Computational Learning Theory

D.M.J. Tax

TU Delft
Delft University of Technology

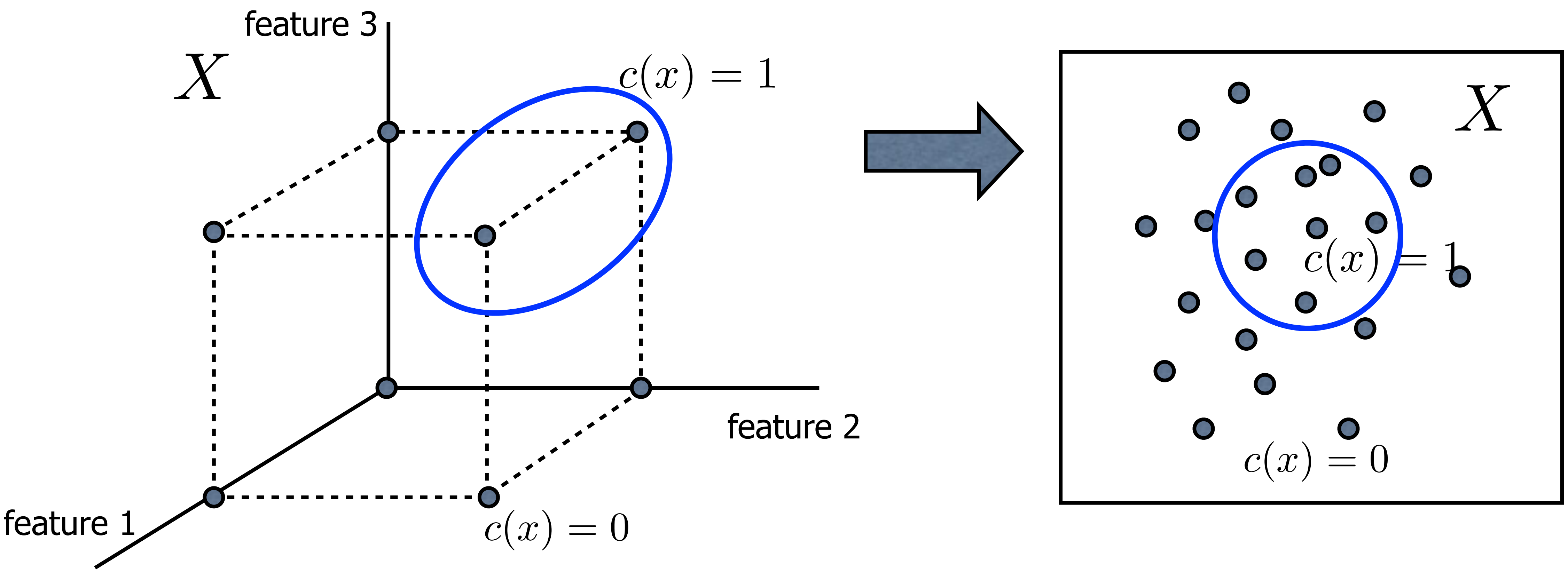
Contents

- PAC learning, the definitions
- Example: Rectangle Learning
- Discrete hypothesis space and Consistent learners
- Continuous hypothesis space: VC-dimension
- ‘No Free Lunch’ theorem

- Weak/strong learning
- Boosting
- AdaBoost

PAC learning

- Probably Approximately Correct: PAC
- Here: restricted to boolean valued concepts from **noise-free** training data (often discrete features, although it can be extended...)
- Goal: learn a concept c from instances randomly drawn from prob.distribution D using learner L.

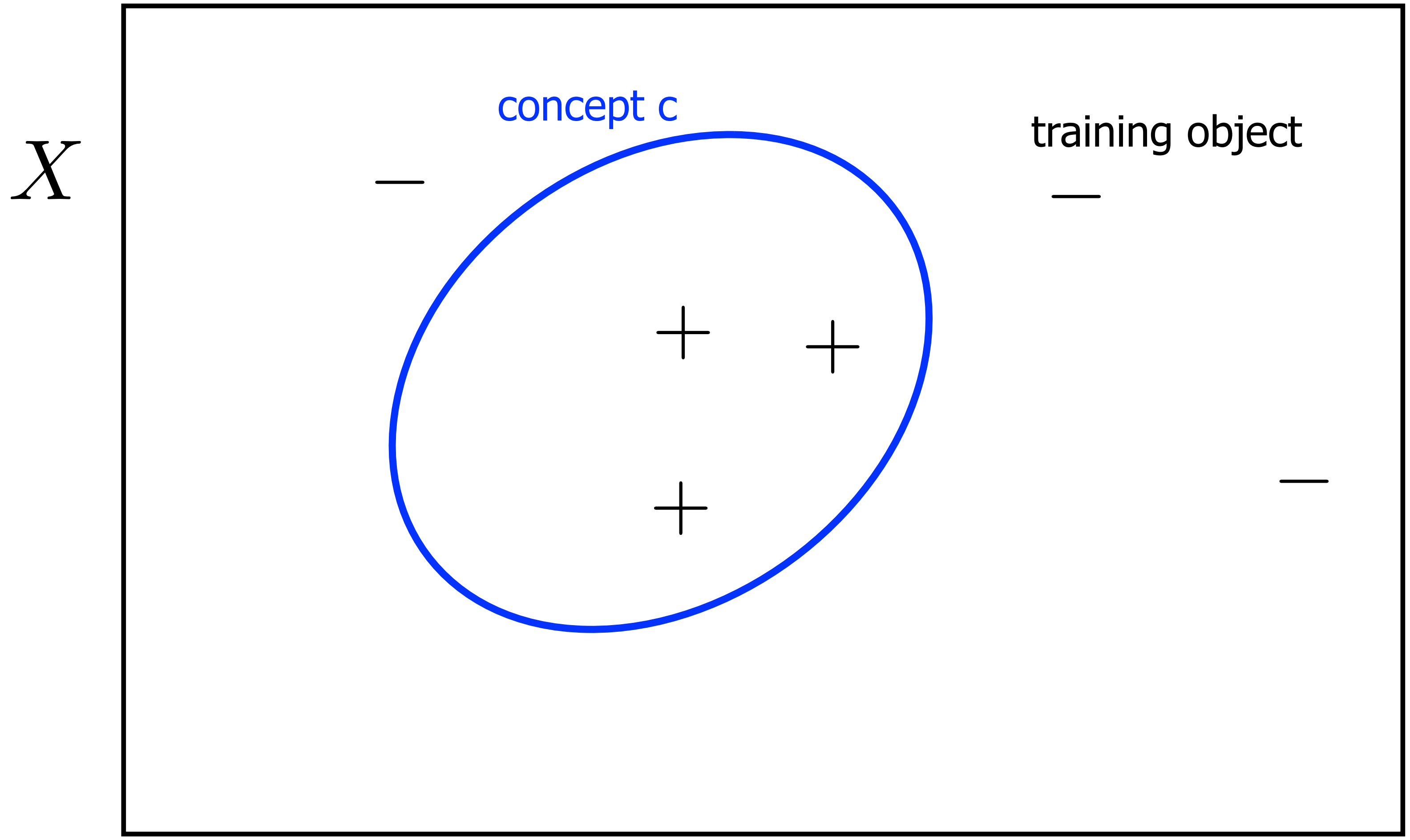


PAC learning

- X: instance space (all possible instances)
- C: set of target concepts that may have to be learned
- c: a concept, a subset of X $c : X \rightarrow \{0, 1\}$
- D: probability distribution over instances x.
- H: possible hypotheses used for approximating the concept c (H should include C)
- L: learner that selects a hypothesis h given a random sample of instances drawn according to D
- error: $\mathrm{error}_D(h) = Pr_{x \in D}\left[c(x) \neq h(x)\right]$

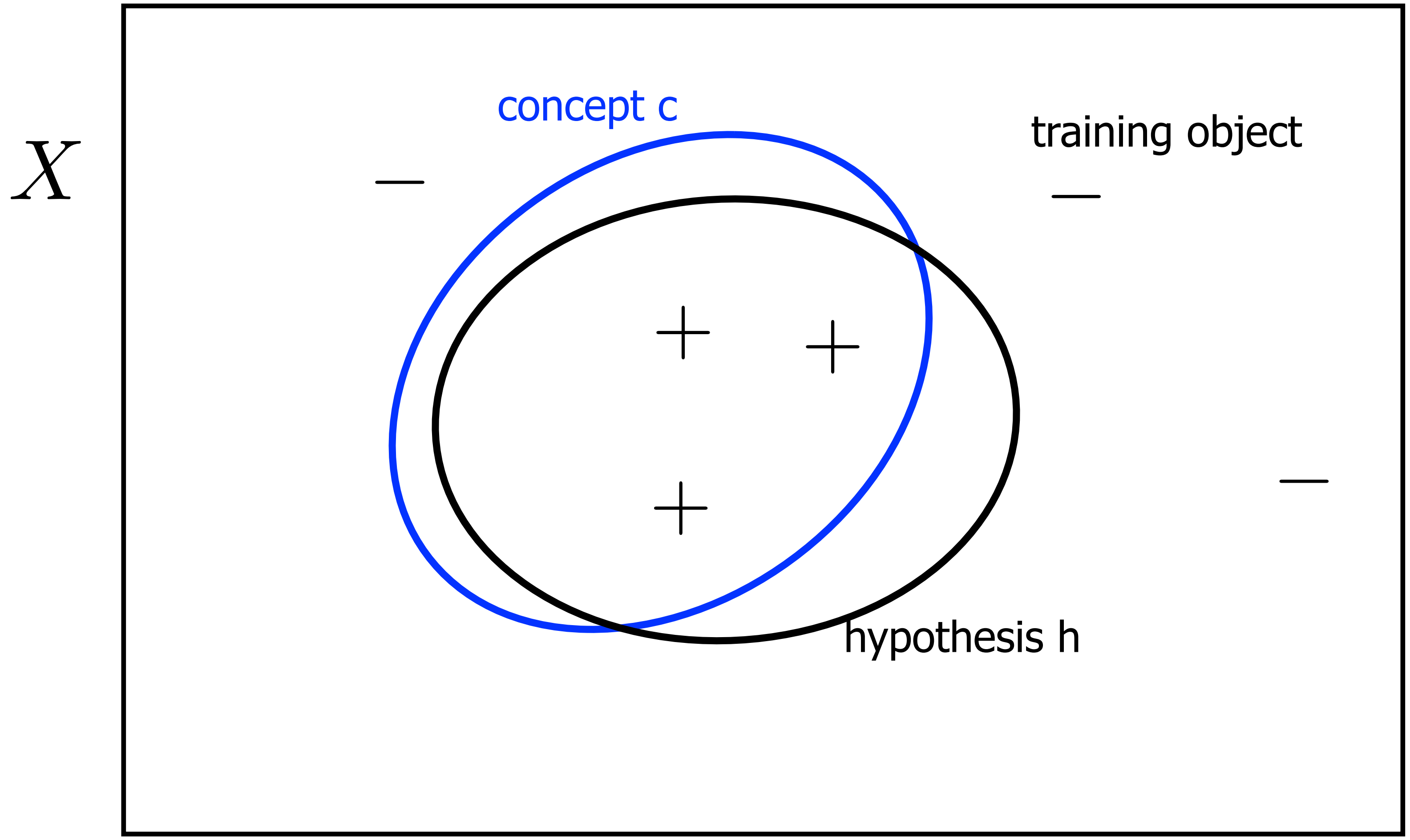
where $Pr_{x \in D}$ excludes objects used in training h.

PAC error



- There is a (hidden) concept
- There is given training data (3 positive, 3 negative)

PAC error



- Here the true error is non-zero, although h and c agree on all six training instances (training error = 0).
- How probable is it that the observed training error gives a misleading estimate of the true error?

PAC learnable

- Characterise target concepts that can be reliably learned from (1) a ‘reasonable’ number of (randomly drawn) training examples and (2) a ‘reasonable’ amount of computation.

PAC learnable

- Characterise target concepts that can be reliably learned from (1) a ‘reasonable’ number of (randomly drawn) training examples and (2) a ‘reasonable’ amount of computation.

- Sometimes we have an unlucky draw of examples
- With finite number of training examples there are hypotheses that work identical on the training examples: how to choose?
- We will not demand zero error, but an arbitrarily small error (approximately correct)
- We will not demand small error on all training sets, but that the failure is bounded (probably correct)

Probably Approximately Correct (PAC)

TU Delft
Delft University of Technology

PAC learnable

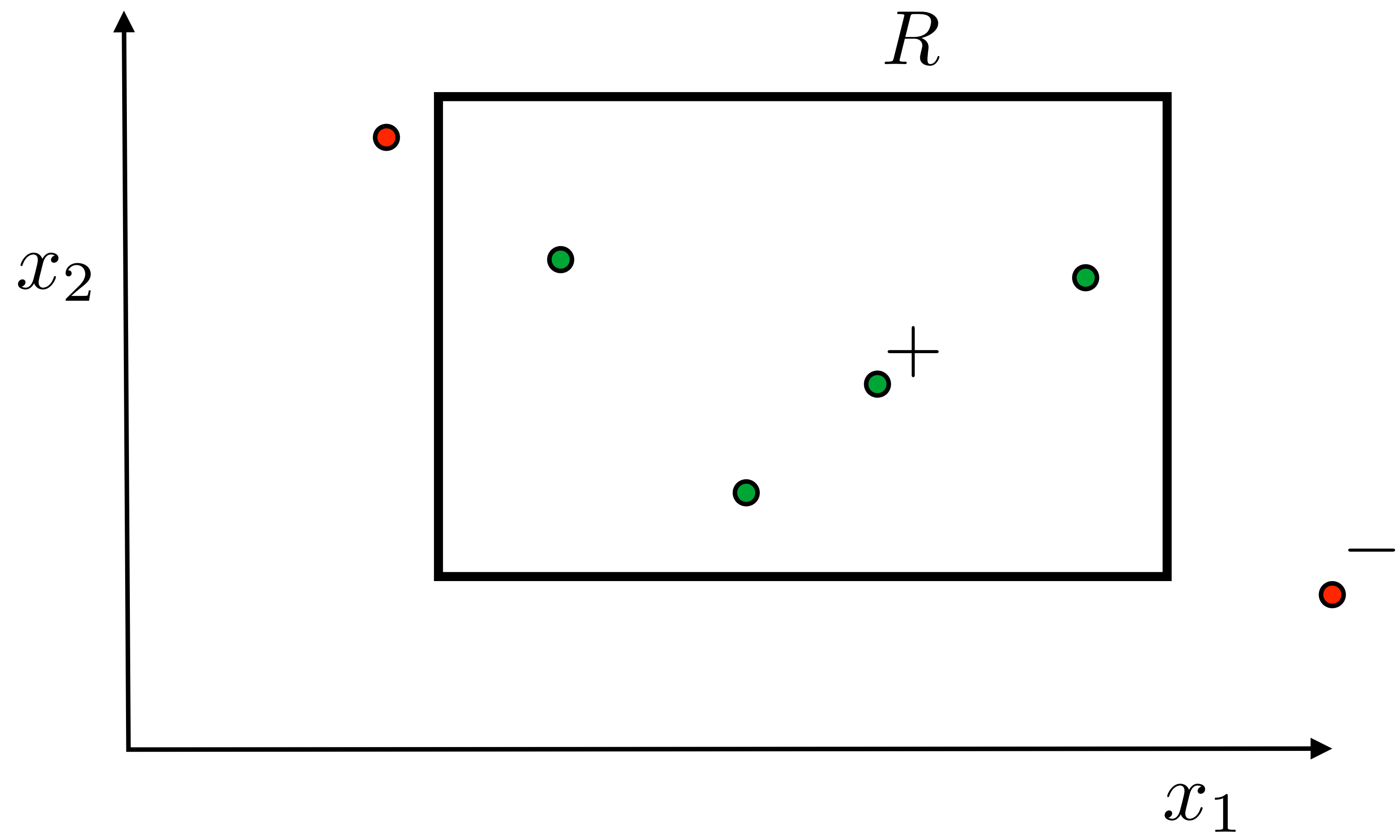
- Characterize target concepts that can be reliably learned from (1) a 'reasonable' number of (randomly drawn) training examples and (2) a 'reasonable' amount of computation.

- We will not demand small error on all training sets, but that the failure is bounded (probably correct)

  Probably Approximately Correct (PAC)

- C is PAC-learnable by L using H if for all c, distribution D, the learner L will with probability at least $(1-\delta)$output a hypothesis h such that $\mathrm{error}_D(h) \leq \varepsilon$ in time/samples that is polynomial in $1/\varepsilon$, $1/\delta$, $\mathrm{size}(c)$, dimensionality n

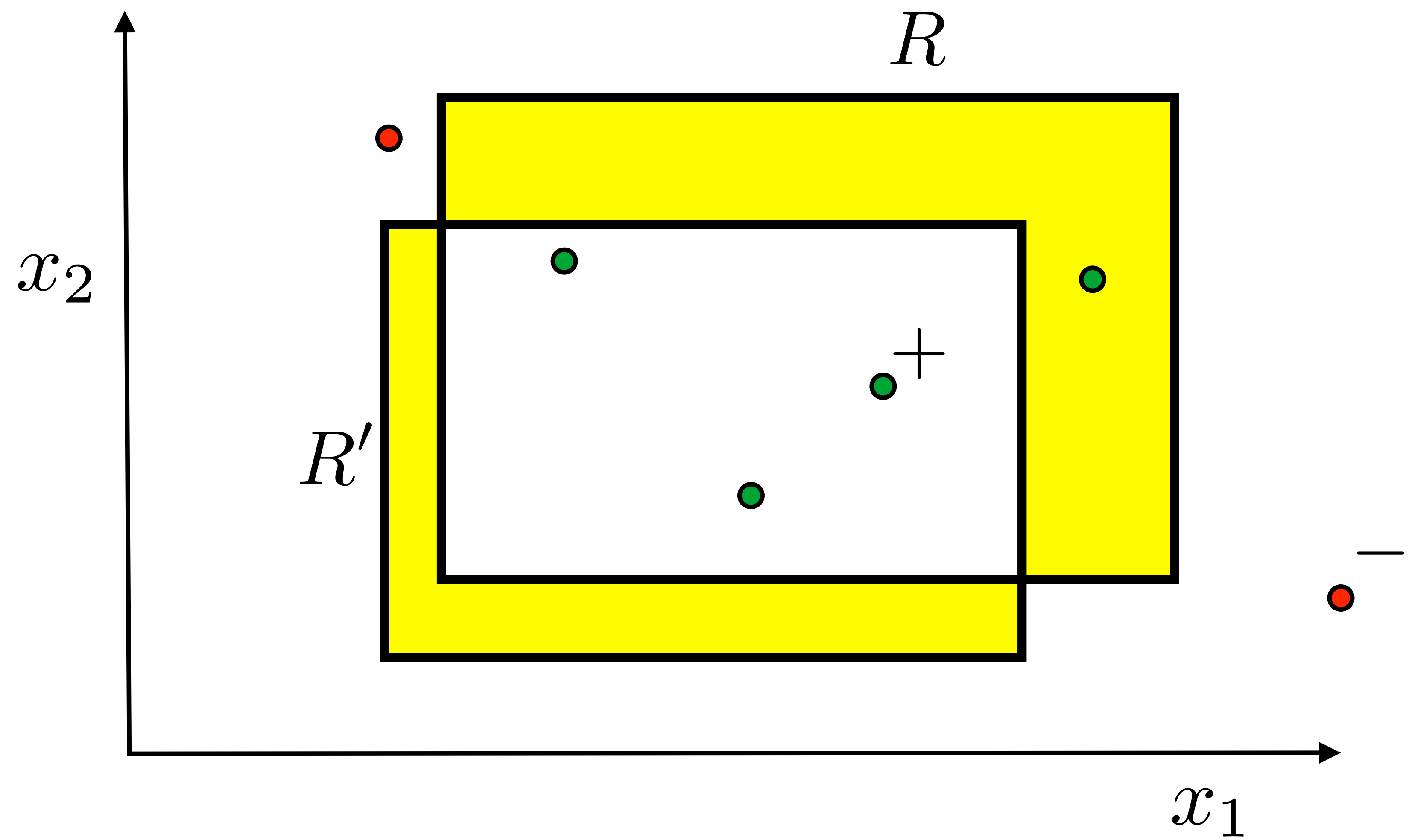
Example: rectangle learning



- Learn an axis-parallel rectangle R from + and - examples in $\mathbb{R}^2$
- Examples are randomly drawn from D
- Adapt hypothesis rectangle R’ to approximate R

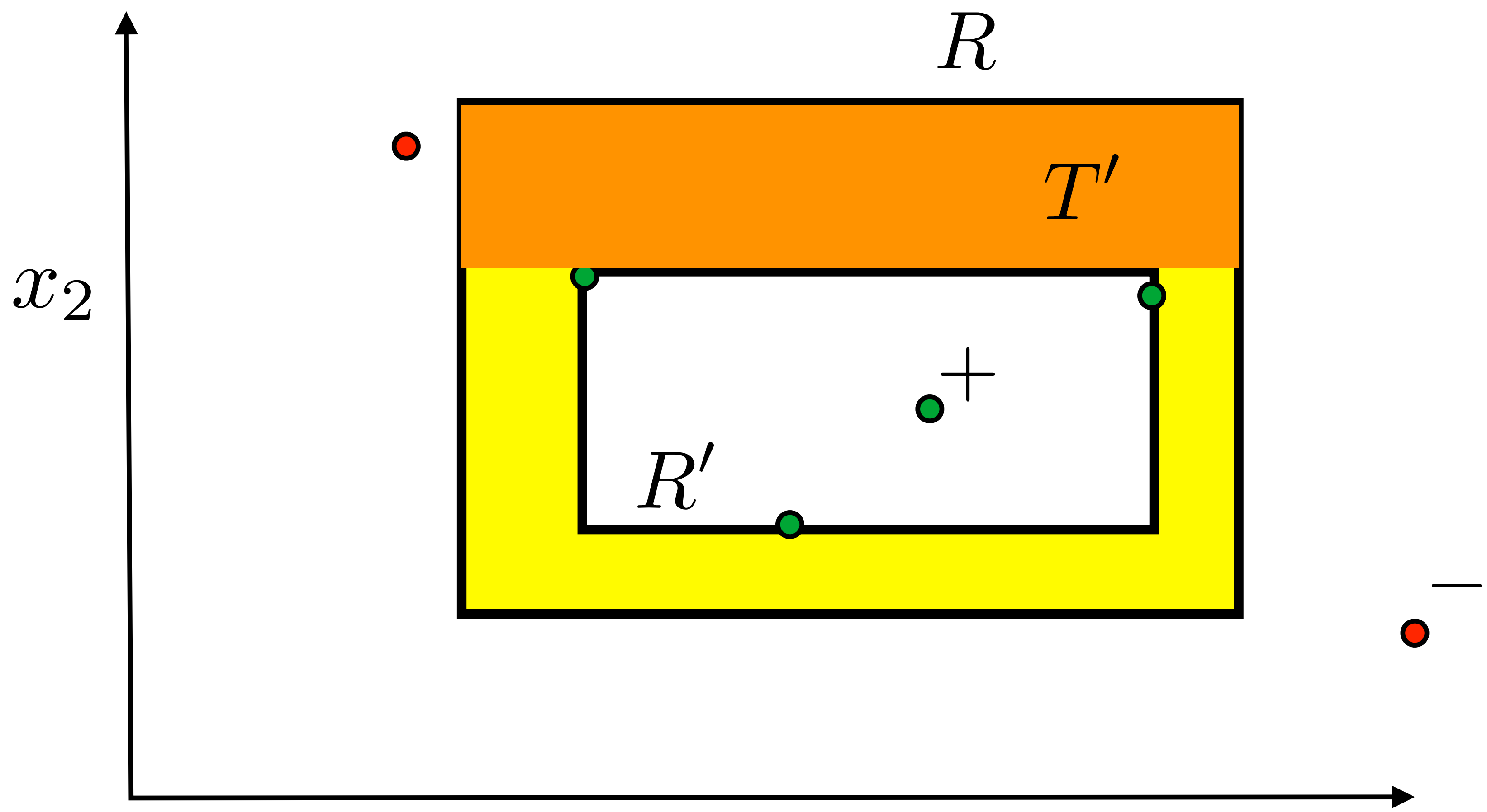
Delft University of Technology

Example: rectangle learning



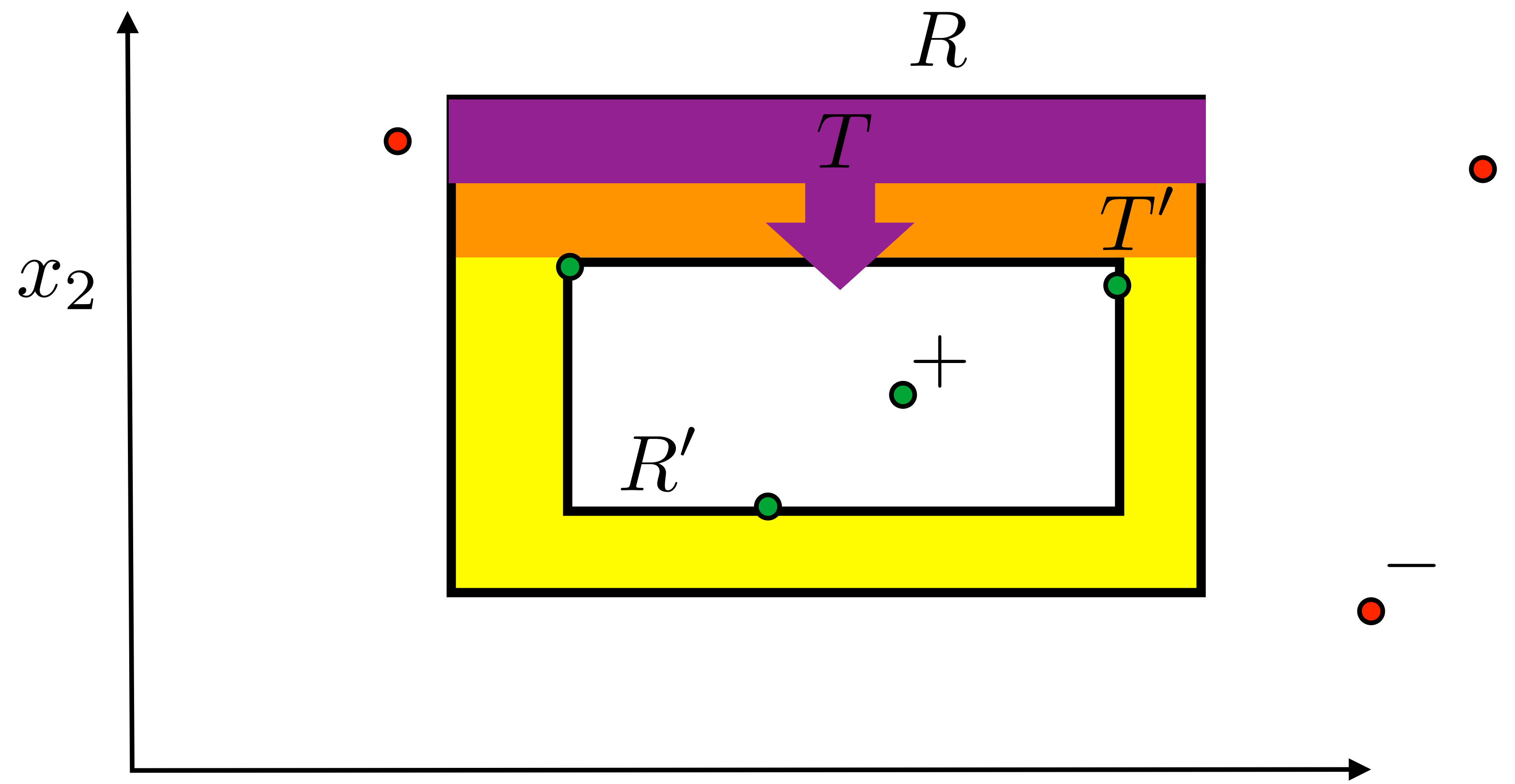
- The error of R′ is $(R - R') \cup (R' - R)$
- What learning strategy to use so we can efficiently learn it?...

Example: rectangle learning



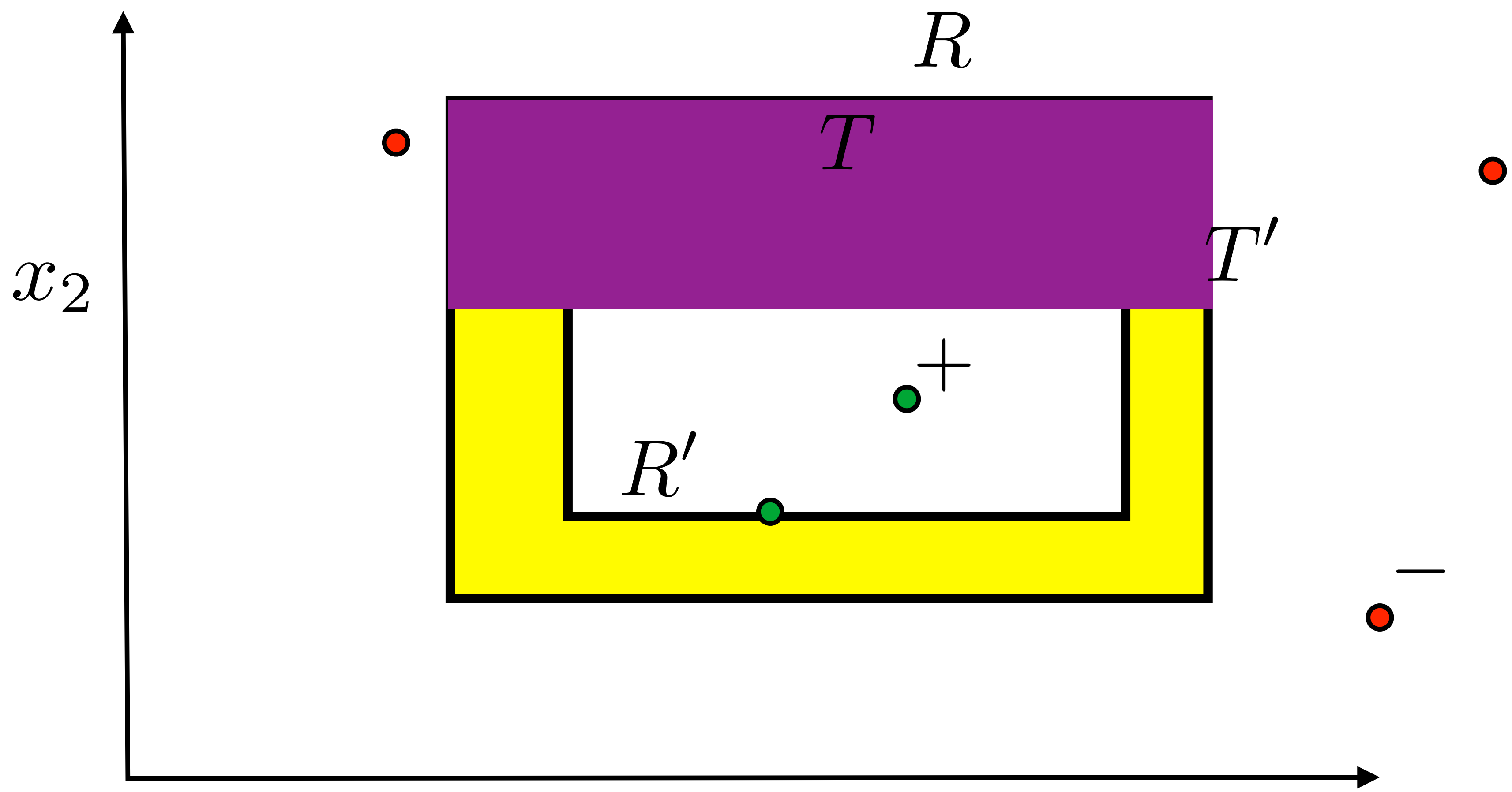
- Use the '**tightest** fit rectangle' (definition of L!): R'
- We make still an error on the test set: R' is always contained in R
- Can we analyse the error? We can split the error in four strips (shown one in orange: T').

Example: rectangle learning



- What is the prob. the learner has error larger than ε?
- Now define a new strip, T
- Strip T is 'grown' such that it covers $\varepsilon/4$ of the prob.mass (for given ε)
- Now T may cover T' or may not cover T'

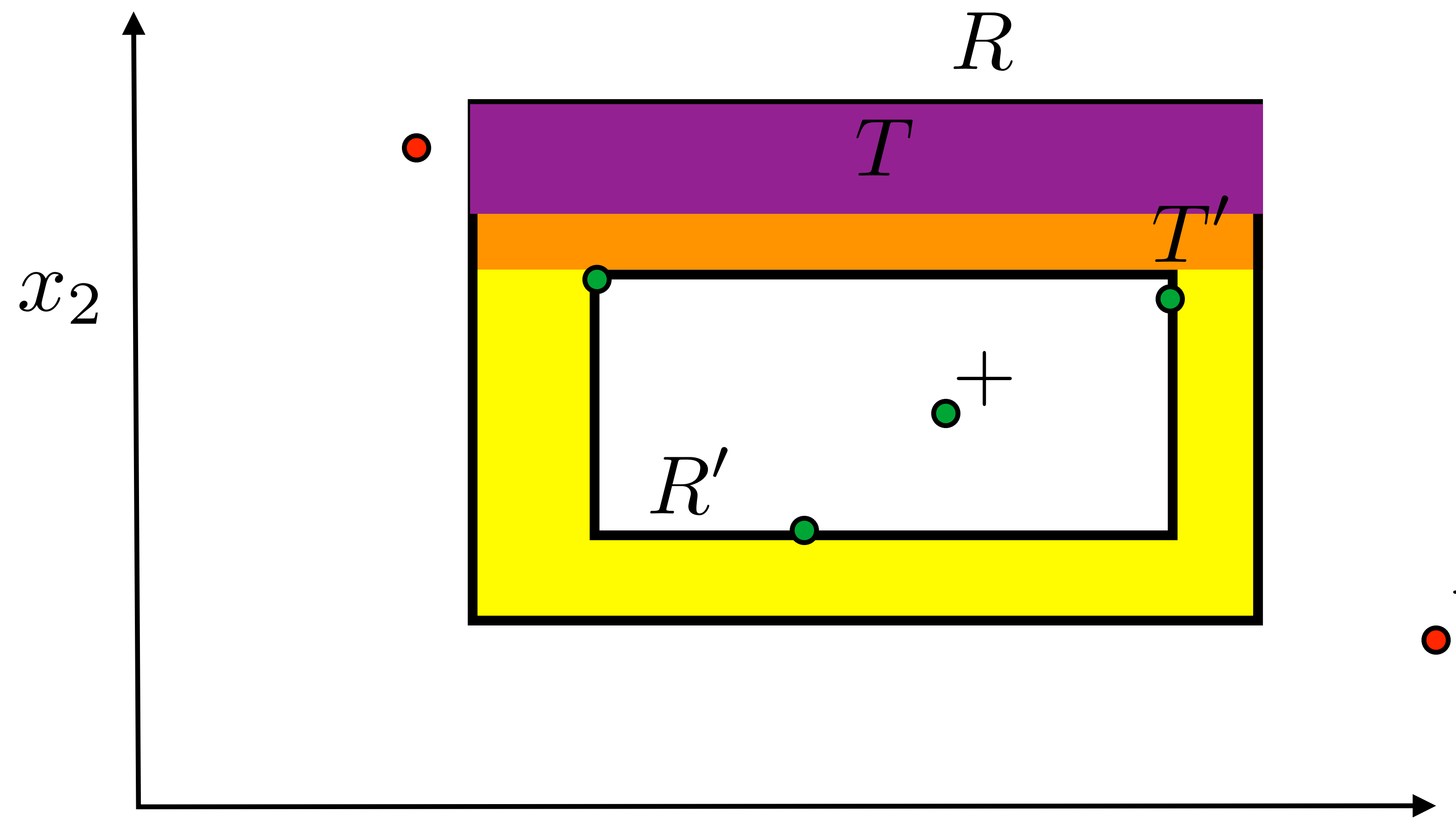
Example: rectangle learning



- If T covers T' and that holds for all strips, then the error

$$P[\text{error}] = P[\text{yellow}] \leq P[T_1] + P[T_2] + P[T_3] + P[T_4]$$
$$= 4(\varepsilon/4) = \varepsilon$$

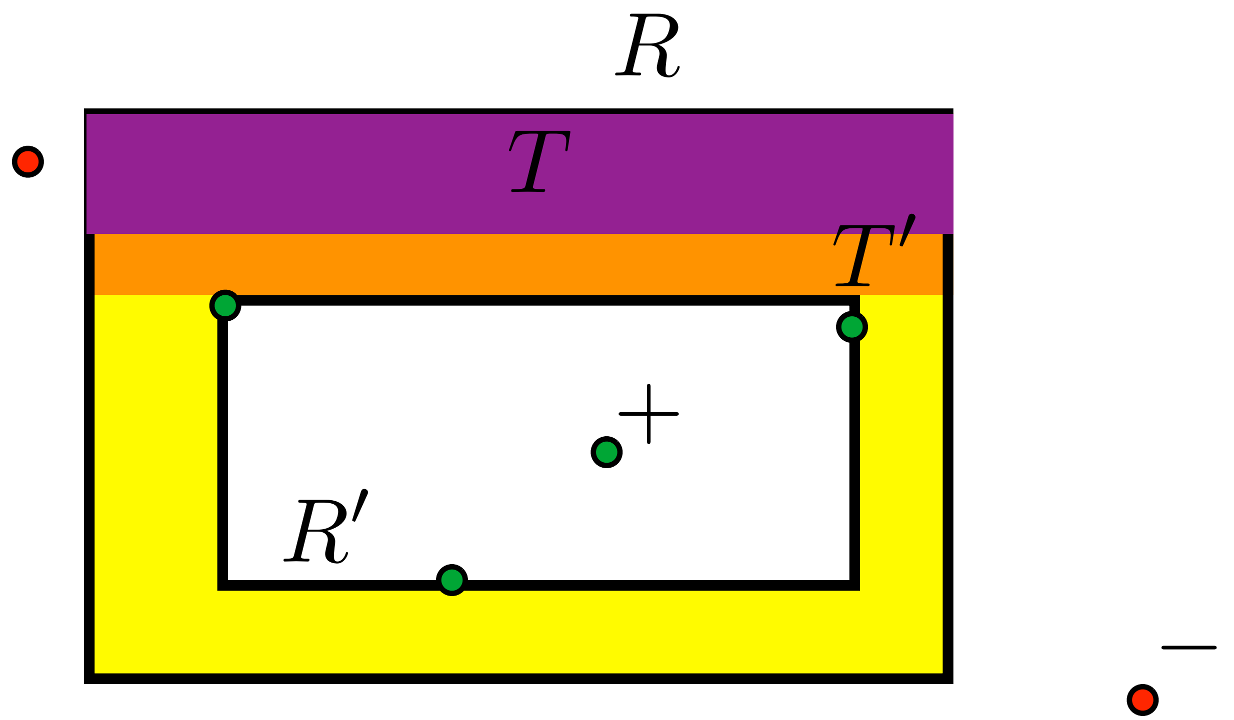
Example: rectangle learning



- Can we now estimate the probability that T does not cover T′ (that the error exceeds ε)?
- Can we show that, with sufficient number of training samples, R′ will always be so large that T covers T′? And how many training samples then?

TU Delft
Delft University of Technology

Example: rectangle learning



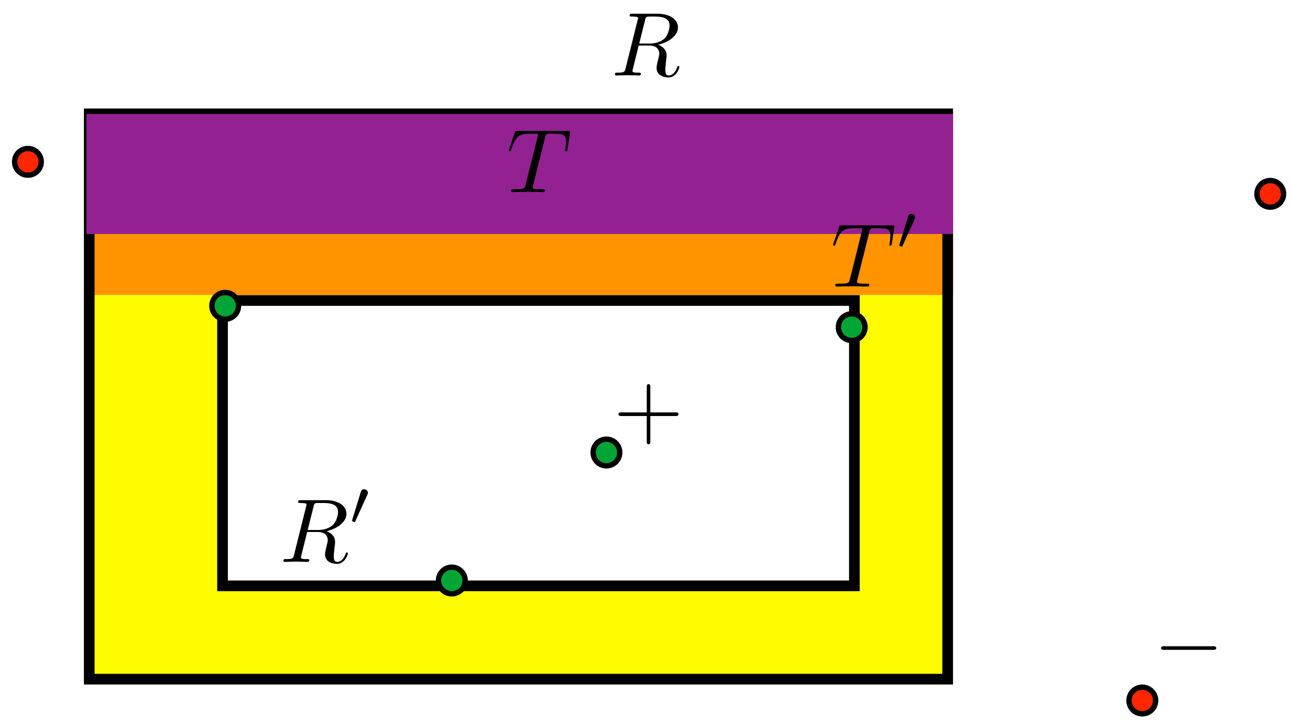
- T would have covered T' when none of the positive samples would have hit area T

$$P[\text{random}\, x\, \text{hits}\ T] = \varepsilon/4$$

$$P[\text{random}\, x\, \text{missed}\ T] = 1 - \varepsilon/4$$

$$P[m\, \text{random}\, x'\text{s}\, \text{miss}\ T] = (1 - \varepsilon/4)^m$$

Example: rectangle learning



(union bound)

- We have 4 strips, so

$$P[m\,\text{random}\,x'\text{s}\,\text{miss any}\;T\text{s}] = P[\text{miss}T_1]\text{or}P[\text{miss}T_2]\text{or}\ldots$$
$$\leq 4(1-\varepsilon/4)^m$$

- So, the probability that our R' has an error larger than ε is something we want to bound:

$$P[R'\text{ has larger error than }\varepsilon] \leq 4(1-\varepsilon/4)^m < \delta$$

Delft University of Technology

Union bound

- For a countable number of events A1,A2,… the probability that **at least one** of the events happens, is no greater than the sum of the probabilities of the individual events:

$$P(\bigcup_i A_i) = P(A_1 \text{ or } A_2 \text{ or} \ldots) \leq \sum_i P(A_i)$$

- If the events are mutually exclusive, then the equality holds

Rectangle learning

- Now we want to bound the chance that our R′ makes an error larger than ε by δ

$$4(1-\varepsilon/4)^m < \delta$$

- Now use: $$e^{-x} \geq (1-x)$$

  and we obtain: $$4e^{-m\varepsilon/4} \geq 4(1-\varepsilon/4)^m$$

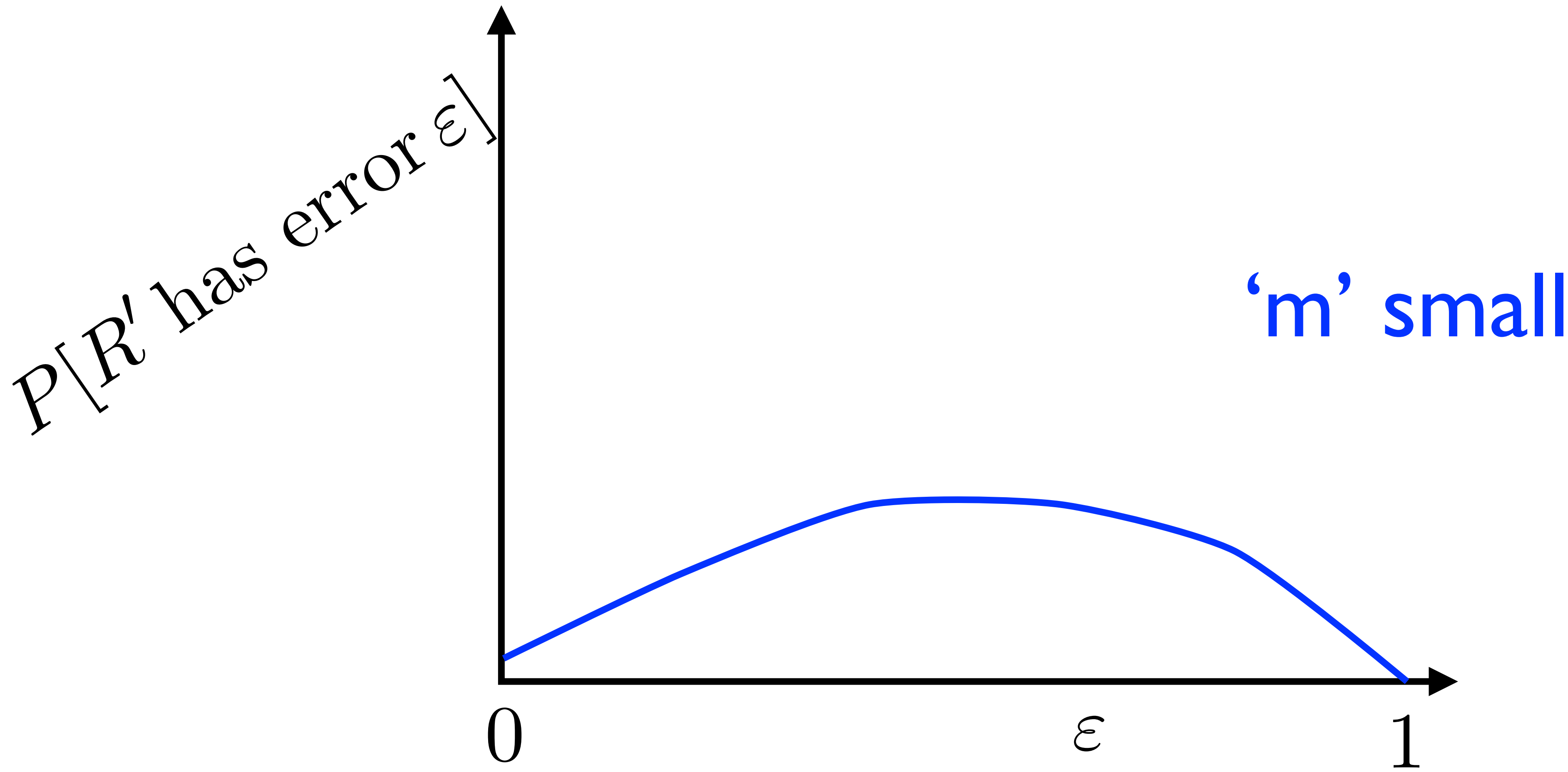
- So instead we can demand: $$4e^{-m\varepsilon/4} < \delta$$

$$-m\varepsilon/4 < \log(\delta/4)$$

$$m\varepsilon/4 > \log(4/\delta)$$

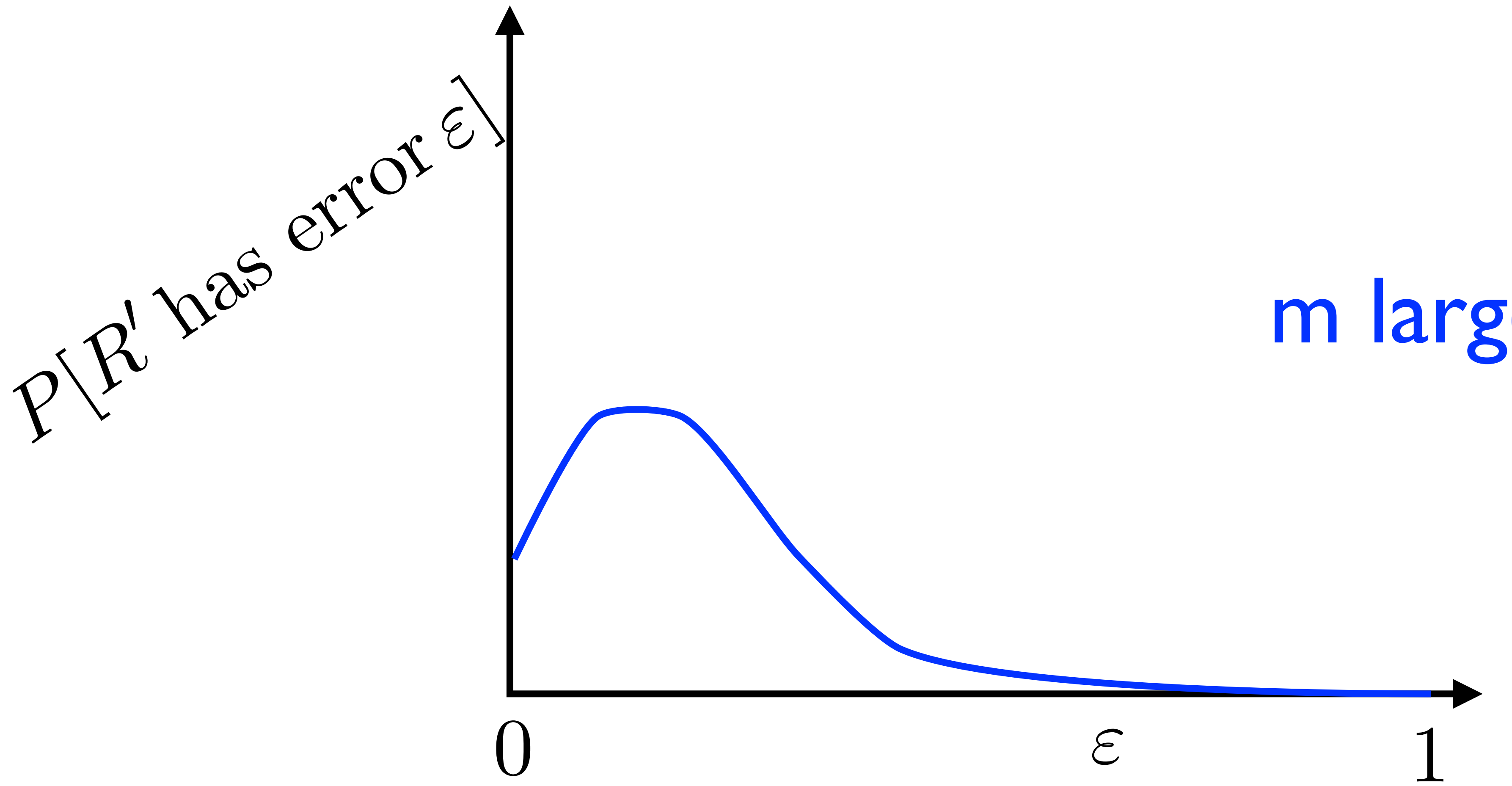
- This R′ is PAC learnable! $$m > (4/\varepsilon)\log(4/\delta)$$

More ‘interpretation’...



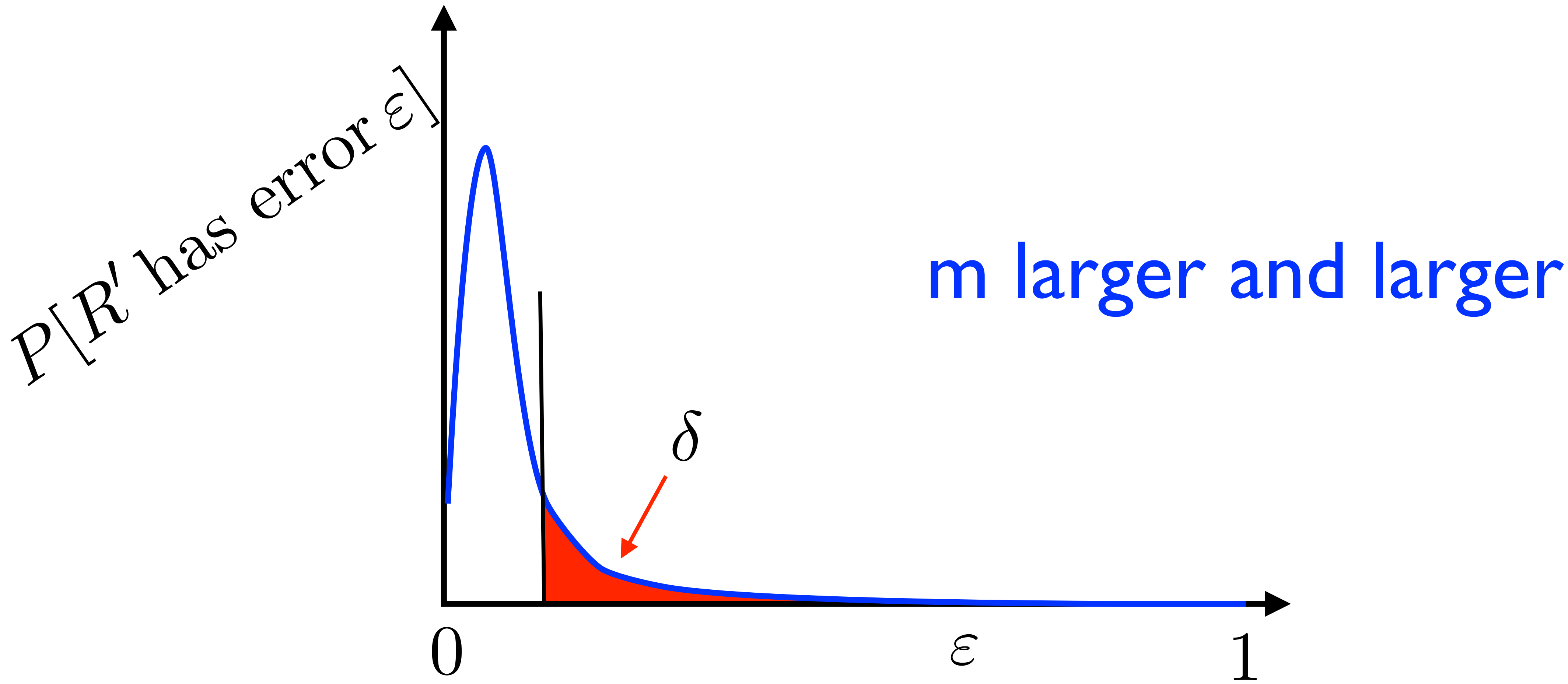
- When I get a few training samples
- then the true error of R’ may still be anything
- ... in particular when m is small.

More interpretation...



- When I get more samples, my error tends to become smaller
- But still, I may be unlucky

More interpretation...



- But still, I may be unlucky: for all errors I still have some probability δ that my classifier R' actually is worse than that (red area)

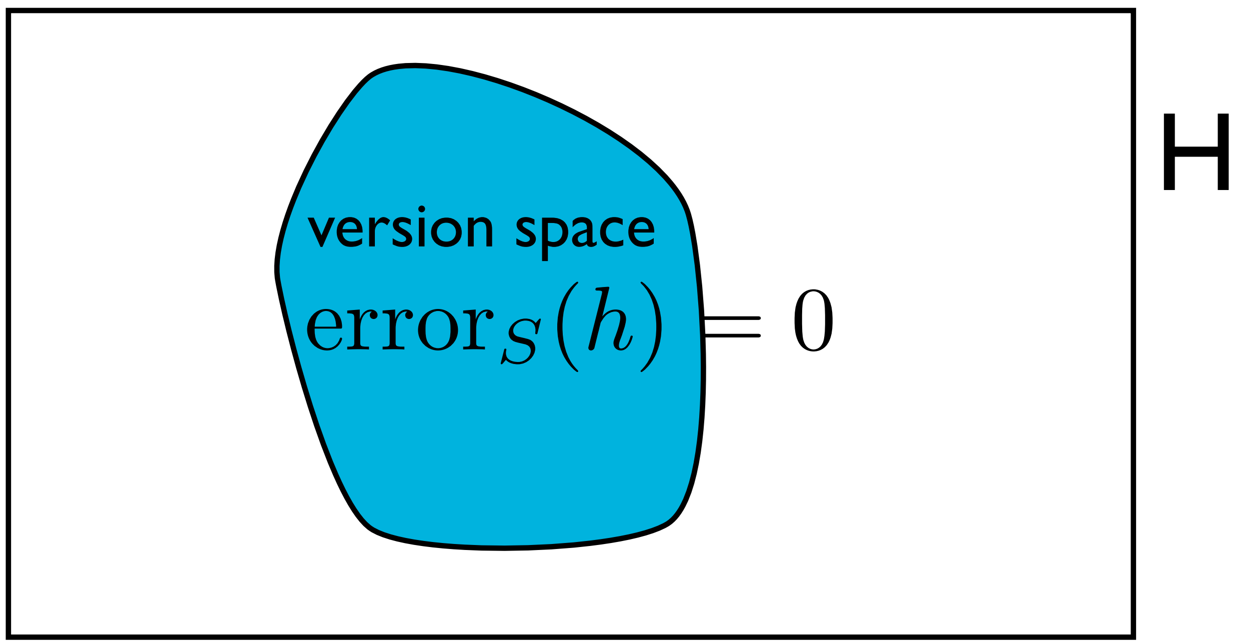
‘Conclusion’

- So the general question in Learning Theory is: How many samples m do I need such that my learner L gives a classifier with small error?
- Is the number of samples m ‘reasonable’ (i.e. not too large)?

Special case:
Discrete Hypothesis spaces
and
Consistent learners

Consistent learner

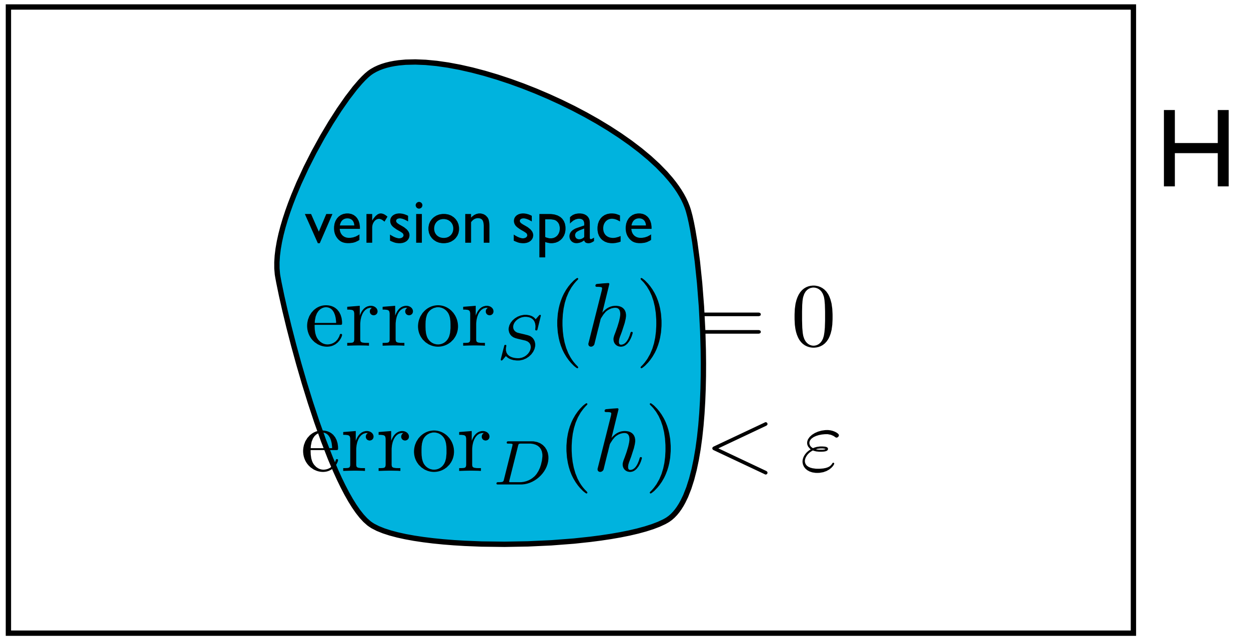
- The Version Space is the collection of all consistent hypotheses (zero error on training set S, test error can be anything)



- Consistent learner has zero error on training set

ε-exhausted version space

- The Version Space is the collection of all consistent hypotheses (zero error on training set S, test error can be anything)



- In an ε-exhausted version space, all hypotheses have an error smaller than ε on the test set. GOOD!

When not exhausted?

- What is the probability that the version space is not ε-exhausted?

- Assume there are k hypotheses with error larger than ε
- We fail to exhaust the version space if any of these hypotheses is consistent with our training sample (with m training objects)
- The probability that a hypothesis with error larger than ε is consistent with m objects is at most $(1-\varepsilon)^m$

The probability is…

- Given k hypotheses with error $> \varepsilon$, the probability that at least one of them is consistent with all m training examples is at most: $k(1-\varepsilon)^m$ (union bound)

- Obviously $k \leq |H|$ and using $(1-x) \leq e^{-x}$

$$k(1-\varepsilon)^m \leq |H|(1-\varepsilon)^m \leq |H|e^{-\varepsilon m}$$

Which means...

- Given k hypotheses with error $> \varepsilon$, the probability that at least one of them is consistent with all m training examples is at most: $k(1-\varepsilon)^m$

- Obviously $k \leq |H|$ and using $(1-x) \leq e^{-x}$

$$k(1-\varepsilon)^m \leq |H|(1-\varepsilon)^m \leq |H|e^{-\varepsilon m}$$

- So when we want to bound the chance of having a failure: $|H|e^{-\varepsilon m} \leq \delta$

  we need: $m \geq \frac{1}{\varepsilon}(\ln|H| + \ln(1/\delta))$

Delft University of Technology

Consistent learners

- We found a very general bound for ANY consistent learner: $$m \geq \frac{1}{\varepsilon}\left(\ln|H| + \ln(1/\delta)\right)$$
- It depends on the (log of the) size of the hypothesis space

- This number m of training examples is sufficient to assure that any consistent hypothesis will be probably (with prob. $(1-\delta)$) approximately (within error ε) correct.
- Note that we assumed consistent algorithms: zero training error in a discrete feature space...

VC-dimension

- We discussed discrete feature spaces and hypothesis spaces with zero class overlap (the learner can perfectly learn the concept)
- Inconsistent learners are also possible (weak learners, later in lecture): the bounds gets less tight
- More class overlap is possible, but too much for this lecture...

- What if we use **continuous** feature/hypotheses spaces?
- We have seen it in the Pattern Recognition course: Vapnik-Chervonenkis dimension

Bounding the true error (Copied from the PR course)

With probability at least $1-\eta$ the inequality holds:

$$\varepsilon \leq \varepsilon_A + \frac{\mathcal{E}(N)}{2}\left(1+\sqrt{1+\frac{\varepsilon_A}{\mathcal{E}(N)}}\right)$$

where

$$\mathcal{E}(N) = 4\frac{h(\ln(2N/h)+1)-\ln(\eta/4)}{N}$$

V.Vapnik, Statistical learning theory, 1998

- When h is small, the true error is close to the apparent error

VC-dimension and samples

- When you have the VC-dimension of a learner L, then holds:

$$m \geq c_0 \left(\frac{1}{\varepsilon} \log \frac{1}{\delta} + \frac{h}{\varepsilon} \log \frac{1}{\varepsilon} \right)$$

(similar to the discrete feature space)

(caused by the continuous feature space)

- This VC-dimension is the analogue of $|H|$
- Similarly, also lower bounds on the number of training samples can be given.
- Only bounds/approximations on the VC-dimension are known for most classifiers

‘No Free Lunch Theorem’

- There are **no** context-independent reasons to favor one learner over another (Wolpert, 1996)

- Averaged over all possible problems, all learners have the same average performance
- So when one algorithm seems to outperform another, it just fits the problem better, and it does not mean that the one algorithm is inherently better
- Claims in literature that this procedure/ algorithm performs ‘best’ overall should be considered with some care...

No free lunch...

- Assume we have a discrete feature space (with size $|X|$), then we need for a consistent learner that the number of samples m:

$$m \geq \frac{1}{\varepsilon} \left(\ln |H| + \ln(1/\delta) \right)$$

- But assume then that **ALL** possible hypotheses are allowed:

$$|H| = 2^{|X|}$$

- For a discrete binary feature space with n features:

$$|X| = 2^n$$

- The number of training examples grows exponentially:

$$m \geq \frac{1}{\varepsilon} \left(2^n \ln 2 + \ln(1/\delta) \right)$$

NOT PAC learnable

Conclusions

- General statements can be made about the number of required training objects, but additional (strong) assumptions on the feature space or hypothesis space have to be made
- Bounds can be given, but are often very loose (and not always easy to interpret)

- Sometimes constructive algorithms are invented (AdaBoost, Support Vector Machines)
- Averaged over all problems, all methods are equally good ... ...

Weak/strong learners

- PAC learning requires that the error ε can be arbitrarily small, and the confidence $1-\delta$ can be set arbitrarily high.
- What if we have a **weak** learner that has a **fixed** error ε_0 and confidence $1-\delta_0$?

- Magically, it appears that there is an algorithm that can use the weak learner to boost it to a full PAC learner (a **strong** learner)
- It also means that PAC learning is very general: the demands on the learner do not have to be that strict (you can always boost it)

Original boosting

- The original idea: resample original data D such that more difficult objects are used to train 2 additional (weak) learners
- The collection of (3) weak learners predict the final label by majority voting

- Later versions do not resample the training set, but split the feature space, or introduce other combinations of weak learners

AdaBoost

- Inspired by boosting a weak classifier to a strong one: Adaptive Boosting
- My explanation starts from assumptions on (1) the model, and (2) the error function. The (PAC) theory is not needed in the derivation.

- Assumption 1: the model is linear additive:

$$F_K(\mathbf{x}) = \sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}) + \alpha_K f_K(\mathbf{x})$$

where

$$f_i(\mathbf{x}) = \pm 1$$

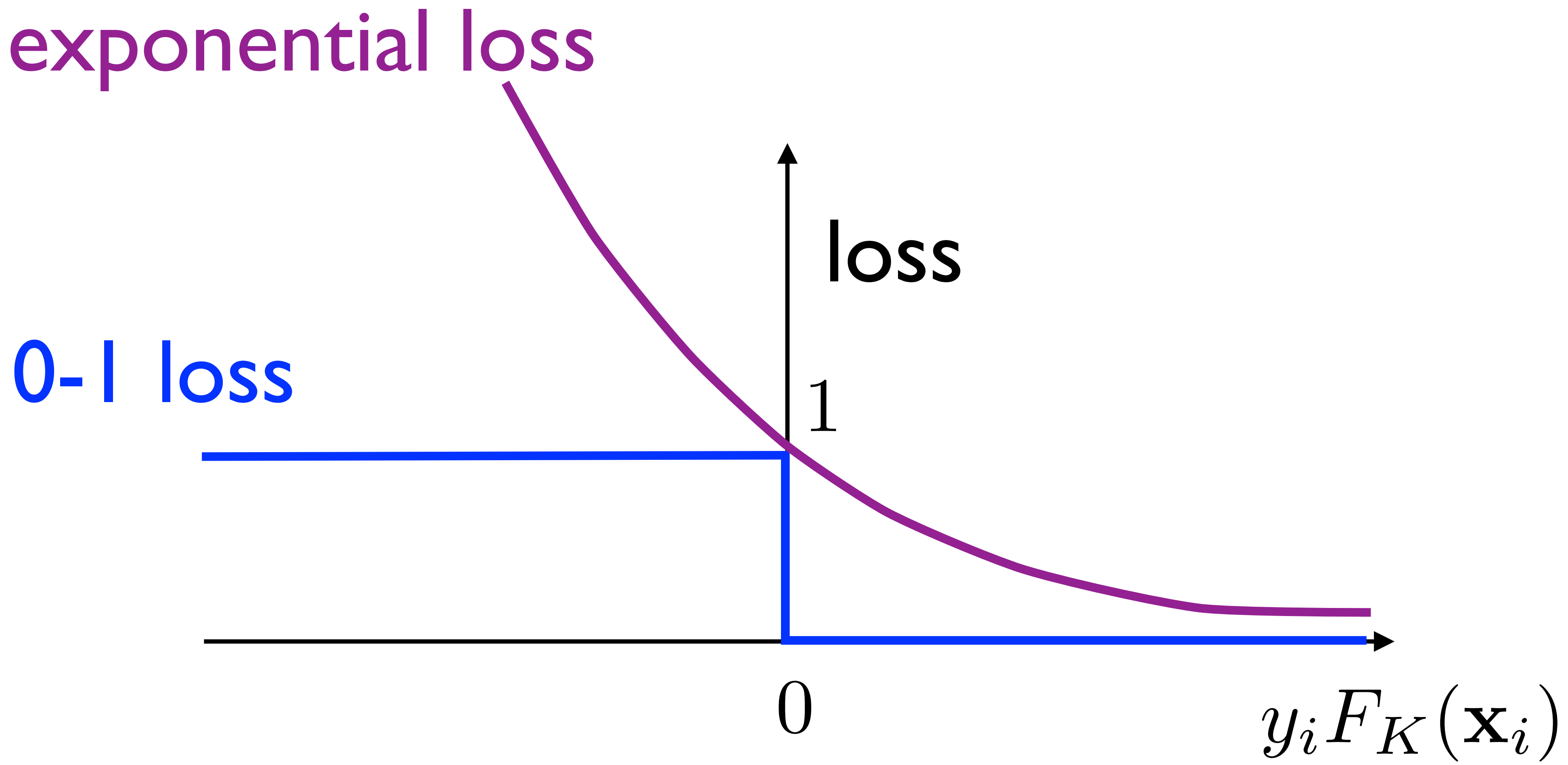
(binary outputs!)

and α_i are weights.

AdaBoost

- Assumption 2: the loss/error on a training set is measured by:

$$L = \sum_{i=1}^{N} \exp\left(-y_i F_K(\mathbf{x}_i)\right)$$



Delft University of Technology

AdaBoost

- To optimize both the weak classifiers $f_i(\mathbf{x})$ and the weights α_i is an open problem
- Instead, do it incrementally:

$$F_K(\mathbf{x}) = \sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}) + \alpha_K f_K(\mathbf{x})$$

AdaBoost

- To optimize both the weak classifiers $f_i(\mathbf{x})$ and the weights α_i is an open problem
- Instead, do it incrementally:

$$F_K(\mathbf{x}) = \sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}) + \alpha_K f_K(\mathbf{x})$$

fix this

optimize these two terms

AdaBoost

- To optimize both the weak classifiers $f_i(\mathbf{x})$ and the weights α_i is an open problem
- Instead, do it incrementally:

$$F_K(\mathbf{x}) = \underbrace{\sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x})}_{\text{fix this}} + \underbrace{\alpha_K f_K(\mathbf{x})}_{\text{optimize these two terms}}$$

fix this

optimize these two terms

- Minimize L:

$$L = \sum_{i=1}^{N} \exp\left(-y_i \left[\sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}_i) + \alpha_K f_K(\mathbf{x}_i)\right]\right)$$

AdaBoost

$$
\begin{aligned}
L &= \sum_{i=1}^{N} \exp\left(-y_i \left[\sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}_i) + \alpha_K f_K(\mathbf{x}_i)\right]\right) \\
&= \sum_{i=1}^{N} \underbrace{\exp\left(-y_i \sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}_i)\right)}_{w_i} \exp\left(-y_i \alpha_K f_K(\mathbf{x}_i)\right) \\
&= \sum_{i=1}^{N} w_i \exp\left(-y_i \alpha_K f_K(\mathbf{x}_i)\right)
\end{aligned}
$$

- Now distinguish correctly and incorrectly classified objects:

$$
\begin{aligned}
y_i f_K(\mathbf{x}_i) = 1 \quad &\rightarrow \mathbf{x}_i \in C_K \quad \text{(correct)} \\
y_i f_K(\mathbf{x}_i) = -1 \quad &\rightarrow \mathbf{x}_i \in W_K \quad \text{(wrong)}
\end{aligned}
$$

TU Delft
Delft University of Technology

AdaBoost

$$
\begin{aligned}
L &= \sum_{i=1}^{N} w_i \exp(-\alpha_K y_i f_K(\mathbf{x}_i)) && \text{(definition)} \\
&= \sum_{C_K} w_i \exp(-\alpha_K) + \sum_{W_K} w_i \exp(\alpha_K) && \text{(previous page)} \\
&= \sum_{i=1}^{N} w_i \exp(-\alpha_K) - \sum_{W_K} w_i \exp(-\alpha_K) + \sum_{W_K} w_i \exp(\alpha_K) && \text{(sum over correct = sum over N - sum over wrongs)} \\
&= \sum_{i=1}^{N} w_i \exp(-\alpha_K) + \sum_{W_K} w_i \left(\exp(\alpha_K) - \exp(-\alpha_K)\right) && \text{(rearrange)} \\
&= \sum_{i=1}^{N} w_i \exp(-\alpha_K) + \sum_{i=1}^{N} w_i \left(\exp(\alpha_K) - \exp(-\alpha_K)\right) \mathcal{I}(f_K(\mathbf{x}_i) \neq y_i) && \text{(replace sum over wrongs by sum over N)}
\end{aligned}
$$

Delft University of Technology

AdaBoost

- To minimize w.r.t. f_K

$$L = \sum_{i=1}^{N} w_i \exp(-\alpha_K) + \sum_{i=1}^{N} w_i \left(\exp(\alpha_K) - \exp(-\alpha_K)\right) \mathcal{I}(f_K(\mathbf{x}_i) \neq y_i)$$

we should minimize $\varepsilon_K = \sum_{i=1}^{N} w_i \mathcal{I}(f_K(\mathbf{x}_i) \neq y_i)$

- Or, in other words, we should find a classifier f_K that minimizes the error where each object is re-weighted by:

$$w_i = \exp\left(-y_i \sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}_i)\right)$$

(how bad was $\mathbf{x}_i$ classified by the previous F_{K-1})

AdaBoost

- Ok, so the classifier should minimize the weighted error, what about the weight α_K?
- Take derivative of the loss with respect to α_K and set it to zero:

$$\frac{\partial L}{\partial \alpha_K} = -\exp(-\alpha_K) \sum_{i=1}^{N} w_i + (\exp(\alpha_K) + \exp(-\alpha_K))\, \varepsilon_K = 0$$

where: $$\varepsilon_K = \sum_{i=1}^{N} w_i \mathcal{I}(f_K(\mathbf{x}_i) \neq y_i)$$

- Solving it:

$$\sum_{i=1}^{N} w_i = (\exp(2\alpha_K) + 1)\, \varepsilon_K$$

$$\alpha_K = \frac{1}{2} \log \left(\frac{\sum_i w_i}{\varepsilon_K} - 1 \right)$$

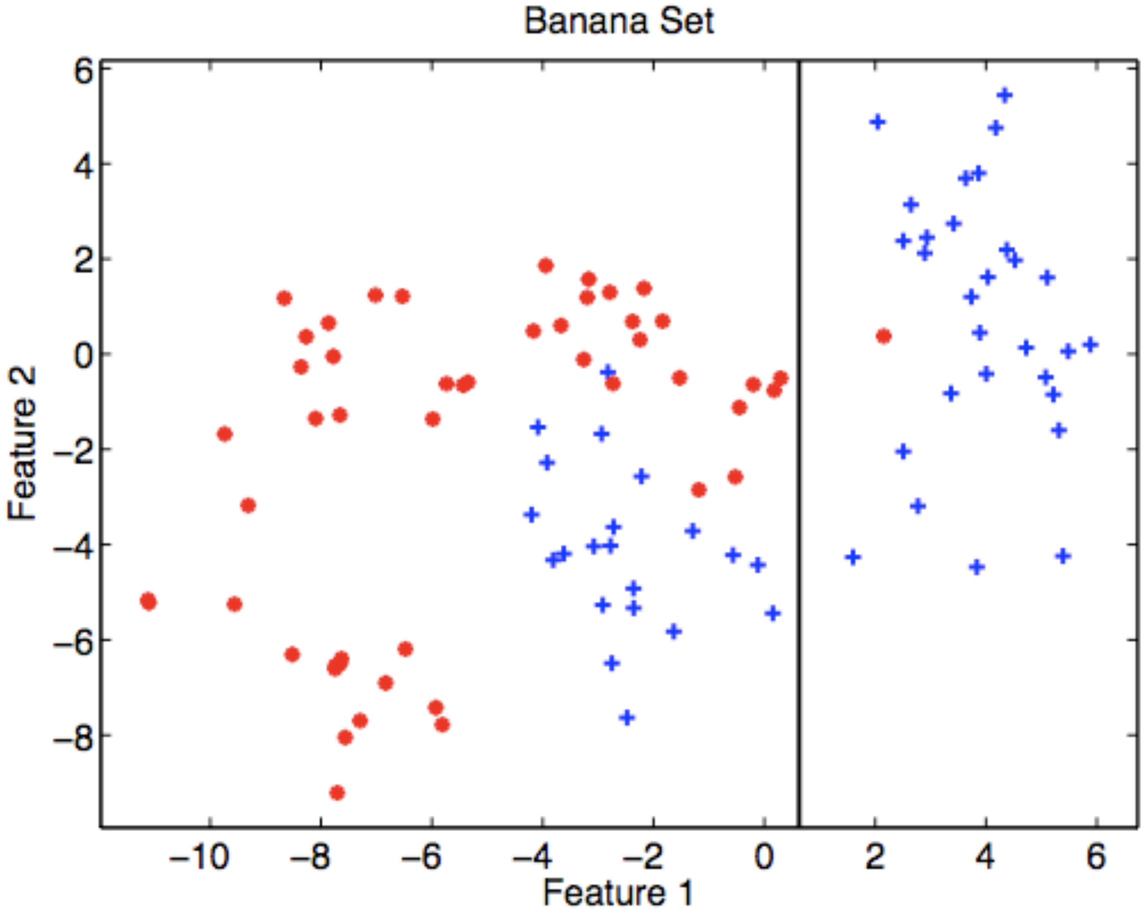
AdaBoost

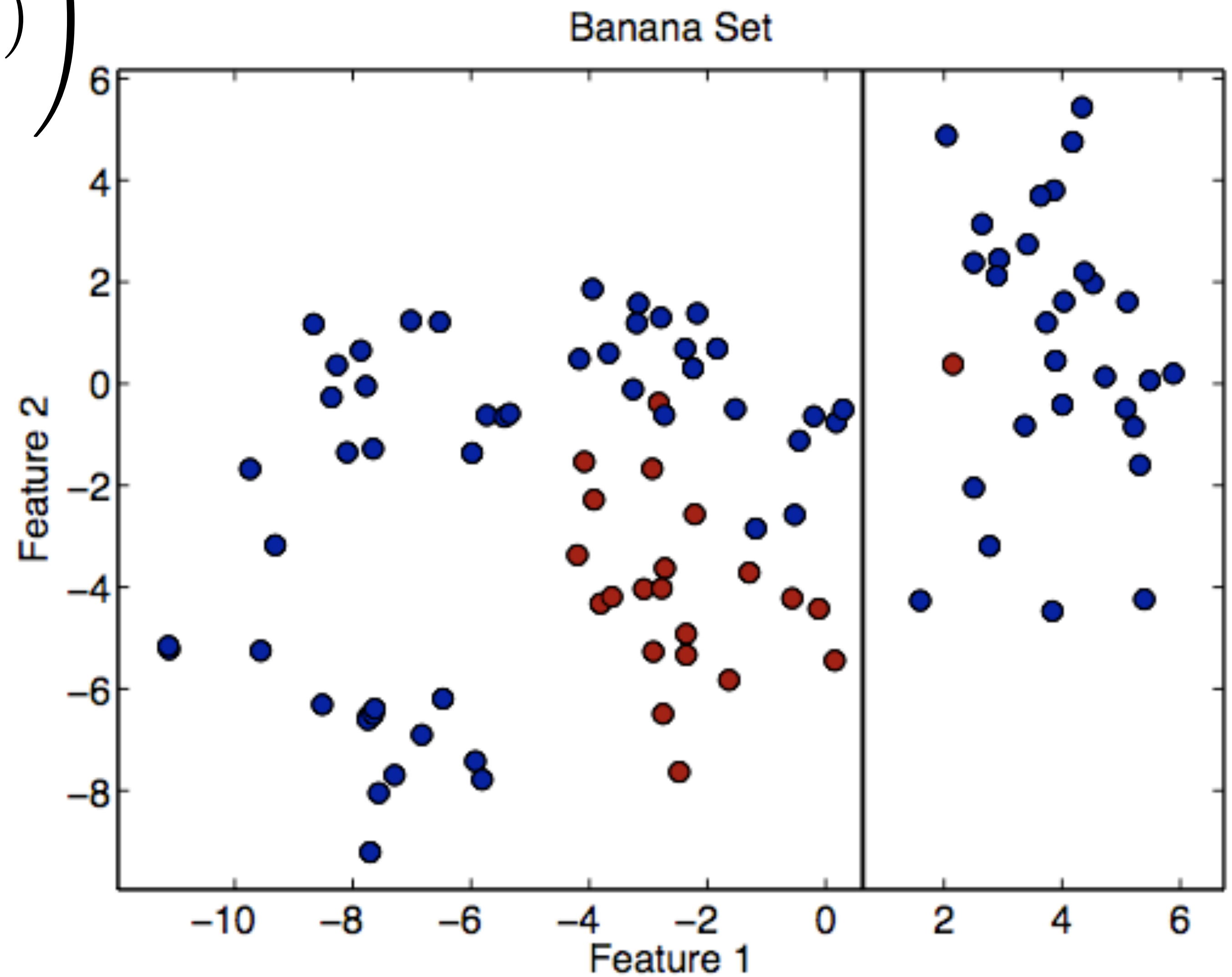
1. Give each object a weight $w_i = 1$
2. Train a classifier that minimizes the weighted error:
$$\varepsilon_K = \sum_{i=1}^{N} w_i \mathcal{I}(f_K(\mathbf{x}_i) \neq y_i)$$
3. Compute the weight of the classifier:
$$\alpha_K = \frac{1}{2} \log \left(\frac{\sum_i w_i}{\varepsilon_K} - 1 \right)$$
4. Compute the new object weights:
$$w_i = \exp \left(-y_i \sum_{k=1}^{K} \alpha_k f_k(\mathbf{x}_i) \right)$$
5. If K not large enough, go to 2, else we're done:
$$\boxed{F_K(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k f_k(\mathbf{x})}$$

AdaBoost

- Use a simple decision stump for weak classifier
- Compute α_K and reweigh each object using

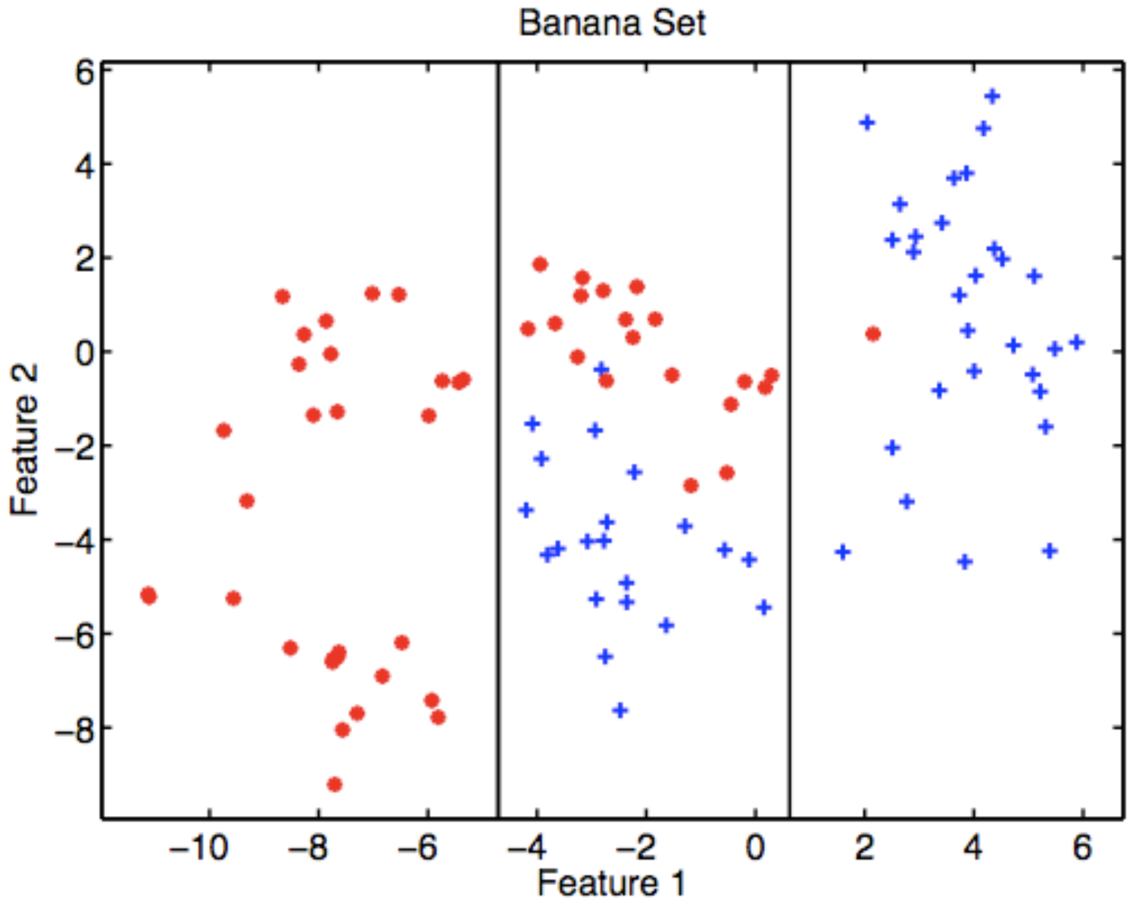
$$w_i = \exp\left(-y_i \sum_{k=1}^{K-1} \alpha_k f_k(\mathbf{x}_i)\right)$$

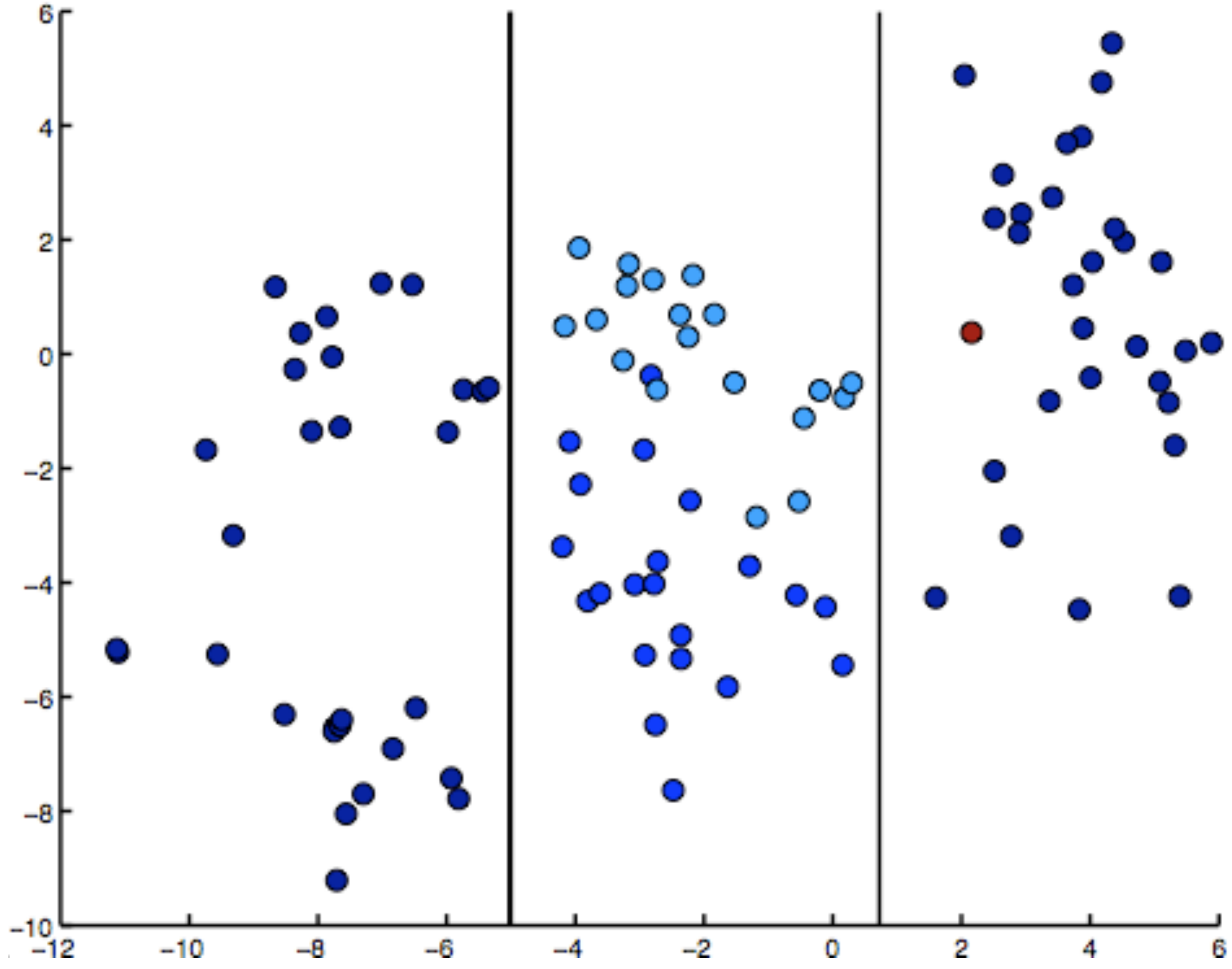




AdaBoost

- Train a new decision stump on reweighted objects
- Recompute α_K and w_i
- Repeat and repeat...

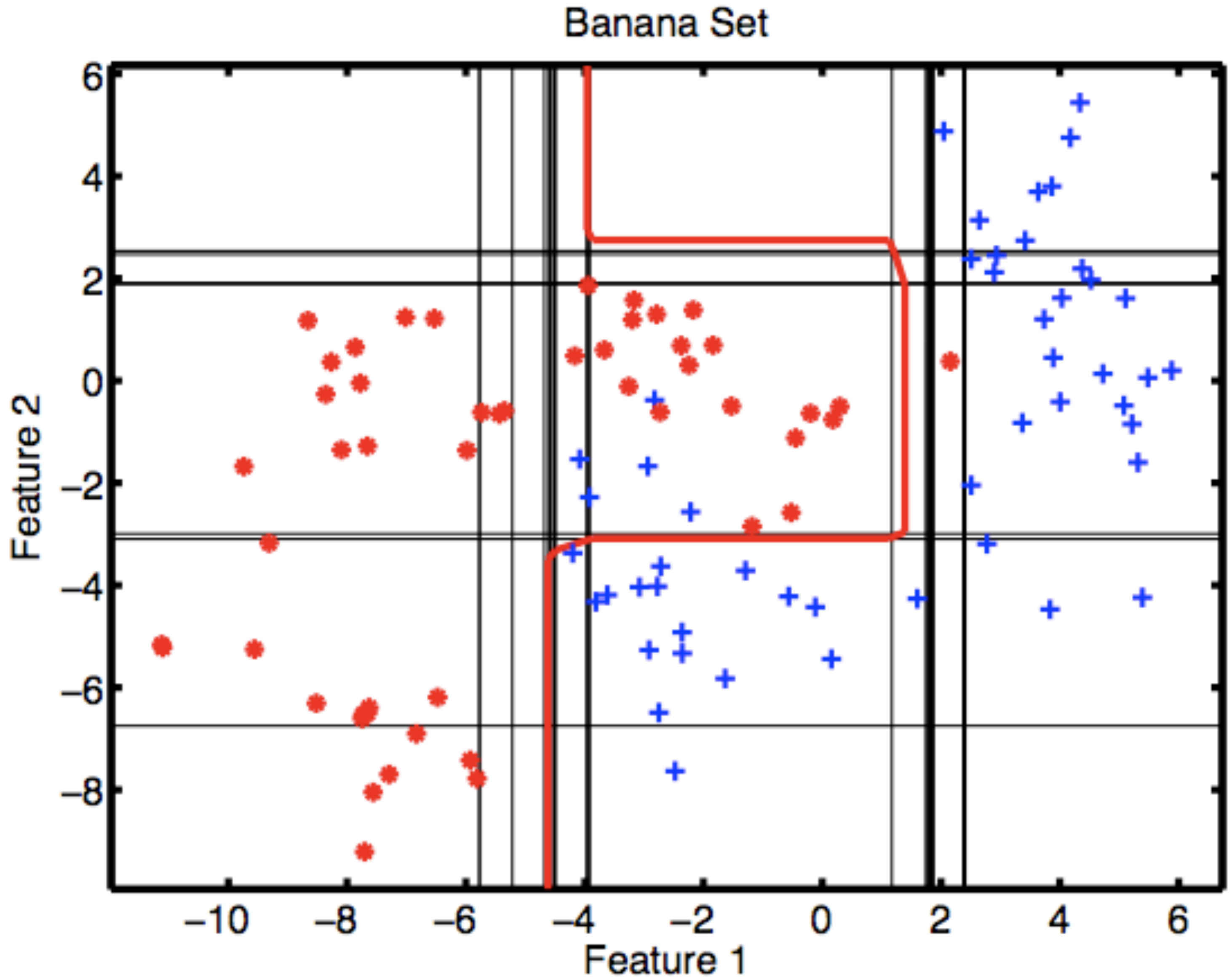




TU Delft
Delft University of Technology

AdaBoost

- Finally, we end up with:



$$F_K(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k f_k(\mathbf{x})$$

Conclusions

- PAC learning: theoretical bounds on errors, required sample sizes
- Bounds are hard to get, and not very tight
- Useful by-product: AdaBoost