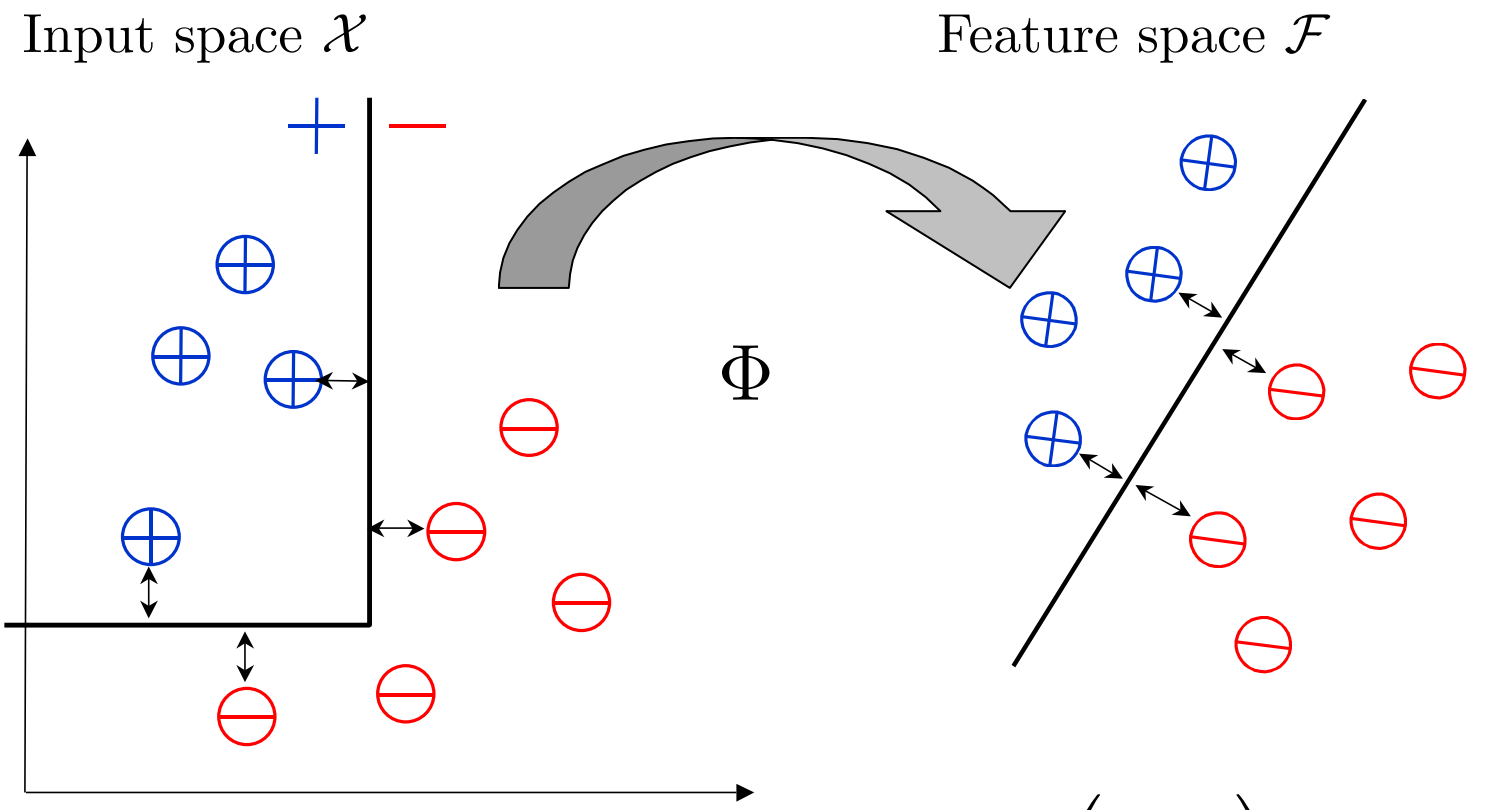
Something more is going on in AdaBoost

- AdaBoost stepwise minimizes a function

$$\mathrm{G}[f_{\boldsymbol{\alpha}}] = \sum_{n=1}^{N} \exp\left\{-y_n f_{\boldsymbol{\alpha}}(\mathbf{x}_n)\right\}$$

- Assume $y_i f_{\boldsymbol{\alpha}}(\mathbf{x}_i) > 0$ for all $i = 1, \ldots, N$
  $\Rightarrow \quad \lim_{\|\boldsymbol{\alpha}\| \to \infty} \mathrm{G}[f_{\boldsymbol{\alpha}}] = 0$
- Many weightings $\boldsymbol{\alpha}$ lead to zero loss!
- Which is the solution generated by AdaBoost?
- Analysis of the margins give some answer.

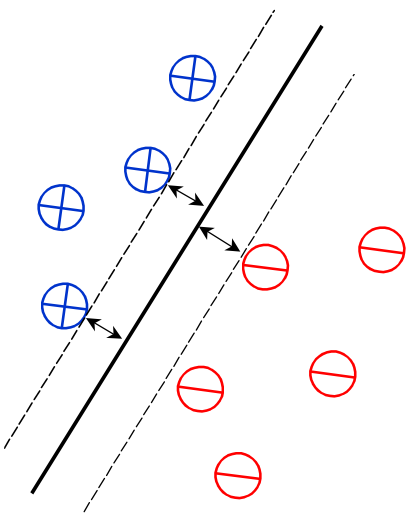
Large Margin and Linear Separation



Linear separation in $\mathcal{F}$ is nonlinear separation in input space

$$\Phi(\mathbf{x}) = \begin{pmatrix} h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \\ \vdots \end{pmatrix}, \mathcal{H} = \{h_1, h_2, \ldots\}$$

Boosting and Large Margins



- Hyperplane = combined hypothesis

$$f_{\boldsymbol{\alpha}}(\mathbf{x}) = (\Phi(\mathbf{x}) \cdot \boldsymbol{\alpha}) = \sum_{j=1}^{J} \alpha_j h_j(\mathbf{x})$$

- margin $\rho(\boldsymbol{\alpha}) = \min_n \underbrace{\frac{y_n(\Phi(\mathbf{x}_n) \cdot \boldsymbol{\alpha})}{\|\boldsymbol{\alpha}\|_1}}_{\text{distance}}$
- distance = ℓ_∞-distance (Mangasarian 1999)

Weak Learner $\mathcal{L}$

- One uses an example weighting $d_1, \ldots, d_N$
- Weak learner $\mathcal{L}$ returns hypotheses $\mathcal{H} \ni h : \mathbf{x} \mapsto \{\pm 1\}$
  ... which are consistently *better than random guessing*

$$\underbrace{\sum_{n=1}^{N} d_n \ \mathrm{I}(h(\mathbf{x}_n) \neq y_n)}_{\text{weighted tr. error of } h} \leq \frac{1}{2} - \frac{1}{2}\gamma$$

  for any weighting $\mathbf{d}$ and some fixed $\gamma > 0$

- For real valued hypotheses $h : \mathbf{x} \mapsto [-1, 1]$

$$\underbrace{\sum_{n=1}^{N} d_n \ y_n h(\mathbf{x}_n)}_{\text{Edge of } h} \geq \gamma$$

Edge and Margin

- Edge of a weak learner $\mathcal{L}$ for a probability vector $\mathbf{d}$ on the examples

$$\gamma(\mathbf{d}) := \max_{h \in \mathcal{H}} \sum_{n=1}^{N} d_n \; y_n h(x_n)$$

- Margin of the combined hypothesis $f_{\boldsymbol{\alpha}}$

$$\rho(\boldsymbol{\alpha}) := \min_{n=1\ldots N} y_n \frac{f_{\boldsymbol{\alpha}}(\mathbf{x}_n)}{\sum_t \alpha_t} = \min_{n=1\ldots N} y_n \sum_{t=1}^{T} \frac{\alpha_t}{\sum_r \alpha_r} h_t(\mathbf{x}_n)$$

Min-Max-Theorem:

$$\min_{\mathbf{d}} \gamma(\mathbf{d}) = \max_{\boldsymbol{\alpha}} \rho(\boldsymbol{\alpha})$$

If $\gamma(\mathbf{d}) \geq \gamma$, the maximal margin is at least γ!

Discussion

What is the margin achieved by AdaBoost?

$\Rightarrow$ finds fast a solution with large, but not maximal margin

$\Rightarrow$ only about a half of the maximal margin

Modify AdaBoost for maximizing margin

- AdaBoost$_\varrho$ generates a solution with margin at least ϱ (if it exists)
  $\Rightarrow$ Arc-GV (Breiman 1999) asymptotically maximizes the margin
  $*$ no converge rates
  $\Rightarrow$ AdaBoost* requires $\frac{2\log(N)}{\epsilon^2}$ iterations to get margin in $[\rho^* - \epsilon, \rho^*]$
- Column Generation uses a Linear Programming solver
  – adds one constraint (hypothesis) per iteration
  – usually very fast
  – no converge rates

The AdaBoost$_\varrho$ Algorithm (Breiman 1997)

1. **Input:** $S = \langle (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \rangle$, No. Iterations T, margin target ϱ
2. **Initialize:** $d_n^1 = 1/N$ for all $n = 1 \ldots N$
3. **Do for** $t = 1, \ldots, T$,
   (a) Train classifier on $\{S, \mathbf{d}^t\}$ and obtain hypothesis $h_t : \mathbf{x} \mapsto [-1, 1]$
   (b) Calculate the edge γ_t of h_t: $\gamma_t = \sum_{n=1}^{N} d_n^t y_n h_t(\mathbf{x}_n)$.
   (c) **if** $\gamma_t \leq \varrho$ or $\gamma_t = 1$, **then** break
   (d) Set $\alpha_t = \frac{1}{2} \log \frac{1+\gamma_t}{1-\gamma_t} \boxed{- \frac{1}{2} \log \frac{1+\varrho}{1-\varrho}}$.
   (e) Update weights: $d_n^{t+1} = d_n^t \exp\{-\alpha_t y_n h_t(\mathbf{x}_n)\} / Z_t$, s.t. $\sum_{n=1}^{N} d_n^{t+1} = 1$.
4. **Output:** $f(\mathbf{x}) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_r \alpha_r} h_t(\mathbf{x})$

Primal and Dual Max Margin LP

$$\max_{\boldsymbol{\alpha},\rho} \quad \rho$$

$$\text{subject to} \quad \underbrace{y_n \sum_{j=1}^{J} \alpha_j h_j(\mathbf{x}_n)}_{\text{margin}} \geq \rho$$

$$\alpha_j \geq 0$$

$$\textstyle\sum_j \alpha_j = 1$$

$$\min_{\mathbf{d},\delta} \quad \delta$$

$$\text{subject to} \quad \underbrace{\sum_{n=1}^{N} d_n y_n h_j(\mathbf{x}_n)}_{\text{edge}} \leq \delta$$

$$d_n \geq 0$$

$$\textstyle\sum_n d_n = 1$$

- AdaBoost$_\varrho$: ϱ and δ are fixed
- Arc-GV and AdaBoost*: ϱ and δ are optimized
- LP Column Generation works in the dual

The number of hypotheses can be very large or infinite!

4. Boosting SVMs: Multiple Kernel Learning



Hill Inlet, Whitsunday Island

Main Sources:

- Bach, Lanckriet, Jordan, 2004
- Sonnenburg, Rätsch, Schäfer (RECOMB'05)

Multiple Kernel Learning: Motivation

- Kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$ used in standard SVM Classifier

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} y_i \alpha_i \mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b\right)$$

- Now: Convex combination of kernels (again a kernel)

$$\mathrm{k}(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{M} \beta_j \, \mathrm{k}_j(\mathbf{x}, \mathbf{x}'), \;\; \beta_j \geq 0$$

- **useful:** Polynomial kernels of different degree, kernels on different domains
- **but:** How to learn weights β_j ?

  $\Rightarrow$ Make it part of the SVM optimization!

Constraining the weights

2 vs. 1 Norm

- in max problem weights certain β_j would grow infinitely
  $\Rightarrow$ **constraining β_j necessary**
  - $\|\boldsymbol{\beta}\|_2 = 1$ (Lin and Zhang 2004)
    - weights are non-sparse
    - harder to interpret
  - $\|\boldsymbol{\beta}\|_1 = 1$ (Bach, Lanckriet and Jordan 2004)
    - convex combination of kernels
    - sparse solution in terms of kernels
    - allows for interpretation of result

We use: $\sum_{j=1}^{N} \beta_j = 1,\ \beta_j \geq 0$

Standard SVM Optimization Problem

This we all know

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N}\xi_n \\ \text{w.r.t.} \quad & \mathbf{w} \in \mathbf{R}^k, \boldsymbol{\xi} \in \mathbf{R}_+^N, b \in \mathbf{R} \\ \text{s.t.} \quad & y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i, \forall i = 1, \ldots, N \end{aligned}$$

MKL Optimization: Primal

$$\begin{aligned}
\min \quad & \frac{1}{2}\left(\sum_{j=1}^{M}\beta_j \left\|\mathbf{w}_j\right\|_2\right)^2 + C\sum_{i=1}^{N}\xi_n \\
\text{w.r.t.} \quad & \mathbf{w}=(\mathbf{w}_1,\ldots,\mathbf{w}_M), \mathbf{w}_j \in \mathbf{R}^{k_j}, \boldsymbol{\xi}\in\mathbf{R}_+^N, \boldsymbol{\beta}\in\mathbf{R}_+^M, b\in\mathbf{R} \\
\text{s.t.} \quad & y_i\left(\sum_{j=1}^{M}\beta_j {\mathbf{w}_j}^\top \mathbf{x}_{i,j} + b\right) \geq 1-\xi_i, \forall i=1,\ldots,N \\
& \sum_{j=1}^{M}\beta_j = 1
\end{aligned}$$

Properties: equivalent to linear SVM for $M = 1$; solution sparse in "blocks"; each block j corresponds to one kernel

MKL Optimization: Dual (Bach et al., 2004)

$$\begin{aligned}
\min \quad & \frac{1}{2}\gamma^2 - \sum_{i=1}^{N} \alpha_i \\
\text{w.r.t.} \quad & \gamma \in \mathbf{R}, \boldsymbol{\alpha} \in \mathbf{R}^N \\
\text{s.t.} \quad & 0 \leq \boldsymbol{\alpha} \leq C, \sum_{i=1}^{N} \alpha_i y_i = 0 \\
& \underbrace{\sum_{r=1}^{N}\sum_{s=1}^{N} \alpha_r \alpha_s y_r y_s K_j(\mathbf{x}_r, \mathbf{x}_s)}_{=:S_j(\boldsymbol{\alpha})} - \gamma^2 \leq 0, \ \forall j = 1, \ldots, M
\end{aligned}$$

"partial Lagrangian:" $(\text{s.t. } 0 \leq \boldsymbol{\alpha} \leq C, \sum_{i=1}^{N} \alpha_i y_i = 0)$

$$L := \frac{1}{2}\gamma^2 - \sum_{i=1}^{N} \alpha_i + \sum_{j=1}^{M} \beta_j (S_j(\boldsymbol{\alpha}) - \gamma^2)$$

MKL Optimization Problem II

Partial Lagrangian: $(\text{s.t. } 0 \leq \boldsymbol{\alpha} \leq C, \sum_{i=1}^{N} \alpha_i y_i = 0)$

$$L := \frac{1}{2}\gamma^2 - \sum_{i=1}^{N} \alpha_i + \sum_{j=1}^{M} \beta_j (S_j(\boldsymbol{\alpha}) - \gamma^2)$$

Solve Saddle Point Problem:

$$\max_{\boldsymbol{\beta}} \min_{\boldsymbol{\alpha}} \sum_{j=1}^{M} \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_{i=1}^{N} \alpha_i \right)$$

$$\text{s.t. } 0 \leq \boldsymbol{\alpha} \leq C, \sum_{i=1}^{N} \alpha_i y_i = 0, \sum_{j=1}^{M} \beta_j = 1$$

MKL Optimization: Reformulation

- Introduce new constraints: $\Rightarrow$ Semi-infinite Linear Program

$$\begin{aligned} \max \quad & \theta \\ \text{w.r.t.} \quad & \theta \in \mathbf{R}, \boldsymbol{\beta} \in \mathbf{R}_+^M \text{ with } \sum_{j=1}^{M} \beta_j = 1 \\ \text{s.t.} \quad & \sum_{j=1}^{M} \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_{i=1}^{N} \alpha_i \right) \geq \theta \\ & \text{for all } \boldsymbol{\alpha} \text{ with } 0 \leq \boldsymbol{\alpha} \leq C \text{ and } \sum_{i=1}^{N} y_i \alpha_i = 0 \end{aligned}$$

$\Rightarrow$ Use Boosting techniques to solve it

- exponential update of the $\boldsymbol{\beta}$'s
- "base hypothesis" finding corresponds to finding active constraint:

$$\sum_{j=1}^{M} \beta_j \left(\frac{1}{2} S_j(\boldsymbol{\alpha}) - \sum_{i=1}^{N} \alpha_i \right) = \frac{1}{2} \sum_{r=1}^{N} \sum_{s=1}^{N} \alpha_r \alpha_s y_r y_s \sum_{j=1}^{M} \beta_j k_j(\mathbf{x}_r, \mathbf{x}_s) - \sum_{i=1}^{N} \alpha_i,$$

$\Rightarrow$ Solve a SVM for some given $\boldsymbol{\beta}$'s

How to solve the Problem?

Properties:

- optimize a convex combination
- infinitely many constraints
- quite easy to identify violated constraints

Use the Boosting like techniques!

- Arc-GV or AdaBoost*
  - known convergence rate $\mathcal{O}(\log(M)/\varepsilon^2)$
- Column Generation
  - faster, but no convergence rate
- SMO like algorithm
  - consider suboptimal SVM solutions: empirically 3-5 times faster
  - $t(N, M, \epsilon) \approx 5 \cdot 10^{-11} M^{2.22} N^{1.68} \log_2(1/\epsilon)^{2.52} + \mathcal{O}(1)$

An Arc-GV like Algorithm

$D^0 = 1,\ \rho^1 = \tau_k^1 = 0, \beta_k^1 = \frac{1}{M}$ for $k = 1, \ldots, M$

for $t = 1, 2, \ldots$ **do**

obtain SVM's $\boldsymbol{\alpha}^k$ with kernel

$$\mathrm{k}^t(\mathbf{x}_i, \mathbf{x}_j) := \sum_{k=1}^{M} \beta_k^t \, \mathrm{k}_k(\mathbf{x}_i, \mathbf{x}_j)$$

for $k = 1, \ldots, M$ **do**

$D_k^t = \frac{1}{2} \sum_{r,s} \alpha_r^t \alpha_s^t y_r y_s \, \mathrm{k}_k(\mathbf{x}_r, \mathbf{x}_s) - \sum_r \alpha_r^t$

end for

$D^t = \sum_{k=1}^{M} \beta_k^t D_k^t$

$\gamma_t = \mathrm{argmin}_{\gamma \in [0,1]} \sum_{k=1}^{M} \beta_k^t \exp\left\{\gamma(D_k - \rho^t)\right\}$

for $k = 1, \ldots, M$ **do**

$\tau_k^{t+1} = \tau_k^t + \gamma_t D_k^t$

$\beta_k^{t+1} = \beta_k^t \exp(\gamma_t D_k^t) / \left(\sum_{k'} \beta_{k'}^t \exp(\gamma_t D_{k'}^t)\right)$

end for

$\rho^{t+1} = \max_k \tau_k^{t+1} / \sum_{r=1}^{t} \gamma_r$

if $|1 - \frac{\rho^{t+1}}{D^t}| \leq \epsilon$ **then break**

end for

5. Application to Sequence Analysis



Hill Inlet, Whitsunday Island

Main Sources:

- Sonnenburg, Rätsch, Schäfer (RECOMB'05)
- Rätsch, Sonnenburg, Schölkopf (ISMB'05)

MKL in Sequence Analysis

- Problem:
  - classify (e.g. nucleic) sequences of a certain length
  - for instance a window around a certain signal
- Use positional string kernels for SVMs

$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{d} \sum_{l=1}^{L-k} \beta_{l,k} \mathrm{I}(\mathbf{u}_{k,l}(\mathbf{x}_i) = \mathbf{u}_{k,l}(\mathbf{x}_j)),$$

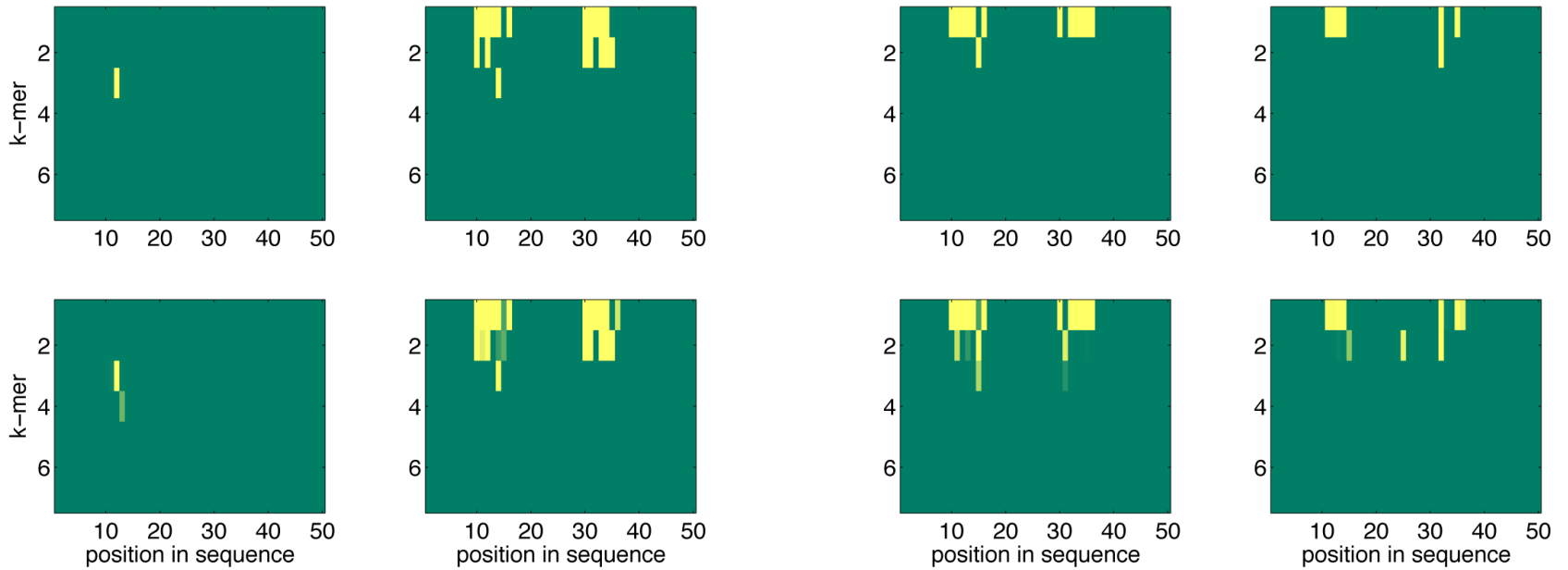
  $\Rightarrow$ hard to interpret SVMs decision boundary: $\mathrm{sign}\left(\sum_i \alpha_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x})\right)$

- **Idea**:
  - Use one kernel per position and oligomer length
  - Optimize convex combination of kernels

  $\Rightarrow$ A small subset of kernels get non-zero weights

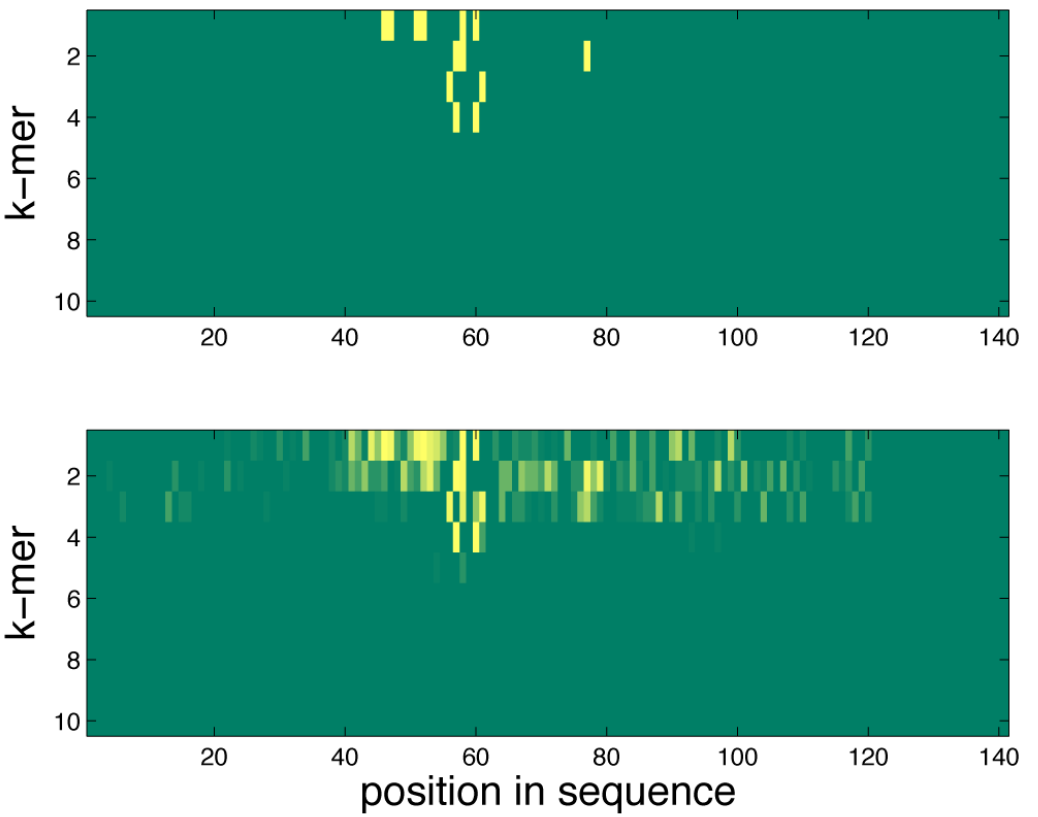
  $\Rightarrow$ Weights can be interpreted

Toy Dataset



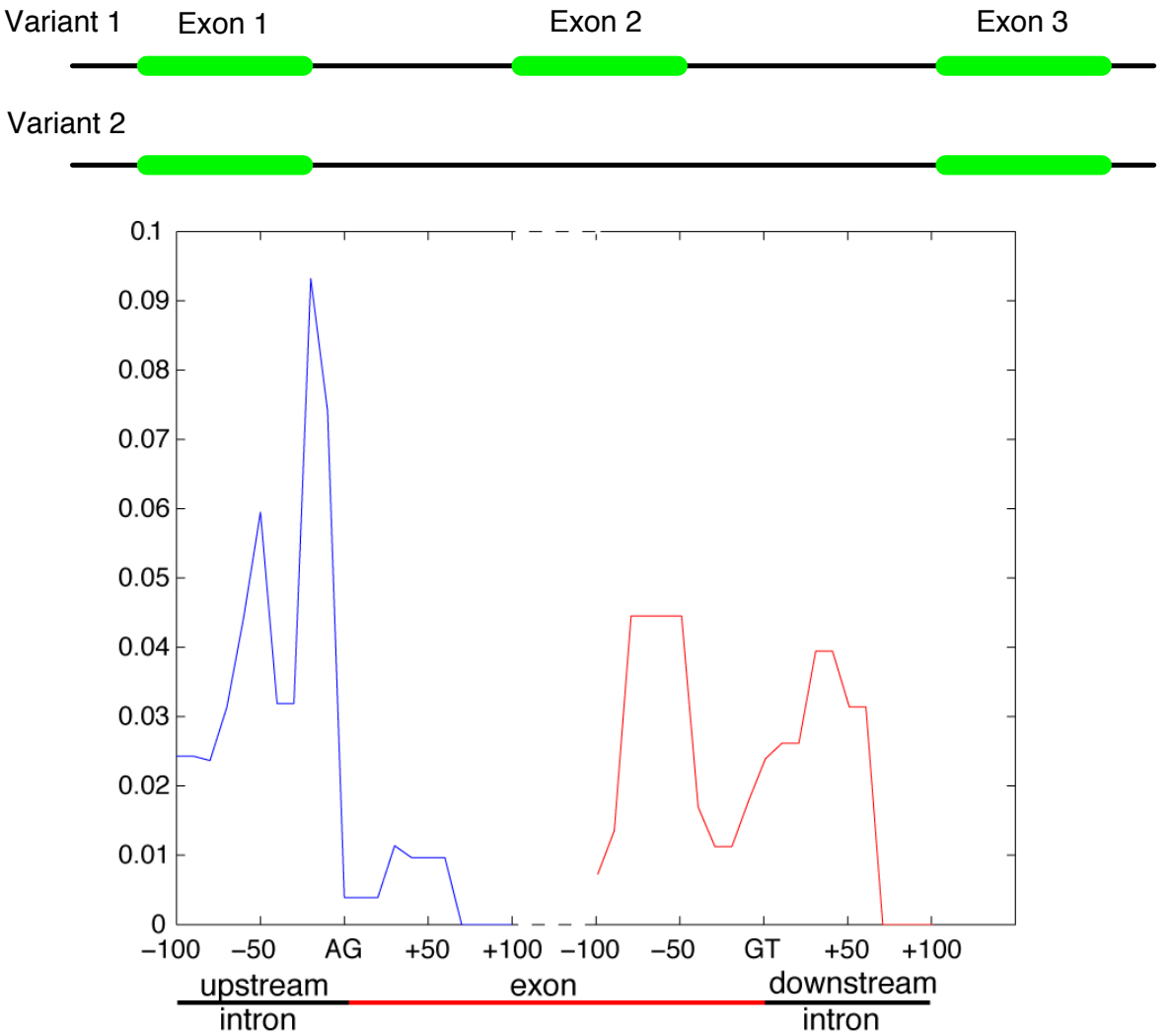
- DNA sequences of length 50 with hidden motifs at 10-16 and 30-36
- 8×50 string kernels with max. word- length 8
- compute significance level by bootstrapping
- columns $\equiv$ noise level
- subplot columns weights used at certain position, rows oligomer length

Application to Splice Detection



- *C.elegans* acceptor splice site dataset (AG at position 61 & 62)
- subplot columns weights used at certain position, rows oligomer length

Recognition of Alt. Spliced Exons (ISMB'05)



- Identify Alt. Spliced Exons from nucleic sequence only (*C. elegans*)
- High Accuracy (AUC $\approx$ 90%)
- Where is the discriminative information located?

$\Leftarrow$ Use MKL (one weight per 10 positions near 5' and 3' end of skipped exon)

Conclusion

- AdaBoost minimizes a certain exponential loss ...
- ... while at the same time approximates the solution of a Semi-infinite Linear Program (SLP)
- Multiple Kernel Learning Problem can be reformulated a SLP
- Boosting techniques can efficiently solve the problem
  (even with hundrets of kernels and thousands of examples)
- MKL helps understanding the SVM's decision in sequence analysis...
- ... by identifying important positions and k-mers