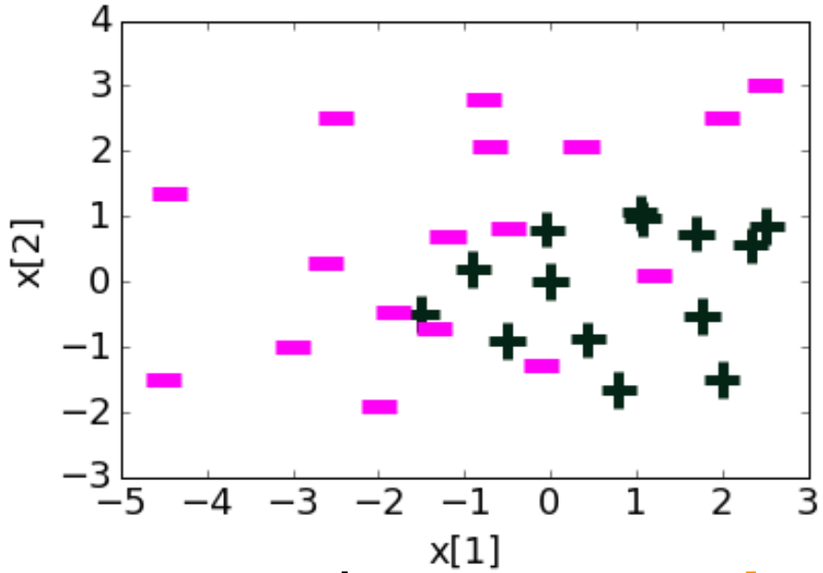
Boosting

Sewoong Oh

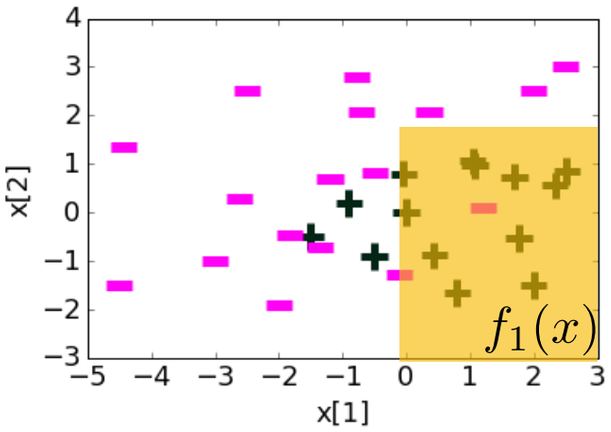
CSE/STAT 416
University of Washington

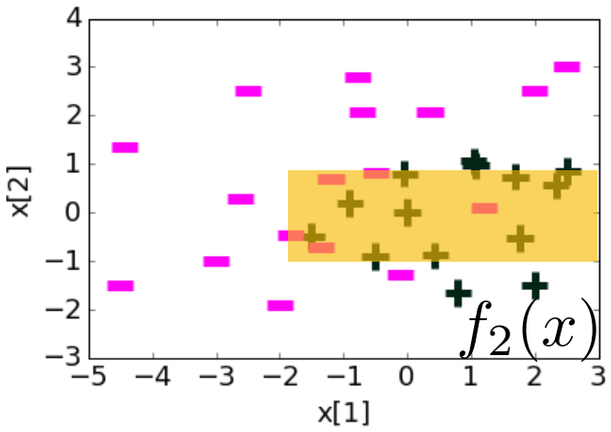
Ensemble classifier:

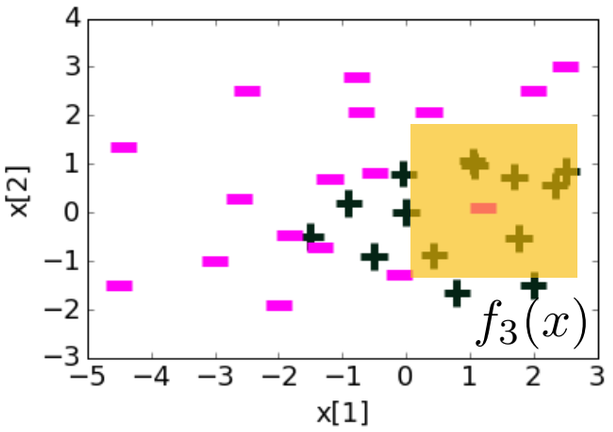
Aggregating weak classifiers

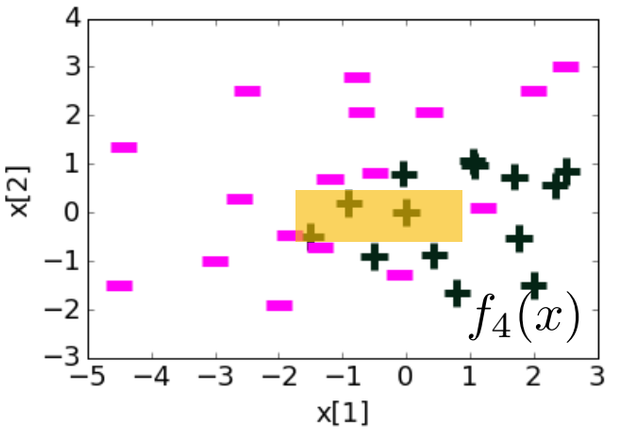


- Consider a scenario where we train many **weak** classifiers on a given data

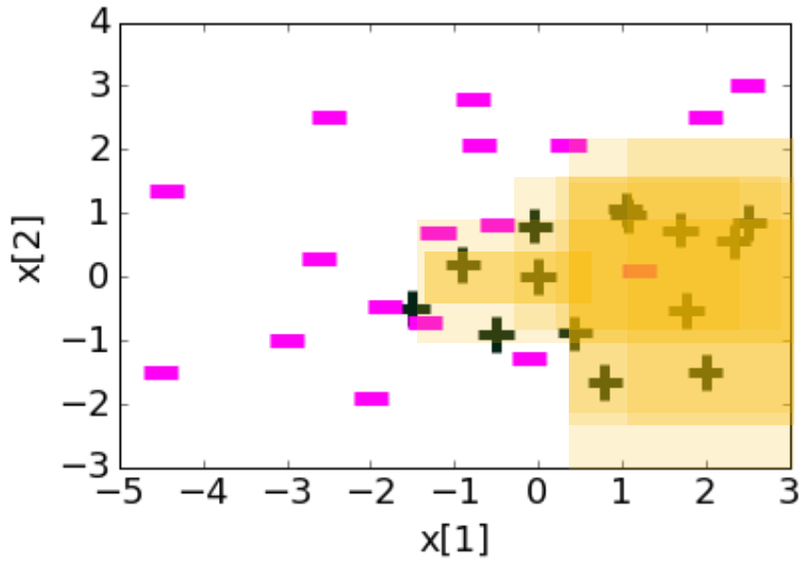




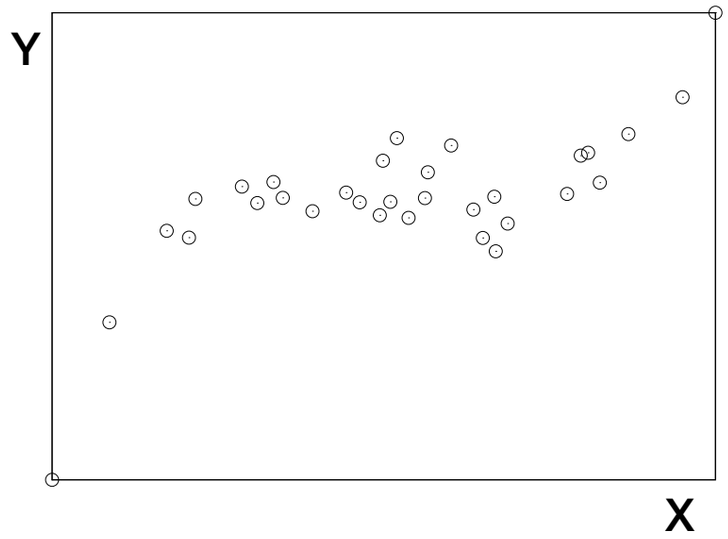




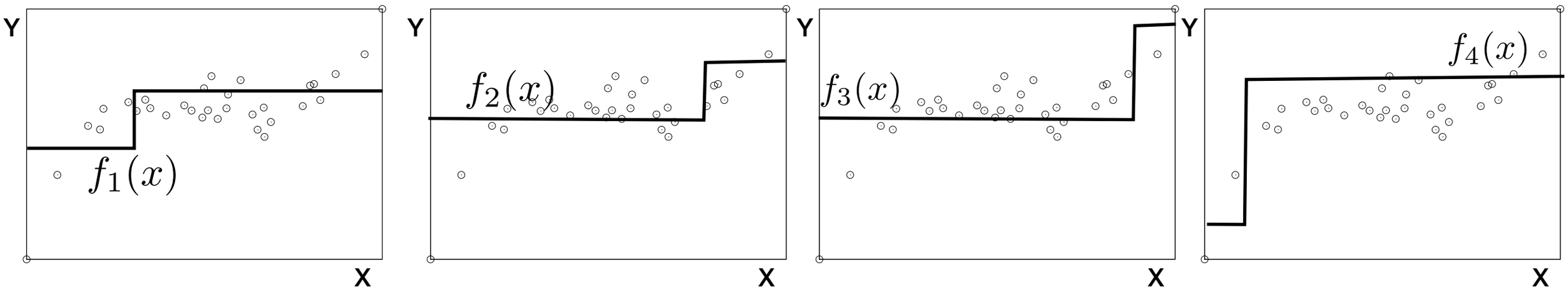
- By taking a weighted average of those weak classifiers with appropriately chosen weights, we might get a **strong** classifier



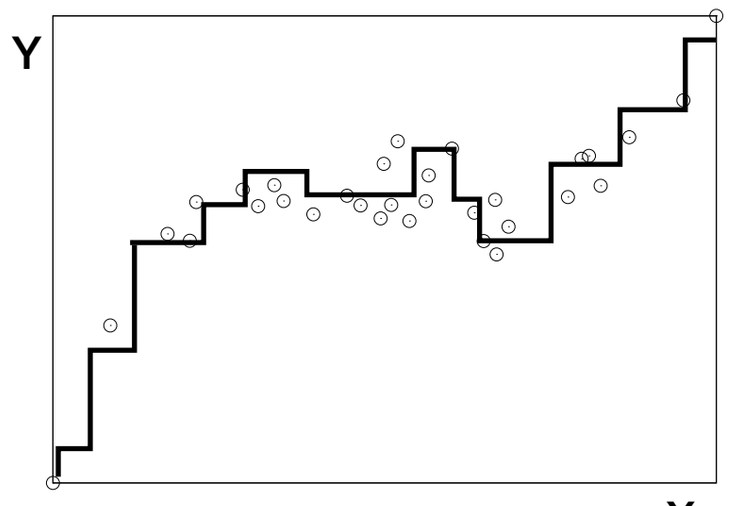
$$f(x) = w_1 f_1(x) + w_2 f_2(x) + \cdots$$



- we train many **weak** regressors on a given data



- By taking a weighted average of those weak regressors with appropriately chosen weights, we might get a **strong** regressor



$$f(x) \;=\; w_1 f_1(x) + w_2 f_2(x) + \cdots$$

Ensemble classifier

- Given data {(x_i, y_i)}
- Train an ensemble of classifiers: $f_1(x), f_2(x), \cdots, f_T(x)$ each with its own model parameters
- Train weight parameters $w_1, w_2, \cdots, w_T$ to predict

$$f(x) = w_1 f_1(x) + w_2 f_2(x) + \cdots$$

- If we need to make a hard prediction in {-1,1}, then

$$\hat{y} = \text{sign}(w_1 f_1(x) + w_2 f_2(x) + \cdots + w_T f_T(x))$$

- But, which classifiers do we use?
  and how do we choose the weights?

Ensemble classifier

- Training a single decision tree often results in either
  - a large tree that is overfitted; or
  - a small tree that is a weak classifier
- Principled averaging methods
  - **Bagging**: primary goal is variance reduction
    - Fit many large trees
    - using **bootstrap-resampled** versions of training data
    - and classify using majority vote
  - **Boosting**: primary goal is bias reduction
    - Fit many small trees
    - using **re-weighted** visions of the training data
    - and classify using weighted majority vote
- In general,
  - Boosting > Bagging > single decision tree
  - **AdaBoost** is the best off-the-shelf classifier; the most popular classifier on Kaggle challenges

Boosting:

How do we find those classifiers to add?

Boosting

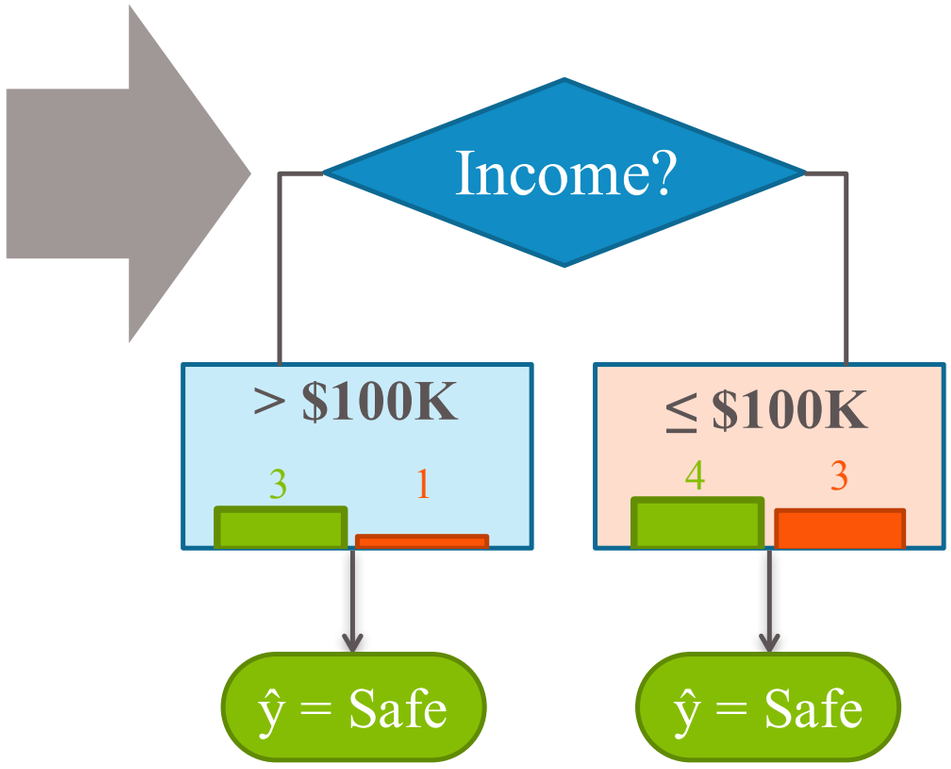
- Idea:
  - Sequentially add a new classifier each round, which are chosen to work well on hard example

- Technique:
  - maintain a appropriately **weighted dataset**
  - at the end of each round, each data point is weighted by α_i, such that if data point (x_i, y_i) is hard to classify with current models, then it is weighted high
  - When training a new model to be added in the next round, data point (x_i, y_i) is counted as α_i data points
  - this ensures that newly added models are different from existing ones, as it can better classify a subset of examples that are hard for existing models

Example: round 1

- In round 1, start with all weights 1, that is $\alpha_i = 1$ for all $i \in \{1, \cdots, N\}$
- and train a simple model, like a one-level decision tree (a.k.a. decision stump)
- At the end of training in round 1, increase α_i for misclassified data points as they are hard
  (and decrease it for correctly classified data points as they are easy)
- We will learn how to update the weights later (called AdaBoost)

Credit	Income	y
A	$130K	Safe
B	$80K	Risky
C	$110K	Risky
A	$110K	Safe
A	$90K	Safe
B	$120K	Safe
C	$30K	Risky
C	$60K	Risky
B	$95K	Safe
A	$60K	Safe
A	$98K	Safe



Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	1.2
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9

Example: round 2

- Train a new classifier (typically a decision tree) with the weighted examples by taking weighted count of the data points
(examples coming soon)

- If it is not decision tree, we train by minimizing weighted loss:

$$\text{minimize } \mathcal{L}(w) \;=\; \frac{1}{N}\sum_{i=1}^{N} \alpha_i \ell(\hat{y}_i, y_i)$$

AdaBoost:

Adaptive Boosting

AdaBoost

- Initialize $\alpha_i = \frac{1}{N}$ for all $i \in \{1, \cdots, N\}$
- For $t = 1, \cdots, T$
  - Train a classifier $f_t(x)$ on weighted training data
  - Compute coefficient $\hat{w}_t$ for the t-th classifier $f_t(x)$
  - Re-compute weights $\{\alpha_i\}_{i=1}^N$
- Final model is prediction:

$$\hat{y} = \text{sign}\Big(\sum_{t=1}^{T} \hat{w}_t f_t(x) \Big)$$

Computing $\hat{w}_t$

- Idea:
  at round ***t***, we want to give larger weight $\hat{w}_t$ to classifier $f_t(x)$, if it is accurate on the hard examples
- Formally,

$$\hat{w}_t = \frac{1}{2}\log\left(\frac{1-\text{WeightedError}(f_t)}{\text{WeightedError}(f_t)}\right)$$

  that is, higher weight to accurate classifier,
  but accuracy is computed on the weighted examples (at round t)

weighted_error(f_t)	$\frac{1-\text{WeightedError}(f_t)}{\text{WeightedError}(f_t)}$	$\hat{w}_t$	
0.01	99	2.3	informative & accurate
0.5	1	0.0	uninformative
0.99	0.01	-2.3	informative but opposite

- where,

$$\text{WeightedError}(f_t) = \frac{1}{N}\sum_{i=1}^{N}\alpha_i \mathbb{I}(\underbrace{\hat{y}_i}_{f_t(x_i)} \neq y_i)$$

- note that weighted error of ZERO or ONE can be problematic

Justification of the coefficient

$$\hat{w}_t = \frac{1}{2}\log\left(\frac{1-\text{WeightedError}(f_t)}{\text{WeightedError}(f_t)}\right)$$

- Consider each classifier ***$f_t(x)$*** as an expert, telling you what he thinks the label is for a point ***x***
- Consider each expert as giving independent advice, which is correct with probability (1-WeightedError)
- Then the Likelihood of the label ***y*** at point ***x*** is

$$\log\left(\frac{\mathbb{P}(\text{expert advices}|\text{true } y=+1)}{\mathbb{P}(\text{expert advices}|\text{true } y=-1)}\right) = f_1(x)\log\left(\frac{1-WeightedError(f_1)}{WeightedError(f_1)}\right)+\cdots+f_t(x)\log\left(\frac{1-WeightedError(f_t)}{WeightedError(f_t)}\right)$$

- Hence the maximum likelihood estimate of the label is

$$\text{sign}(\hat{w}_1 f_1(x) + \cdots \hat{w}_t f_t(x))$$

which is exactly the weights (or coefficients) used in AdaBoost

- But, note that in practice, ***$f_t(x)$***'s are not independent experts

Re-compute α_i's

- Idea: each data point ***(x_i,y_i)*** is weighted by α_i to reflect how difficult it is to correctly classify that point with the current averaged classifier $f(x) = \hat{w}_1 f_1(x) + \cdots + \hat{w}_t f_t(x)$

- AdaBoost uses the exponential weight to measure how hard that data point is:

$$\alpha_i \;=\; e^{-y_i f(x_i)} \;=\; e^{-\hat{w}_1(y_i f_1(x_i)) - \cdots - \hat{w}_t(y_i f_t(x_i))}$$

- this can be computed by iteratively updating the weights as follows:

$$\alpha_i \longleftarrow \begin{cases} e^{-\hat{w}_t}\alpha_i & \text{if correct: } f_t(x_i) = y_i \\ e^{\hat{w}_t}\alpha_i & \text{if error: } f_t(x_i) \neq y_i \end{cases}$$

- Implications of the weight update

$$\alpha_i \longleftarrow \begin{cases} e^{-\hat{w}_t}\alpha_i & \text{if correct: } f_t(x_i) = y_i \\ e^{\hat{w}_t}\alpha_i & \text{if error: } f_t(x_i) \neq y_i \end{cases}$$

$f_t(x_i)=y_i$?	$\hat{w}_t$	Multiply α_i by	Implication
yes	2.3	$e^{-2.3} = 0.1$	decrease importance of easy example
yes	0	$e^{0} = 1$	no change if predictor is bad
yes	-2.3	$e^{2.3} = 10$	increase weight for easy examples, for bad predictors
no	2.3	$e^{2.3} = 10$	increase importance of hard example
no	0	$e^{0} = 1$	no change if predictor is bad
no	-2.3	$e^{-2.3} = 0.1$	decrease weight for hard examples, for bad predictors

Normalizing weights

- As we update the weights multiplicatively, AdaBoost can become numerically unstable if some weights are too large

If x_i often mistake, weight α_i gets very large

If x_i often correct, weight α_i gets very small

Can cause numerical instability after many iterations

Normalize weights to add up to 1 after every iteration

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

AdaBoost

- Start with same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(x)$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathbf{w}}_t f_t(\mathbf{x})\right)$$

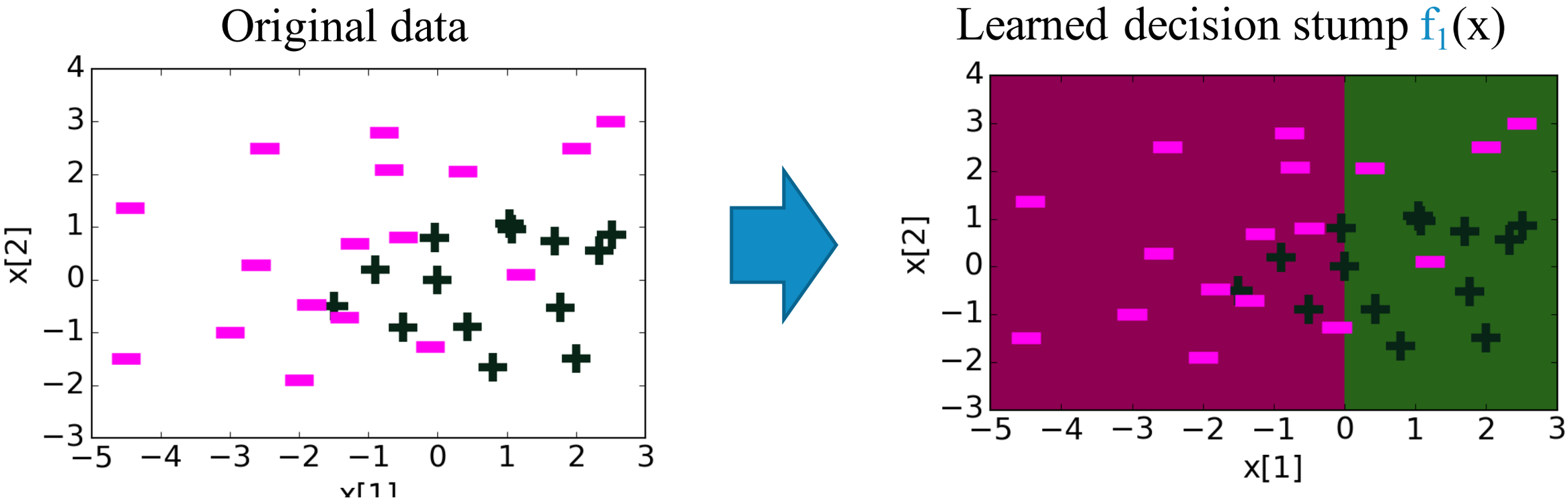
$$\hat{\mathbf{w}}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(x_i)=y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(x_i)\neq y_i \end{cases}$$

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

Example

- At t=1, it is the standard training of a decision tree but we train a simple model, in this case a decision stump (which is another name for a decision tree of level 1)



- Then compute the weight of the classifier, based on the error: 9/30 -> weight=0.5 log (21/9) = 0.424

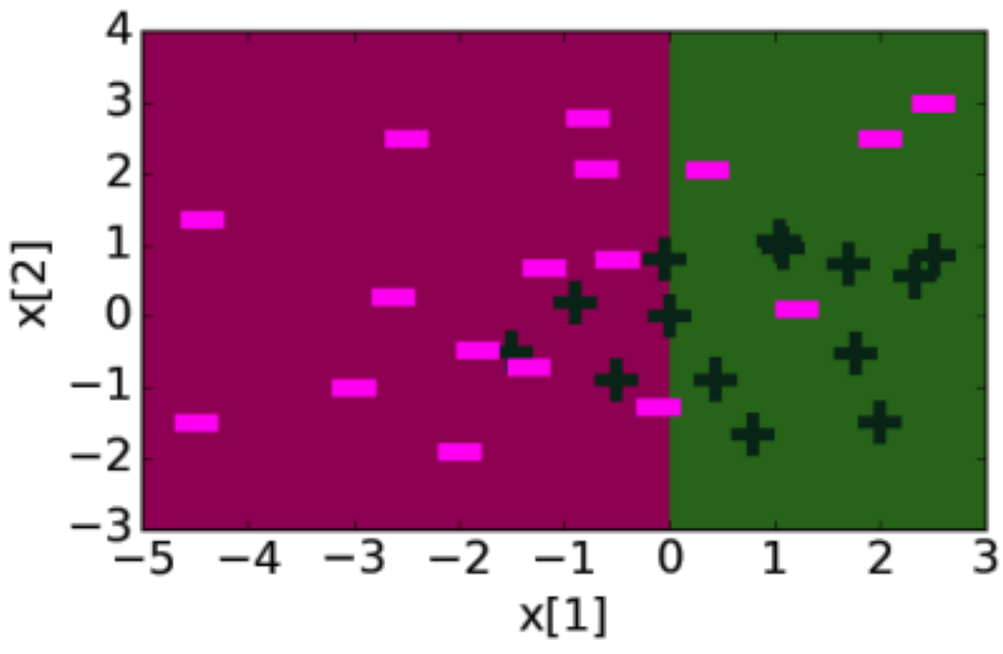
$$\hat{w}_t = \frac{1}{2}\log\left(\frac{1-\text{WeightedError}(f_t)}{\text{WeightedError}(f_t)}\right)$$

- At t=1, we then update the weights of the data points

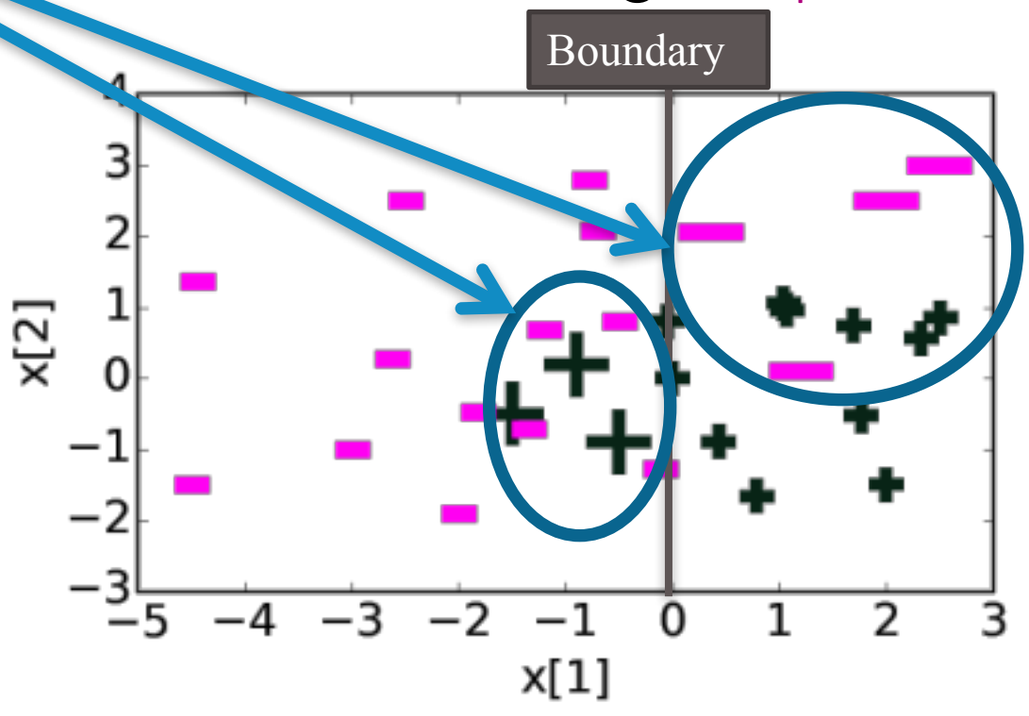
WeightedError was 9/30
then, $e^{\hat{w}_1} = \sqrt{\frac{1-WeightedError}{WeightedError}} = \sqrt{7/3} = 1.53$

Increase weight α_i of misclassified points

Learned decision stump $f_1(x)$



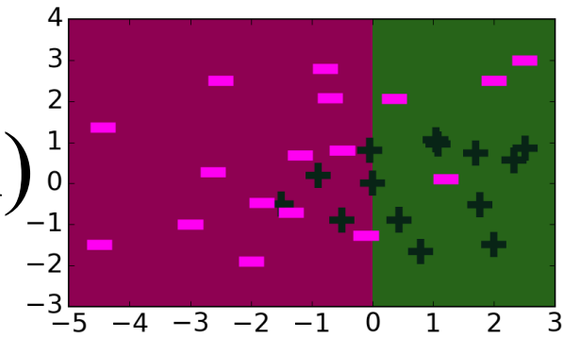
New data weights α_i



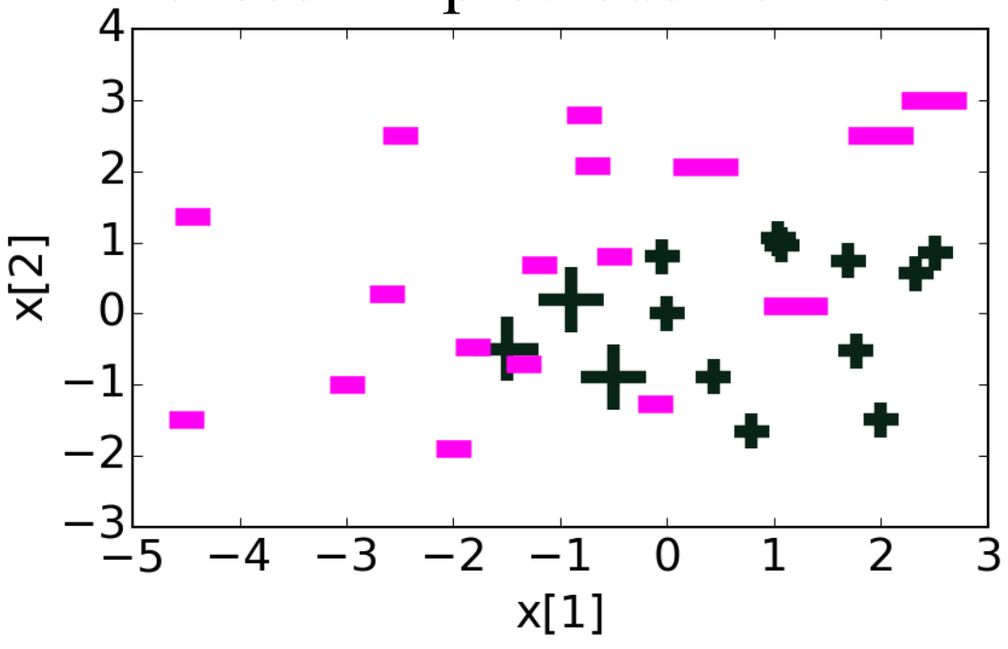
$$\alpha_i \longleftarrow \begin{cases} e^{-\hat{w}_t}\alpha_i & \text{if correct: } f_t(x_i) = y_i \\ e^{\hat{w}_t}\alpha_i & \text{if error: } f_t(x_i) \neq y_i \end{cases}$$

- At t=2, we train a new classifier on the weighted data

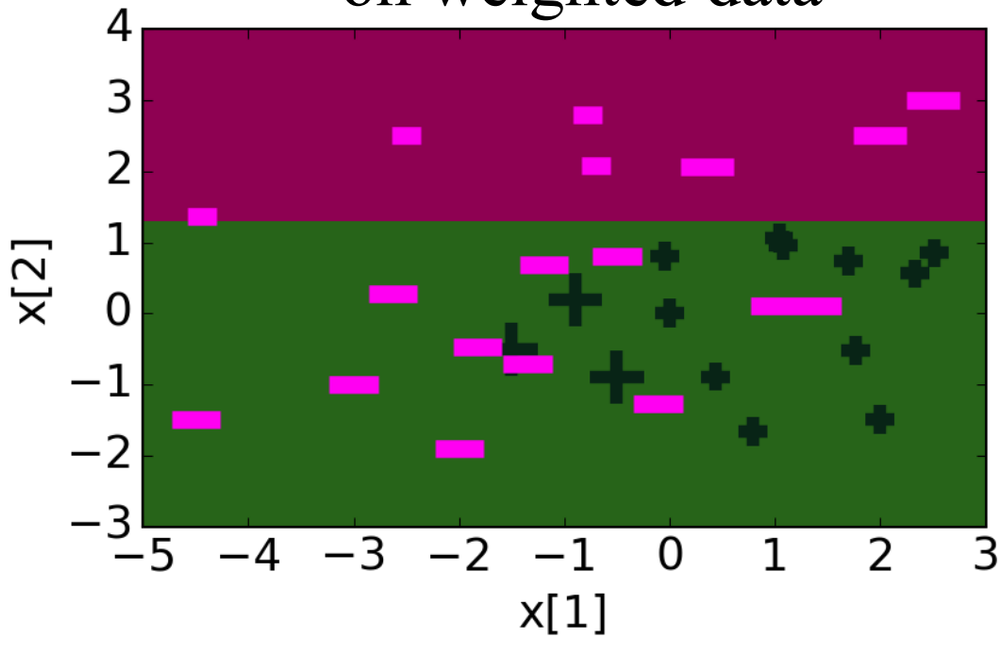
$f_1(x)$



Weighted data: using α_i chosen in previous iteration

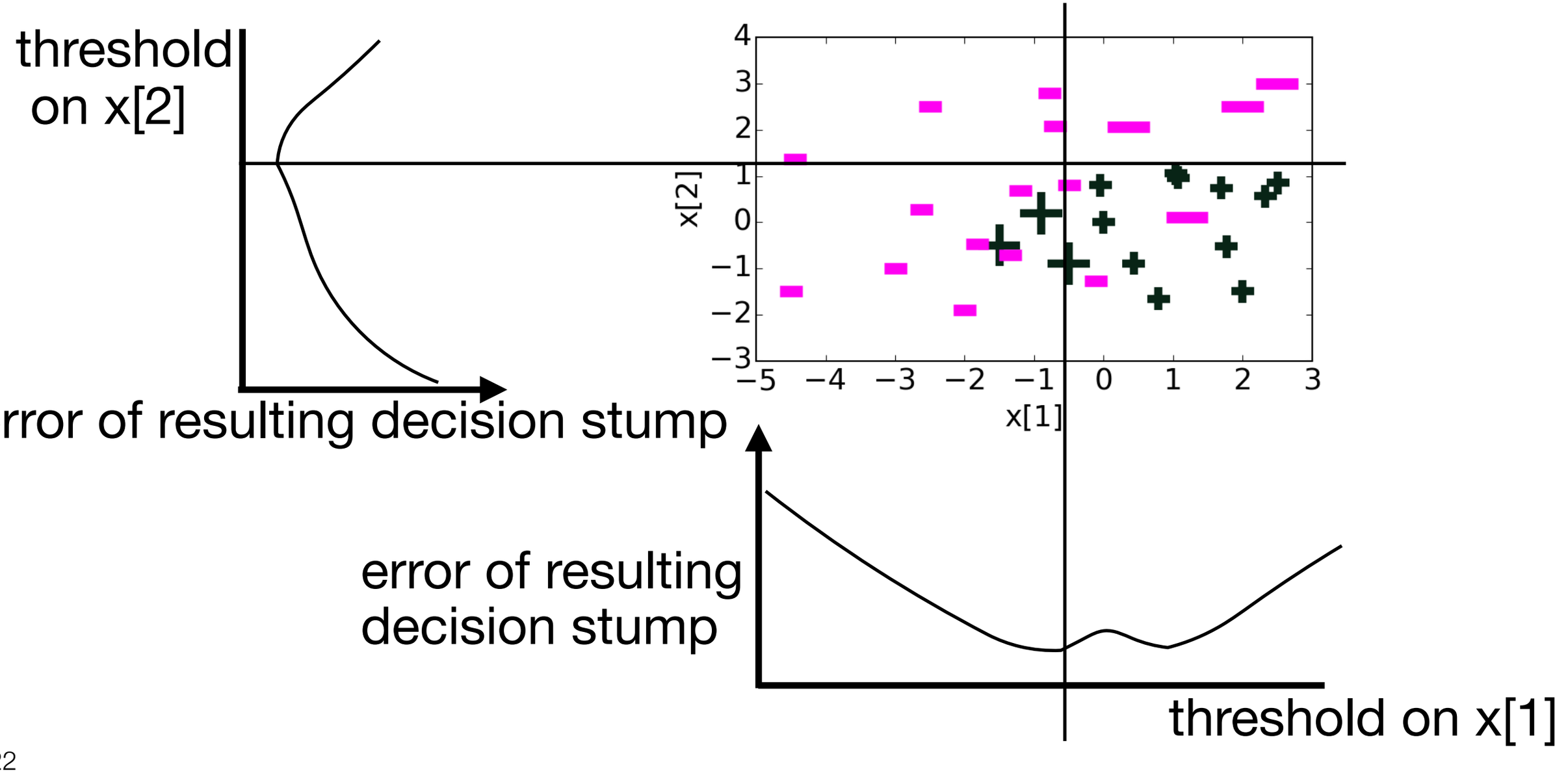


Learned decision stump $f_2(x)$ on weighted data

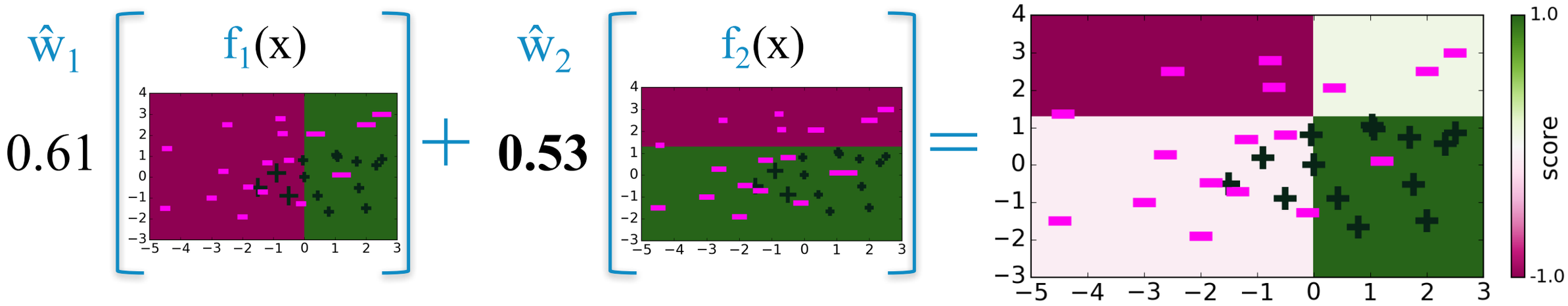


- then we compute the weight of the classifier

- When training a decision stump on weighted samples, it is exactly like training a decision stump on (unweighted) samples, but competing each data point according to the weight
- Concretely, for each feature x[1] and x[2], we consider splitting at any possible point that is in between two data points, and find the one that minimizes the error



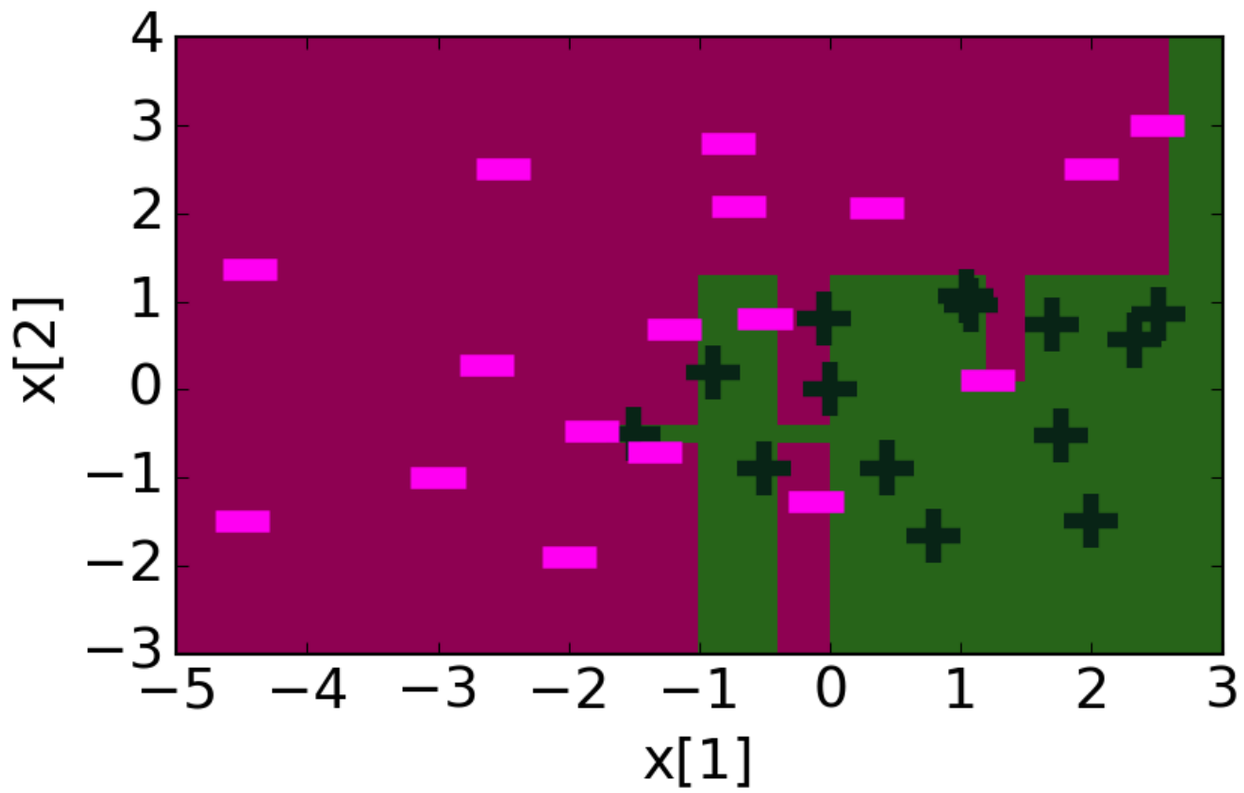
- weighted sum of the learned predictors give a new ensemble predictor



- At t=2, we then update the weights fo the data points

After 30 iterations

- after enough iterations, we get zero training error
- but probably overfitted

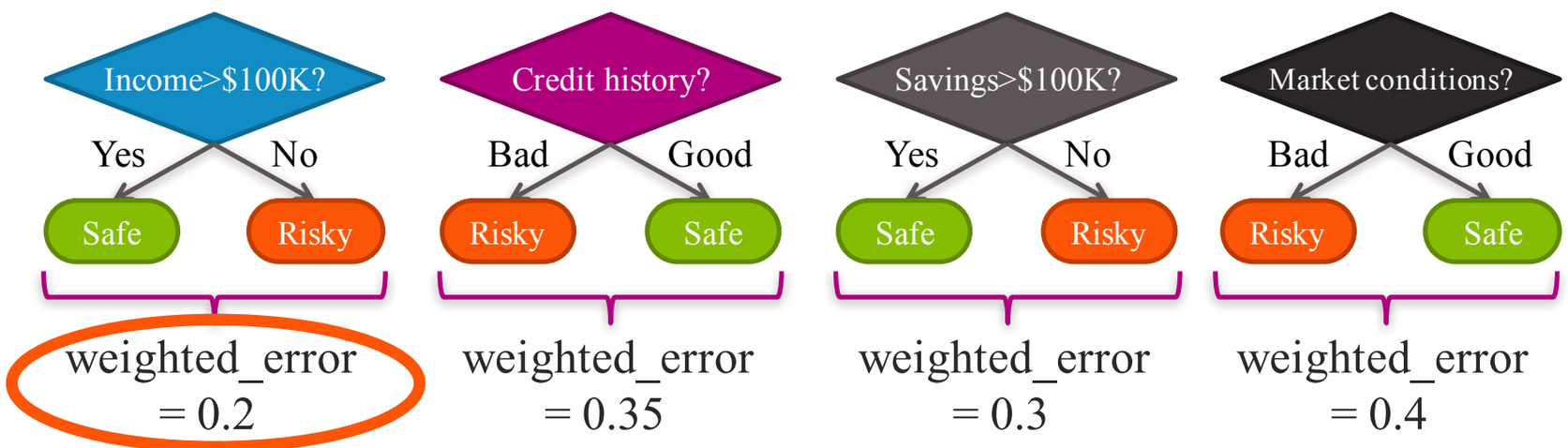


training_error = 0

Example: loan default

- finding the best decision stump on weighted data points
- and compute the new eight of the learned decision stump

Consider splitting on each feature:



$f_t =$

$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right) = 0.69$$

As WeightedError=0.2 and $\hat{w}_t = 0.5\log(\frac{1-WeightedError}{WeightedError})$

$$e^{\hat{w}_t} = 2$$

- Updating the weights of the data points

$$\alpha_i \longleftarrow \begin{cases} e^{-\hat{w}_t}\alpha_i & \text{if correct: } f_t(x_i) = y_i \\ e^{\hat{w}_t}\alpha_i & \text{if error: } f_t(x_i) \neq y_i \end{cases}$$

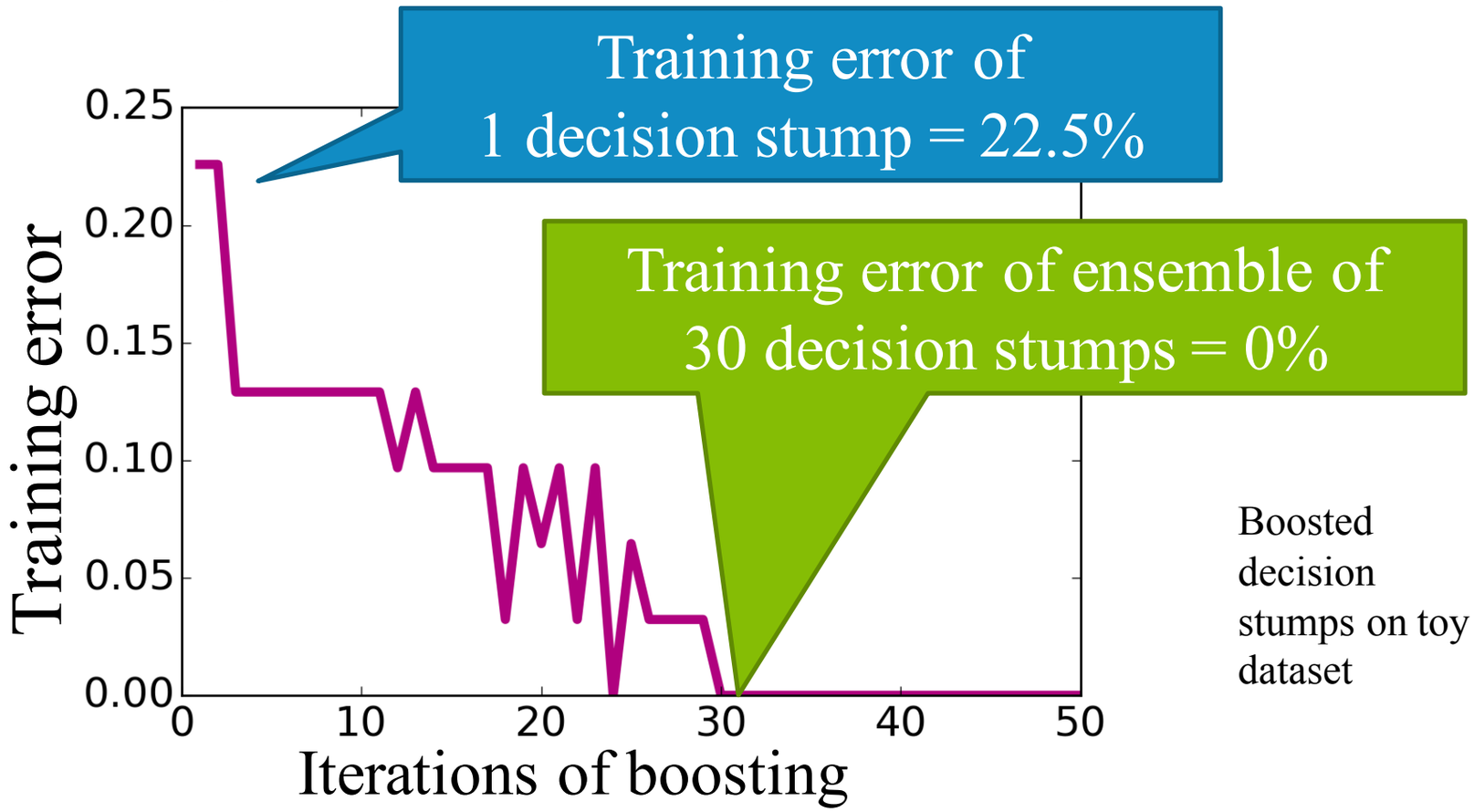
$$e^{\hat{w}_t} = 2$$

Credit	Income	y	ŷ	Previous	New
A	$130K	Safe	Safe	0.5	0.5/2 = 0.25
B	$80K	Risky	Risky	1.5	0.75
C	$110K	Risky	Safe	1.5	2 * 1.5 = 3
A	$110K	Safe	Safe	2	1
A	$90K	Safe	Risky	1	2
B	$120K	Safe	Safe	2.5	1.25
C	$30K	Risky	Risky	3	1.5
C	$60K	Risky	Risky	2	1
B	$95K	Safe	Risky	0.5	1
A	$60K	Safe	Risky	1	2
A	$98K	Safe	Risky	0.5	1

Convergence and Overfitting in Boosting

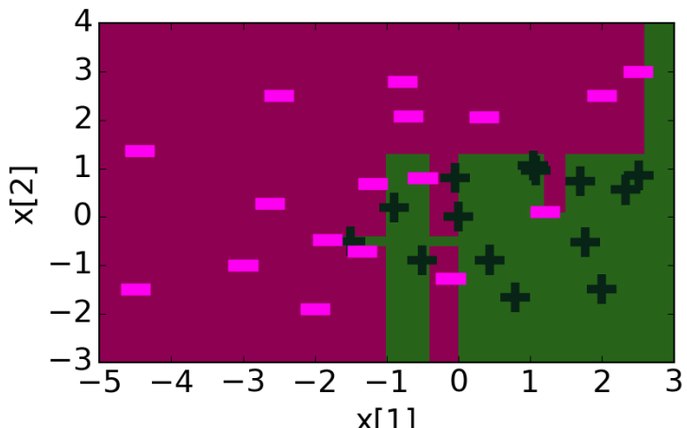
Boosting example

- training error goes to zero, after some iterations!



Boosted decision stumps on toy dataset

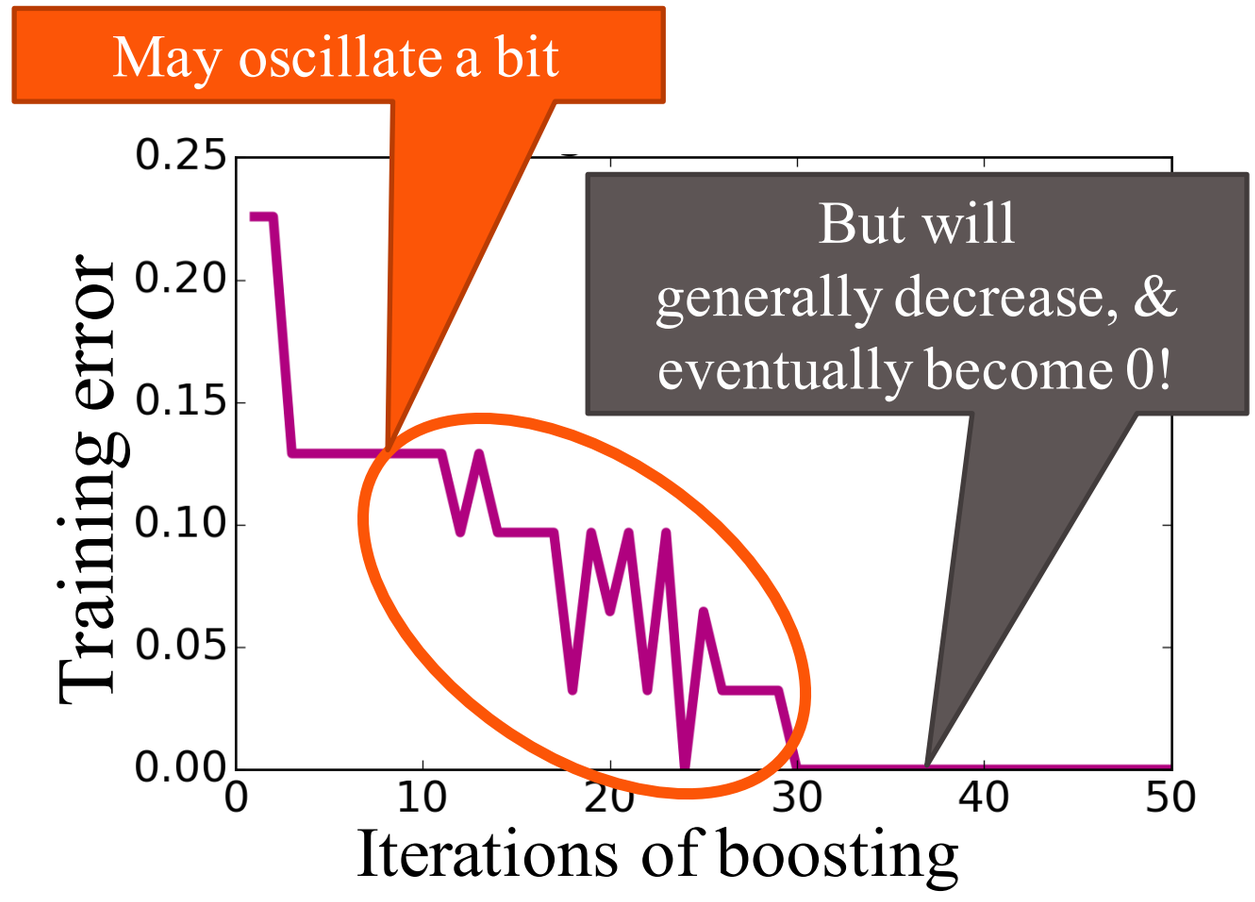
- This implies that
  - 1. average of decision stumps can **represent** any complex function
  - 2. via Boosting, the weighted averaging of simple decision stumps has in the end correct **learned** the function



AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞



Conditions of AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞

Condition = At every t, can find a weak learner with weighted_error(f_t) < 0.5

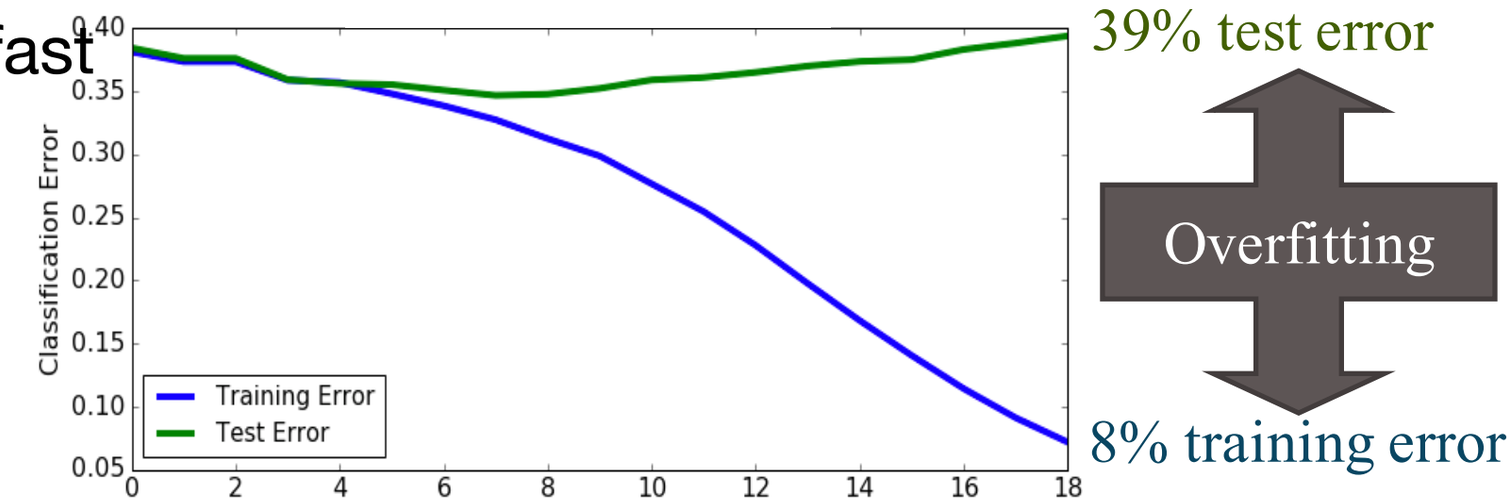
Not always possible

Extreme example: No classifier can separate a +1 on top of -1

Nonetheless, boosting often yields great training error

Comparisons

- Single decision tree vs. Boosting
  - Single decision tree:
    - best test error is 34%
    - training error decreases fast
    - overfits fast



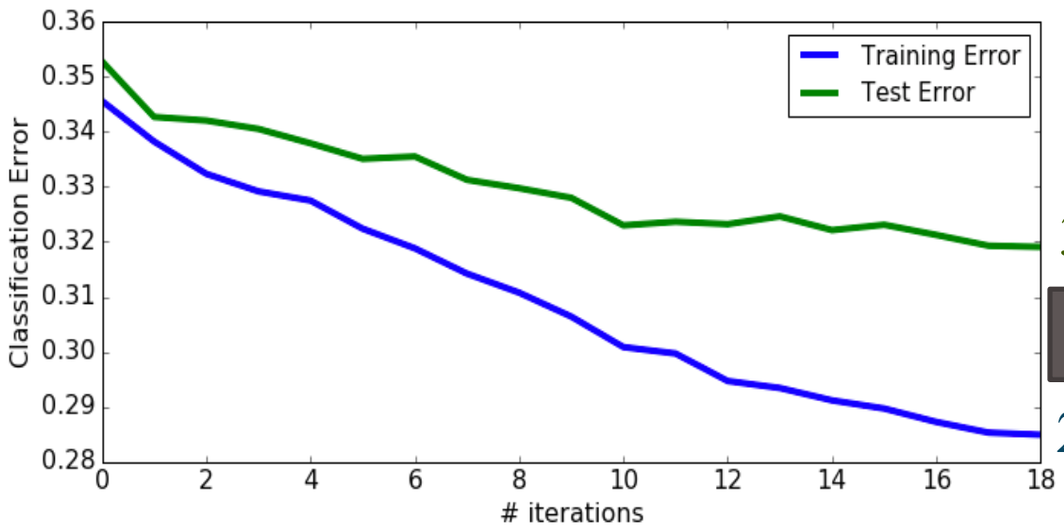
39% test error

Overfitting

8% training error

  - AdaBoost:
    - best test error is 32%
    - training error decreases slow
    - overfits less

Boosted decision stumps on loan data

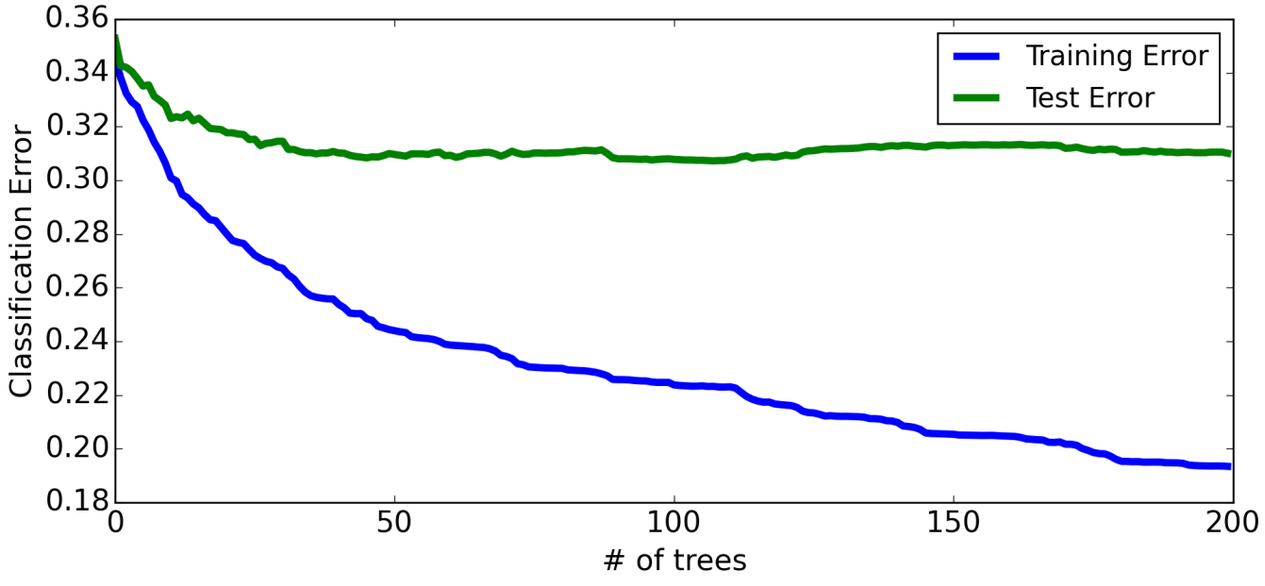


32% test error

Better fit & lower test error

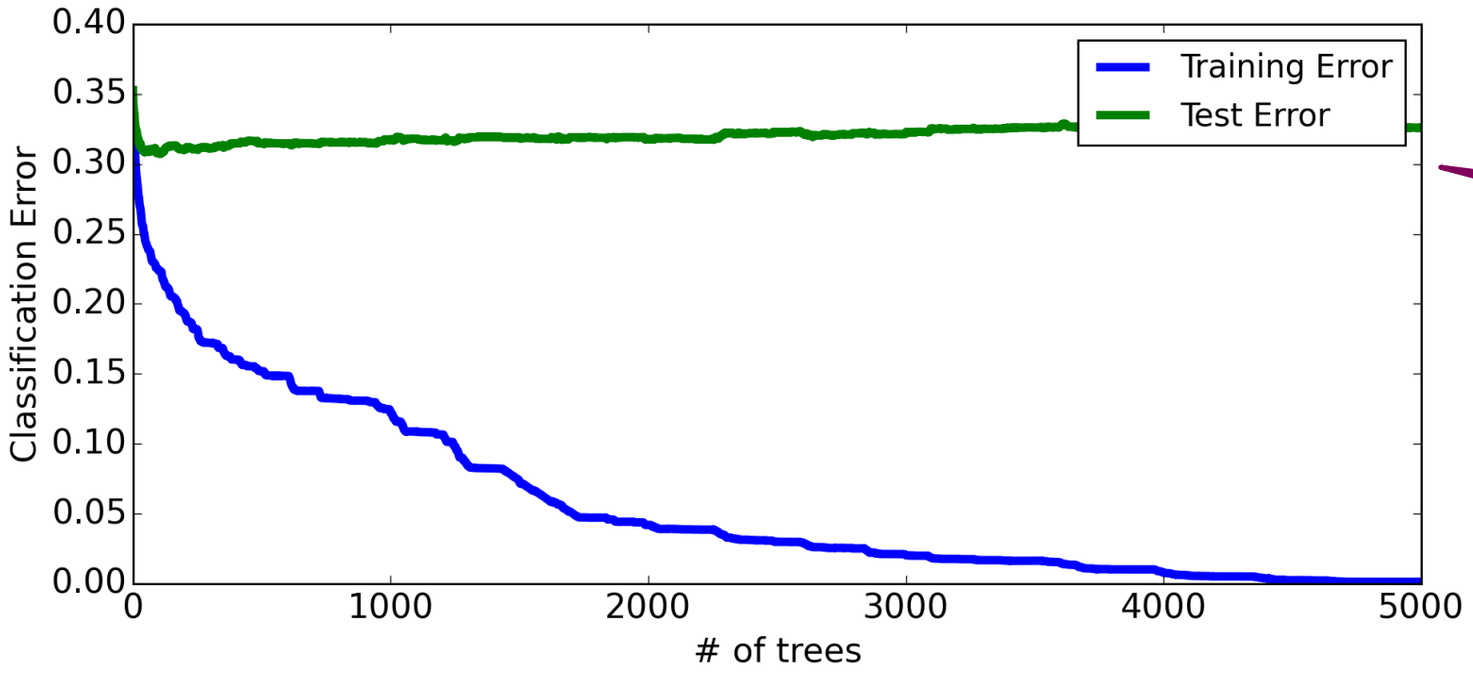
28.5% training error

Boosting tends to be robust to overfitting



Test set performance about constant for many iterations
➔ Less sensitive to choice of T

- But will eventually overfit



Best test error around 31%

Test error eventually increases to 33% (overfits)

- We can use cross-validation or validation to choose # of iterations

Why does boosting work so well in practice?

- If the training data can be perfectly classified by some function (also known as separable)
  then eventually boosting can achieve ZERO training error
- However, at this point, usually we overfitted training data so test error is bad
- in many examples, further training with boosting after training error has reached zero **improves test error**
  (note that training error stays at zero after this point)
- Reason for boosting's robustness to overfitting
  - further updates only change a small subset of dimensions
  - classifier parameters and classifier weights are separately updated
- Boosting can be thought of as logistic regression over many simple classifiers, trying to minimize

Variants of boosting and related algorithms

There are hundreds of variants of boosting, most important:

Gradient boosting

- Like AdaBoost, but useful beyond basic classification

Many other approaches to learn ensembles, most important:

Random forests

- Bagging: Pick random subsets of the data
  - Learn a tree in each subset
  - Average predictions
- Simpler than boosting & easier to parallelize
- Typically higher error than boosting for same # of trees (# iterations T)

Impact of boosting (*spoiler alert... HUGE IMPACT*)

Amongst most useful ML methods ever created

Extremely useful in computer vision

- Standard approach for face detection, for example

Used by most winners of ML competitions (Kaggle, KDD Cup,…)

- Malware classification, credit fraud detection, ads click through rate estimation, sales forecasting, ranking webpages for search, Higgs boson detection,…

Most deployed ML systems use model ensembles

- Coefficients chosen manually, with boosting, with bagging, or others