Ensemble Learning

Roberto Esposito
Dipartimento di Informatica
Università di Torino

Acknowledgements

Most of the material is based on Nicholaus Kushmerick's slides.
You can find his original slideshow at:
www.cs.ucd.ie/staff/nick/home/COMP-4030/L14,15.ppt

Several pictures are taken from the slides by Thomas Dietterich.
You can find his original slideshow (see slides about Bias/Variance theory) at:
http://web.engr.oregonstate.edu/~tgd/classes/534/index.html

Reading

- Dietterich: **Ensemble methods in machine learning** (2000).
- Schapire: **A brief introduction to boosting** (1999). *[Sec 1-2, 5-6]*
- Dietterich & Bakiri: **Solving multiclass learning problems via error-correcting output codes** (1995). *[Skim]*

Agenda

- 1. What is ensemble learning
- 2. Bagging
- 3. Boosting
- 4. Error-correcting output coding
- 5. Why does ensemble learning work?

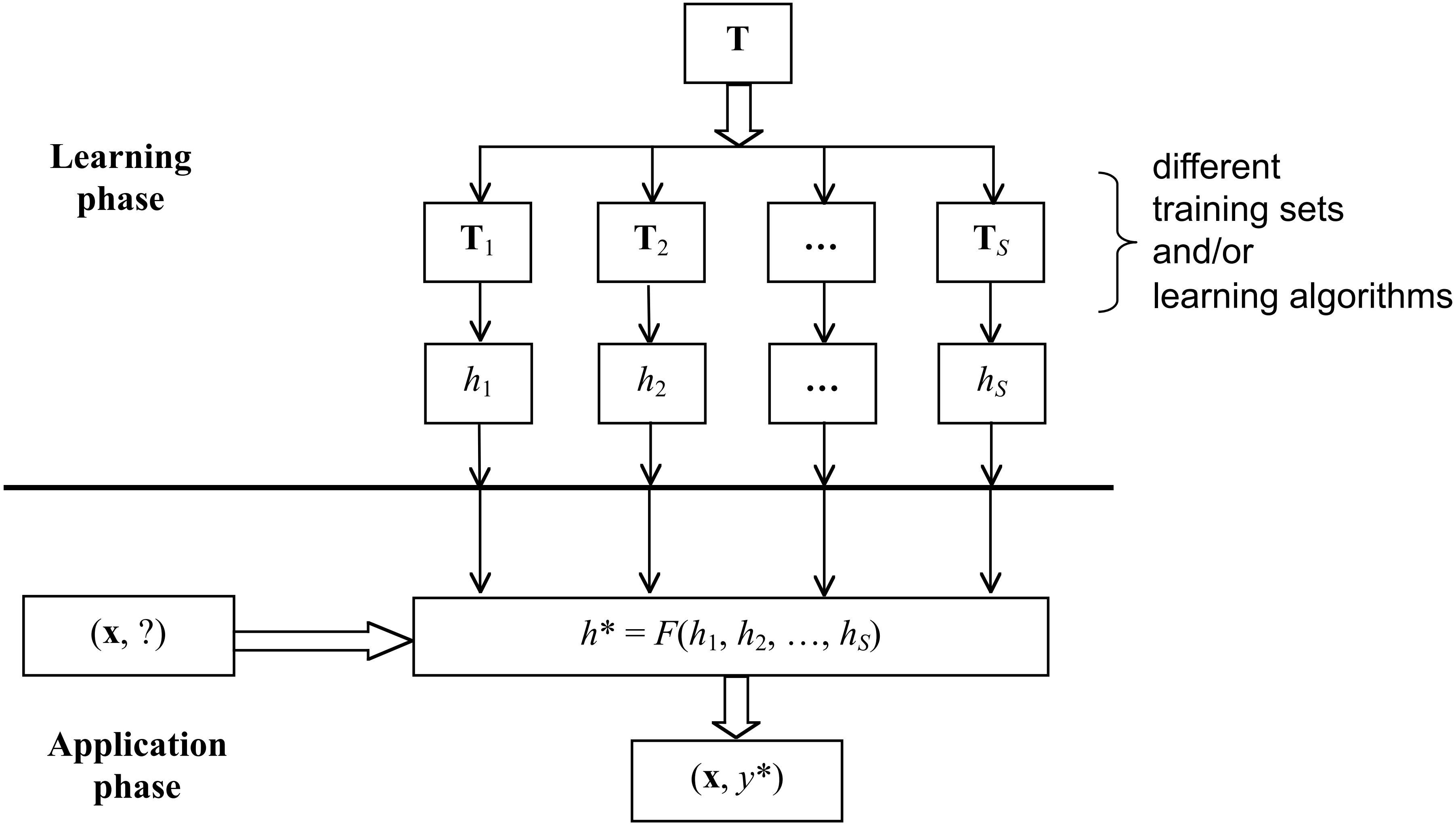
part 1. What is ensemble learning?

Ensemble learning refers to a collection of methods that learn a target function by training a number of individual learners and combining their predictions

A gambler, frustrated by persistent horse-racing losses and envious of his friends' winnings, decides to allow a group of his fellow gamblers to make bets on his behalf. He decides he will wager a fixed sum of money in every race, but that he will apportion his money among his friends based on how well they are doing. Certainly, if he knew psychically ahead of time which of his friends would win the most, he would naturally have that friend handle all his wagers. Lacking such clairvoyance, however, he attempts to allocate each race's wager in such a way that his total winnings for the season will be reasonably close to what he would have won had he bet everything with the luckiest of his friends.

[Freund & Schapire, 1995]

Ensemble learning



How to make an effective ensemble?

Two basic decisions when designing ensembles:

1. How to generate the base classifiers? h_1, h_2, …
2. How to integrate/combine them? $F(h_1(x), h_2(x), \ldots)$

Question 2: How to integrate them

- Usually take a weighted vote:

  $$\text{ensemble}(x) = f\left(\sum_i w_i \, h_i(x) \right)$$

  - w_i is the "weight" of hypothesis h_i
  - $w_i > w_j$ means "h_i is more reliable than h_j"
  - typically $w_i>0$ (though could have $w_i<0$ meaning "h_i is more often wrong than right")

- (Fancier schemes are possible but uncommon)

Question 1: How to generate base classifiers

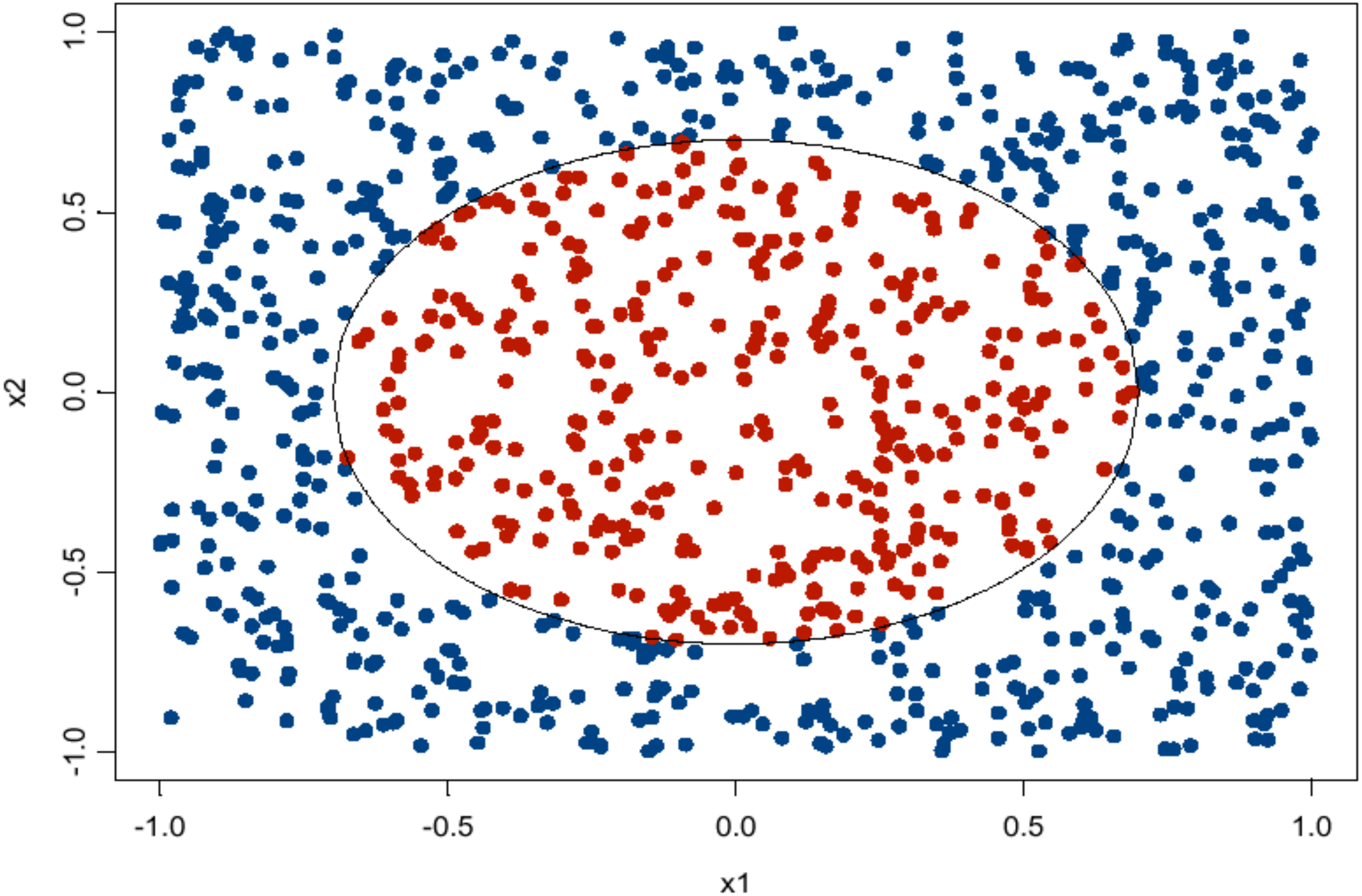
- Lots of approaches…
- A. Bagging
- B. Boosting
- …

PART 2: BAGGing = Bootstrap AGGregation

(Breiman, 1996)

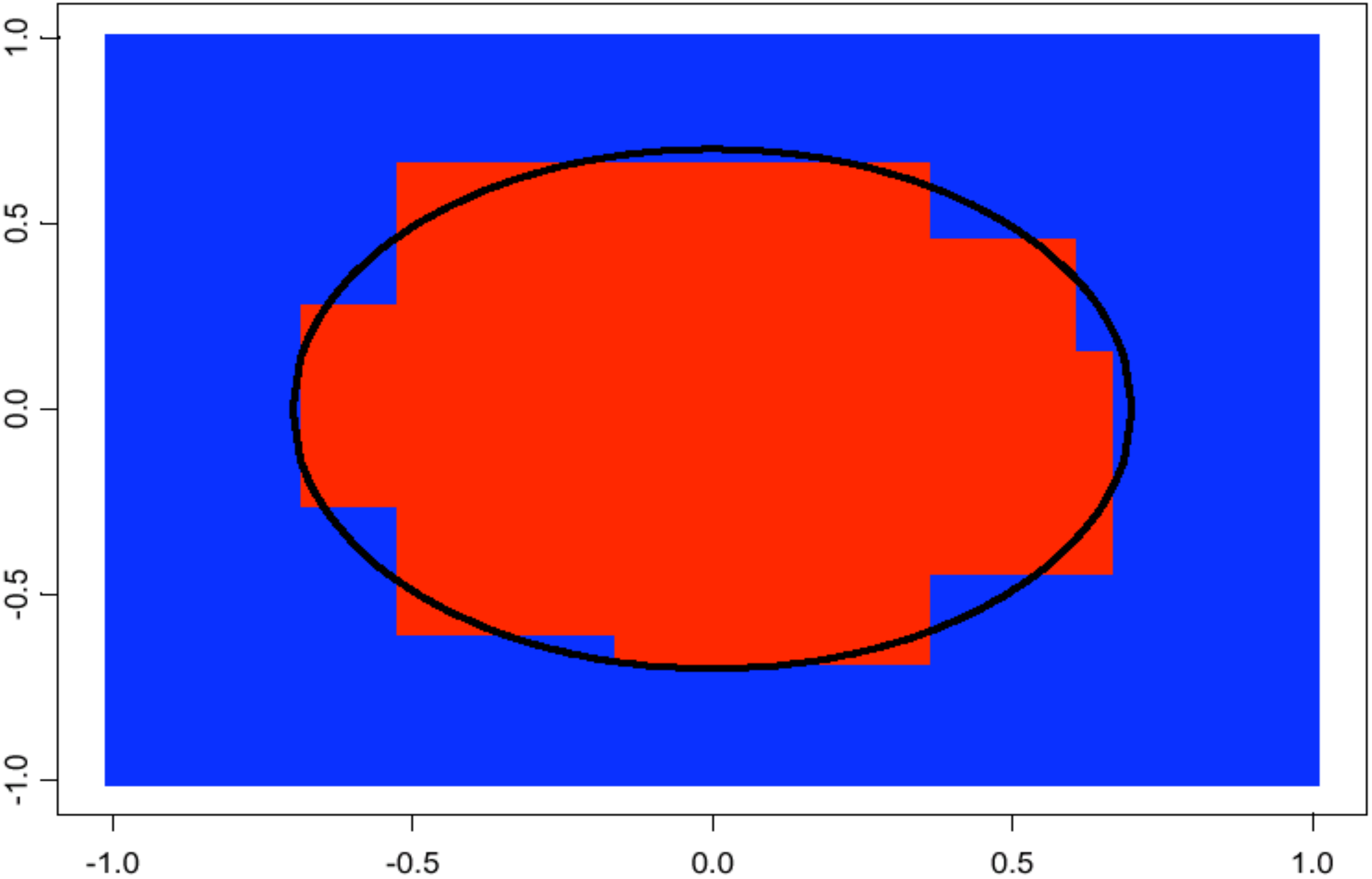
- for i = 1, 2, …, K:
  - $T_i \leftarrow$ randomly select M training instances with replacement
  - $h_i \leftarrow$ learn(T_i) *[ID3, NB, kNN, neural net, …]*
- Now combine the h_i together with uniform voting (w_i=1/K for all i)

Bagging Example

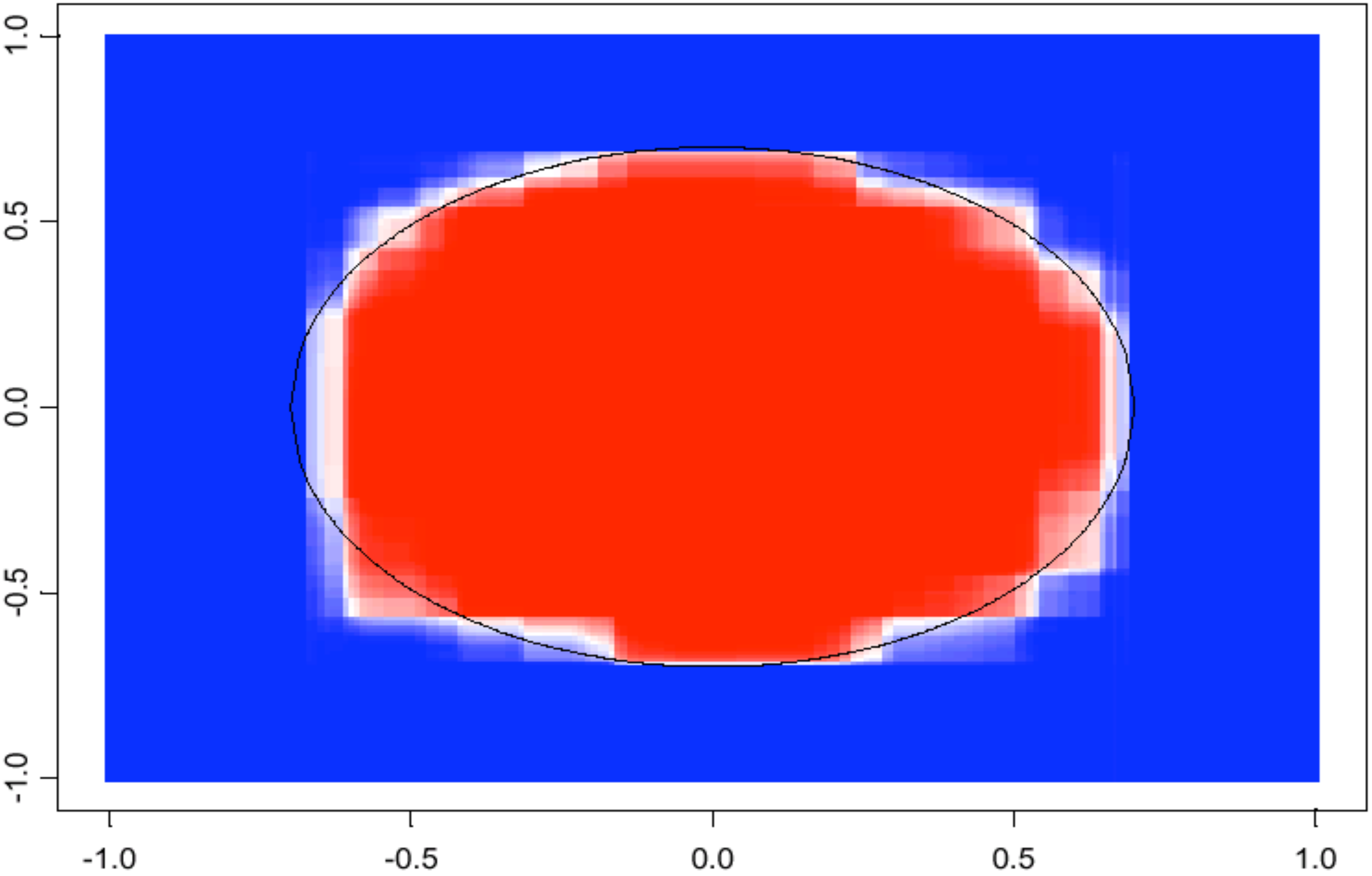


decision tree learning algorithm; along the lines of ID3

CART decision boundary



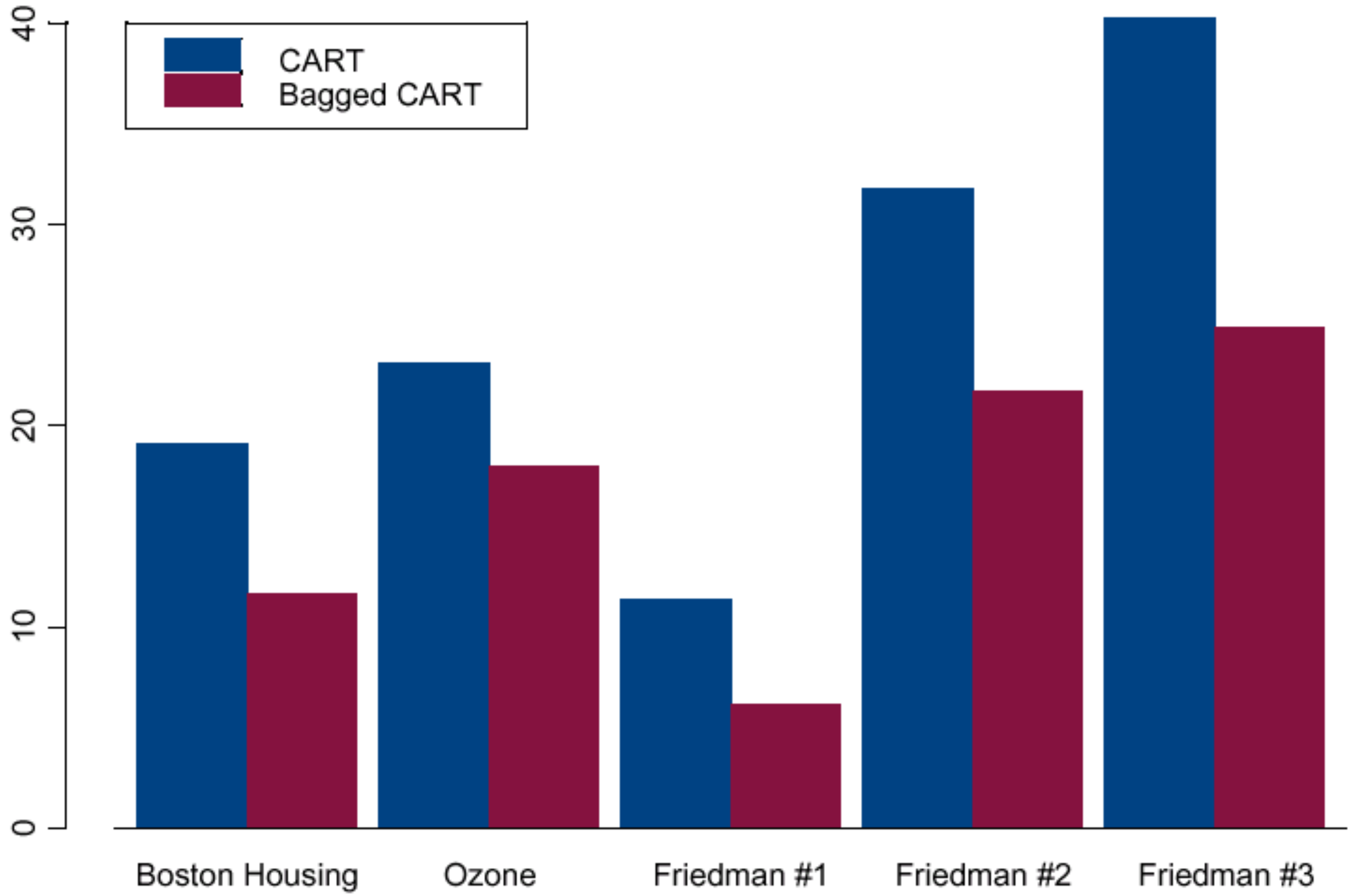
100 bagged trees



shades of blue/red indicate strength of vote for particular classification

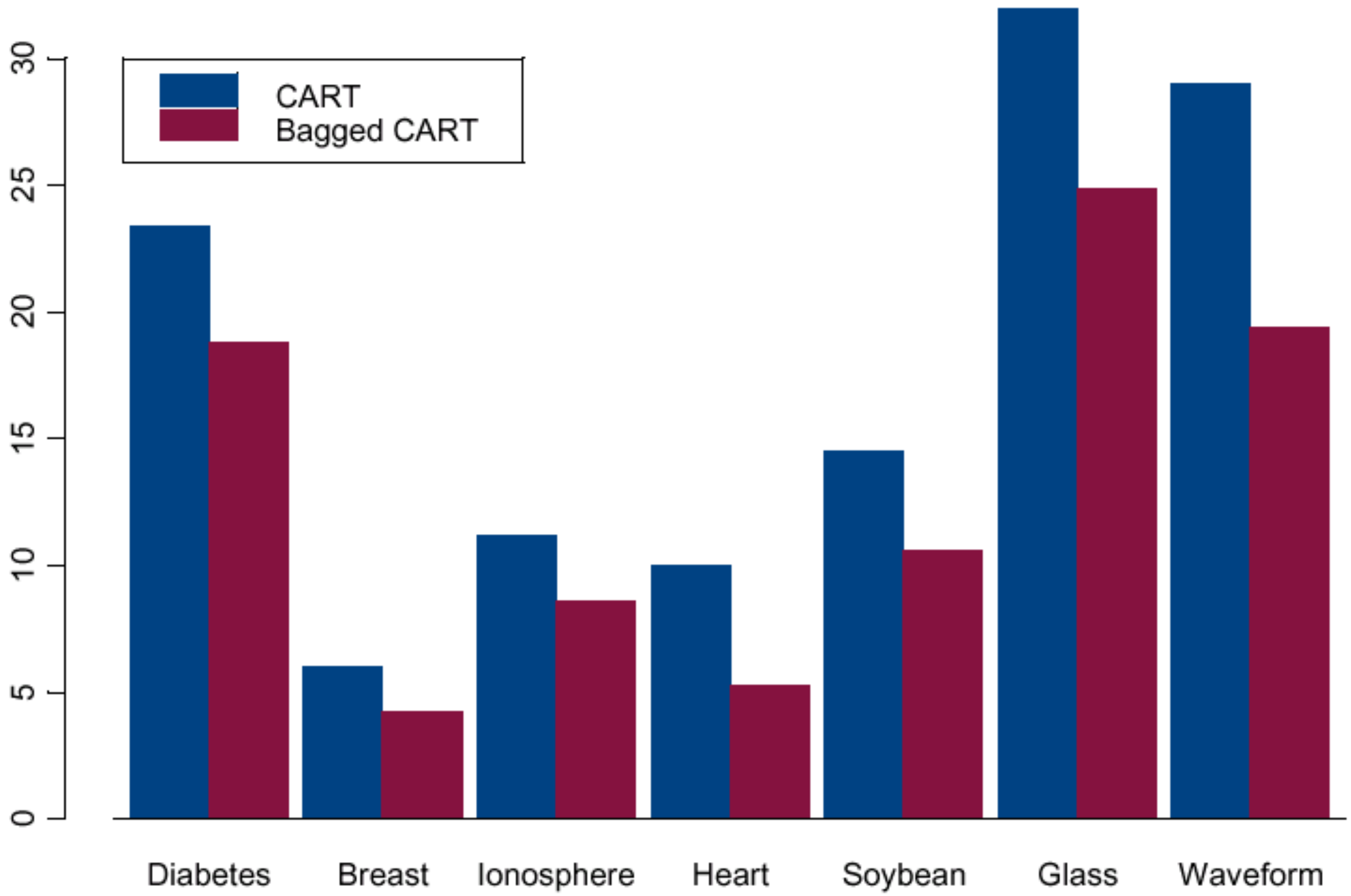
Regression results

Squared error loss



Classification results

Misclassification rates



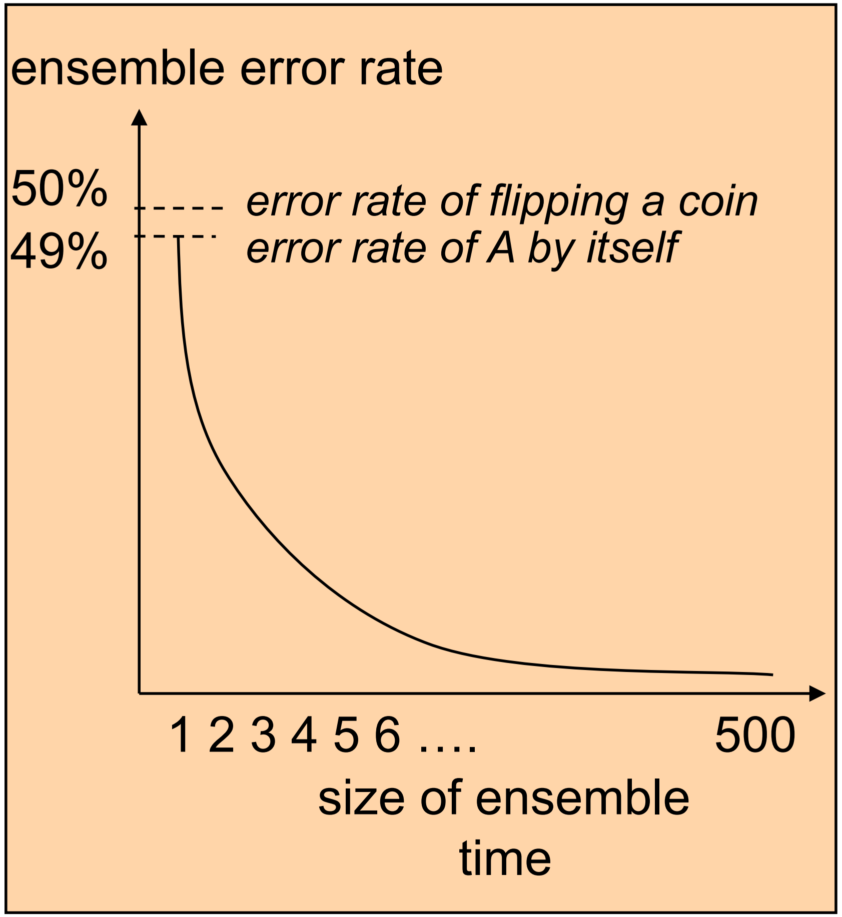
Bagging References

- Leo Breiman's homepage *www.stat.berkeley.edu/users/breiman/*
- Breiman, L. (1996) "Bagging Predictors," *Machine Learning*, 26:2, 123-140.
- Friedman, J. and P. Hall (1999) "On Bagging and Nonlinear Estimation" *www.stat.stanford.edu/~jhf*

Part 3: Boosting

- Bagging was one simple way to generate ensemble members with trivial (uniform) vote weighting
- Boosting is another….

- "**<u>Boost</u>**" as in "**<u>give a hand up to</u>**"
  - suppose *A* can learn a hypothesis that is better than rolling a dice – but perhaps only a tiny bit better
  - **<u>Theorem</u>**: Boosting *A* yields an ensemble with arbitrarily low error on the training data!



Boosting

Idea:

- assign a weight to every training set instance
- initially, all instances have the same weight
- as boosting proceedgs, adjusts weights based on how well we have predicted data points so far
  - data points correctly predicted →low weight
  - data points mispredicted → high weight

Results: as learning proceeds, the learner is forced to focus on portions of data space not previously well predicted

Generic boosting algorithm

Equally weight the observations $(y,\boldsymbol{x})_i$

For t in $1,\ldots,T$

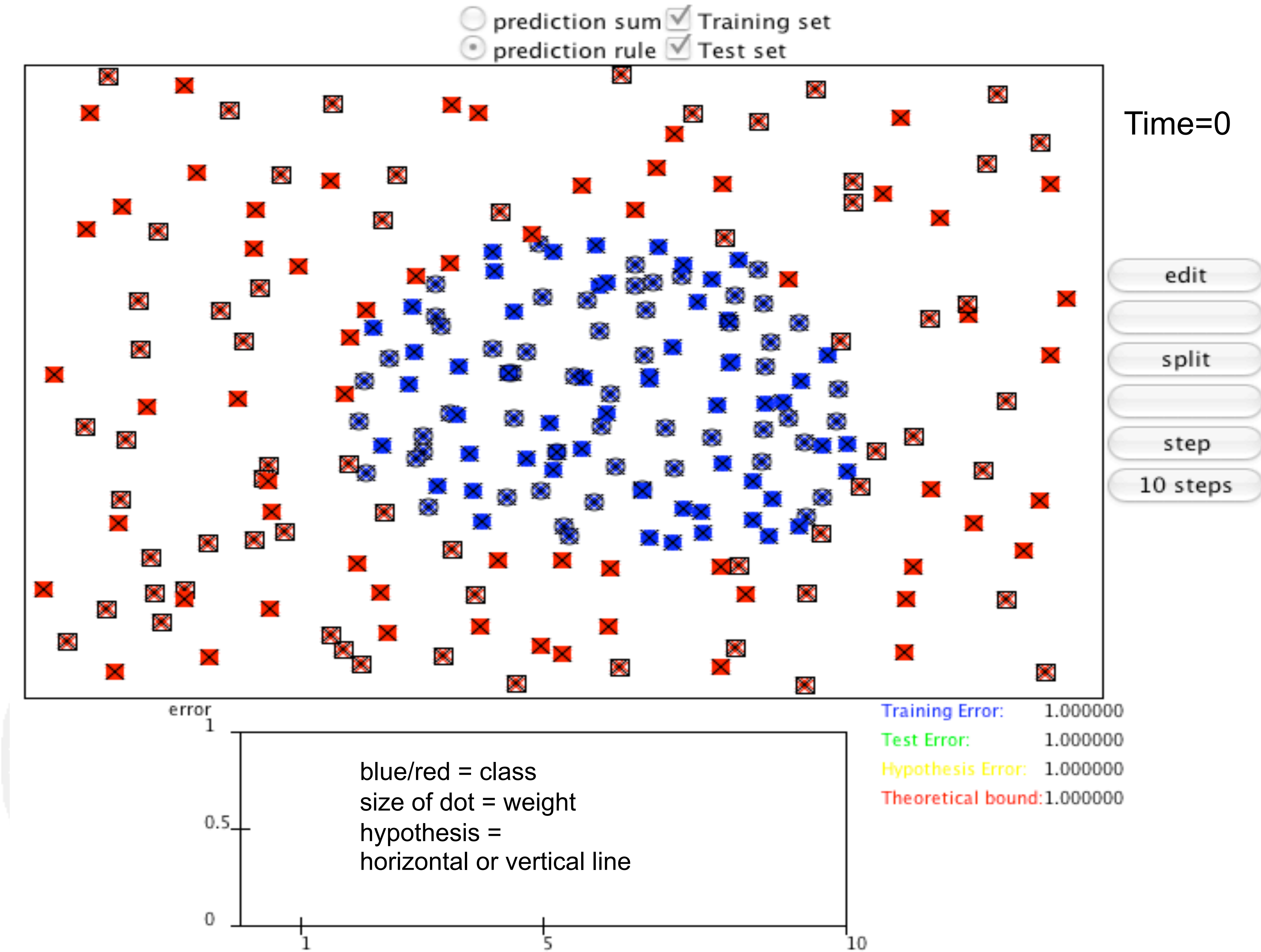
Using the weights, fit a classifier $f_t(\boldsymbol{x}) \to y$

Upweight the poorly predicted observations

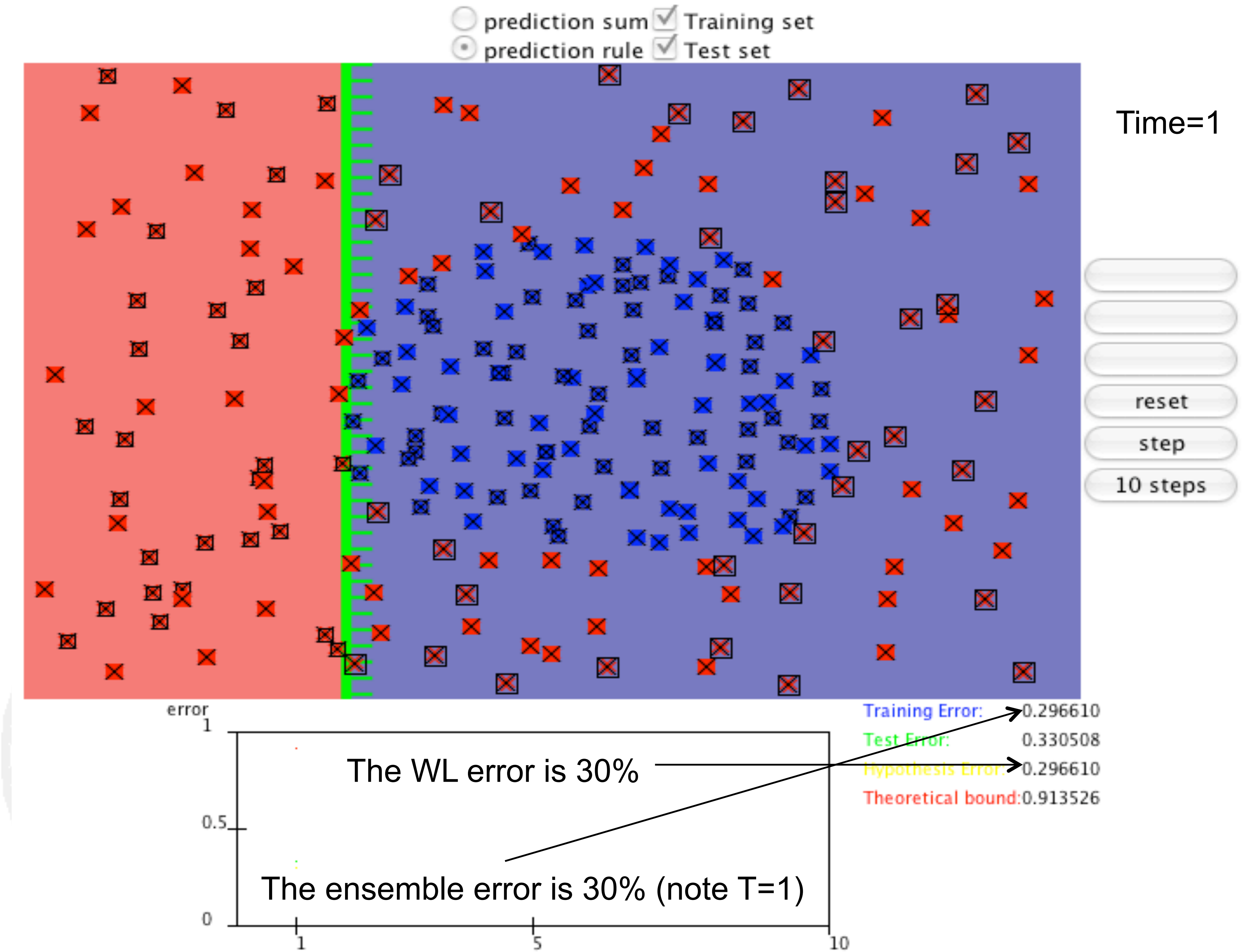
Downweight the well-predicted observations

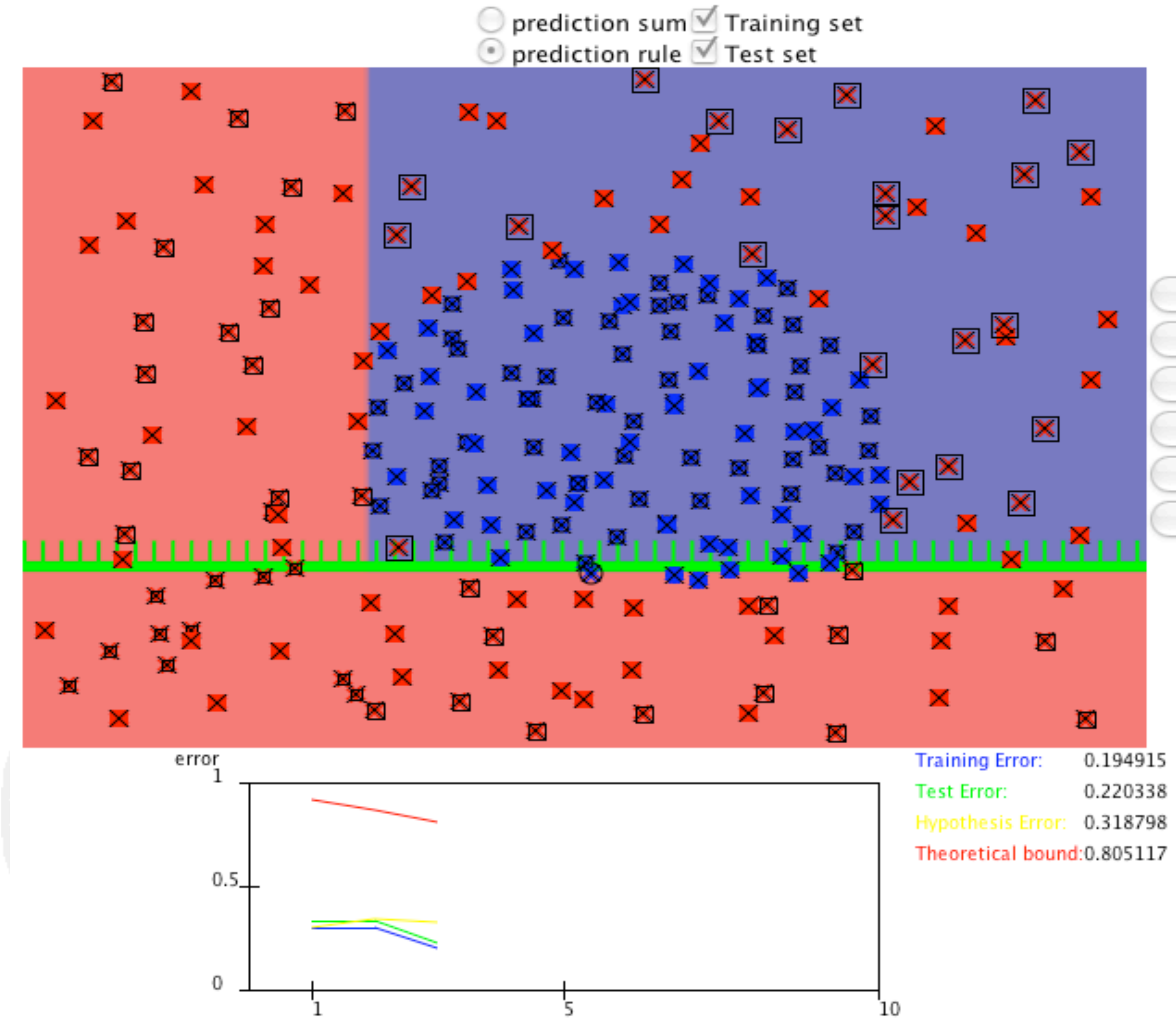
Merge $f_1,\ldots,f_T$ to form the boosted classifier

Time=0

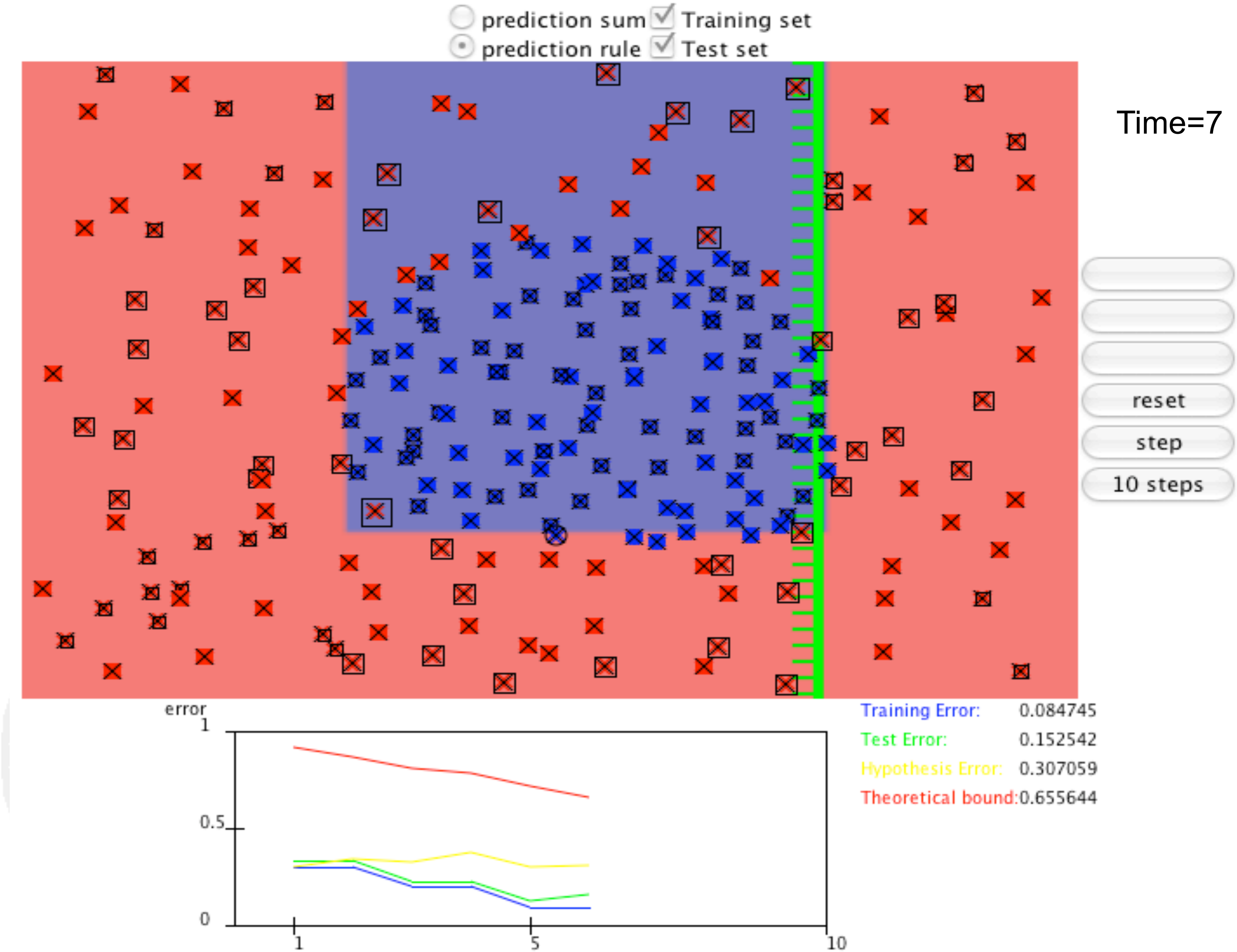


blue/red = class
size of dot = weight
hypothesis =
horizontal or vertical line

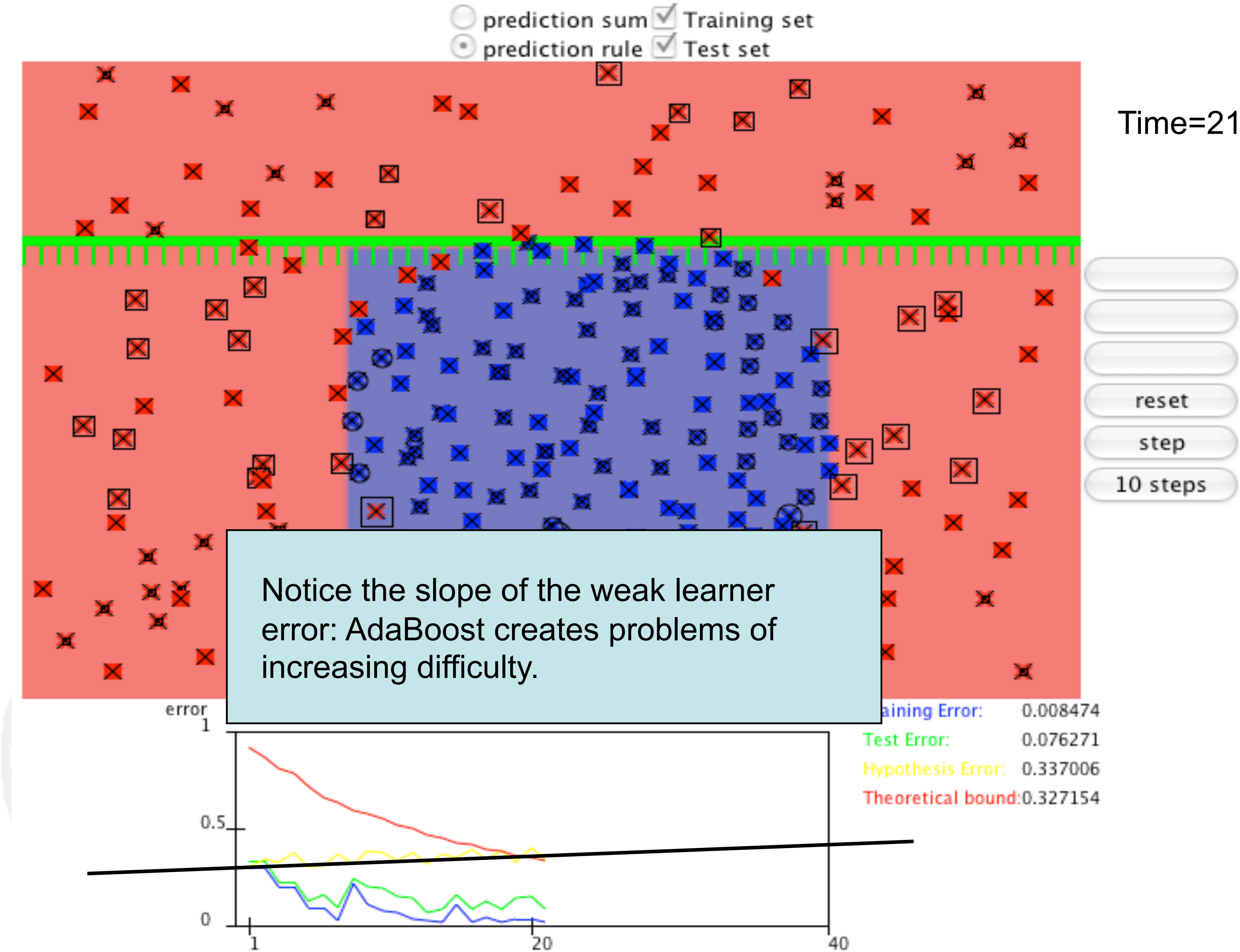

prediction sum
prediction rule
Training set
Test set
Time=1
reset
step
10 steps
error
1
0.5
0
1
5
10
Training Error: 0.296610
Test Error: 0.330508
Hypothesis Error: 0.296610
Theoretical bound: 0.913526
The WL error is 30%
The ensemble error is 30% (note T=1)


prediction sum
Training set
prediction rule
Test set
reset
step
10 steps
error
1
0.5
0
1
5
10
Training Error: 0.194915
Test Error: 0.220338
Hypothesis Error: 0.318798
Theoretical bound:0.805117

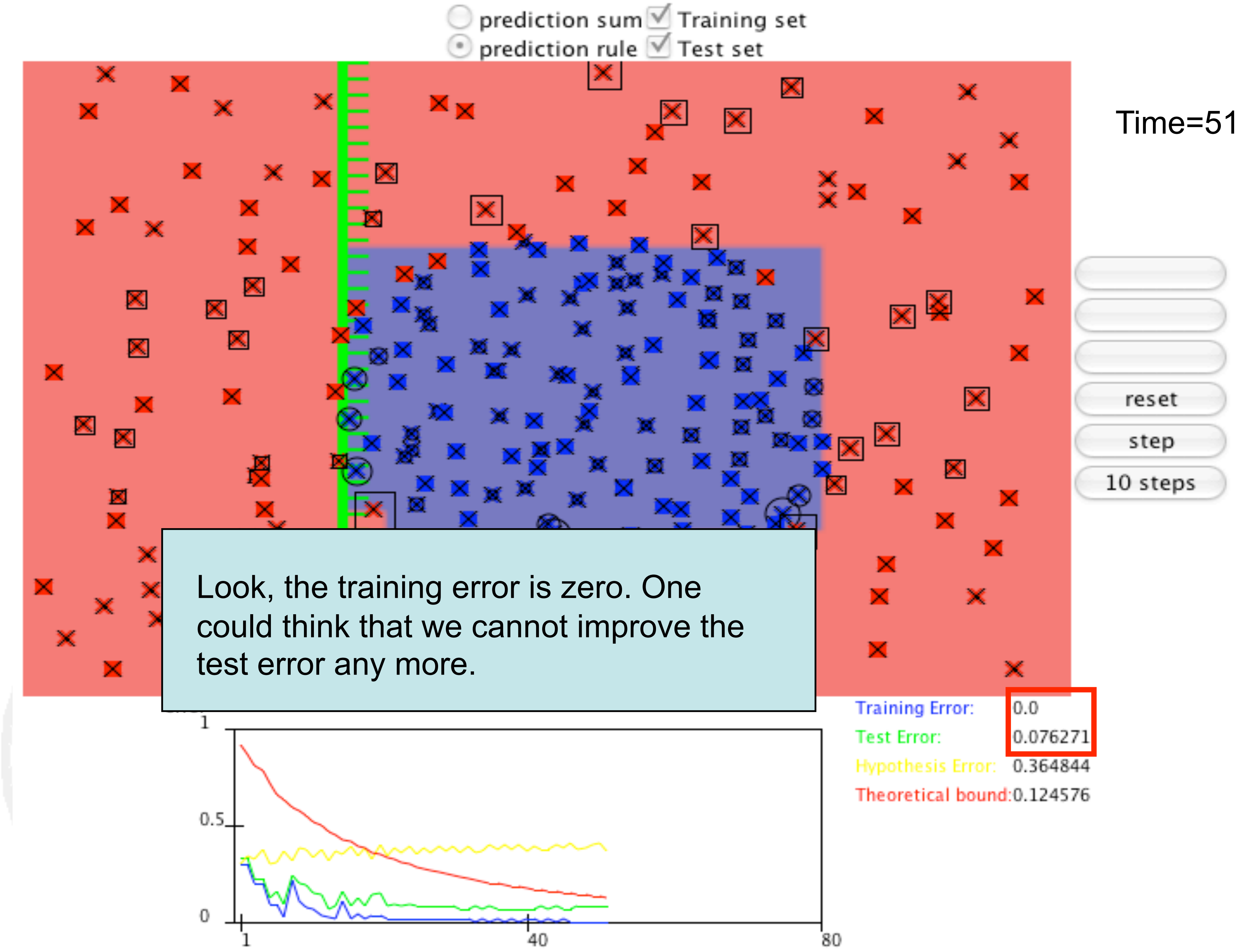
Time=3


prediction sum
prediction rule
Training set
Test set
Time=7
reset
step
10 steps
error
1
0.5
0
1
5
10
Training Error: 0.084745
Test Error: 0.152542
Hypothesis Error: 0.307059
Theoretical bound:0.655644

Time=21



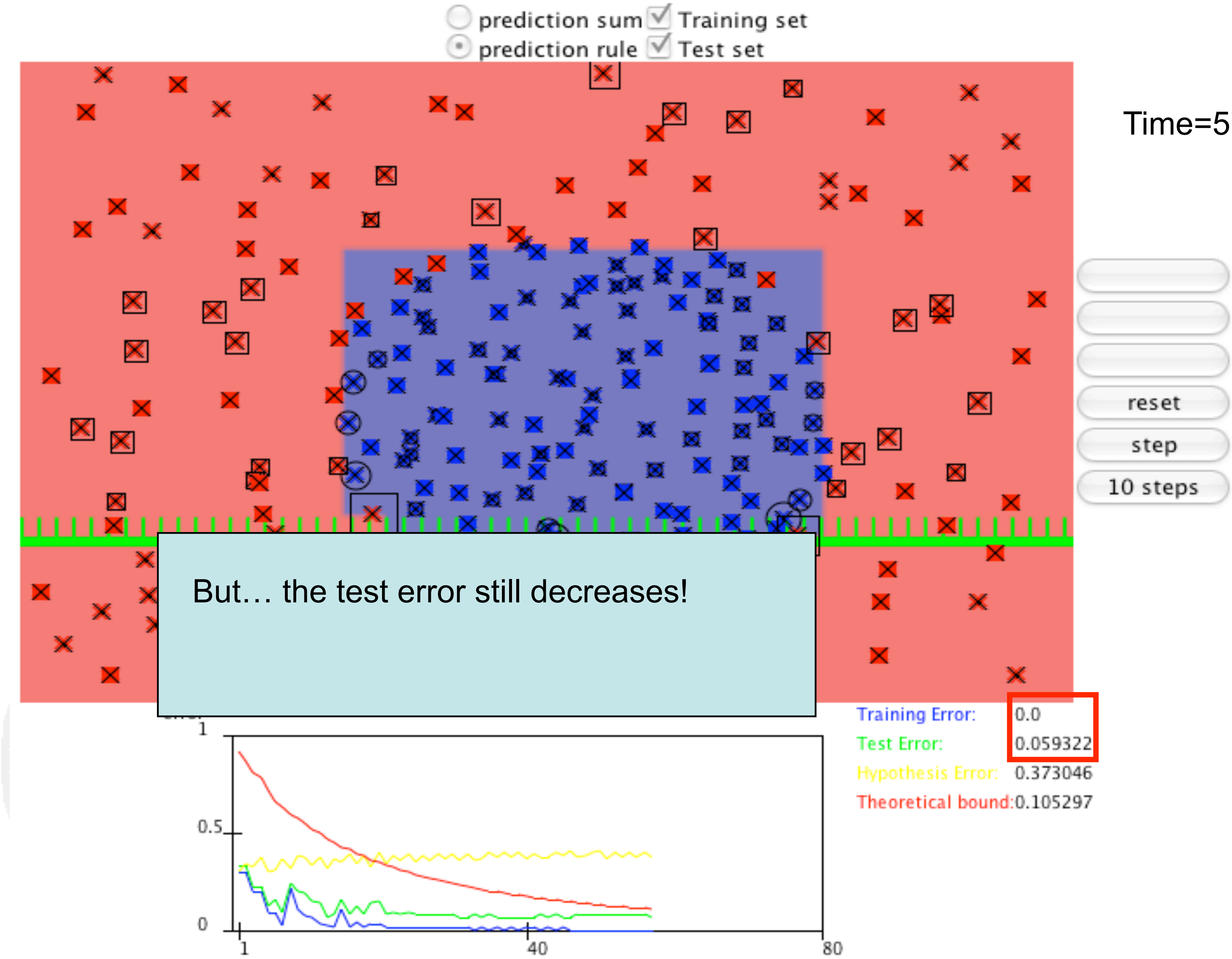
Notice the slope of the weak learner error: AdaBoost creates problems of increasing difficulty.



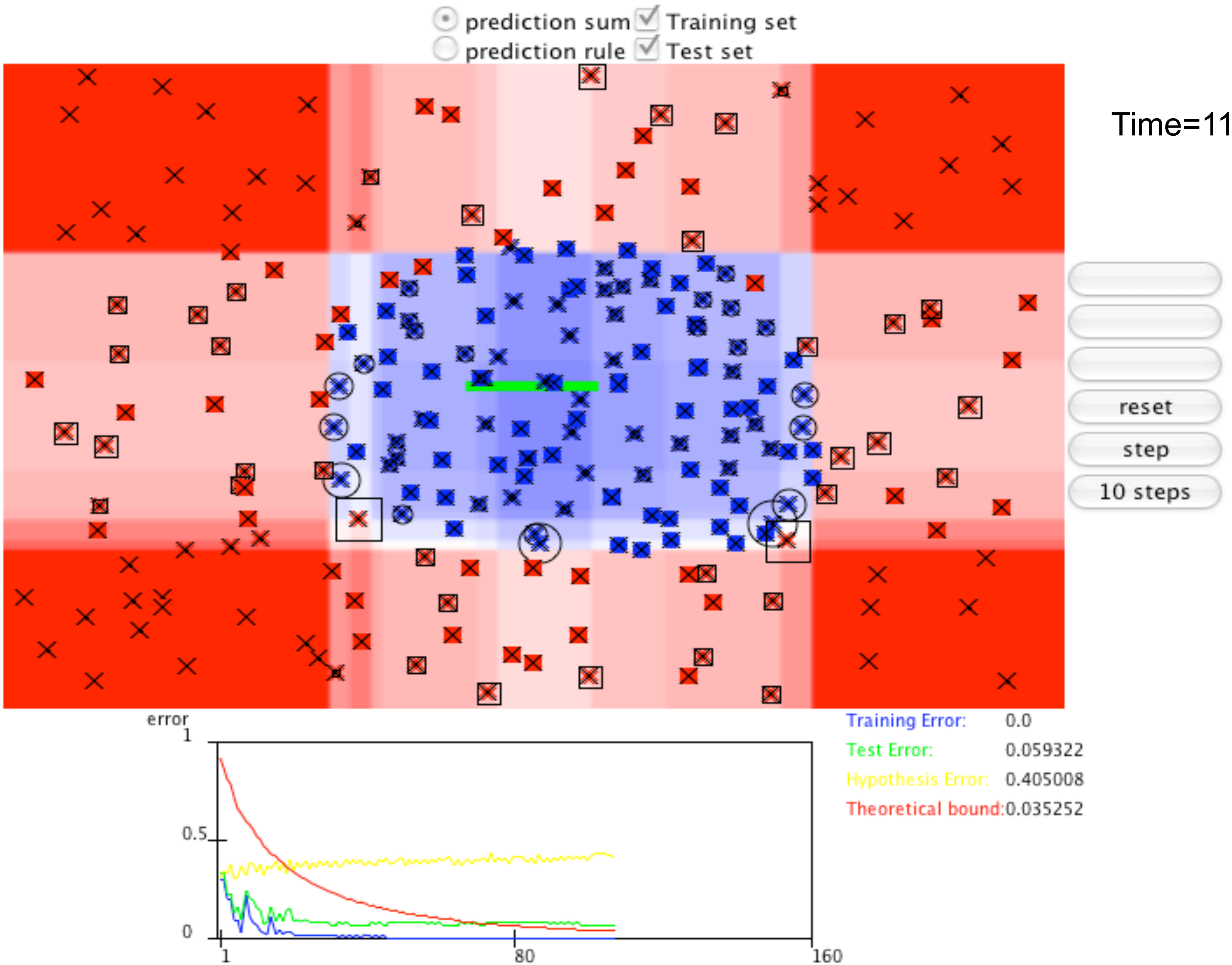
Time=51

Look, the training error is zero. One could think that we cannot improve the test error any more.

Time=57



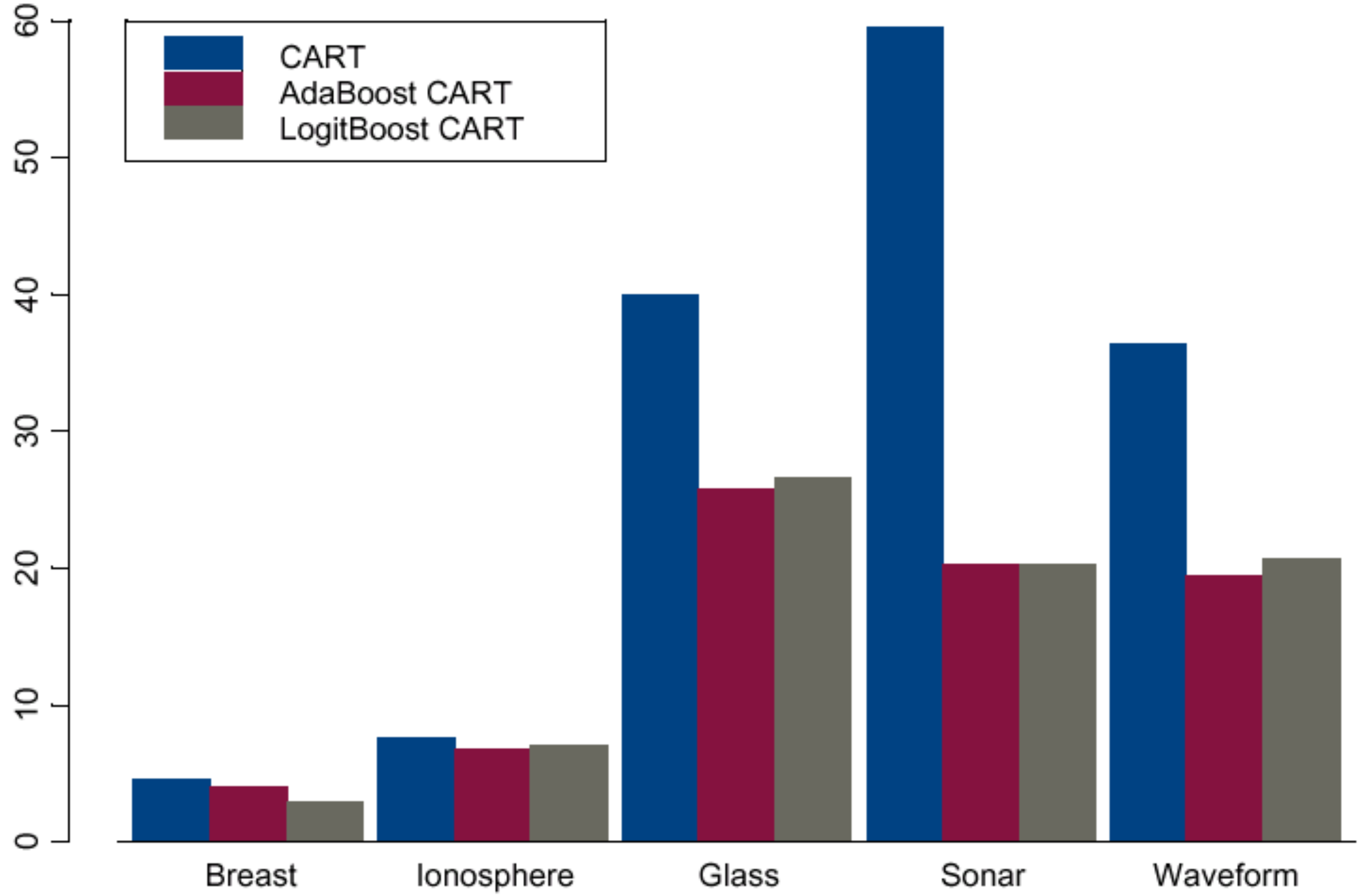
But… the test error still decreases!


prediction sum
prediction rule
Training set
Test set
reset
step
10 steps
error
1
0.5
0
1
80
160
Training Error: 0.0
Test Error: 0.059322
Hypothesis Error: 0.405008
Theoretical bound: 0.035252

Time=110

Misclassification rates

Friedman, Hastie, Tibshirani [1998]



AdaBoost (Freund and Schapire)

Input: sequence of N labeled examples $\langle(x_1, y_1), \ldots, (x_N, y_N)\rangle$

binary class y $\in$ {0,1}

weak learning algorithm **WeakLearn**

integer T specifying number of iterations

Initialize the weight vector: w_i^1 = 1/N for $i = 1, \ldots, N$.

Do for $t = 1, 2, \ldots, T$

1. Set

$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$$

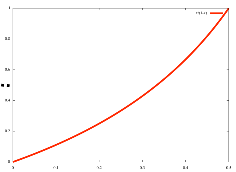
normalize w^t to get a probability distribution p^t
$\sum_I p^t_i = 1$

2. Call **WeakLearn**, providing it with the distribution $\mathbf{p}^t$; get back a hypothesis $h_t : X \rightarrow$[0,1]

3. Calculate the error of h_t: $\epsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|$.

penalize mistakes on high-weight instances more

4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.



5. Set the new weights vector to be

$$w_i^{t+1} = w_i^t \beta_t^{1 - |h_t(x_i) - y_i|}$$

if h_t correctly classify x_i
multiply weight by β_t < 1
otherwise
multiple weight by 1

Output the hypothesis

$$h_f(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^T \left(\log \frac{1}{\beta_t}\right) h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \log \frac{1}{\beta_t} \\ 0 & \text{otherwise} \end{cases}.$$

weighted vote, with $w_t = \log(1/\beta_t)$

Learning from weighted instances?

- One piece of the puzzle missing…

2. Call **WeakLearn**, providing it with the distribution $\mathbf{p}^t$; get back a hypothesis $h_t : X \to$ [0,1]

- So far, learning algorithms have just taken as input a set of equally important learning instances.

Reweighting

•What if we also get a weight vector saying how important each instance is?

•Turns out.. it's very easy to modify most learning algorithms to deal with weighted instances:

–ID3: Easy to modify entropy, information-gain equations to take into consideration the weights associated to the examples, rather than to take into account only the count (which simply assumes all weights=1)

–Naïve Bayes: ditto

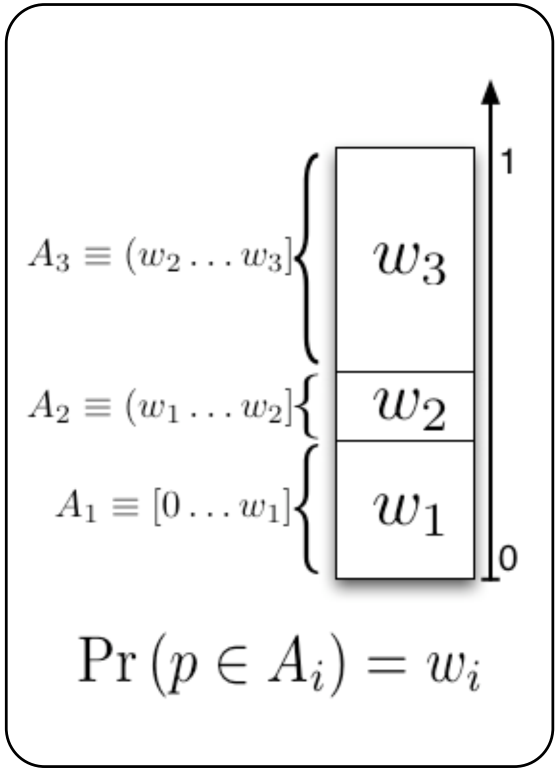
–k-NN: multiple vote from an instance by its weight

Learning from weighted instances?

Resampling

As an alternative to modify learning algorithms to support weighted datasets, we can build a new dataset which is not weighted but it shows the same properties of the weighted one.

1. Let **L’** be the empty set
2. Let $(w_1, \ldots, w_n)$ be the weights of examples in **L** sorted in some fixed order (we assume w_i corresponds to example x_i)
3. Draw $n \in [0..1]$ according to $U(0,1)$
4. set **L’**←**L’**∪$\{x_k\}$ where k is such that $\sum_{i=1}^{k-1} w_i < n \leq \sum_{i=1}^{k} w_i$
5. if enough examples have been drawn return **L’**
6. else go to 3



Learning from weighted instances?

- **How many examples are “enough”?**

  The higher the number, the better **L’** approximate a dataset following the distribution induced by **W**.

  As a rule of thumb: |**L’**|=|**L**| usually works reasonably well.

- **Why don’t we always use resampling instead of reweighting?**

  Resampling can be always applied, unfortunately it requires more resources and produces less accurate results. One should use this technique only when it is too costly (or unfeasible) to use reweighting.

Part 4: ECOC

- So far, we’ve been building the ensemble by tweaking the set of training instances
- ECOC involves tweaking the output (class) to be learned

Example: Handwritten number recognition

 → 7, 4, 3, 5, 2

"obvious" approach: learn function: Scribble → {0,1,2,…,9}

➔ doesn't work very well (too hard!)

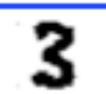
What if we "decompose" the learning task into six "subproblems"?

Class	Code Word					
	vl	hl	dl	cc	ol	or
0	0	0	0	1	0	0
1	1	0	0	0	0	0
2	0	1	1	0	1	0
3	0	0	0	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	0	0	1	0	0
9	0	0	1	1	0	0

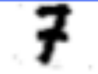
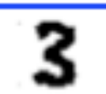
Abbreviation	Meaning
vl	contains vertical line
hl	contains horizontal line
dl	contains diagonal line
cc	contains closed curve
ol	contains curve open to left
or	contains curve open to right

1. learn an ensemble of classifiers, one specialized to each of the 6 "sub-problems"
2. to classify a new scribble, invoke each ensemble member. then predict the class whose code-word is closest (Hamming distance) to the predicted code

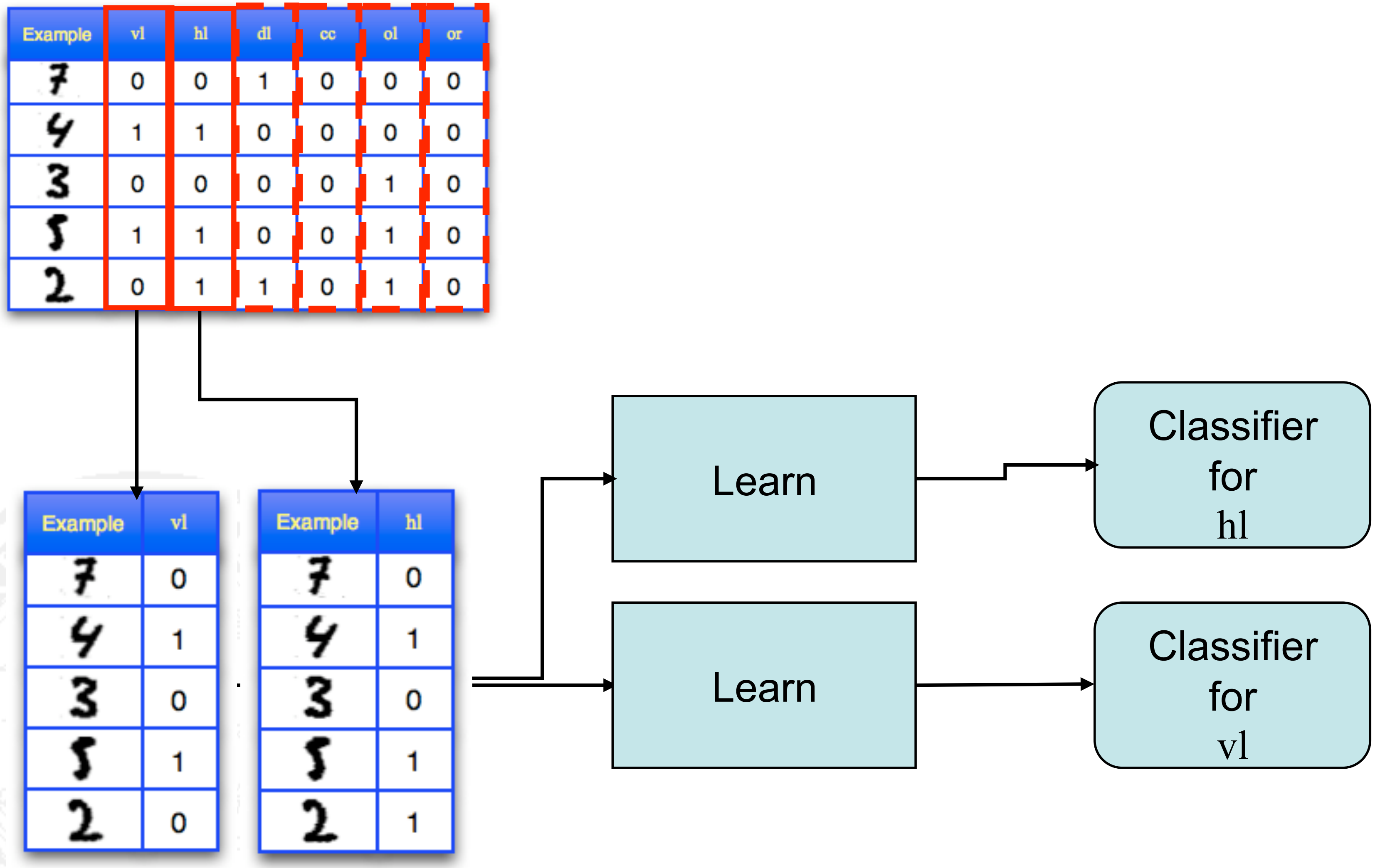
ECOC: Coding

Example	Class
	7
	4
	3
	5
	2

Coding

Example	vl	hl	dl	cc	ol	or
	0	0	1	0	0	0
	1	1	0	0	0	0
	0	0	0	0	1	0
	1	1	0	0	1	0
	0	1	1	0	1	0

ECOC: Learning

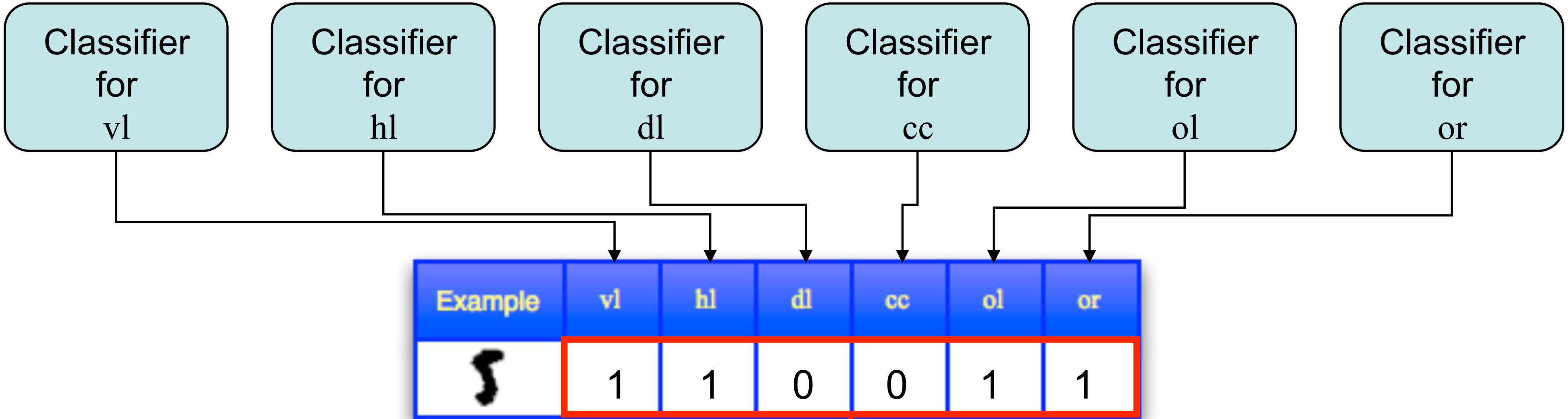


Example	vl	hl	dl	cc	ol	or
7	0	0	1	0	0	0
4	1	1	0	0	0	0
3	0	0	0	0	1	0
5	1	1	0	0	1	0
2	0	1	1	0	1	0

Example	vl
7	0
4	1
3	0
5	1
2	0

Example	hl
7	0
4	1
3	0
5	1
2	1

ECOC:Classification



Find nearest code…

Class	Code Word vl	hl	dl	cc	ol	or
0	0	0	0	1	0	0
1	1	0	0	0	0	0
2	0	1	1	0	1	0
3	0	0	0	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	0	0	1	0	0
9	0	0	1	1	0	0

… and classify accordingly

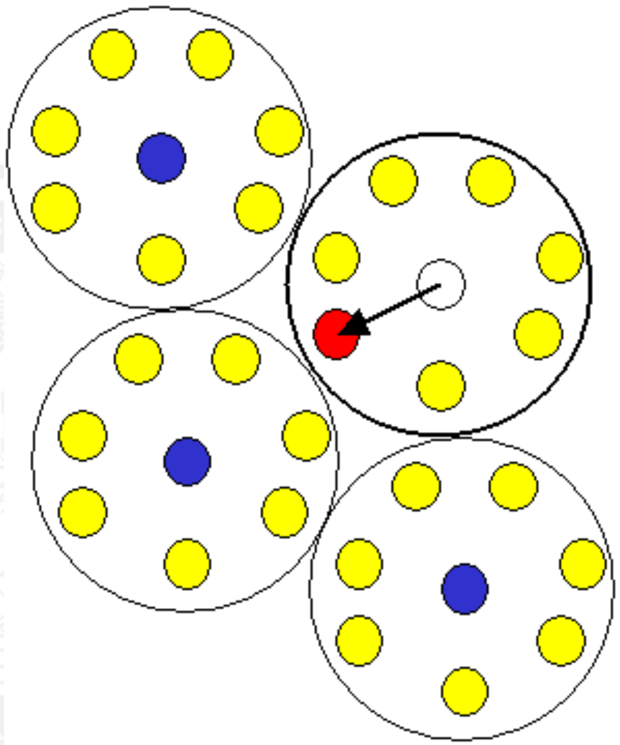
Error-correcting codes

Suppose we want to send n-bit messages through a noisy channel.

To ensure robustness to noise, we can map each n-bit message into an m-bit code (m>n) – note |codes| >> |messages|

When receive a code, translate it to message corresponding to the “nearest” (Hamming distance) code

Key to robustness: assign the codes so that each n-bit “clean” message is surrounded by a “buffer zone” of similar m-bit codes to which no other n-bit message is mapped.

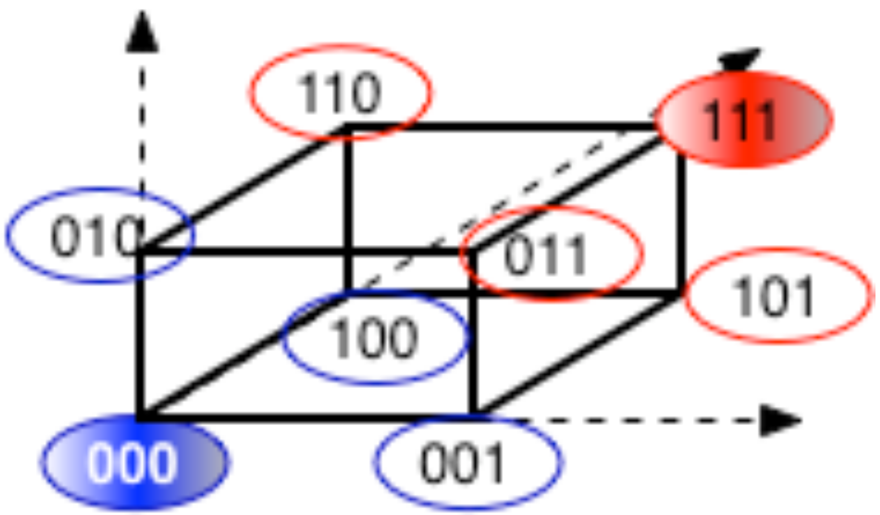


The corrupted word still lies in its original unit sphere. The center of this sphere is the corrected word.

blue = message (n bits)
yellow = code (m bits)

white = intended message
red = received code

A coding example



Consider a situation in which **three** bits are used to code **two** messages. If we select codewords which differs in more than two places, we can detect and correct any “single digit” error.

Designing code-words for ECOC learning

- Coding: k labels → m bit codewords
- Good coding:
  - 1. row separation: want "assigned" codes to be well-separated by lots of "unassigned" codes

class	1	2	3	4	5	6	7	8
Monday	0	0	1	0	0	0	1	0
Tuesday	0	0	1	1	1	0	0	1
Wednesday	0	0	1	0	0	0	1	0
Thursday	0	0	0	1	0	1	1	0
Friday	0	1	1	1	1	0	0	0
Saturday	1	1	1	1	0	0	0	1
Sunday	1	1	1	1	0	0	1	1

  - 2. column separation: each bit i of the codes should be uncorrelated with all other bits j

- Selecting good codes is hard!
  (See paper for details)

Bad codes

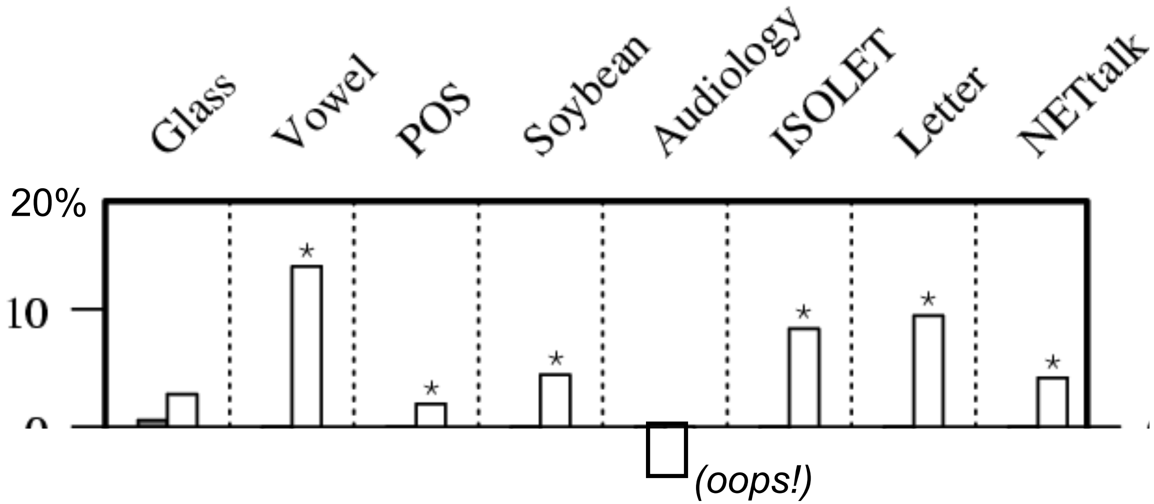
class	1	2	3	4	5	6	7	8
Monday	0	0	1	0	0	0	1	0
Tuesday	0	0	1	1	1	0	0	1
Wednesday	0	0	1	0	0	0	1	0
Thursday	0	0	0	1	0	1	1	0
Friday	0	1	1	1	1	0	0	0
Saturday	1	1	1	1	0	0	0	1
Sunday	1	1	1	1	0	0	1	1

correlated rows ➔ bad

correlated columns ➔ bad

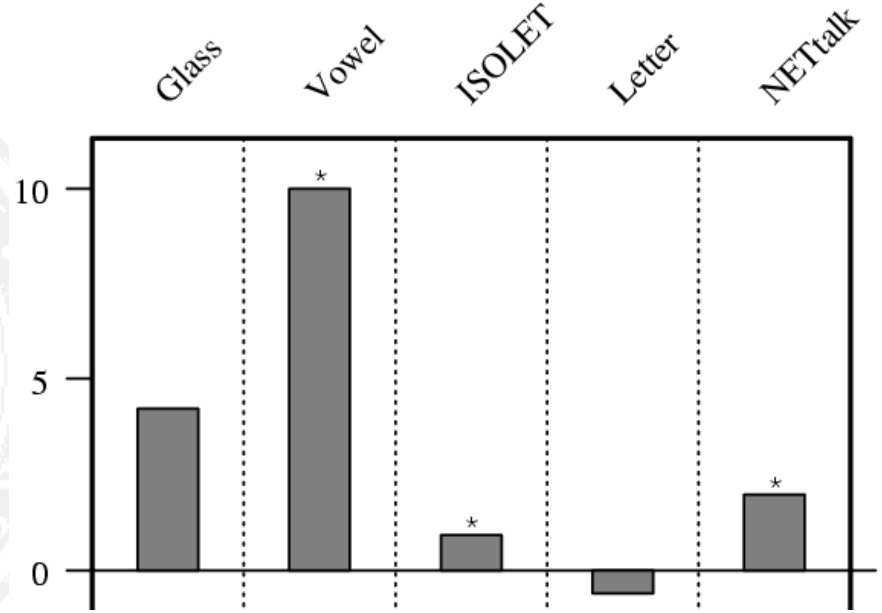
The simplest approach is to select the codewords *at random*. It can be showed that if $2^m >> k$ then we obtain a “good” code with high probability (also Dietterich [1995] mentions that such codes seems to work well in practice).

Results



% decrease in error of ECOC over an ID3-like learning algorithm

% decrease in error of ECOC over a neural network learner



Part 5: Why do ensemble work?

Several reasons justify the ensemble approach.

- Bias/Variance decomposition
- A(nother) statistical motivation
- A motivation based on representational issues
- A motivation based on computational issues

Bias/Variance decomposition

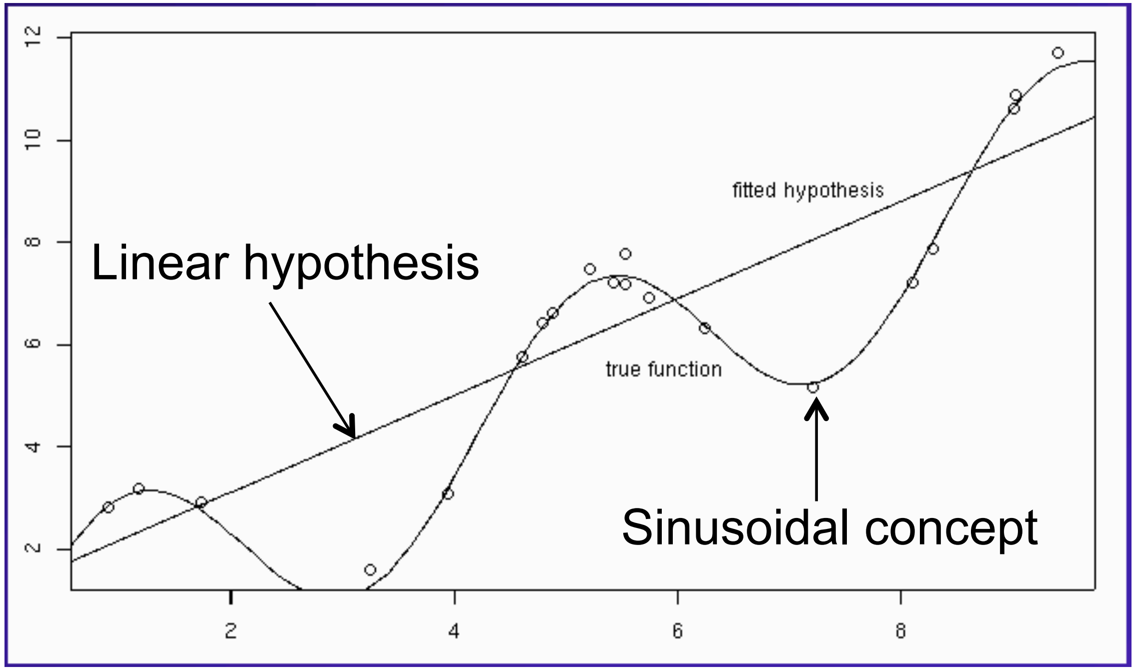
Let $E[\varepsilon(x)]$ be the average error of an algorithm A on an example x (the average is taken repeating the algorithm on many learning sets).

It can be showed that $E[\varepsilon(x)]$ can be decomposed as follows:

$$E[\varepsilon(x)] = Bias(x) + Variance(x) + Noise(x)$$

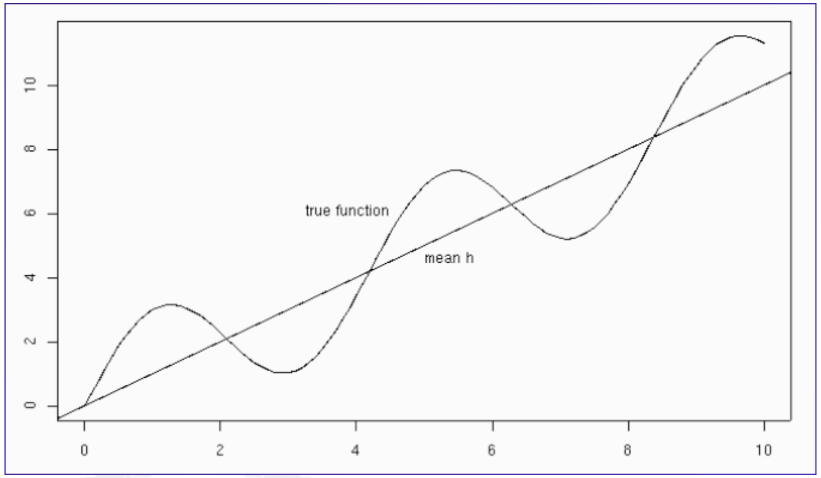
Let us consider the following example:

We want to fit a dataset using a linear concept. "Unfortunately" the true function is sinusoidal.

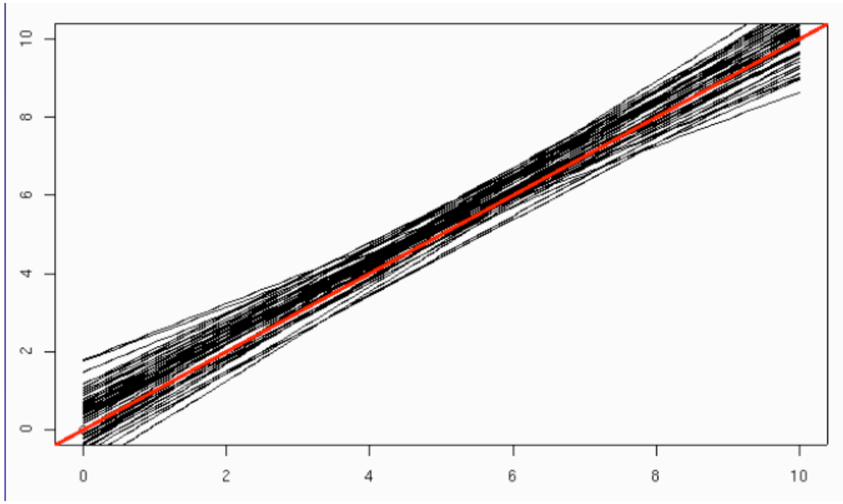


Bias Variance Decomposition

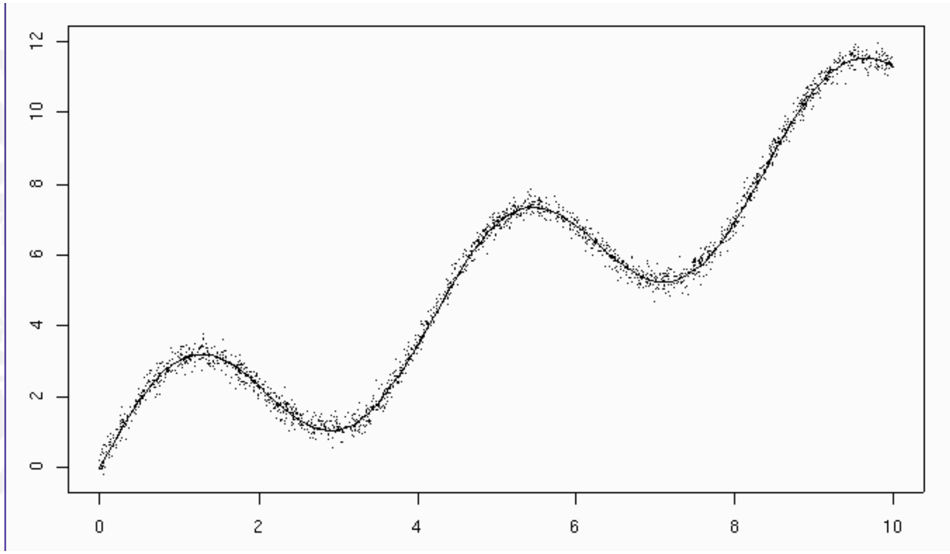
Bias



Variance



Noise



Why do ensembles work? (Bagging)

There exists **empirical and theoretical** evidence that *Bagging* acts as **variance reduction** machine (i.e., it reduces the variance part of the error).

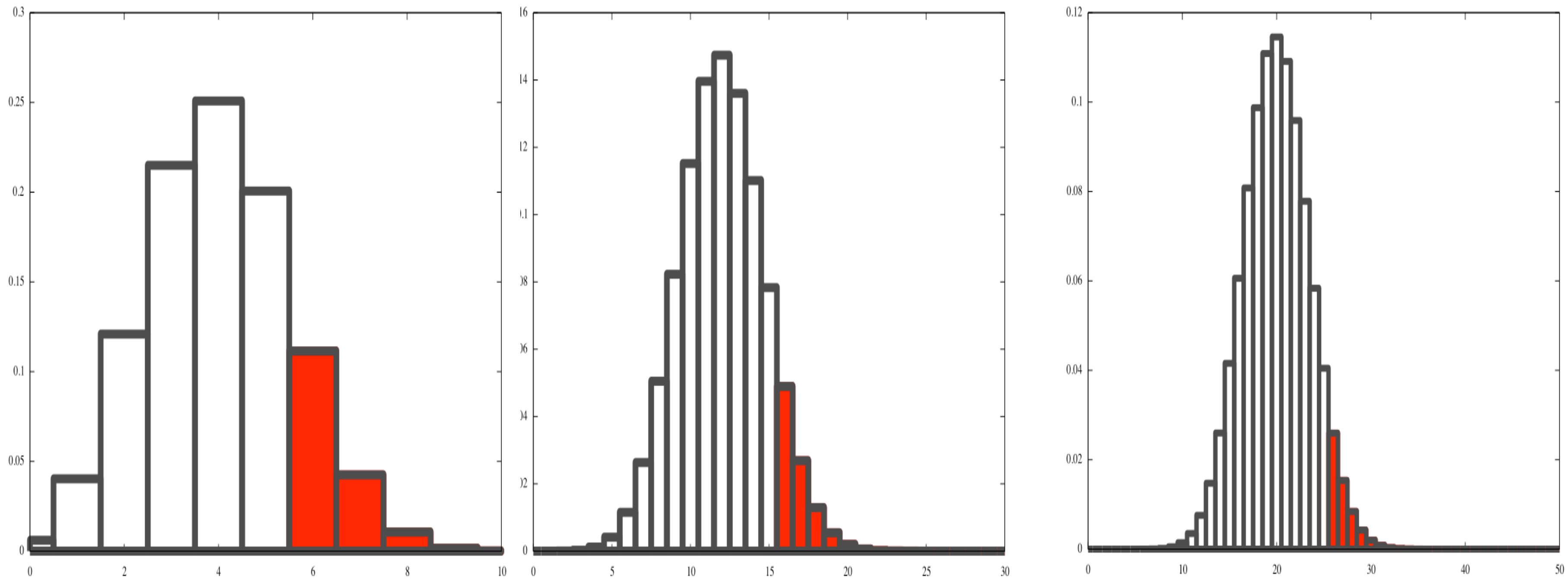
The theoretical arguments depends on the behavior of the binomial distribution when the number of combined hypotheses grows.
Consider the probability *p* that an hypothesis learnt on a bootstrap replicate of the original training set classifies incorrectly a given example. The probability that the majority vote of *T* hypotheses is wrong is:

$$\Pr\left\{\left\lfloor \frac{T}{2} \right\rfloor + 1 \text{ hp are wrong}\right\} + \Pr\left\{\left\lfloor \frac{T}{2} \right\rfloor + 2 \text{ hp are wrong}\right\} + \ldots + \Pr\left\{T \text{ hp are wrong}\right\}$$

Clearly, the random variable X which count the number of hypotheses that are wrong when *T* are extracted is binomially distributed with parameters (*p*,T), i.e. X~Bin(*p*,T)

Why do ensembles work? (Bagging)

It is simple to verify that if $p<0.5$ then the probability X is larger than T/2 approaches zero as T grows.



Why do ensembles work? (AdaBoost)

Empirical evidence suggests that *AdaBoost* reduces both the **bias** and the **variance** part of the error.

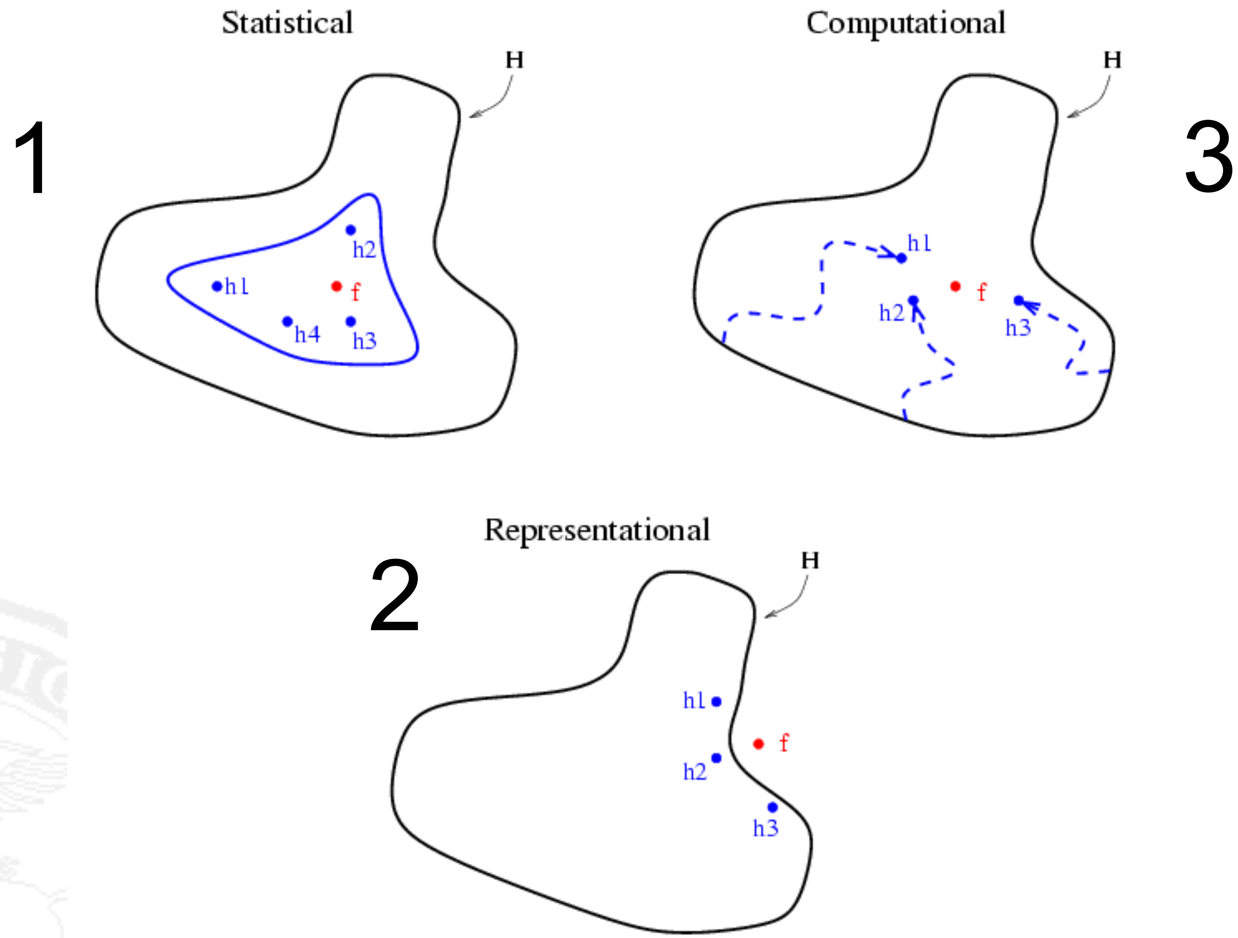
In particular, it seems that bias is mostly reduced in early iterations, while variance in later ones.

Lesson learned?

- **Use Bagging with low bias and high variance classifiers** (e.g., decision trees, 1-nn, ...)

- **Always try AdaBoost** (;-)). Most of the times, it produces excellent results. It has been showed to work very well with very simple learners (e.g., decision stumps inducer) as well as with more complex ones (e.g. C4.5).

Other explanations?

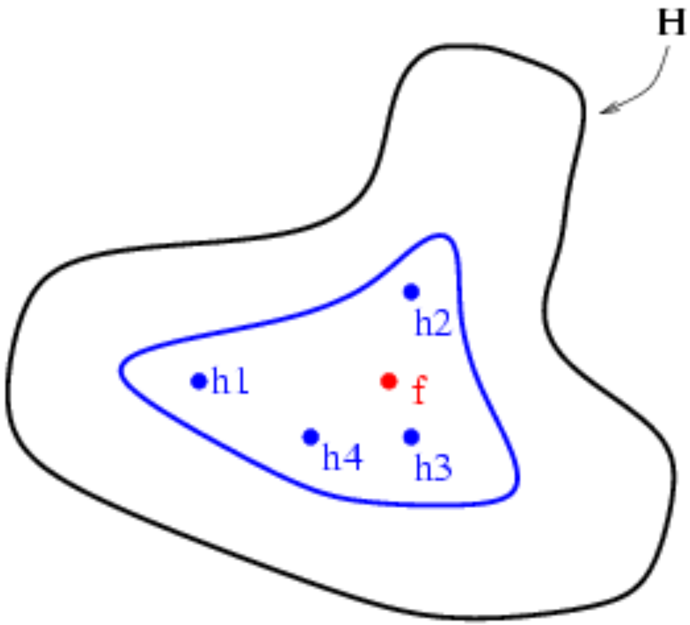


[T. G. Dietterich. *Ensemble methods in machine learning*. Lecture Notes in Computer Science, 1857:1–15, 2000.]

1. Statistical

- Given a finite amount of data, many hypothesis are typically equally good. How can the learning algorithm select among them?

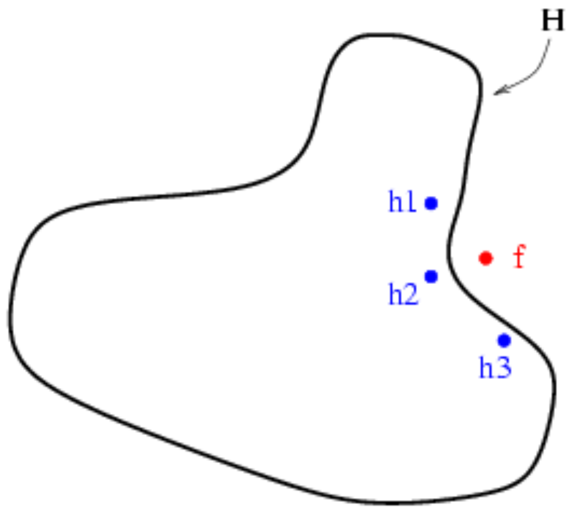
Optimal Bayes classifier recipe: take a *weighted* majority vote of *all* hypotheses weighted by their posterior probability. That is, put most weight on hypotheses consistent with the data.



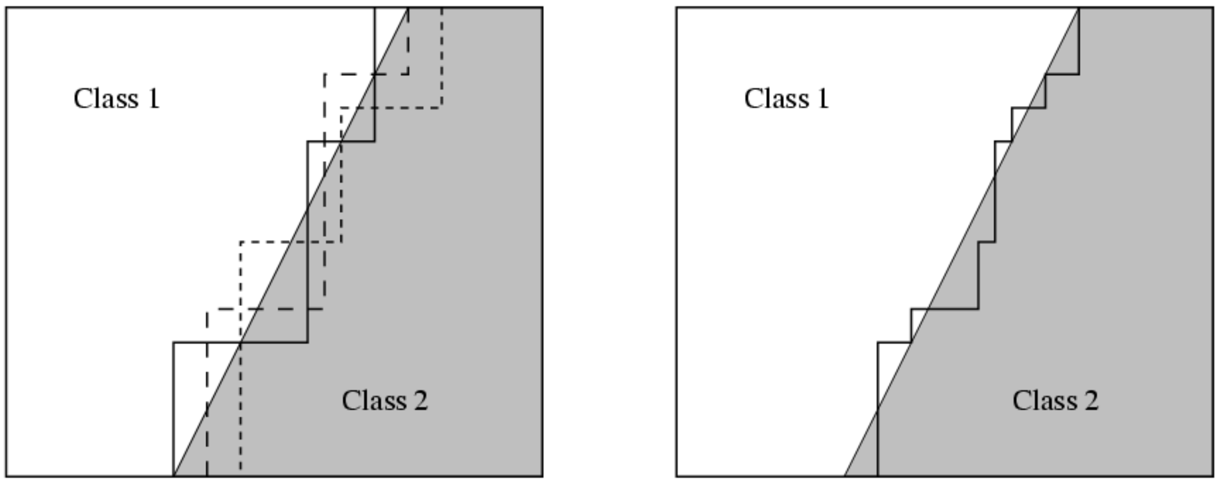
Hence, ensemble learning may be viewed as an approximation of the Optimal Bayes rule (which is **provably** the best possible classifier).

2. Representational

The desired target function may not be implementable with individual classifiers, but may be approximated by ensemble averaging

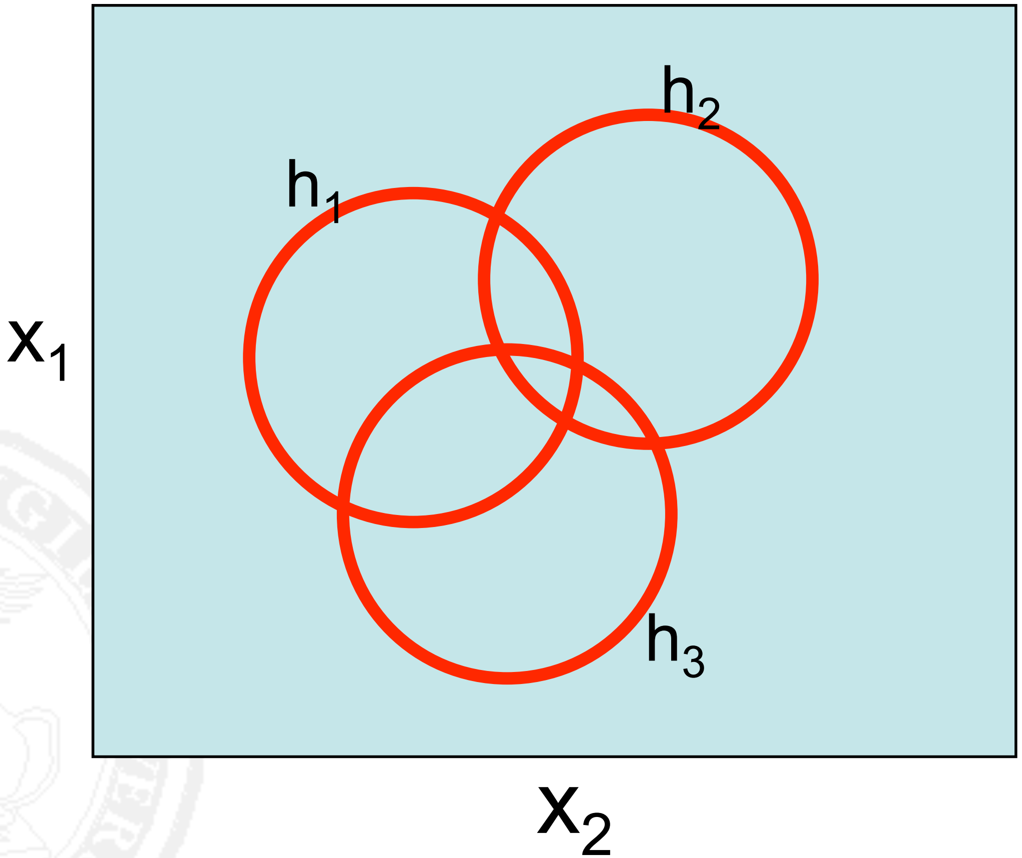


Suppose you want to build a decision boundary with decision trees The decision boundaries of decision trees are hyperplanes parallel to the coordinate axes.   By averaging a large number of such “staircases”, the diagonal decision boundary can be approximated with arbitrarily good accuracy



Representational (another example)

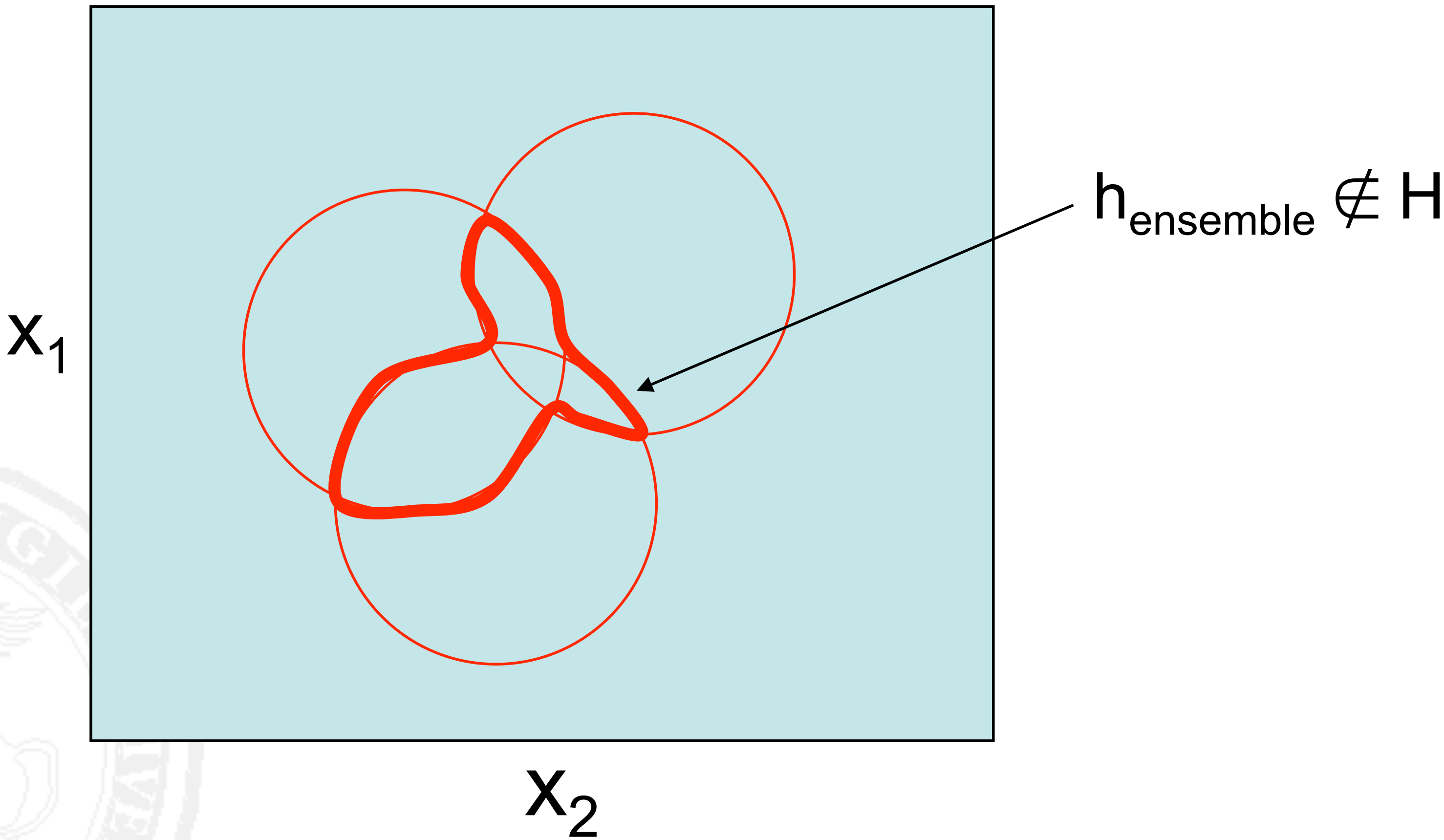
- Consider a binary learning task over [0,1] x [0,1], and the hypothesis space H of “discs”



$h_1, h_2, h_3 \in H$

Representational (another example)

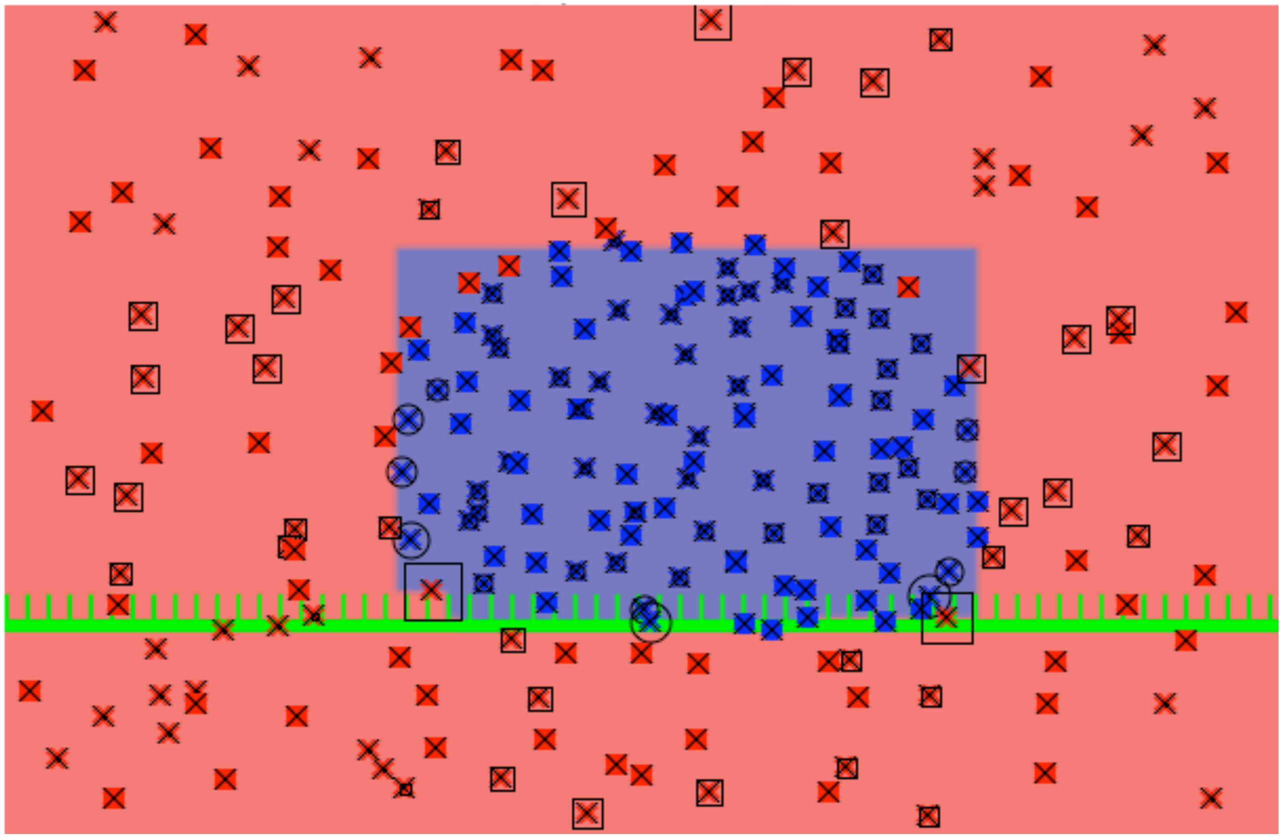
- $H_{ensemble}$ = vote together h_1, h_2, h_3



➔ Even if target concept $\notin$ H, a mixture of hypothesis $\in$ H might be highly accurate

Representational (yet another example)

As we have seen, despite the fact that **no linear concept** can acquire a rectangular concept, AdaBoost was quite successful in finding such an hypothesis by combining several linear concepts.



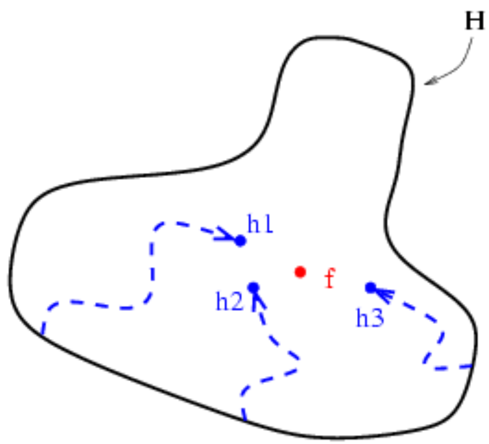
3. Computational

- All learning algorithms do some sort of search through some space of hypotheses to find one that is “good enough” for the given training data

- Since interesting hypothesis spaces are huge/ infinite, heuristic search is essential (eg ID3 does greedy search in space of possible decision trees)

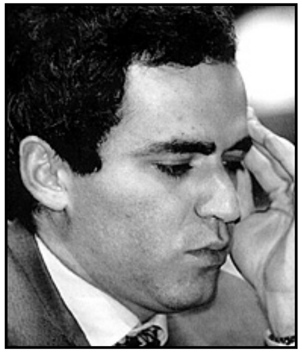
- So the learner might get stuck in a local minimum

- One strategy for avoiding local minima: repeat the search many times with random restarts
  ➔ bagging



Summary…

versus



- Ensembles: basic motivation – creating a **committee of experts** is typically more effective than trying to derive a single **super-genius**
- Key issues:
  - Generation of base models
  - Integration of base models
- Popular ensemble techniques
  - manipulate training data: bagging and boosting (ensemble of "experts", each specializing on different portions of the instance space)
  - manipulate output values: error-correcting output coding (ensemble of "experts", each predicting 1 bit of the {multibit} full class label)
- Why does ensemble learning work?