

Ensemble methods

- A lot of our ML models are *s / o w* to train and predict
 - ▶ e.g., big transformer language models
- But some are much faster
 - ▶ e.g., decision trees, logistic regression, small deep nets
- Means we could afford to train and use many models:
ensemble methods
- What benefits can we unlock?
 - ▶
 - ▶
- Drawbacks: ↓ interpretability, ↓ speed
- Methods in today's slides were SOTA 10–15 years ago, but
 - ▶ still have some advantages (speed, parallelizability)
 - ▶ good to know the ideas since tech is rapidly changing

Quantify uncertainty



from MNIST

- Suppose we managed to train a diverse set of 100 different models θ_k (e.g., decision trees) from the same training dataset $\mathcal{D}$ for handwriting recognition
- Given test instance $\mathbf{x}$ at left:
 - ▶ 37 models predict $y = 9$ (digit 9)
 - ▶ 48 models predict $y = a$ (lowercase a)
 - ▶ 15 models predict $y = o$ (lowercase O)
- Interpretation: posterior probability of label $y \mid \mathbf{x}$ (the ***predictive distribution***) is 37%, 48%, 15% on these outcomes
 - ▶ this is great because the true predictive distribution is intractable: $\int P(y \mid \theta) P(\theta \mid \mathcal{D}) d\theta$
 - ▶ ensemble can provide a good heuristic approximation

Quantify uncertainty



from MNIST

- Suppose we managed to train a diverse set of 100 different models θ_k (e.g., decision trees) from the same training dataset $\mathcal{D}$ for handwriting recognition
- Given test instance $\mathbf{x}$ at left:
 - ▶ 37 models predict $y = 9$ (digit 9)
 - ▶ 48 models predict $y = 0$ (digit 0)
 - ▶ 15 models predict $y = 1$ (digit 1)
- Interpretation: predictive distribution of label y | $\mathbf{x}$ (the probability of label y | $\mathbf{x}$ if we train our 100 models) is 37%, 48%, 15% on these outcomes
 - ▶ this is great because the true predictive distribution is intractable: $\int P(y | \theta) P(\theta | \mathcal{D}) d\theta$
 - ▶ ensemble can provide a good heuristic approximation

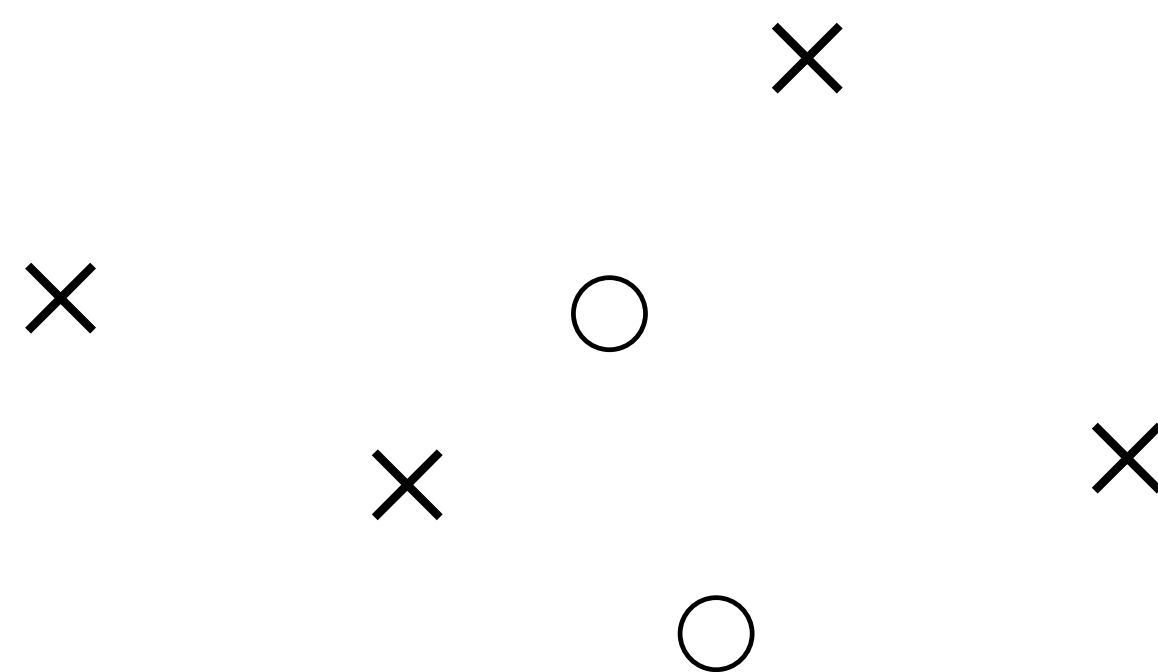
Key question: how should we train our 100 models?

Idea: *subsample* $\mathcal{D}$

- Train different models on different subsets of $\mathcal{D}$
 - ▶ e.g., k -fold cross-validation trains k models
 - ▶ e.g., *bootstrap...*

Bootstrap

- Repeat T times:
 - ▶ take a bootstrap resample D_t from $\mathcal{D}$: select $(\mathbf{x}^{(i)}, y^{(i)})$ iid from $\mathcal{D}$ *with replacement* $|\mathcal{D}|$ times — called a *bag*
 - ▶ same size as original sample, but some datapoints are excluded and some are repeated
 - ▶ in expectation $\frac{1}{e}$ excluded (about 37%)
 - ▶ train h_t on bag D_t , test h_t on excluded (*out-of-bag*) points $\mathcal{D} \setminus D_t$
(points are actually exact duplicates)

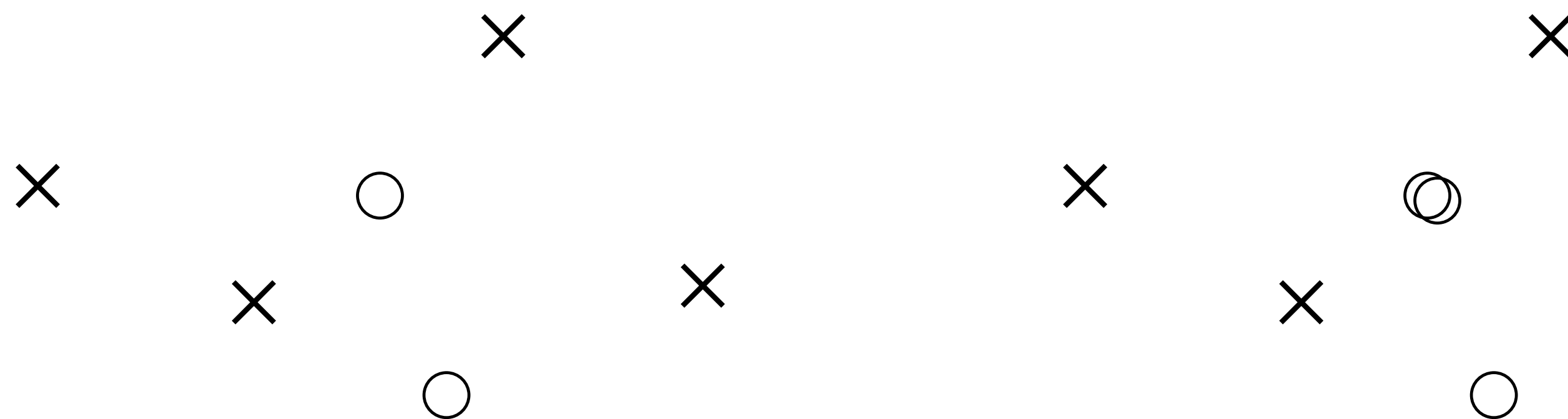


original sample

bootstrap resample

Bootstrap

- Repeat T times:
 - ▶ take a bootstrap resample D_t from $\mathcal{D}$: select $(\mathbf{x}^{(i)}, y^{(i)})$ iid from $\mathcal{D}$ *with replacement* $|\mathcal{D}|$ times — called a *bag*
 - ▶ same size as original sample, but some datapoints are excluded and some are repeated
 - ▶ in expectation $\frac{1}{e}$ excluded (about 37%)
 - ▶ train h_t on bag D_t , test h_t on excluded (*out-of-bag*) points $\mathcal{D} \setminus D_t$
- (points are actually exact duplicates)

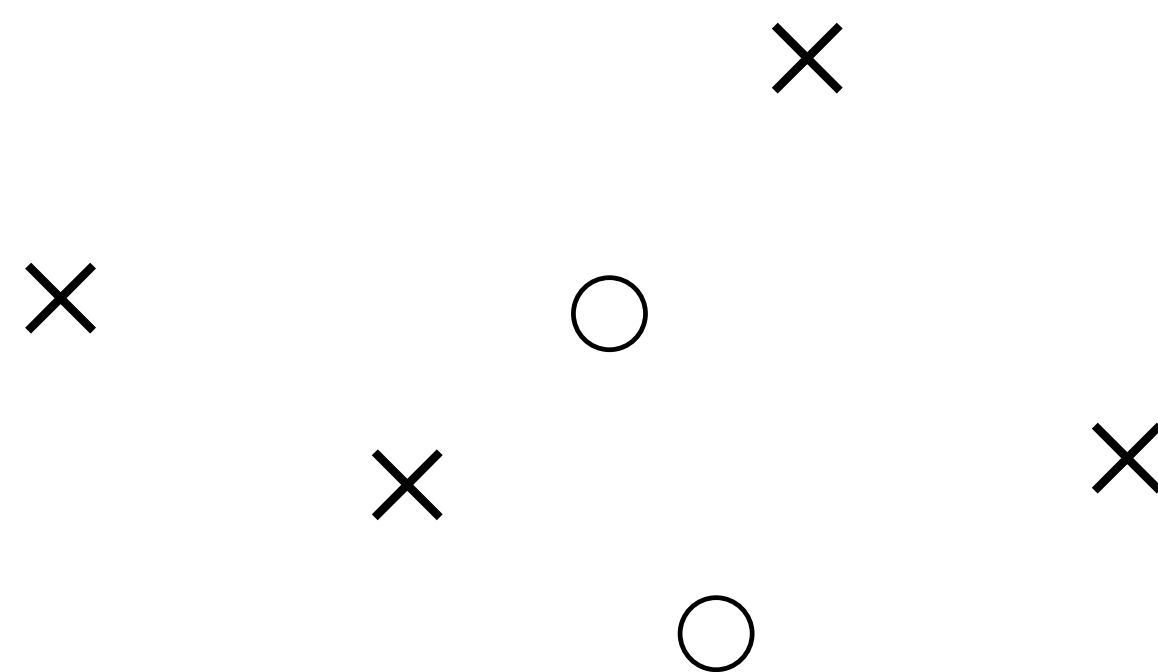


original sample

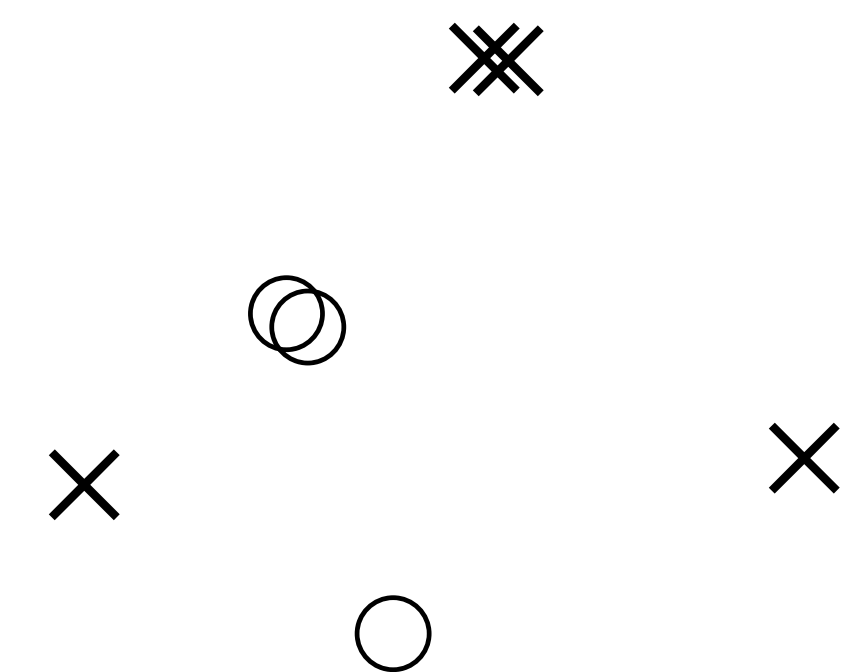
bootstrap resample

Bootstrap

- Repeat T times:
 - ▶ take a bootstrap resample D_t from $\mathcal{D}$: select $(\mathbf{x}^{(i)}, y^{(i)})$ iid from $\mathcal{D}$ *with replacement* $|\mathcal{D}|$ times — called a *bag*
 - ▶ same size as original sample, but some datapoints are excluded and some are repeated
 - ▶ in expectation $\frac{1}{e}$ excluded (about 37%)
 - ▶ train h_t on bag D_t , test h_t on excluded (*out-of-bag*) points $\mathcal{D} \setminus D_t$
- (points are actually exact duplicates)



original sample

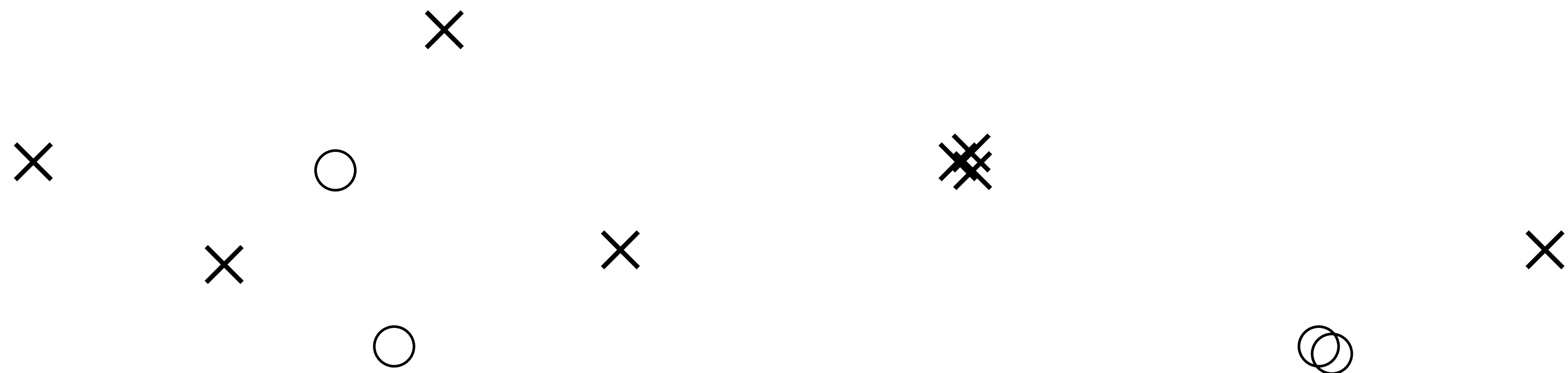


bootstrap resample

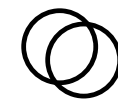
Bootstrap

- Repeat T times:
 - ▶ take a bootstrap resample D_t from $\mathcal{D}$: select $(\mathbf{x}^{(i)}, y^{(i)})$ iid from $\mathcal{D}$ *with replacement* $|\mathcal{D}|$ times — called a *bag*
 - ▶ same size as original sample, but some datapoints are excluded and some are repeated
 - ▶ in expectation $\frac{1}{e}$ excluded (about 37%)
 - ▶ train h_t on bag D_t , test h_t on excluded (*out-of-bag*) points $\mathcal{D} \setminus D_t$

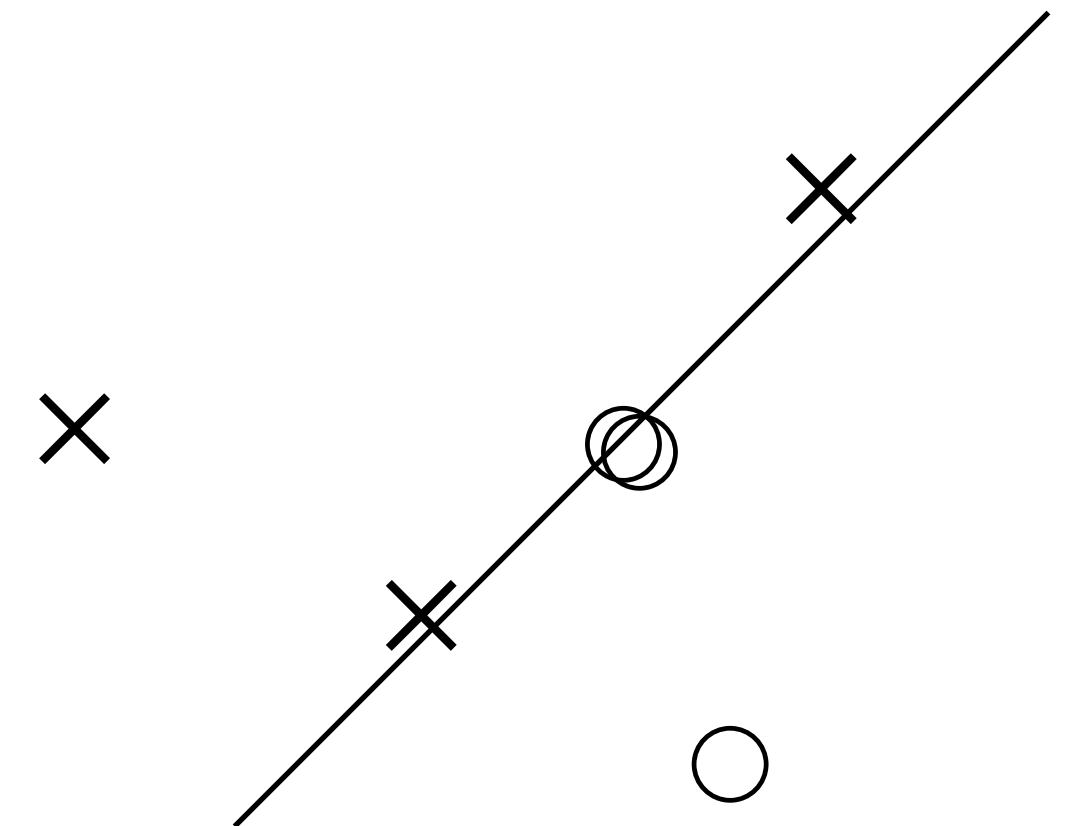
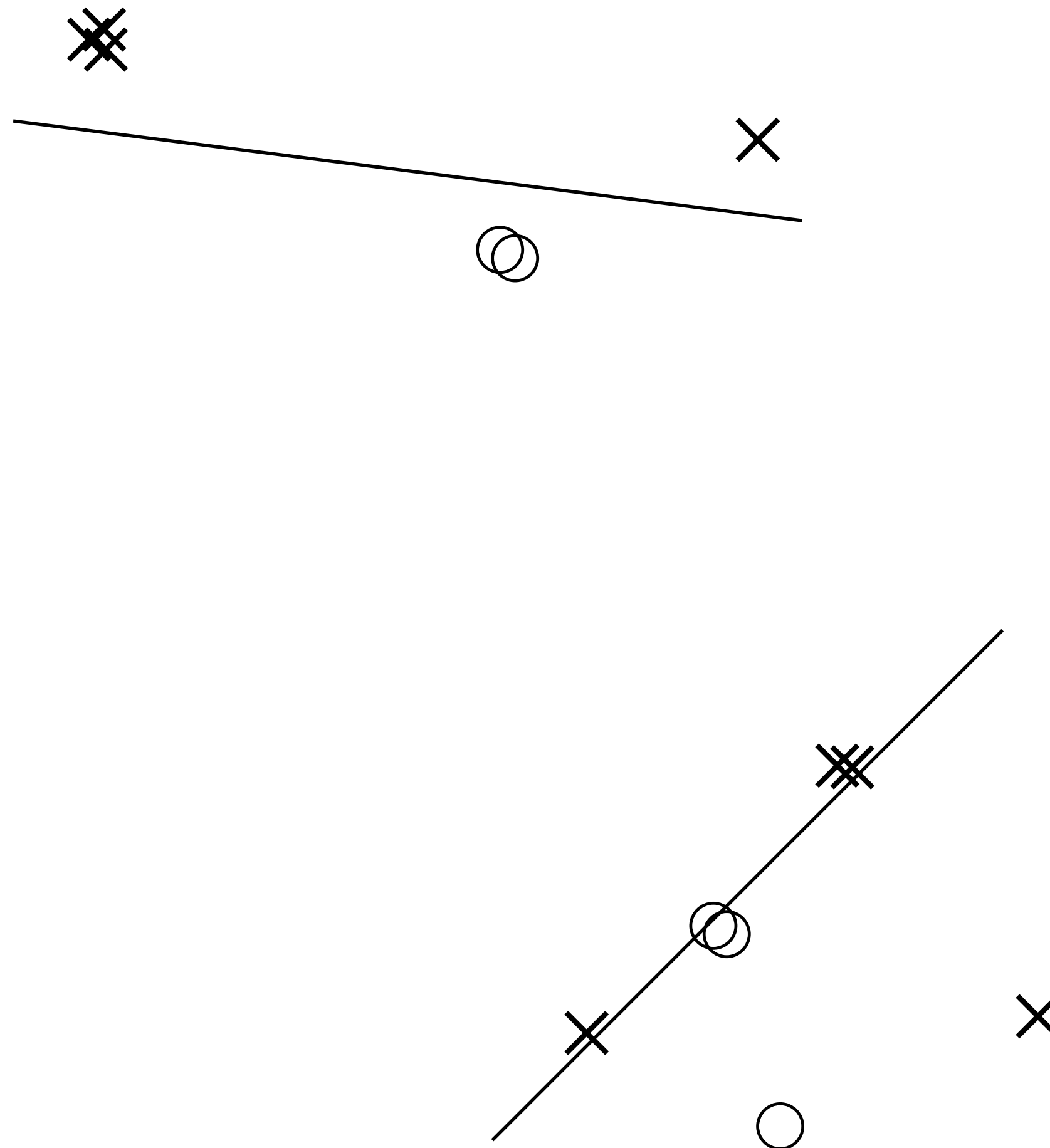
(points are actually exact duplicates)



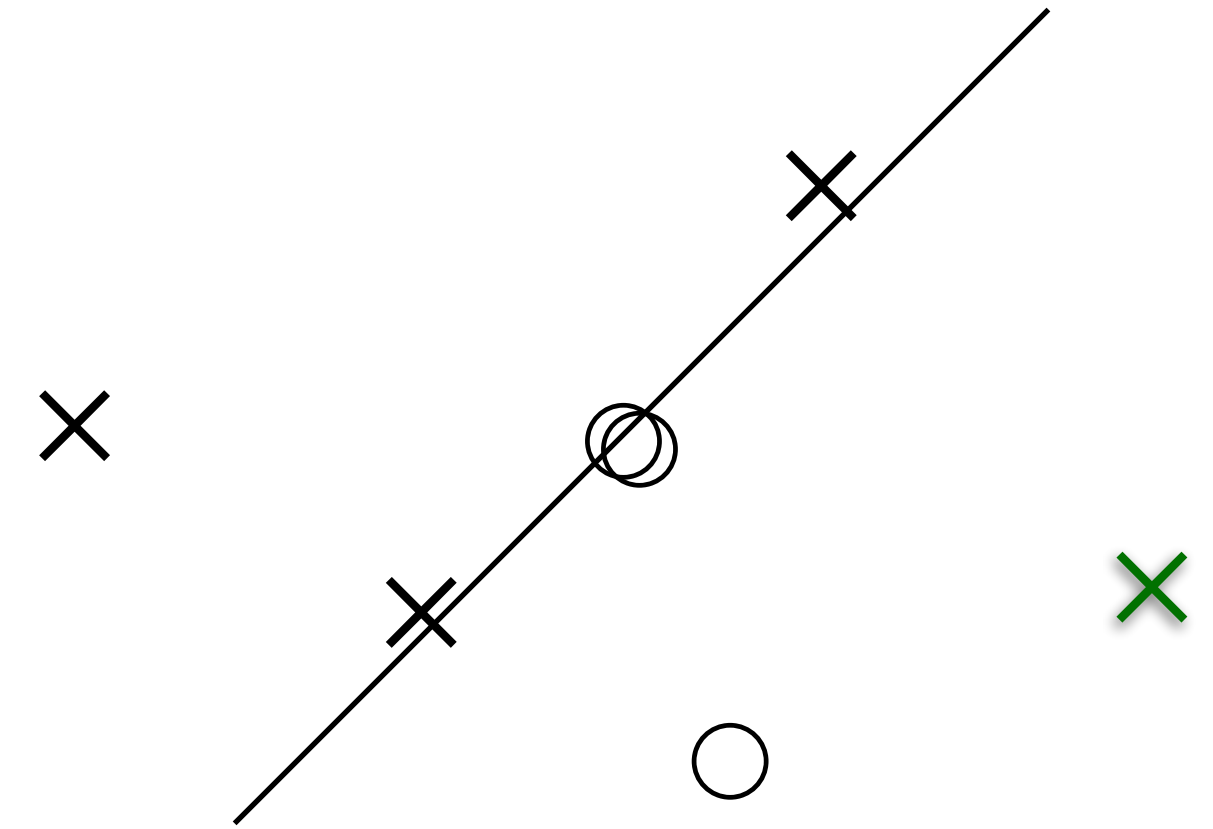
***Train on
each
bootstrap
sample***



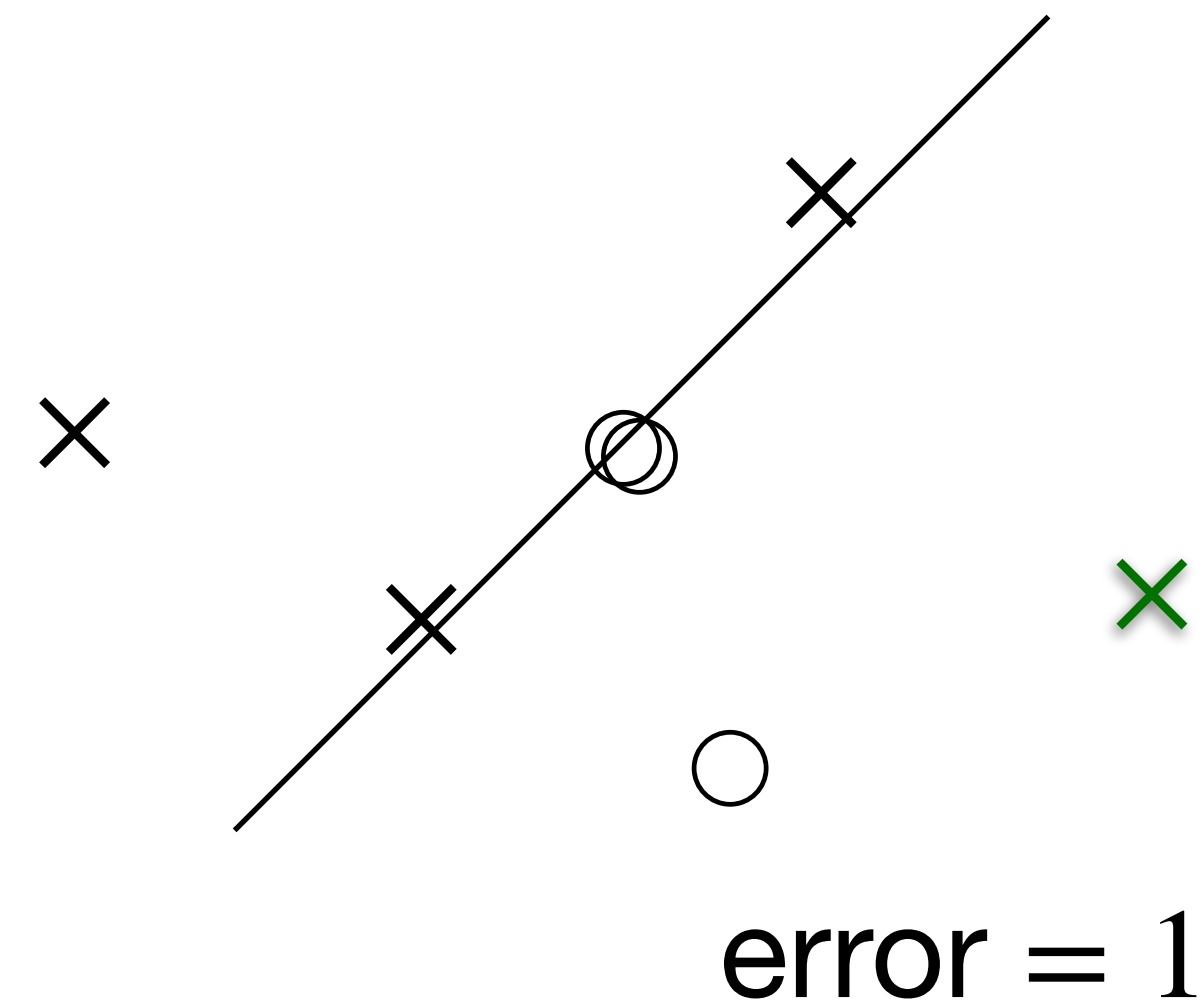
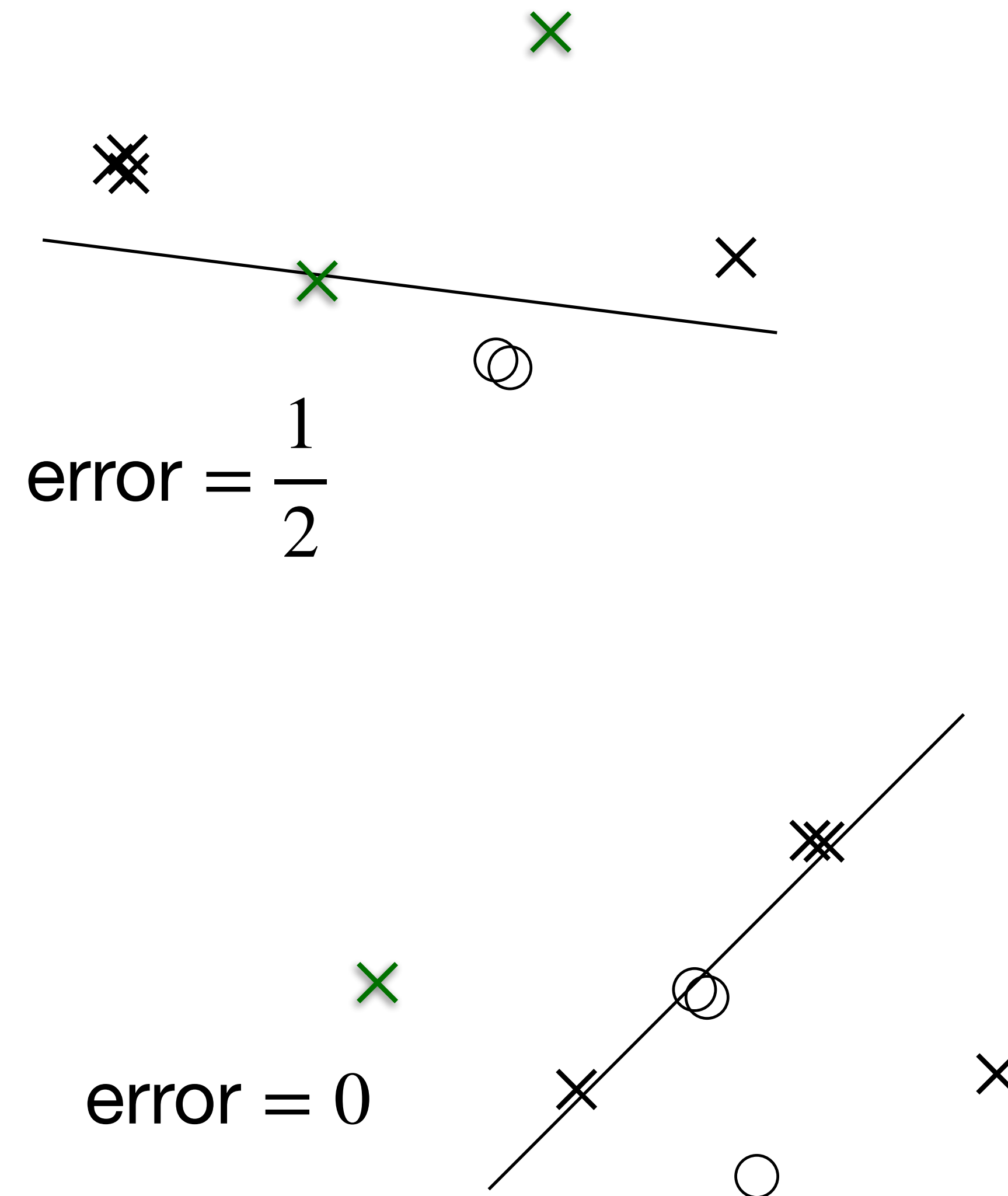
***Train on
each
bootstrap
sample***



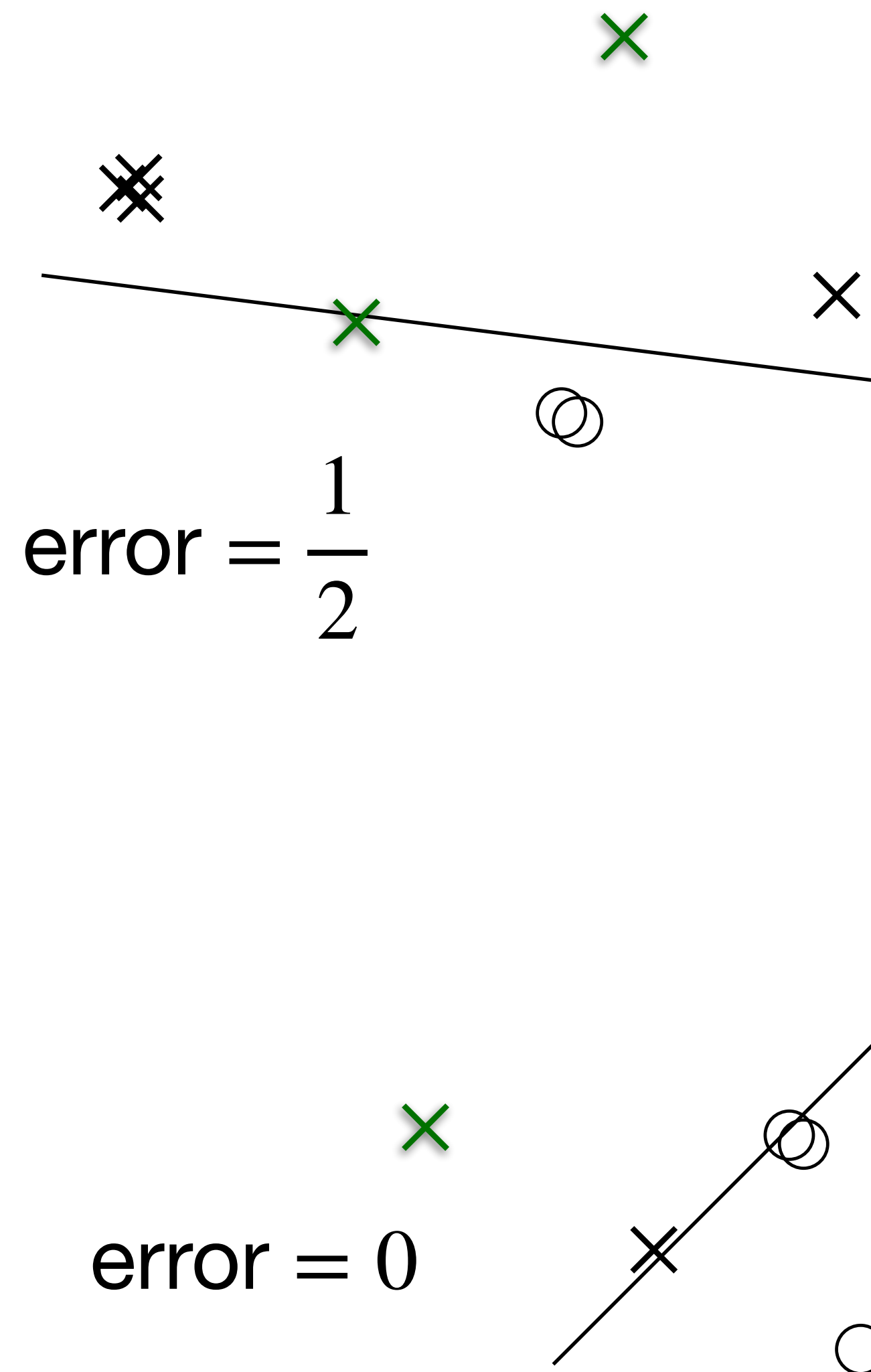
[Faint, illegible handwritten text]



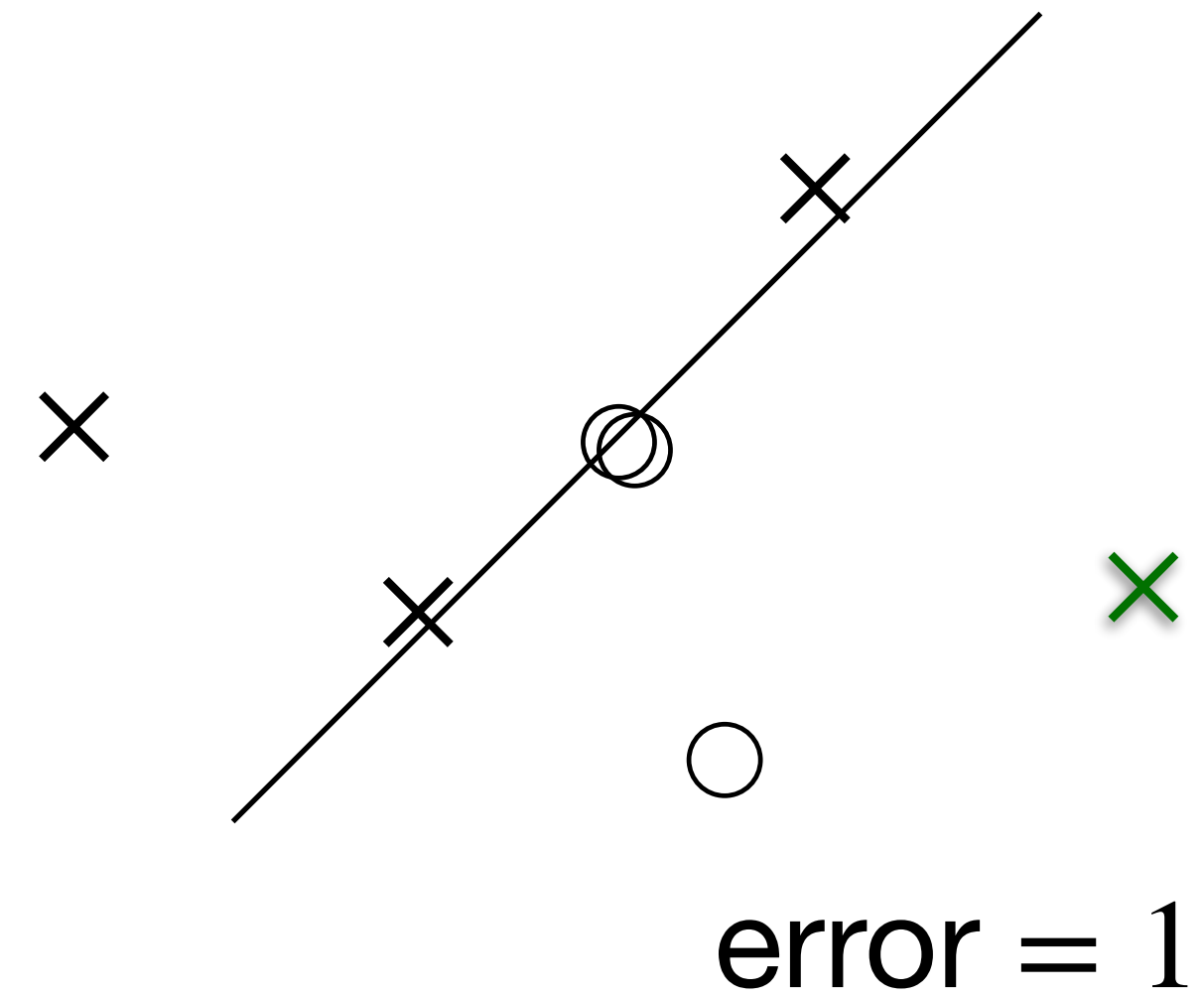
*Test on
o.o.b.*



***Test on
o.o.b.***



overall average error = $\frac{1}{2}$



Bootstrap

- Repeat T times:
 - ▶ take a bootstrap resample D_t of $\mathcal{D}$: select $(\mathbf{x}^{(i)}, y^{(i)})$ iid from $\mathcal{D}$ *with replacement* $|\mathcal{D}|$ times
 - ▶ train h_t on bag, test h_t on out-of-bag points
- Return:
 - ▶ the accuracy estimate (average of o.o.b. accuracies)
 - ▶ the model ensemble $\{h_t\}_{t=1}^T$
 - ▶ to predict for a new $\mathbf{x}$, return average $\frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x})$ — called **bagging** for **bootstrap aggregating**
 - ▶ threshold it if we want a single prediction

Hyper-parameter: how many rounds

- More bootstrap rounds T :
 - ▶ more expensive (both train and test)
 - ▶ finer quantification of uncertainty
 - ▶ approaches convergence (different training runs → similar final hypothesis) due to law of large numbers
- In fact, under some assumptions, bootstrap prediction → true posterior over y given $\mathbf{x}$ and model class
 - ▶ but these assumptions don't ever really hold for modern ML methods, so it's really more of a nice heuristic

Column subsampling

- Bootstrap resamples serve to increase diversity of set of learned classifiers — diversity helps cover posterior
- Empirical observation: bagging alone doesn't yield enough diversity for best performance
- Idea: change hypothesis class slightly for each bag
 - ▶ we probably aren't 100% certain we had the exact right hypothesis class anyway
- ***Column subsampling***: use only a subset of features for each classifier

***Column
subsampling***

Task: will Geoff like this movie?

Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------

***Column
subsampling***

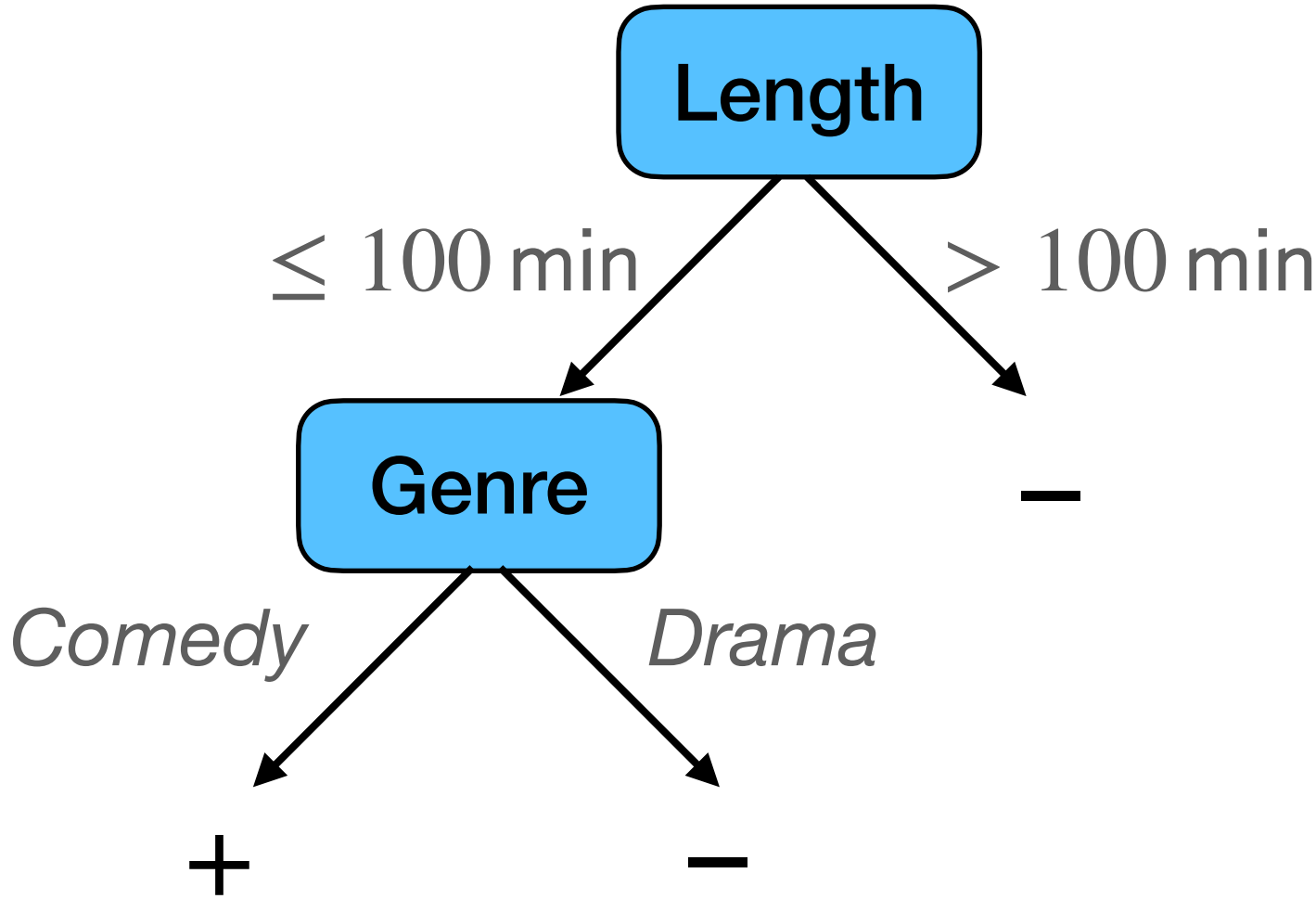
Task: will Geoff like this movie?

Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------

*Column
subsampling*

Task: will Geoff like this movie?

Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------



***Column
subsampling***

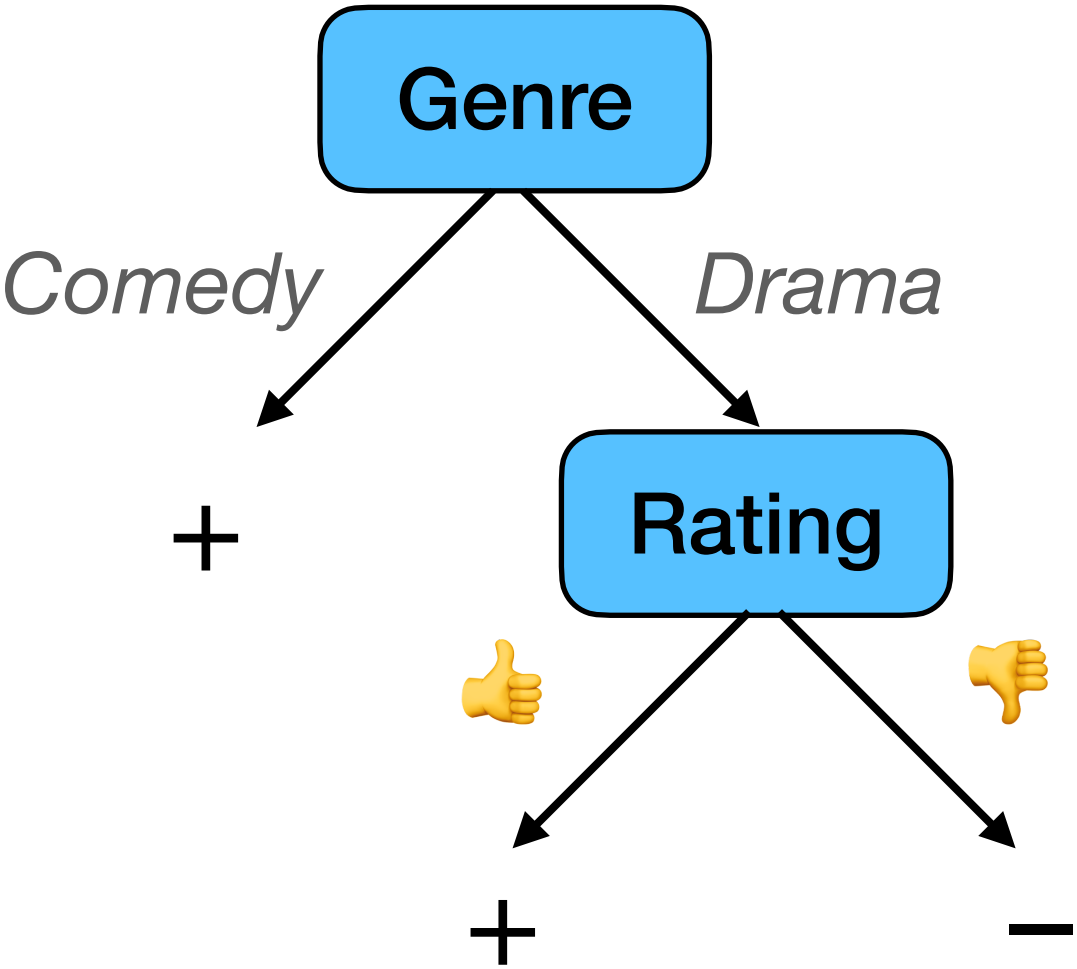
Task: will Geoff like this movie?

Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------

*Column
subsampling*

Task: will Geoff like this movie?

Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------



***Column
subsampling***

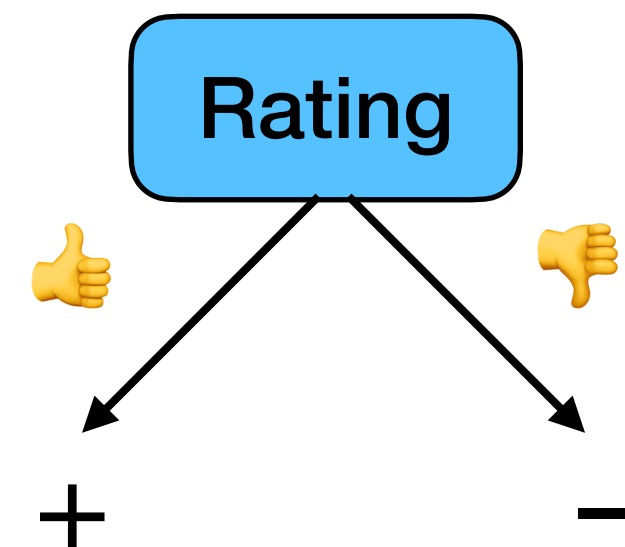
Task: will Geoff like this movie?

Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------

Column subsampling

Task: will Geoff like this movie?

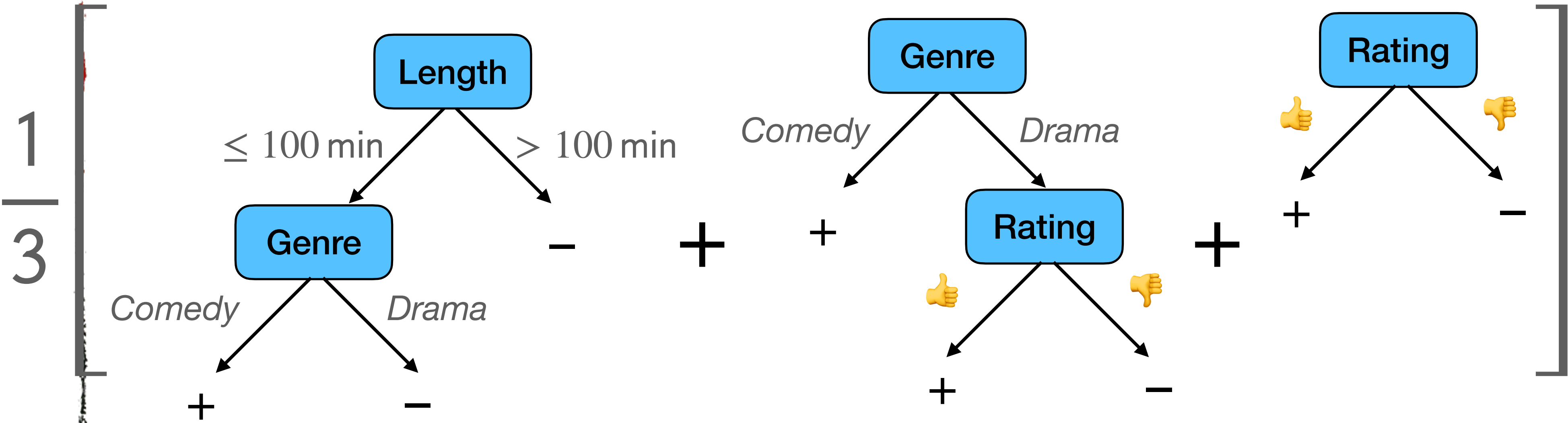
Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------



Task: will Geoff like this movie?

Length	Genre	Budget	Year	Rating
--------	-------	--------	------	--------

Column
subsampling



Variants

- If base classifier = decision trees, we can resample columns...
 - ▶ once at the beginning of training for each bag (= ordinary column subsampling)
 - ▶ once for every level of the tree (by-level subsampling)
 - ▶ once for each split (by-node subsampling)
 - ▶ last one is also called ***split-feature randomization***
- More randomization → more diversity (but maybe more noise)

Random forests

Table 2. Normalized scores for each learning algorithm by metric (average over eleven problems)

MODEL	CAL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN	OPT-SEL
BST-DT	PLT	.843*	.779	.939	.963	.938	.929*	.880	.896	.896	.917
RF	PLT	.872*	.805	.934*	.957	.931	.930	.851	.858	.892	.898
BAG-DT	—	.846	.781	.938*	.962*	.937*	.918	.845	.872	.887*	.899
BST-DT	ISO	.826*	.860*	.929*	.952	.921	.925*	.854	.815	.885	.917*
RF	—	.872	.790	.934*	.957	.931	.930	.829	.830	.884	.890
BAG-DT	PLT	.841	.774	.938*	.962*	.937*	.918	.836	.852	.882	.895
RF	ISO	.861*	.861	.923	.946	.910	.925	.836	.776	.880	.895
BAG-DT	ISO	.826	.843*	.933*	.954	.921	.915	.832	.791	.877	.894
SVM	PLT	.824	.760	.895	.938	.898	.913	.831	.836	.862	.880
ANN	—	.803	.762	.910	.936	.892	.899	.811	.821	.854	.885
SVM	ISO	.813	.836*	.892	.925	.882	.911	.814	.744	.852	.882
ANN	PLT	.815	.748	.910	.936	.892	.899	.783	.785	.846	.875
ANN	ISO	.803	.836	.908	.924	.876	.891	.777	.718	.842	.884

source: (Caruana & Niculescu-Mizil, 2006)

- A popular combination (because it won a bunch of ML competitions ca. 2000s):
 - ▶ bagging
 - ▶ + column subsampling or variants
 - ▶ w/ small decision trees as the base classifier — no pruning!
- Lots of random trees = a *random forest*
- Software: XGBoost [Chen & Guestrin, 2016]

Random forests

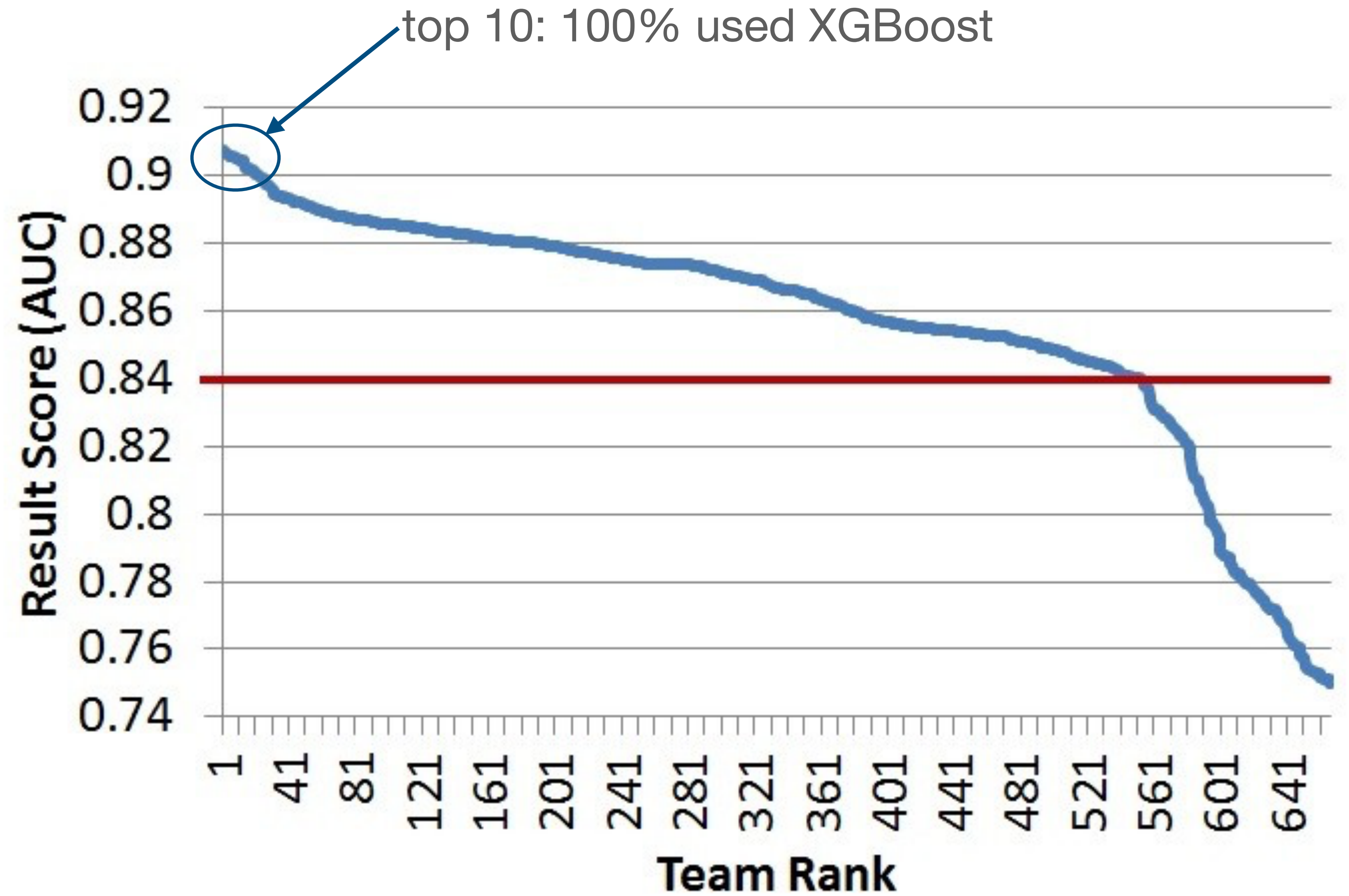
Table 2. Normalized scores for each learning algorithm by metric (average over eleven problems)

MODEL	CAL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN	OPT-SEL
BST-DT	PLT	.843*	.779	.939	.963	.938	.929*	.880	.896	.896	.917
RF	PLT	.872*	.805	.934*	.957	.931	.930	.851	.858	.892	.898
BAG-DT	—	.846	.781	.938*	.962*	.937*	.918	.845	.872	.887*	.899
BST-DT	ISO	.826*	.860*	.929*	.952	.921	.925*	.854	.815	.885	.917*
RF	—	.872	.790	.934*	.957	.931	.930	.829	.830	.884	.890
BAG-DT	PLT	.841	.774	.938*	.962*	.937*	.918	.836	.852	.882	.895
RF	ISO	.861*	.861	.923	.946	.910	.925	.836	.776	.880	.895
BAG-DT	ISO	.826	.843*	.933*	.954	.921	.915	.832	.791	.877	.894
SVM	PLT	.824	.760	.895	.938	.898	.913	.831	.836	.862	.880
ANN	—	.803	.762	.910	.936	.892	.899	.811	.821	.854	.885
SVM	ISO	.813	.836*	.892	.925	.882	.911	.814	.744	.852	.882
ANN	PLT	.815	.748	.910	.936	.892	.899	.783	.785	.846	.875
ANN	ISO	.803	.836	.908	.924	.876	.891	.777	.718	.842	.884

source: (Caruana & Niculescu-Mizil, 2006)

- A popular combination (because it won a bunch of ML competitions ca. 2000s):
 - ▶ bagging
 - ▶ + column subsampling or variants
 - ▶ w/ small decision trees as the base classifier — no pruning!
- Lots of random trees = a *random forest*
- Software: XGBoost [Chen & Guestrin, 2016]

KDD Cup 2015



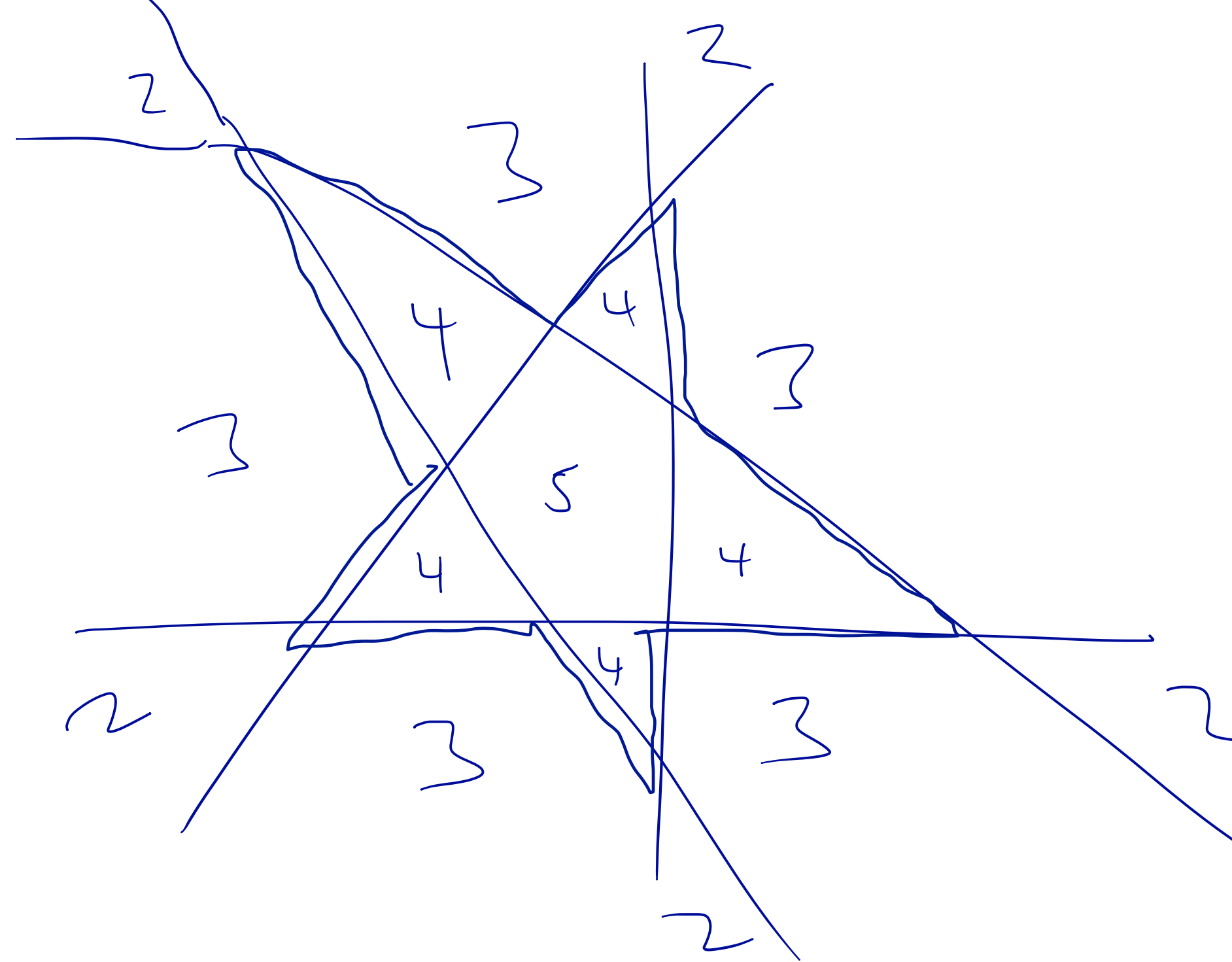
plot credit: Ron Bekkerman

Geoff Gordon

Bagging for multi-class and regression

- Bagging works well for multi-class or regression too
- For regression:
 - ▶ prediction is the average of individual hypotheses
 - ▶ if desired we can express uncertainty by reporting quantiles of the list of predictions
 - ▶ e.g., “we predict $y = 4.01$ given $\mathbf{x}$; 90% of predictions for $y \mid \mathbf{x}$ are in the range $[3.71, 4.12]$ ”

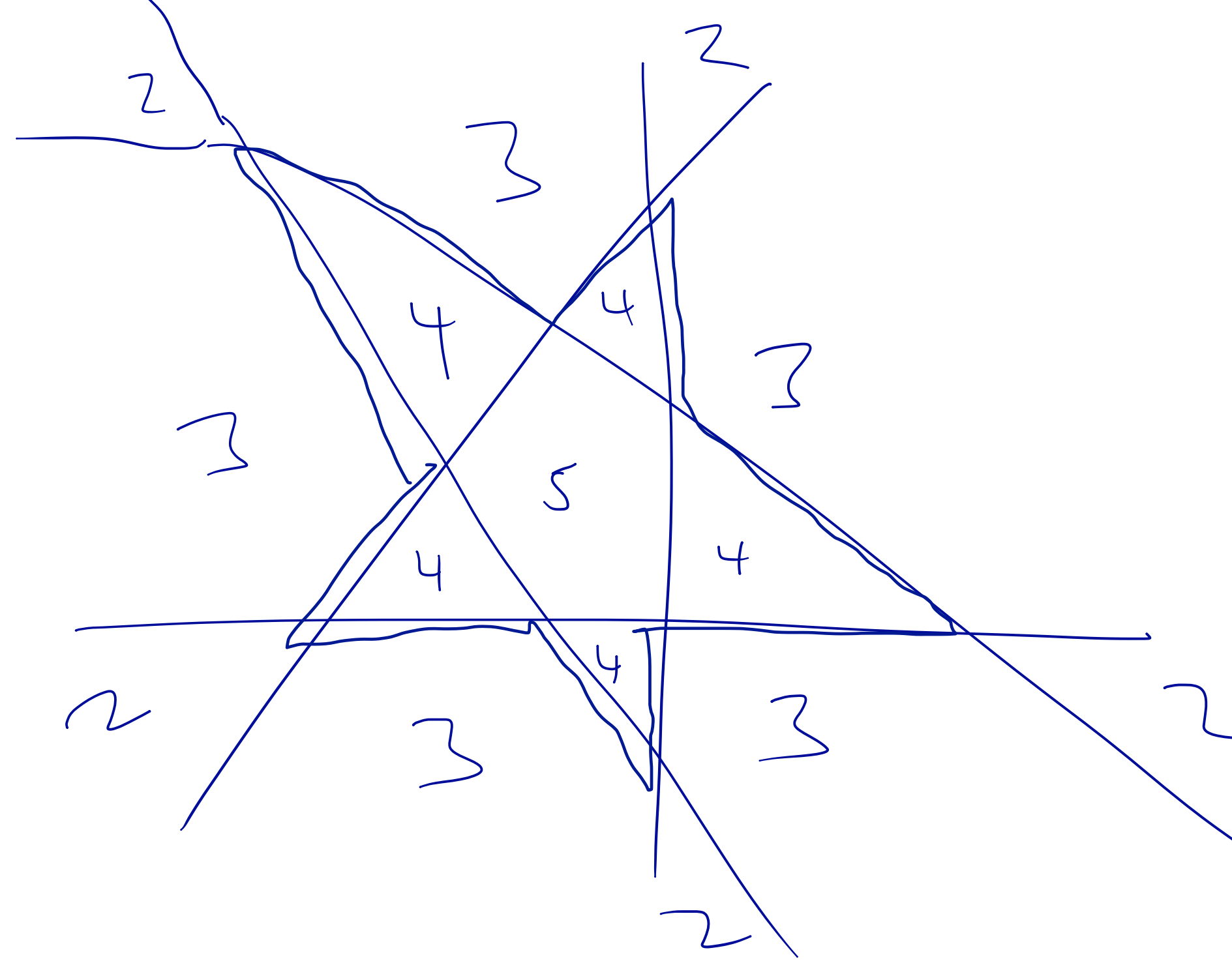
Voted classifiers



ex: five voted
halfspaces

- Bagging for binary classification is a *voted classifier*:
 - ▶ each sub-classifier $h_t : \mathbb{R}^d \rightarrow [-1,1]$ or $\mathbb{R}^d \rightarrow \{-1,1\}$
 - ▶ vote: is $\sum_t h_t(\mathbf{x}) \geq 0$?
 - ▶ or weighted vote: is $\sum_t \alpha_t h_t(\mathbf{x}) \geq 0$? (weights α_t)
 - ▶ wlog $\alpha_t > 0$ for all t (hypothesis set closed under negation)
 - ▶ normalize if desired: $\sum_t \alpha_t = 1$ (constant factor is irrelevant)

Second benefit: better accuracy



ex: five voted
halfspaces

- Note: voted classifier is *strictly more expressive* than original classifier (★ region is not a halfspace)
- Can we use higher expressiveness to get higher accuracy?

Boosting

- Boosting is a wrapper we can put around any binary classifier (perceptron, decision tree, ...)
 - ▶ base classifier solves many (related) learning problems
 - ▶ each one gives us a classifier h_t
 - ▶ final classifier is a weighted vote: $\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \stackrel{?}{\geq} 0$
- Idea: if base algorithm (the *weak learner*) can keep doing just better than chance (error $\frac{1}{2} - \epsilon$)
 - ▶ then final boosted classifier (the *strong learner*) can get zero error on the training set
 - ▶ and (w/ assumptions) do well on the test set too

$$|\mathcal{H}| = 3$$

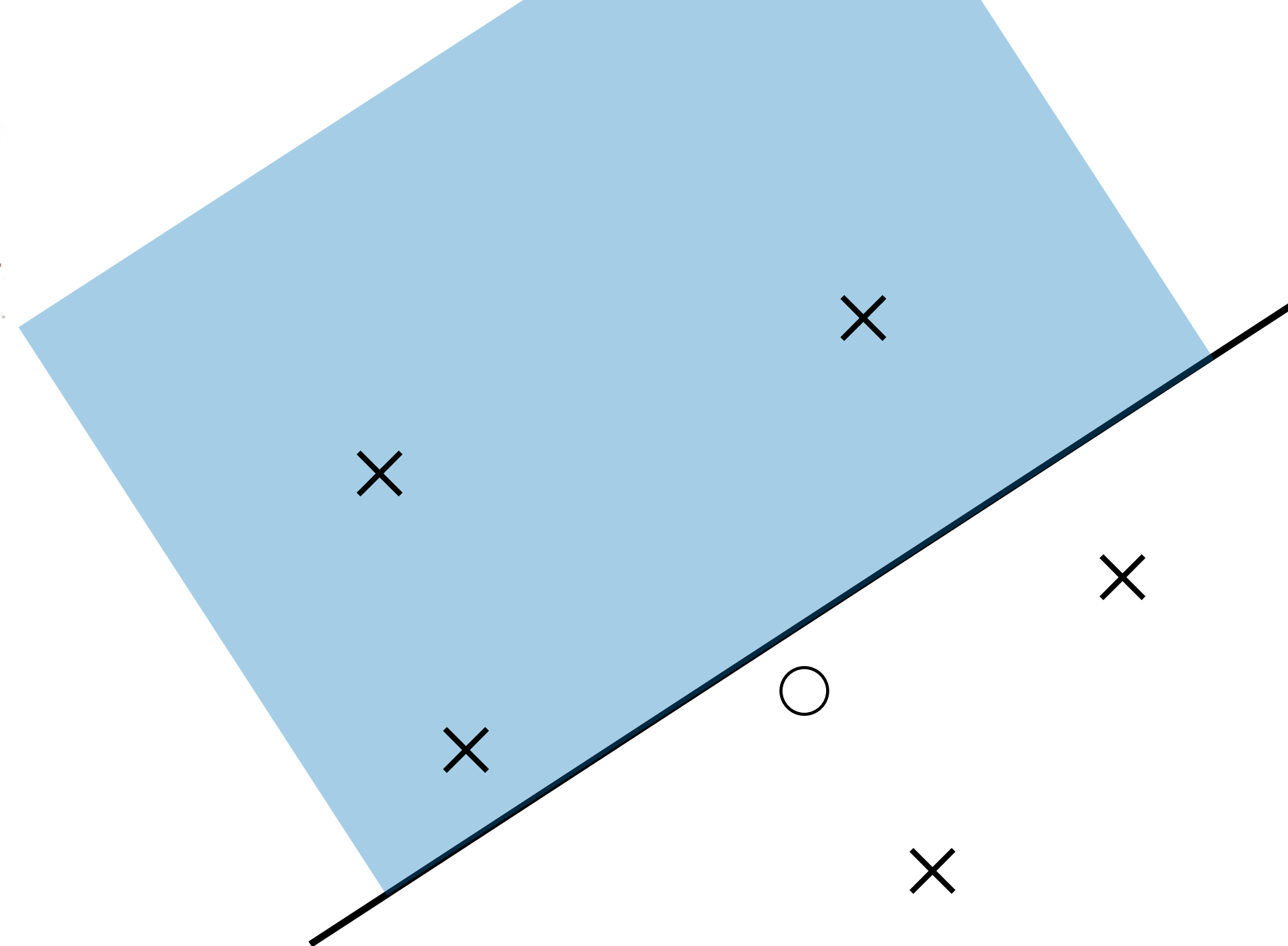
Warmup: finite hypothesis class

- Suppose we have a finite set $\mathcal{H} = \{h_1, h_t, \dots, h_T\}$ of classifiers, and want to learn a weighted vote

$$\sum_{t=1}^{|\mathcal{H}|} \alpha_t h_t(\mathbf{x})$$

$$|\mathcal{H}| = 3$$

Warmup: finite hypothesis class

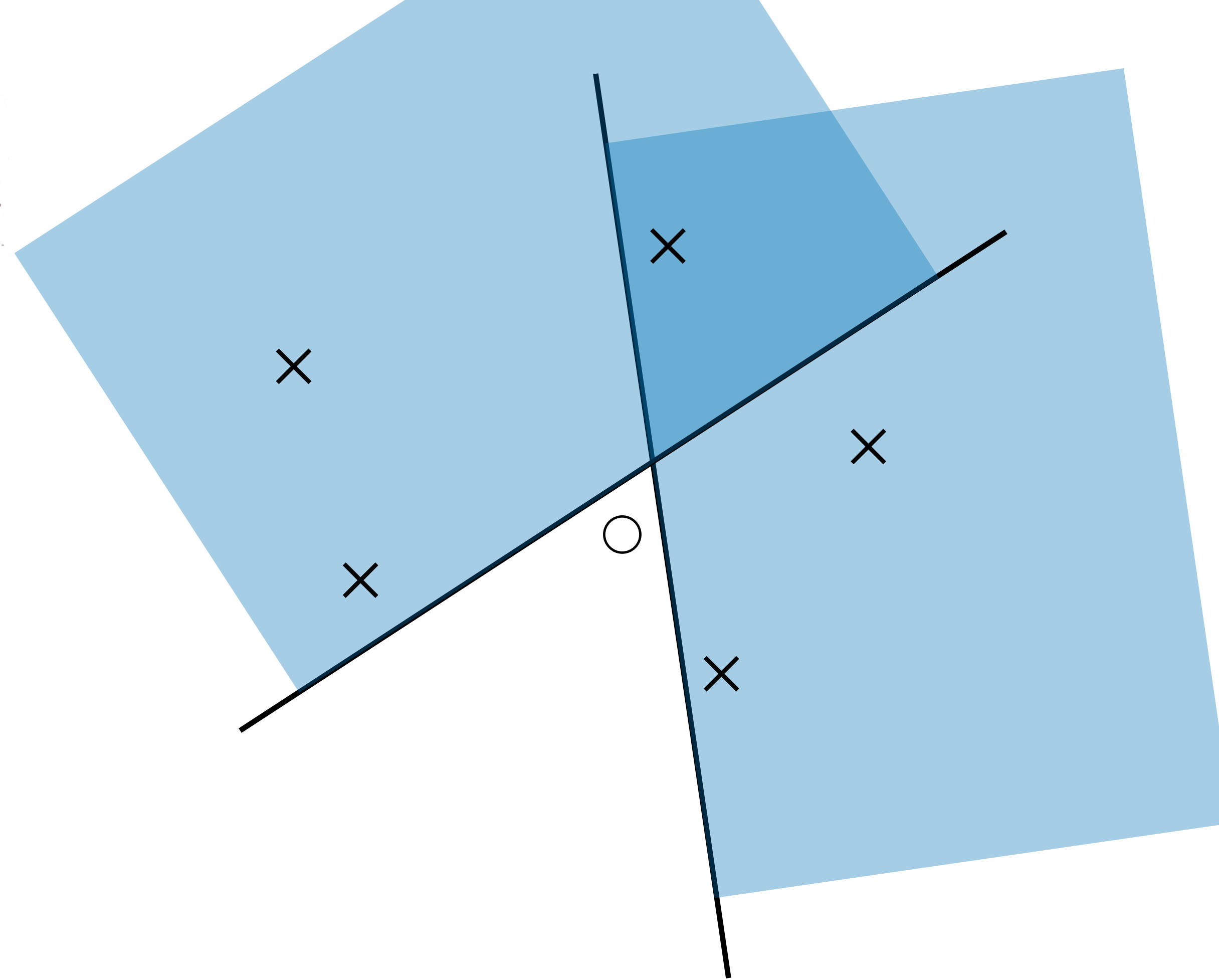


- Suppose we have a finite set $\mathcal{H} = \{h_1, h_t, \dots, h_T\}$ of classifiers, and want to learn a weighted vote

$$\sum_{t=1}^{|\mathcal{H}|} \alpha_t h_t(\mathbf{x})$$

$$|\mathcal{H}| = 3$$

Warmup: finite hypothesis class

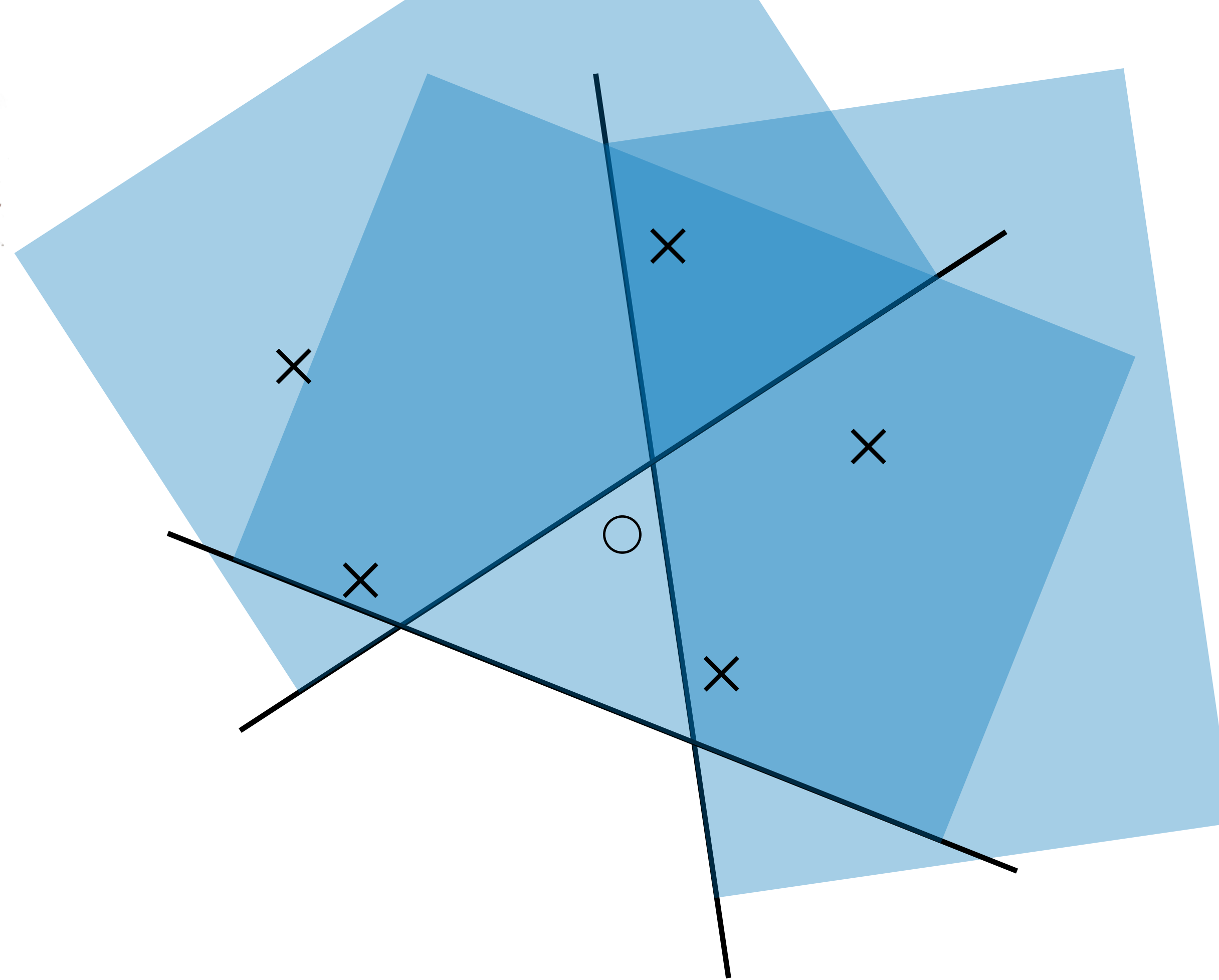


- Suppose we have a finite set $\mathcal{H} = \{h_1, h_t, \dots, h_T\}$ of classifiers, and want to learn a weighted vote

$$\sum_{t=1}^{|\mathcal{H}|} \alpha_t h_t(\mathbf{x})$$

$$|\mathcal{H}| = 3$$

Warmup: finite hypothesis class

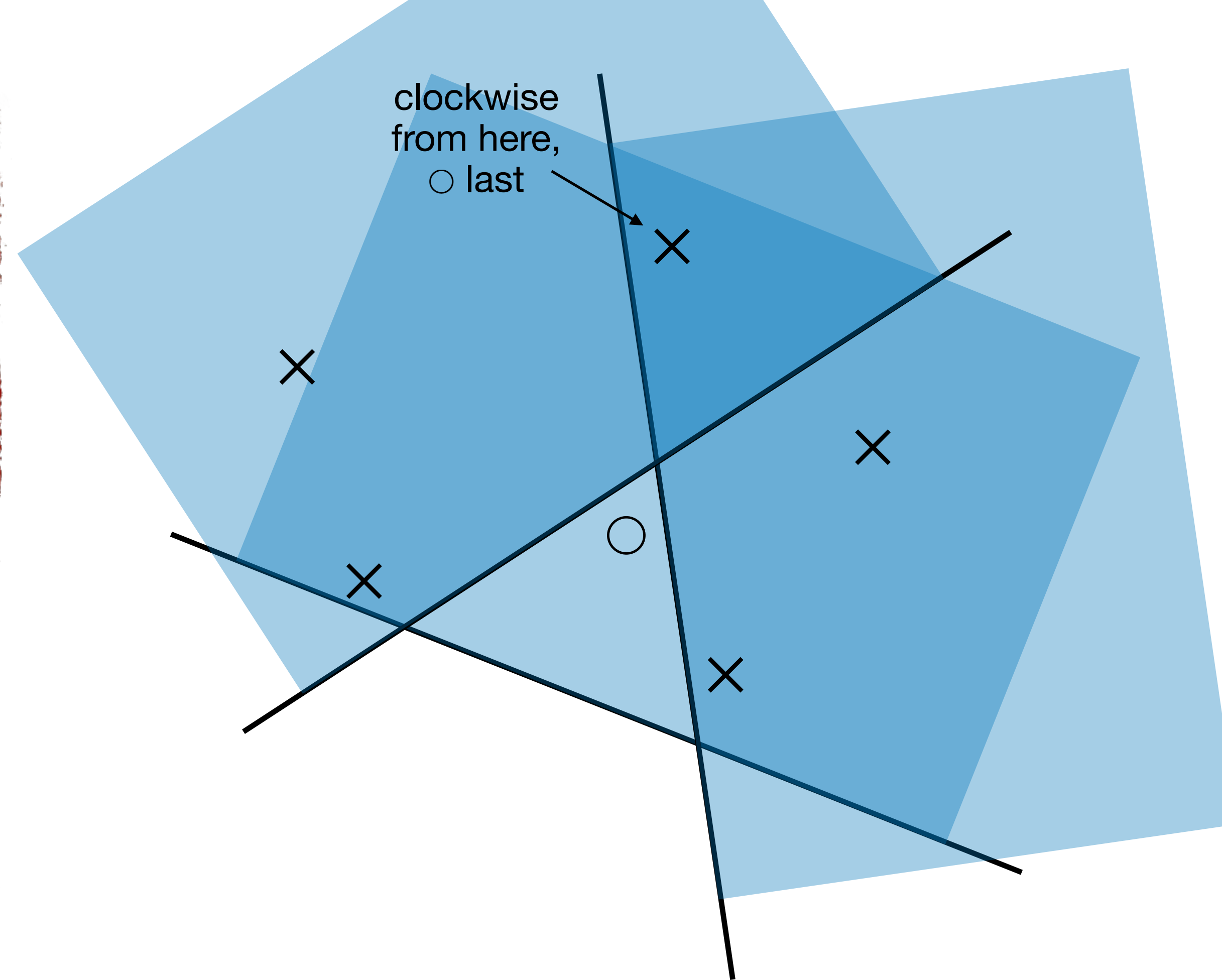


- Suppose we have a finite set $\mathcal{H} = \{h_1, h_t, \dots, h_T\}$ of classifiers, and want to learn a weighted vote

$$\sum_{t=1}^{|\mathcal{H}|} \alpha_t h_t(\mathbf{x})$$

$$|\mathcal{H}| = 3$$

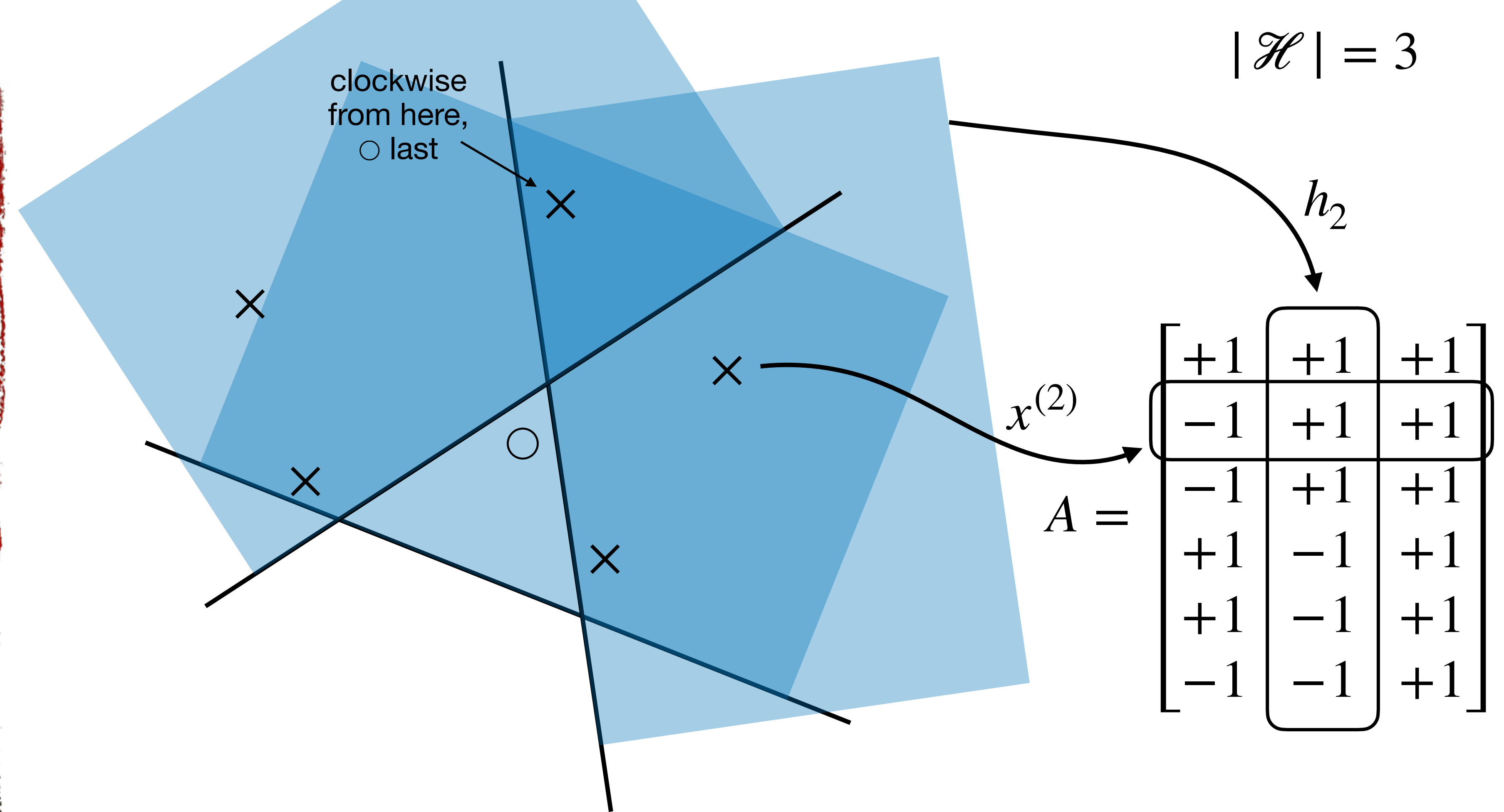
Classifiers as features



$$A = \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & +1 \\ -1 & +1 & +1 \\ +1 & -1 & +1 \\ +1 & -1 & +1 \\ -1 & -1 & +1 \end{bmatrix}$$

- Can treat each classifier as a *feature*: value $+1$ for points classified positive, value -1 for points classified negative, making a data matrix A with $A_{it} = h_t(\mathbf{x}^{(i)})$

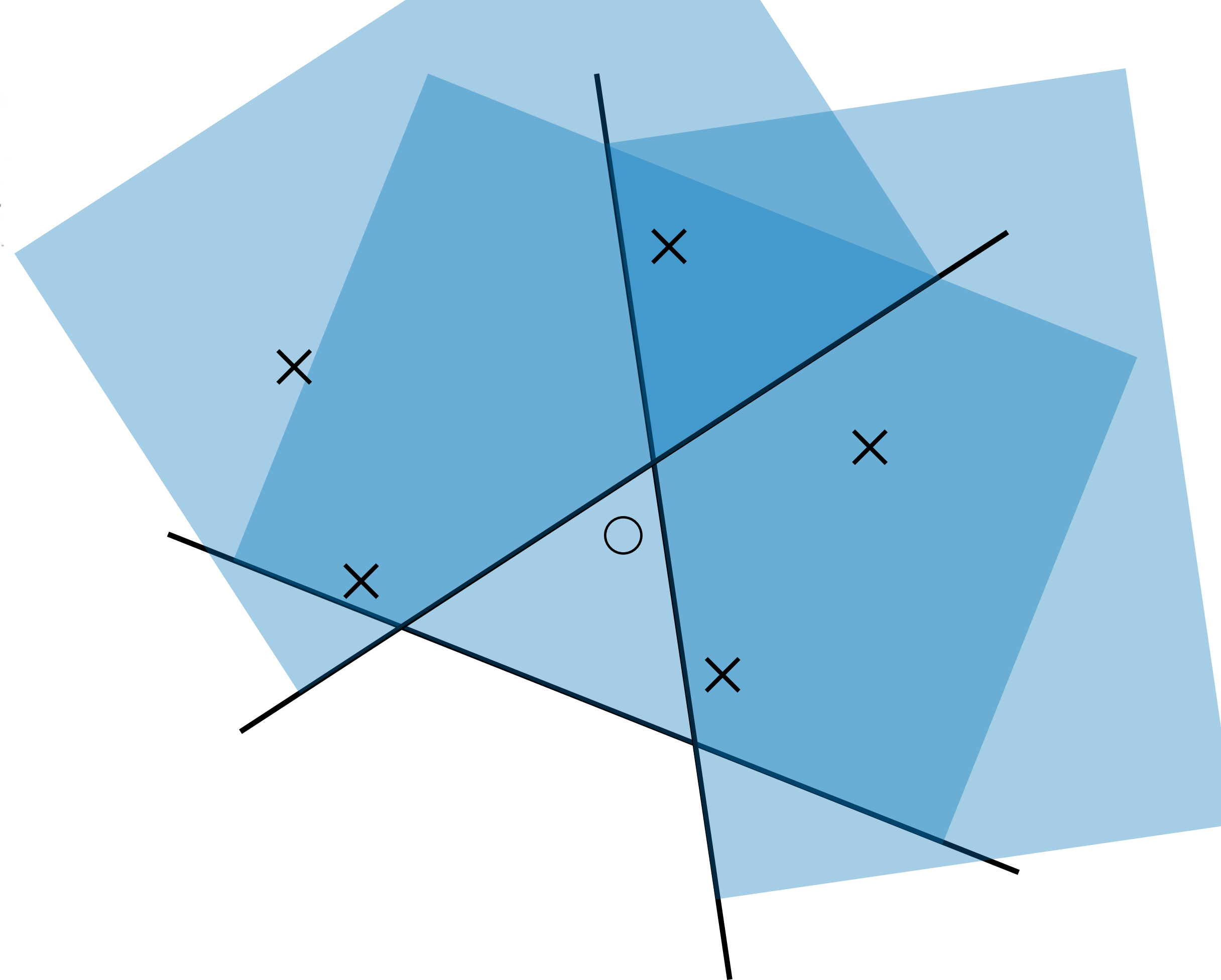
Classifiers as features



- Each column of A corresponds to a single classifier h_t (evaluated on all training examples $\mathbf{x}^{(i)}$)
- Each row of A corresponds to a single training example $\mathbf{x}^{(i)}$ (classified by all hypotheses h_t): $\mathbf{a}^{(i)} = (h_t(\mathbf{x}^{(i)}))_{t=1}^{|\mathcal{H}|}$

$$|\mathcal{H}| = 3$$

Classifiers as features



$$A = \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & +1 \\ -1 & +1 & +1 \\ +1 & -1 & +1 \\ +1 & -1 & +1 \\ -1 & -1 & +1 \end{bmatrix}$$

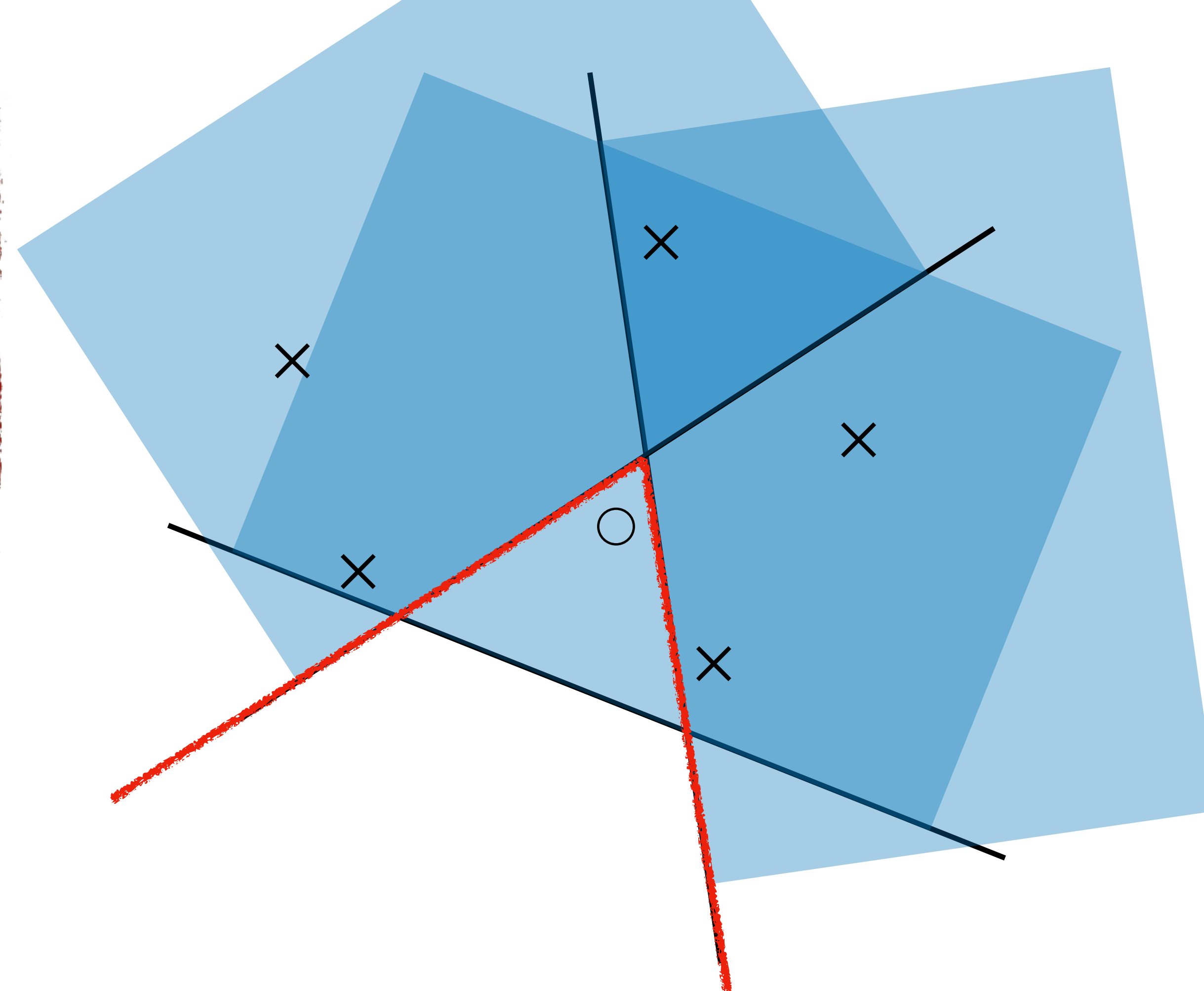
- Vote $\sum_{t=1}^{|\mathcal{H}|} \alpha_t h_t(\mathbf{x})$ becomes a linear classifier $\boldsymbol{\alpha} \cdot \mathbf{a}$ on top of the features $\mathbf{a}$ (so, **nonlinear** in $\mathbf{x}$)

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$A\boldsymbol{\alpha} = \begin{bmatrix} 3 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

$$|\mathcal{H}| = 3$$

Classifiers as features



$$A = \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & +1 \\ -1 & +1 & +1 \\ +1 & -1 & +1 \\ +1 & -1 & +1 \\ -1 & -1 & +1 \end{bmatrix}$$

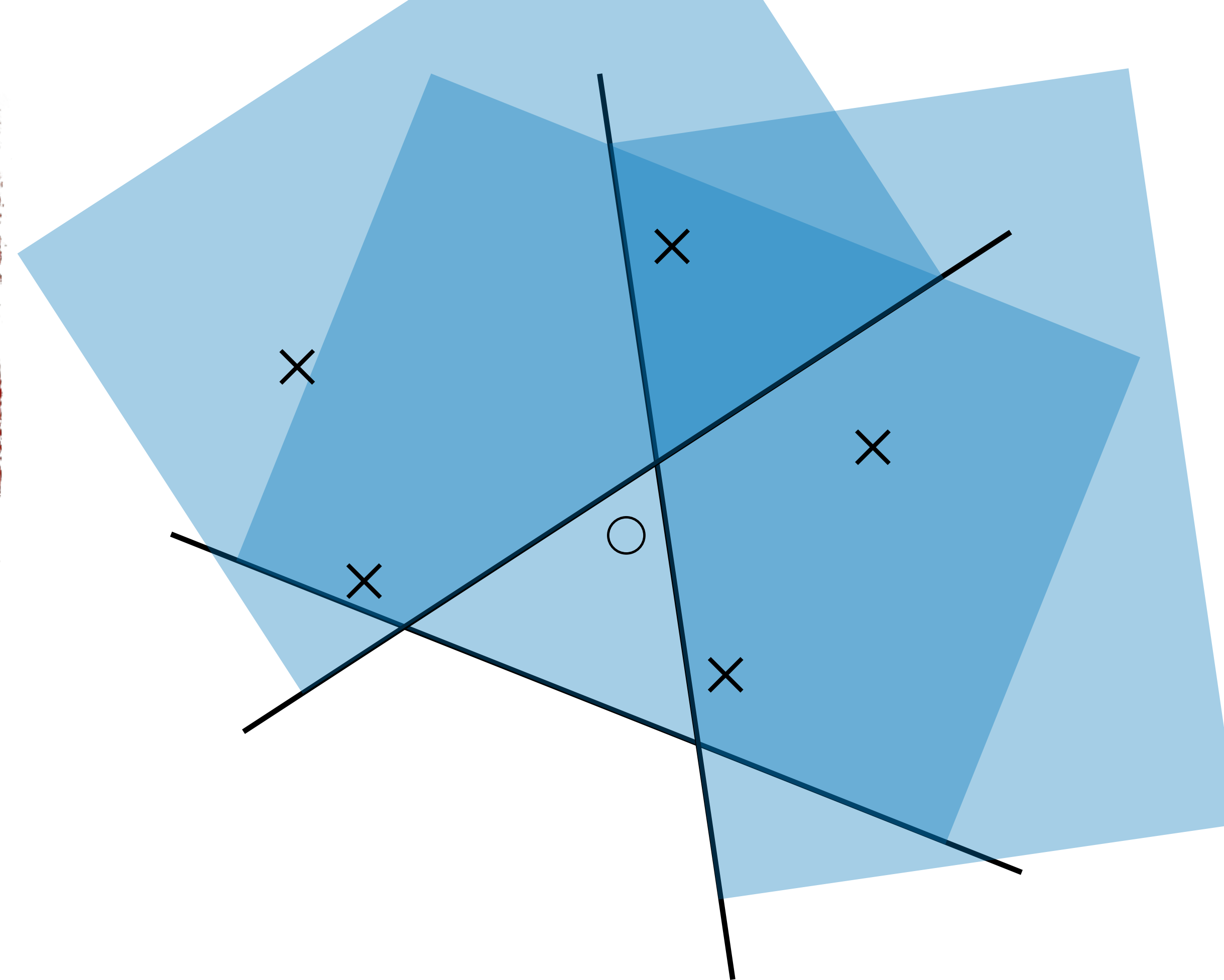
- Vote $\sum_{t=1}^{|\mathcal{H}|} \alpha_t h_t(\mathbf{x})$ becomes a linear classifier $\alpha \cdot \mathbf{a}$ on top of the features $\mathbf{a}$ (so, **nonlinear** in $\mathbf{x}$)

$$\alpha = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$A\alpha = \begin{bmatrix} 3 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

$$|\mathcal{H}| = 3$$

*What's the
best vote
weight α ?*

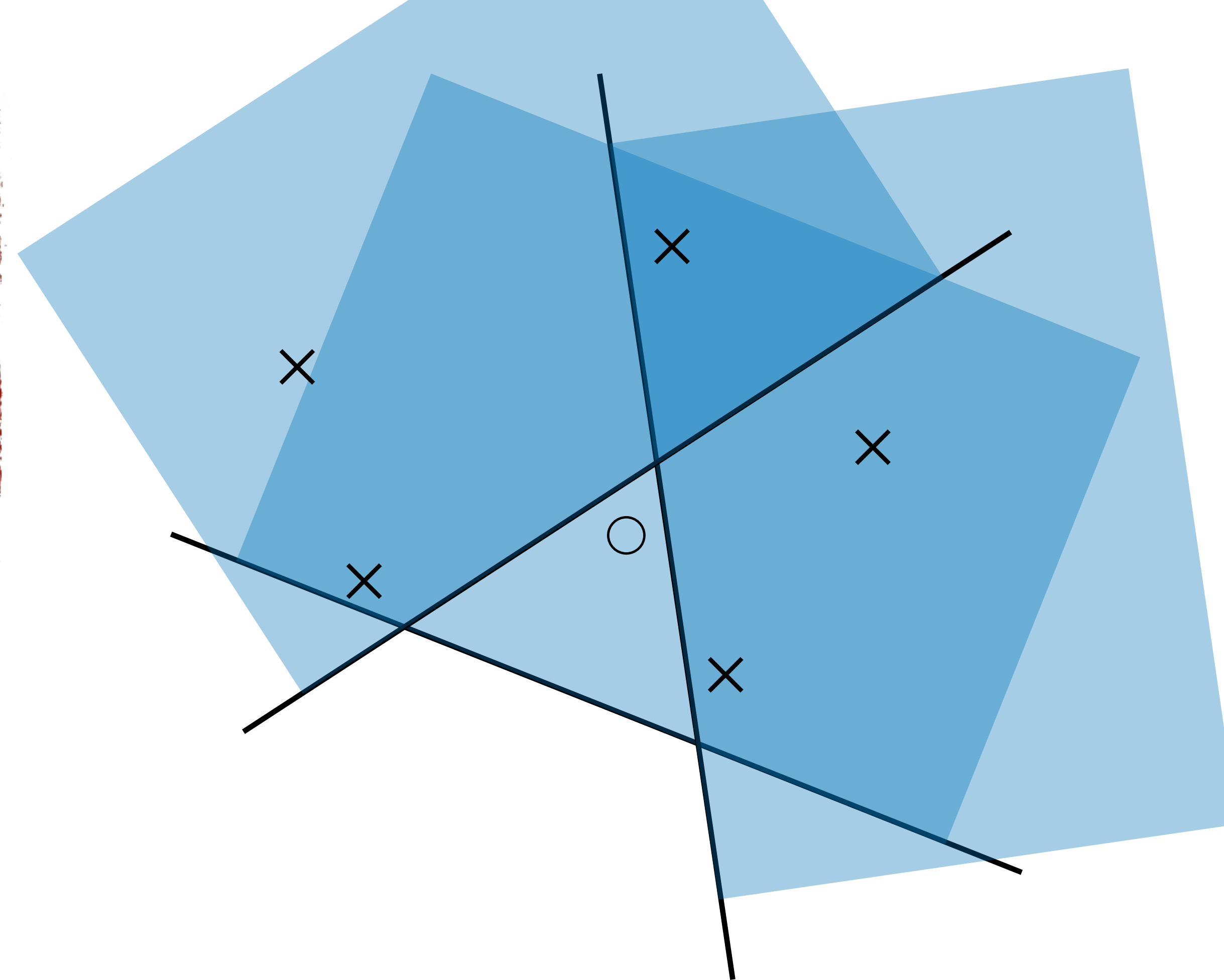


$$A = \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & +1 \\ -1 & +1 & +1 \\ +1 & -1 & +1 \\ +1 & -1 & +1 \\ -1 & -1 & +1 \end{bmatrix}$$

- First try: perceptron algorithm
 - ▶ each time we're wrong on an example $\mathbf{x}^{(i)}$ (equivalently $\mathbf{a}^{(i)}$), add 1 to weights of classifiers that vote for the correct answer $y^{(i)}$, subtract 1 for those that don't

$$|\mathcal{H}| = 3$$

What's the best vote weight α ?



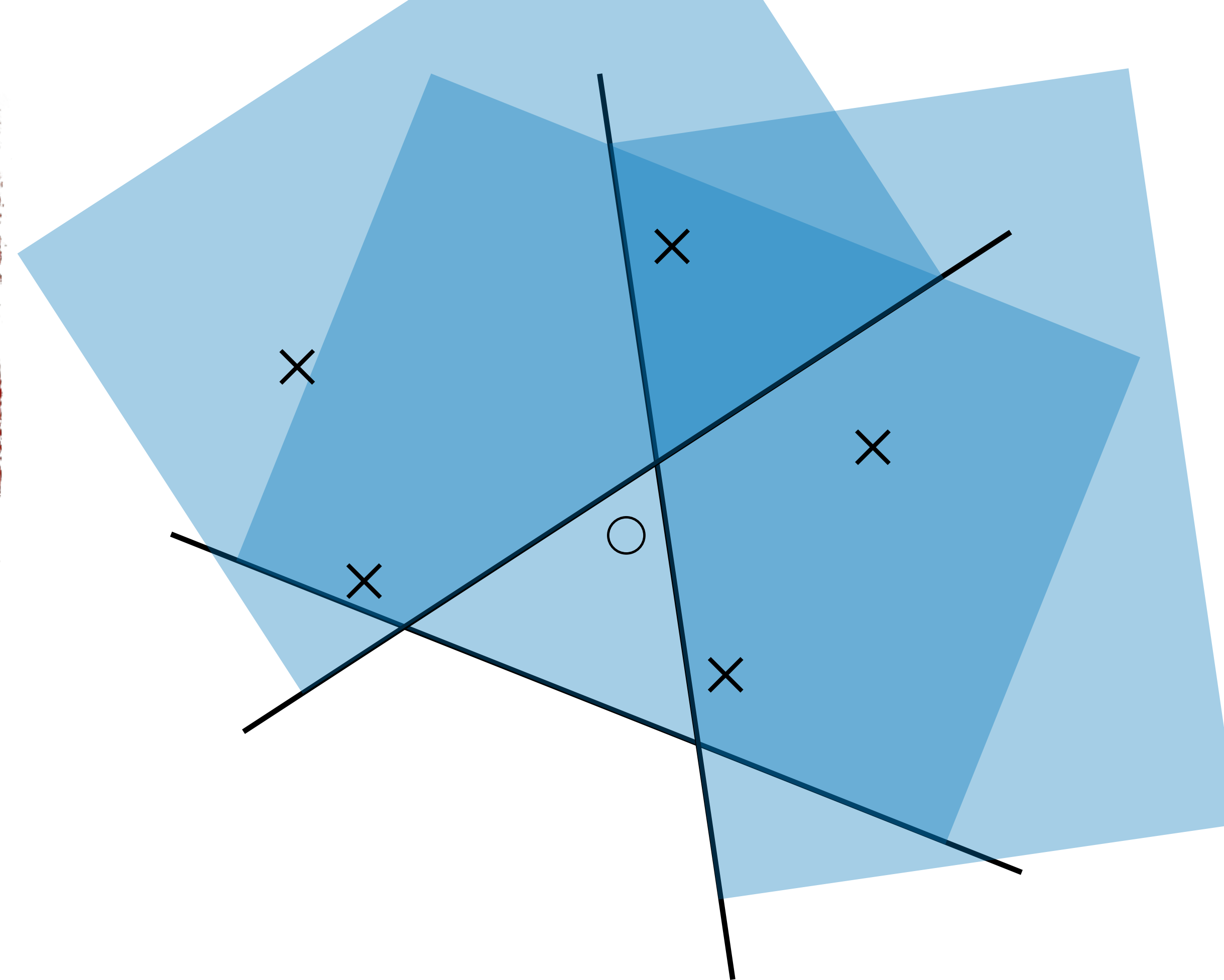
$$A = \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & +1 \\ -1 & +1 & +1 \\ +1 & -1 & +1 \\ +1 & -1 & +1 \\ -1 & -1 & +1 \end{bmatrix}$$

- Pro: would try to get positive margin $y^{(i)} \mathbf{a}^{(i)} \cdot \boldsymbol{\alpha}$ for all i
 - ▶ recall: positive margin $\rightarrow$ we get example i correct
- Cons:

margin is
in $\mathbb{R}^{|\mathcal{H}|}$
not $\mathbb{R}^2$

$$|\mathcal{H}| = 3$$

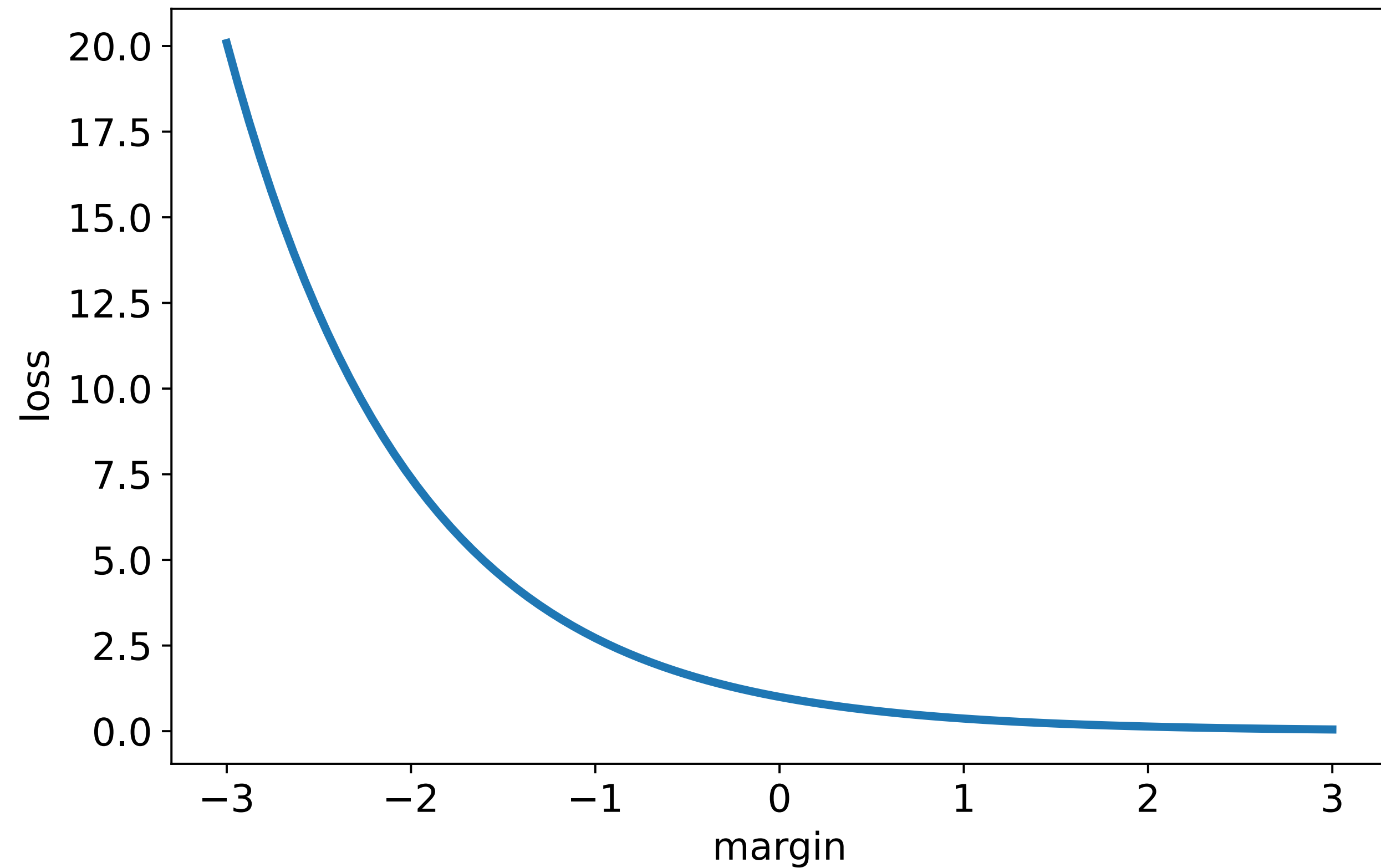
*What's the
best vote
weight α ?*



$$A = \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & +1 \\ -1 & +1 & +1 \\ +1 & -1 & +1 \\ +1 & -1 & +1 \\ -1 & -1 & +1 \end{bmatrix}$$

- Second try: optimize a loss that favors positive margin
 $y^{(i)} \mathbf{a}^{(i)} \cdot \boldsymbol{\alpha} > 0$ for all i

***(Approximately)
maximize
smallest margin***



- Ideally: maximize $\min_{i \in \{1, 2, \dots, 6\}} y^{(i)} \mathbf{a}^{(i)} \cdot \frac{\boldsymbol{\alpha}}{\|\boldsymbol{\alpha}\|}$
 - ▶ but this isn't convex
- Instead: minimize $L(\boldsymbol{\alpha}) = \sum_{i=1}^6 \exp(-y^{(i)} \mathbf{a}^{(i)} \cdot \boldsymbol{\alpha})$
 - ▶ most-negative margin dominates the sum

Coordinate descent

- Which optimizer should we use?
 - ▶ could use SGD, but same problems as perceptron
- Instead, use a simpler method that will scale better:
coordinate descent
- While not converged
 - ▶ pick a coordinate (some $h_t \in \mathcal{H}$)
 - ▶ adjust its weight α_t to reduce loss

} how?

Weight update

$$L(\alpha) = \sum_{i=1}^N \exp(-y^{(i)} \mathbf{a}^{(i)} \cdot \boldsymbol{\alpha})$$

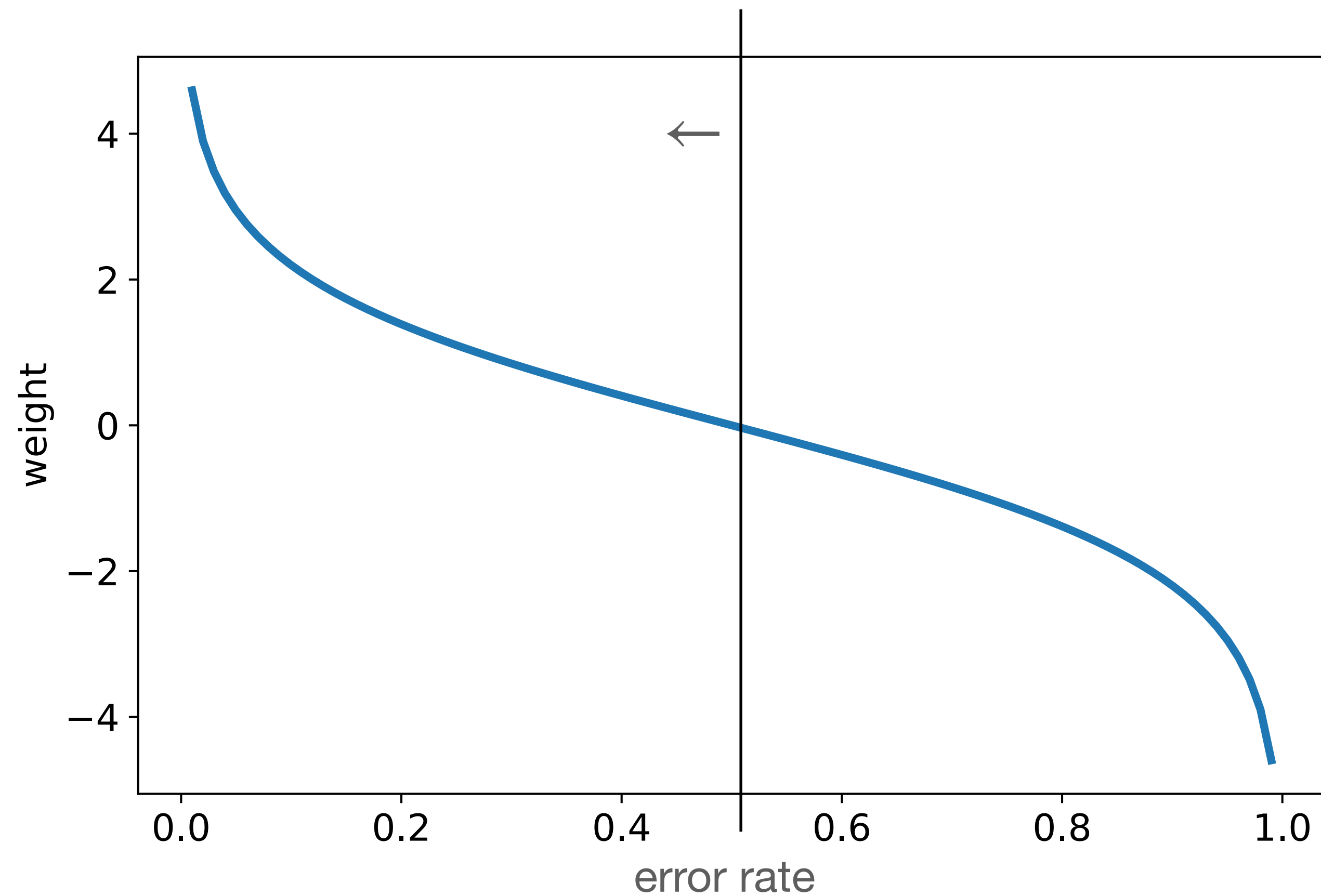
$$\frac{d}{d\alpha_t} L =$$

- normalize, split by $y_i A_{it}$, define weighted error rate ϵ
- consider updating $\alpha_t \rightarrow \alpha_t + \Delta\alpha$

***Weight
update***

$$0 = (1 - \epsilon)e^{-\Delta\alpha} + \epsilon e^{\Delta\alpha}$$

Effect of update



$\Delta\alpha = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$: this is positive, increases as $\epsilon \rightarrow 0$

$$\omega_i \leftarrow \omega_i \cdot \begin{cases} e^{-\Delta\alpha/Z} & \text{if } y^{(i)} = h_t(\mathbf{x}^{(i)}) \\ e^{\Delta\alpha/Z} & \text{o/w} \end{cases} \rightarrow \text{upweight mistakes}$$

Which coordinate?

- Which coordinate h_t should we pick to update?
- Better weighted error $\rightarrow$ bigger update, bigger decrease in L
 - ▶ best $h_t =$ lowest error
 - ▶ OK to pick any h_t with error $\leq \frac{1}{2} - \text{const}$

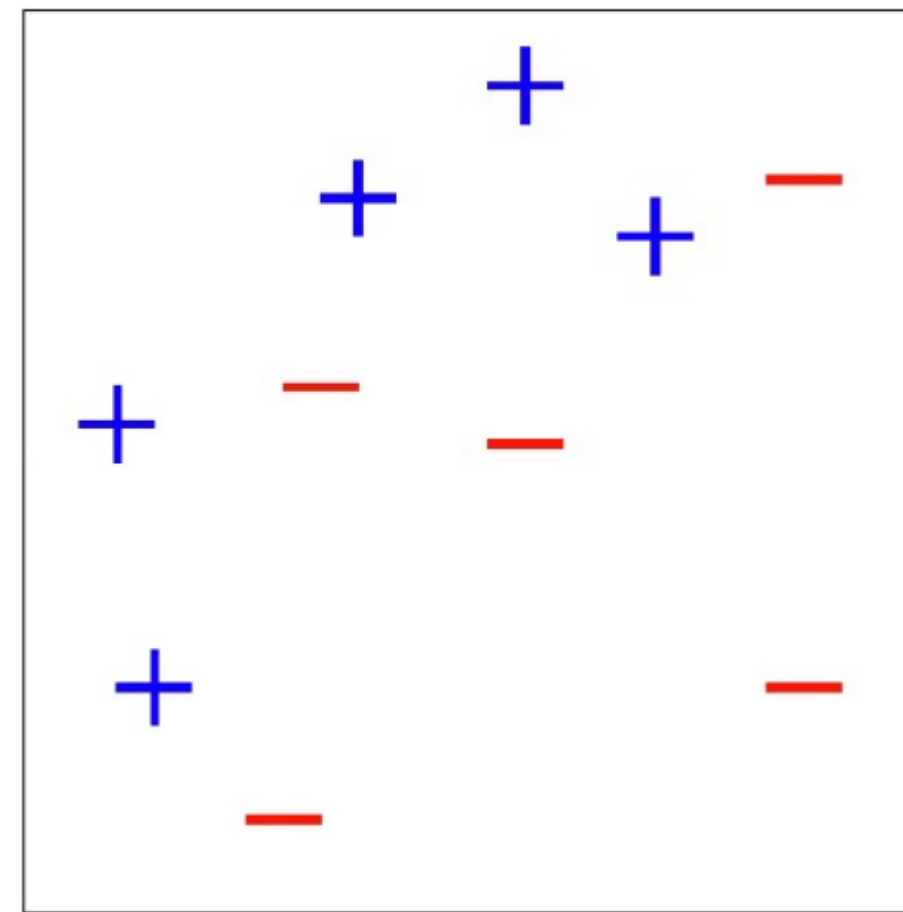
Infinite $\mathcal{H}$

- Our algorithm doesn't depend on $|\mathcal{H}|$ any more!
 - ▶ only way we access $\mathcal{H}$ is to pick h_t with low error
 - ▶ this is a weighted classification problem $\rightarrow$ we have plenty of ML methods to solve it
 - ▶ once we have h_t , finding α_t is a 1-D minimization problem, fast/easy to solve exactly
- Only code change needed is to train a classifier (the weak learner) instead of looking at each element of $\mathcal{H}$ to find a good one
- Called ***AdaBoost***

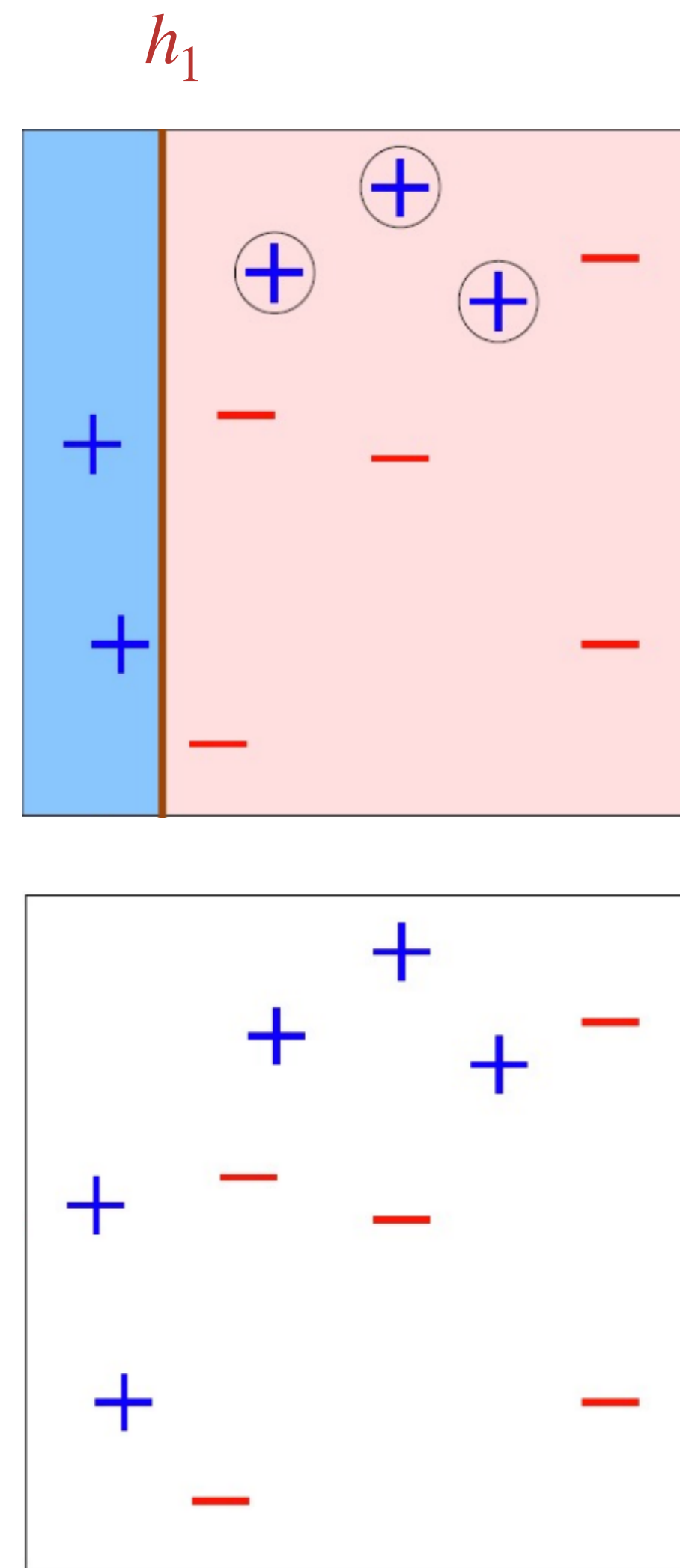
AdaBoost summary

- Input: number of rounds T , dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$
- Initialize datapoint weights $\omega_0^{(i)} = \frac{1}{N}$ for all i
- For $t = 1, \dots, T$:
 - ▶ Train a weak learner h_t by minimizing *weighted* training error on $\mathcal{D}$ (weights $\omega_{t-1}^{(i)}$)
 - ▶ Find weighted training error of h_t :
$$\epsilon_t = \sum_{i=1}^N \omega_{t-1}^{(i)} \mathbb{I}(y^{(i)} \neq h_t(\mathbf{x}^{(i)}))$$
 - ▶ Compute the vote weight of h_t : $\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$
 - ▶ Update datapoint weights:
$$\omega_t^{(i)} = \frac{\omega_{t-1}^{(i)}}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & y^{(i)} = h_t(\mathbf{x}^{(i)}) \\ e^{\alpha_t} & \text{o/w} \end{cases} \quad (Z_t \text{ is normalizer})$$
- Return: weighted vote classifier $\hat{y} = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$

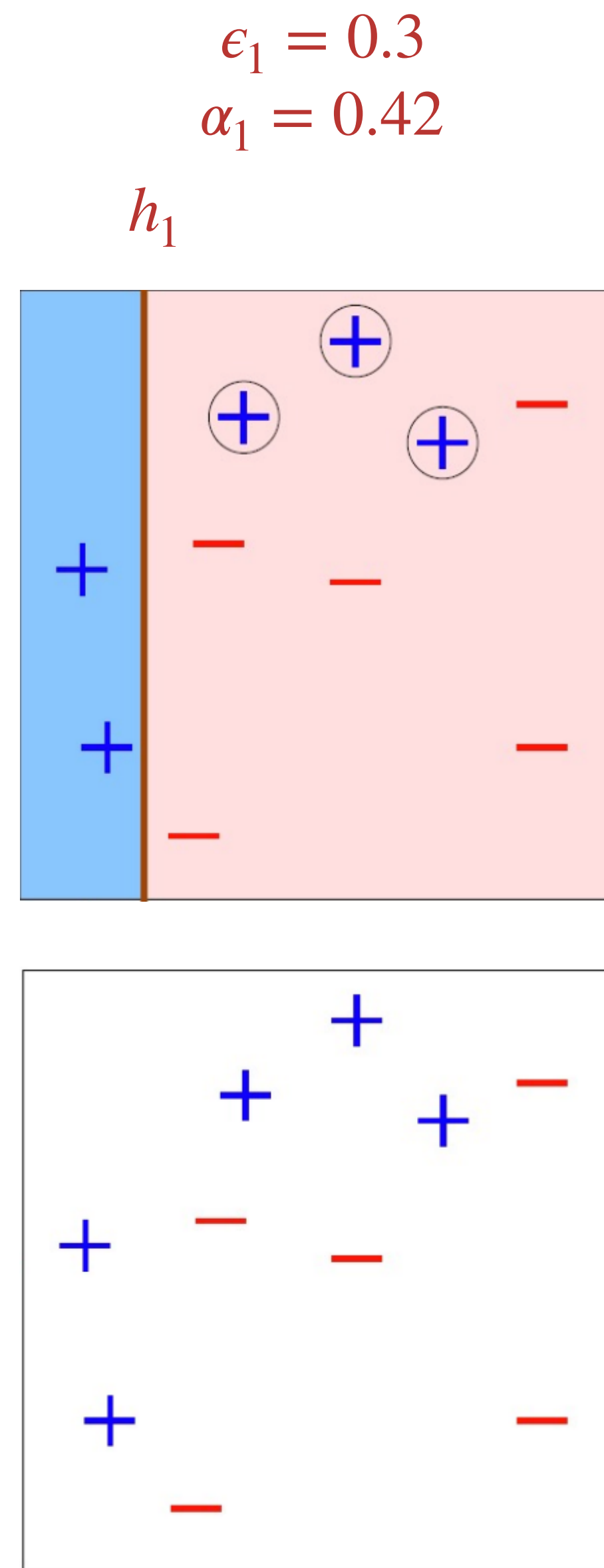
AdaBoost example



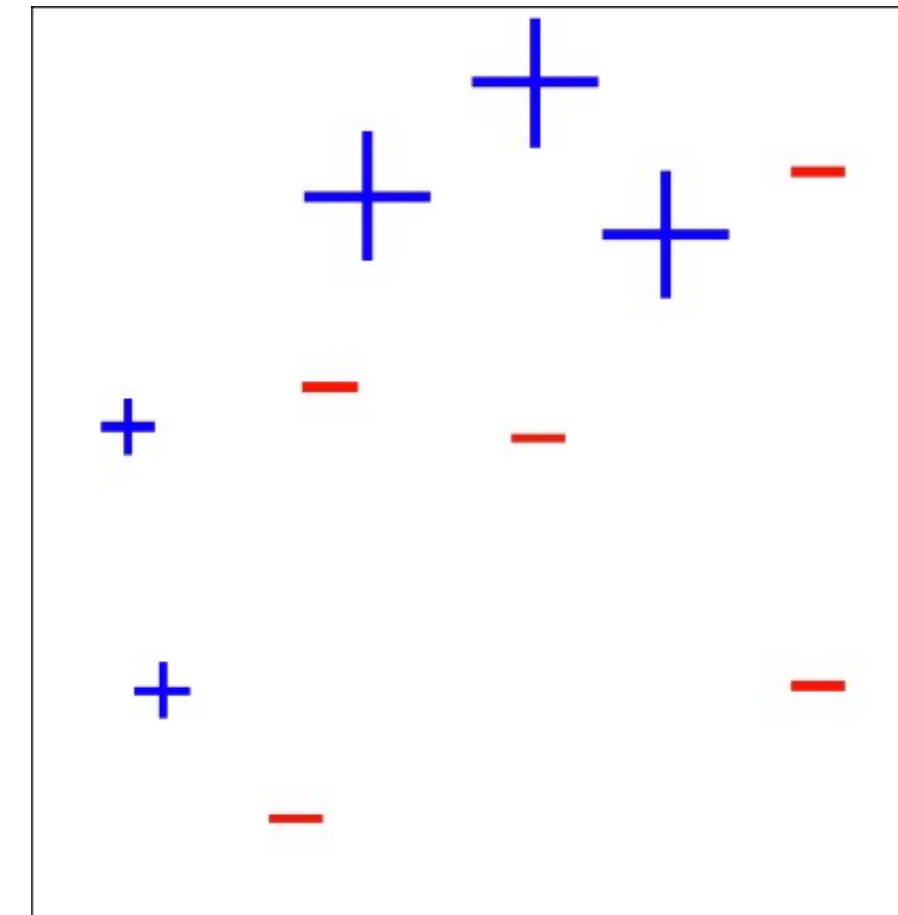
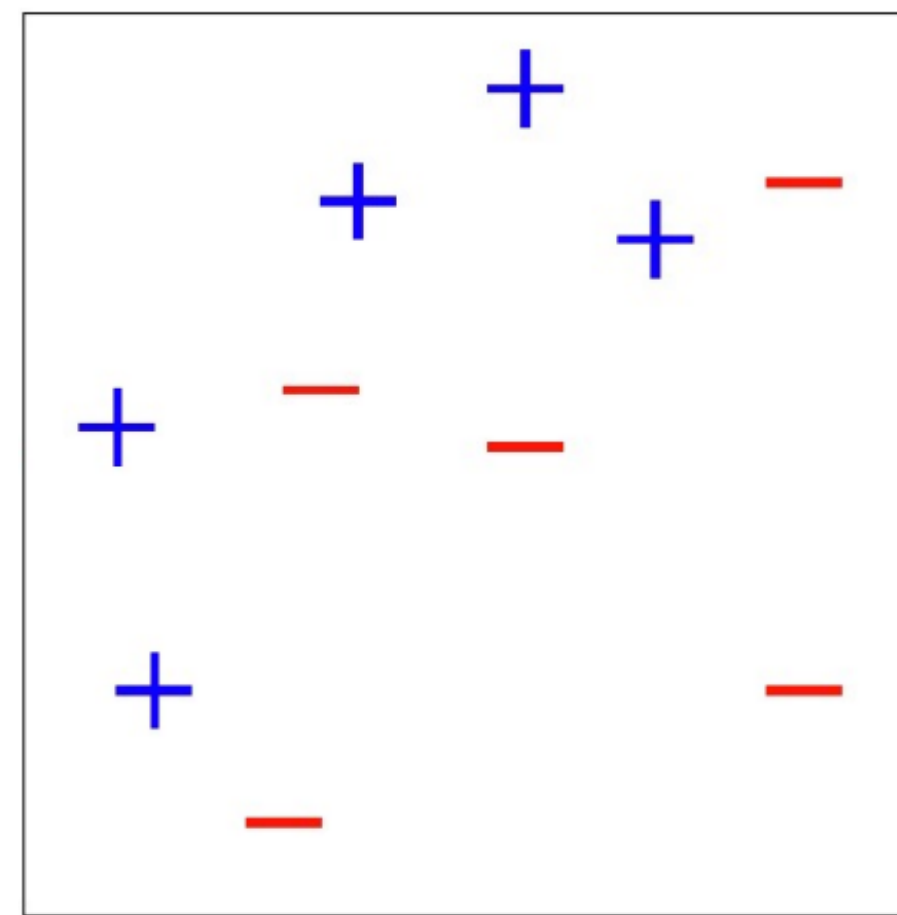
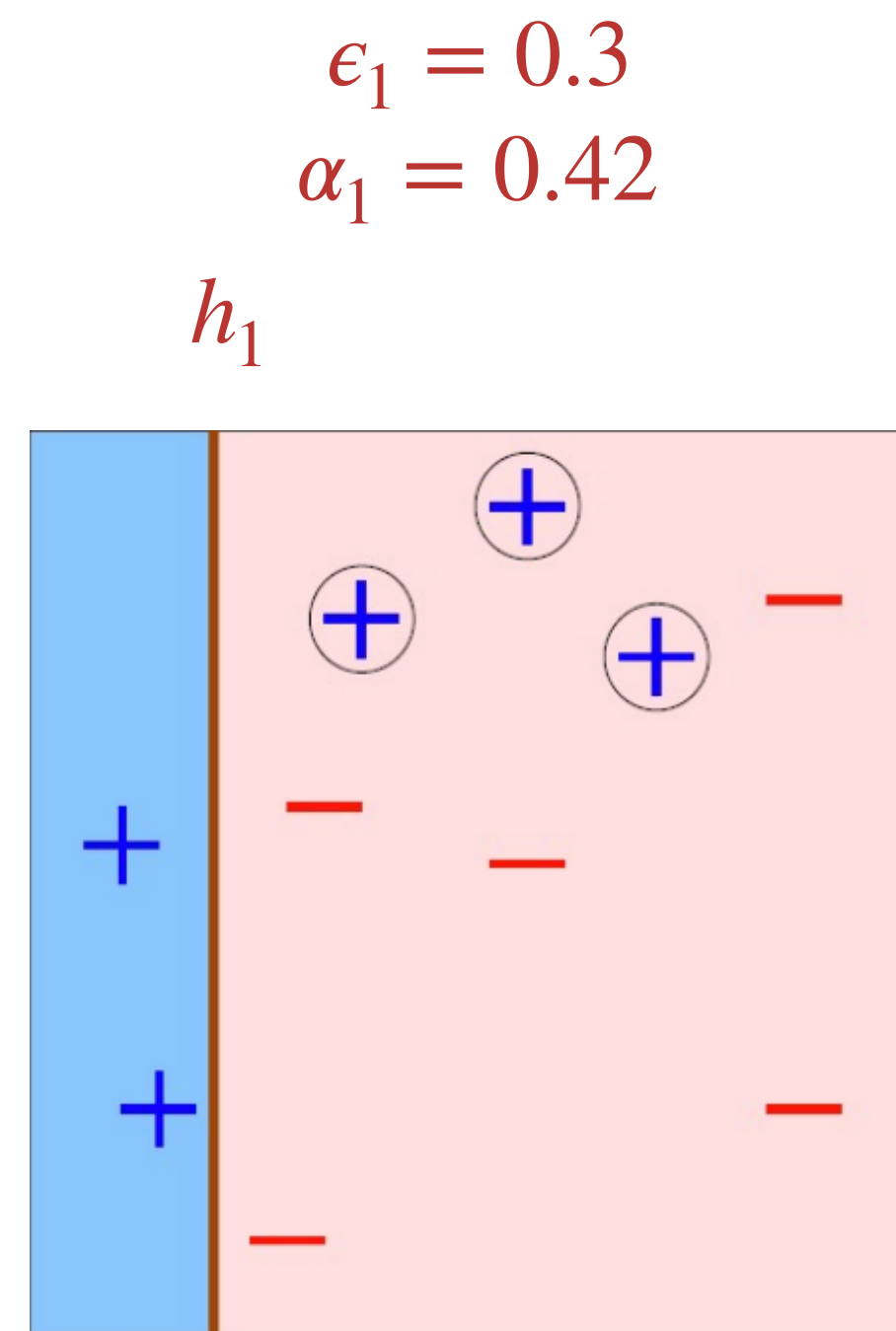
AdaBoost example



AdaBoost example



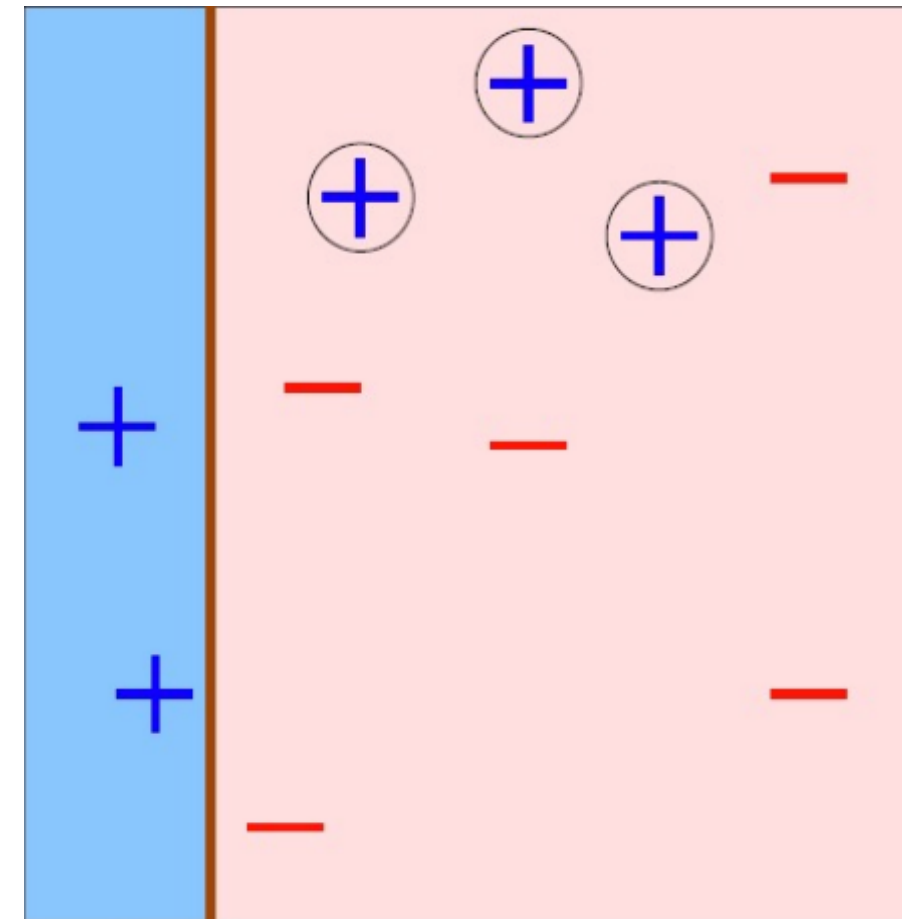
AdaBoost example



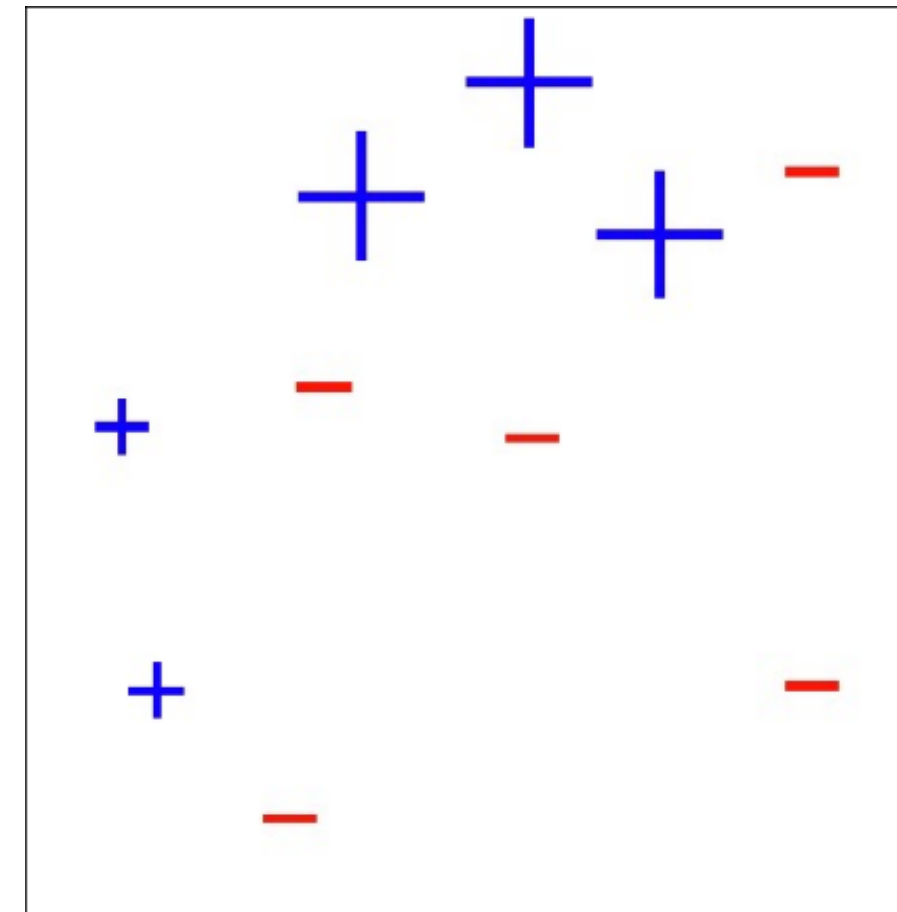
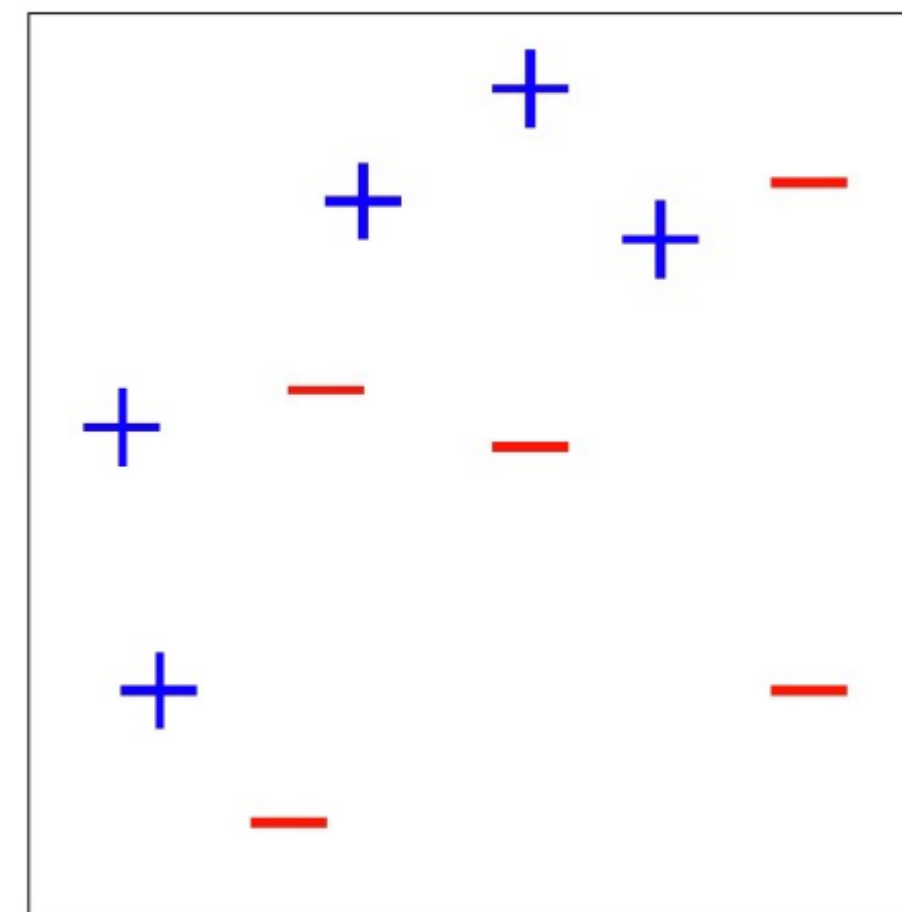
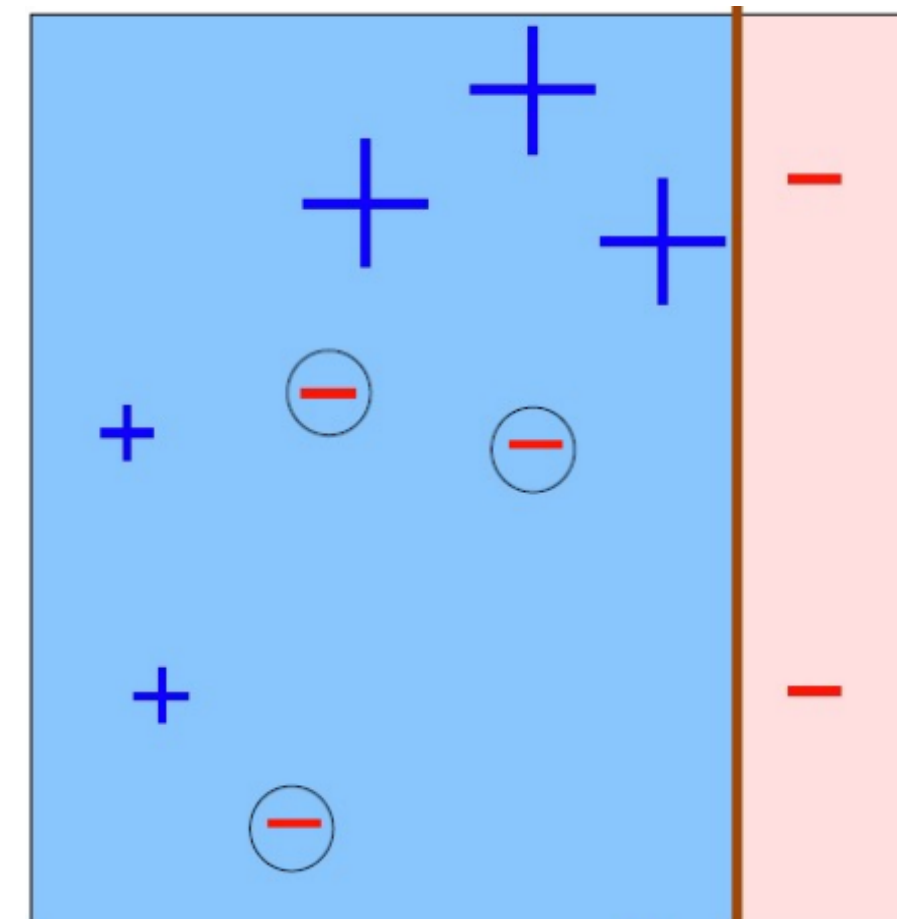
AdaBoost example

$$\epsilon_1 = 0.3$$
$$\alpha_1 = 0.42$$

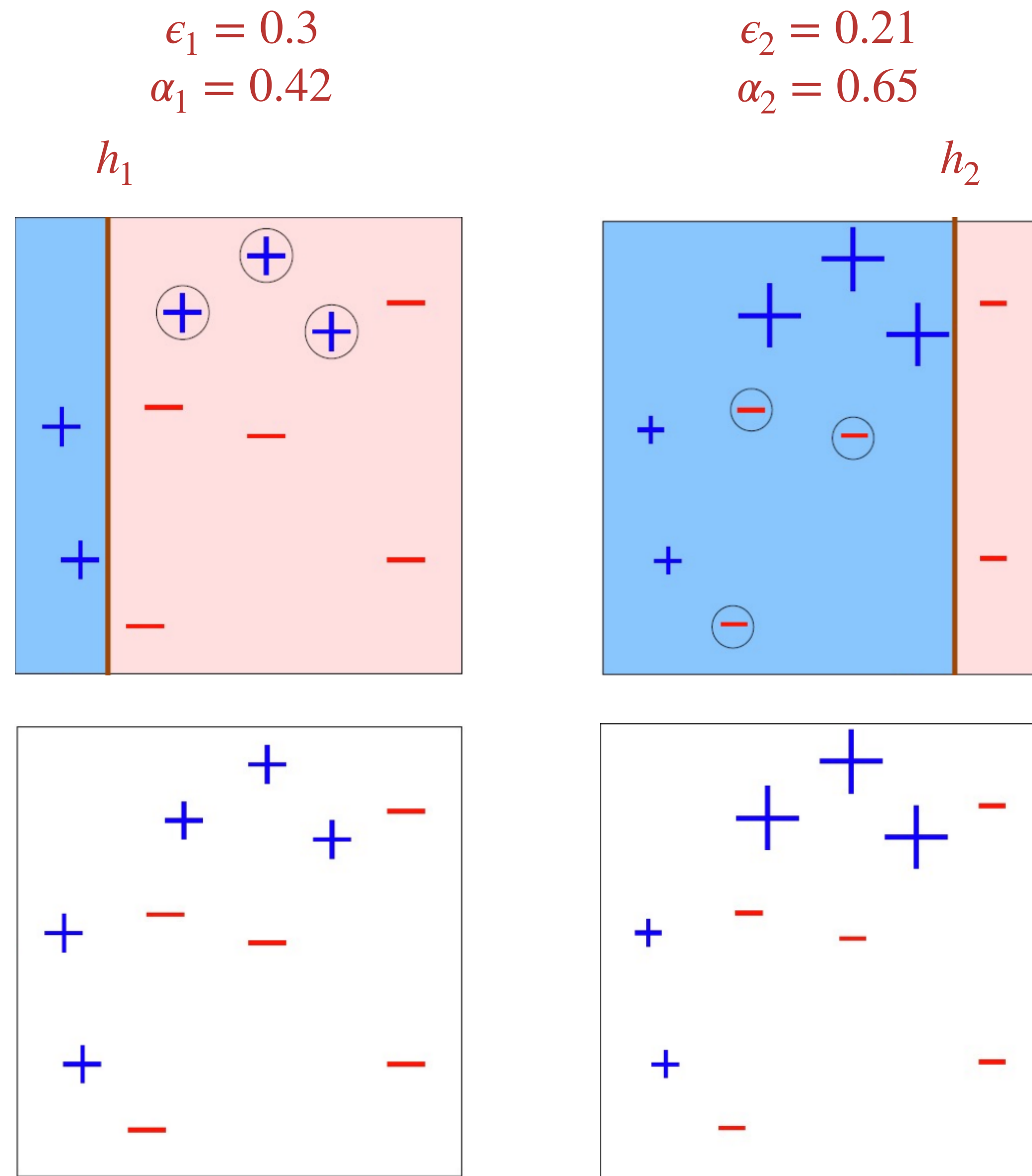
h_1



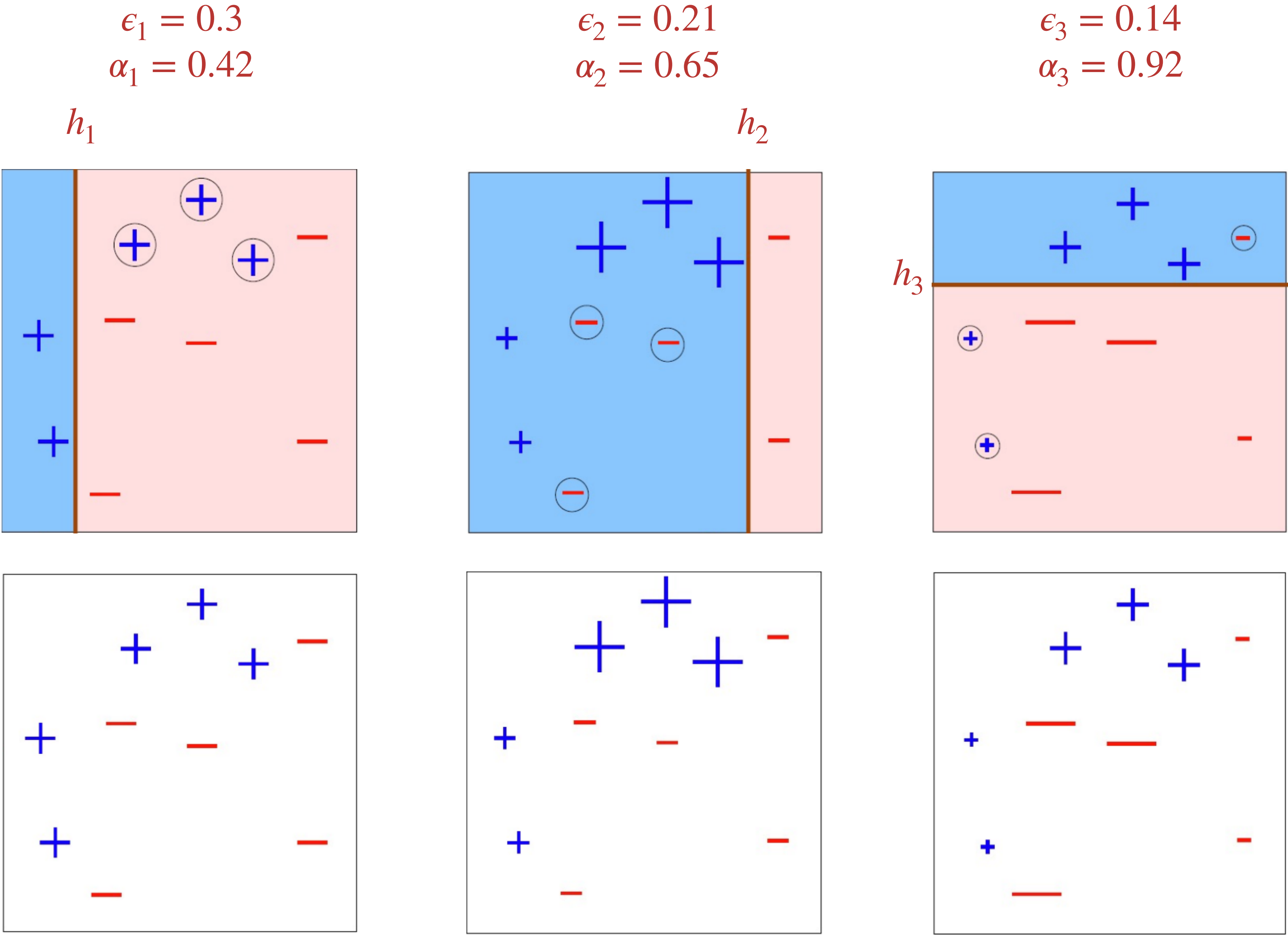
h_2



AdaBoost example

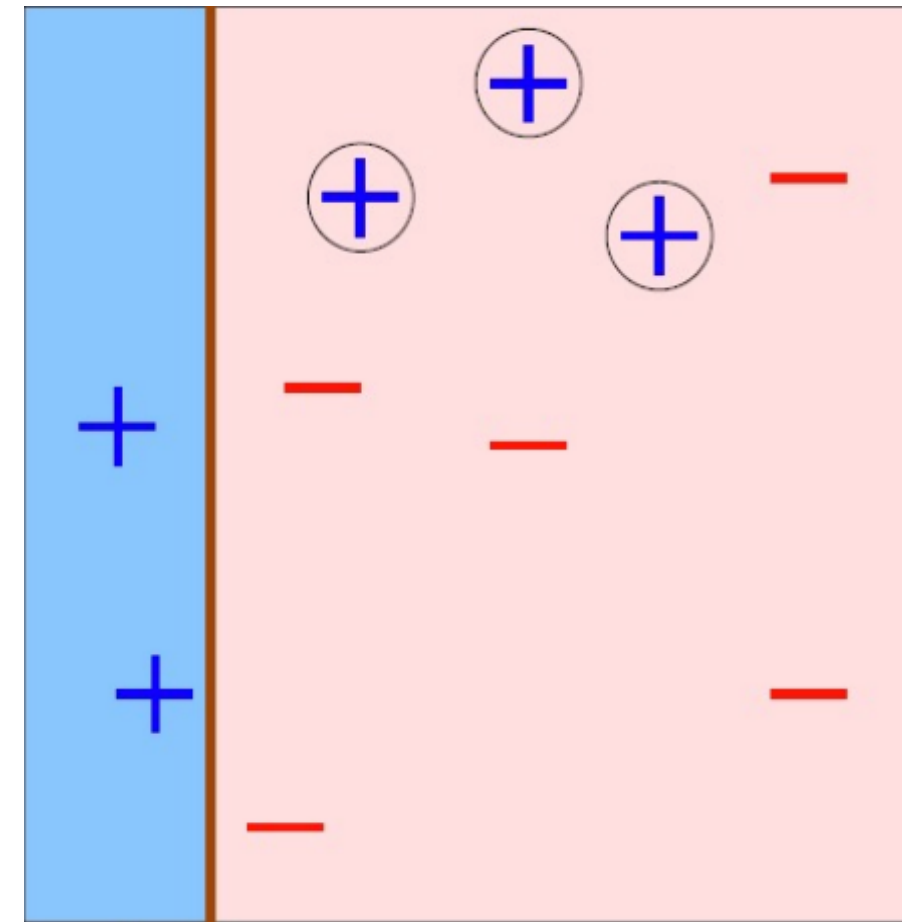


AdaBoost example



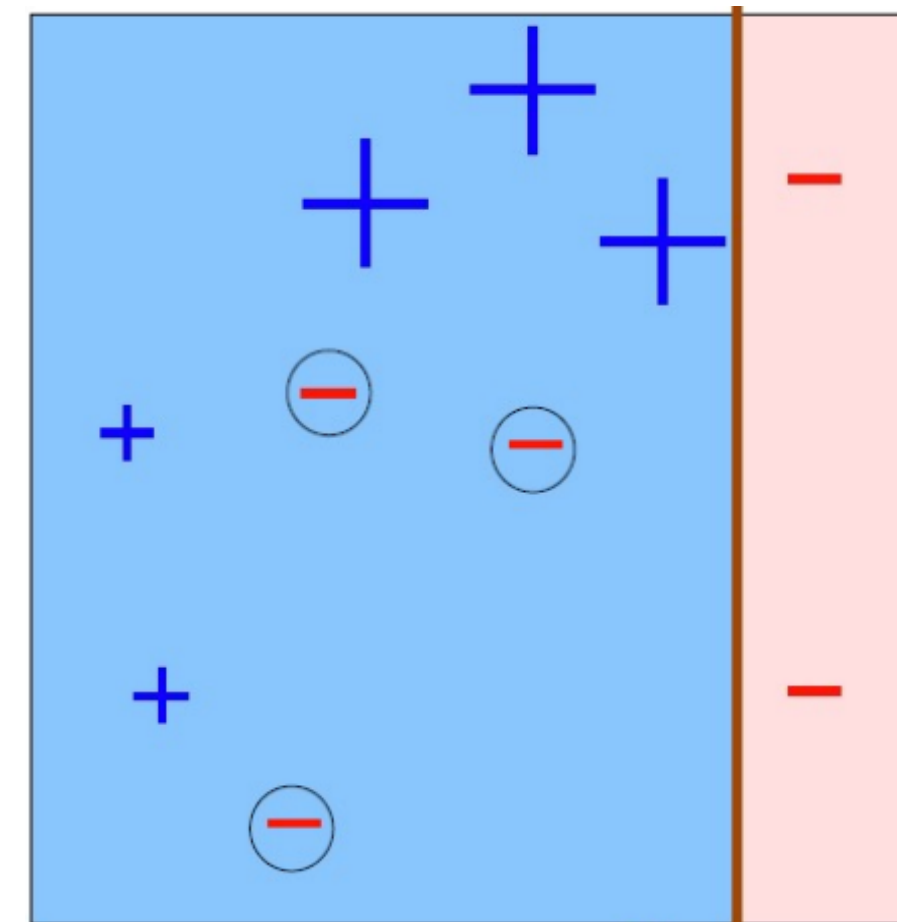
AdaBoost example

0.42 h_1



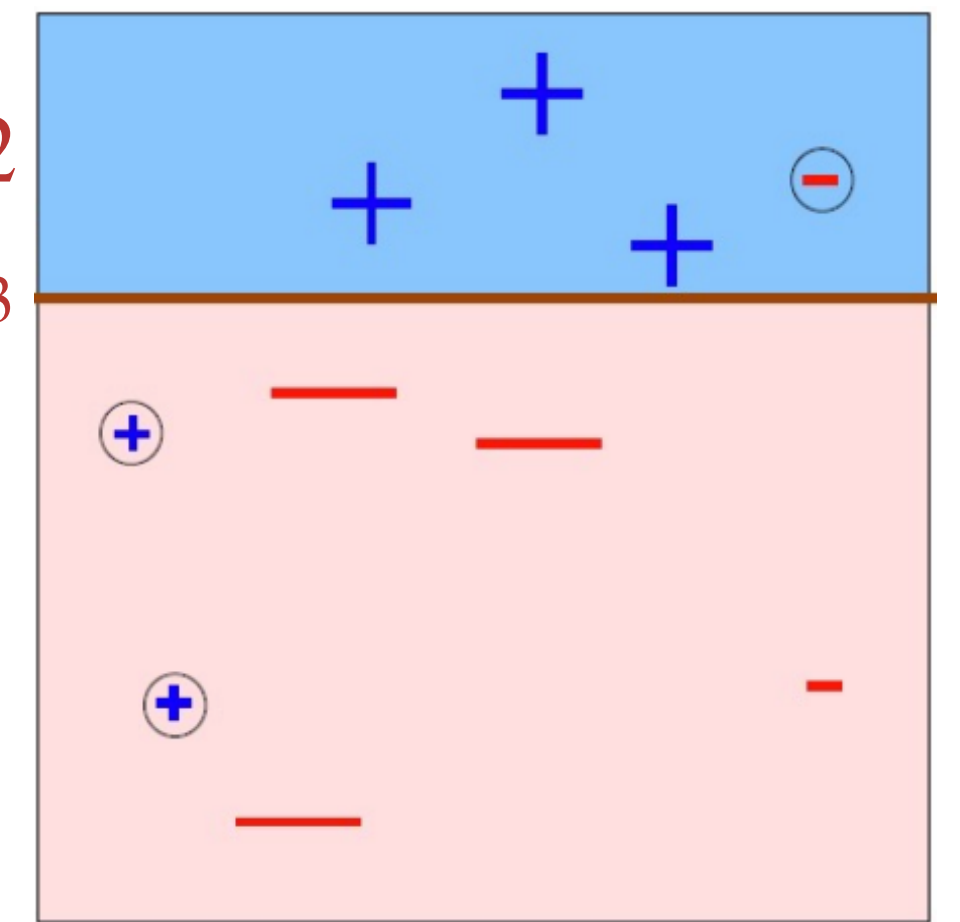
+

0.65 h_2



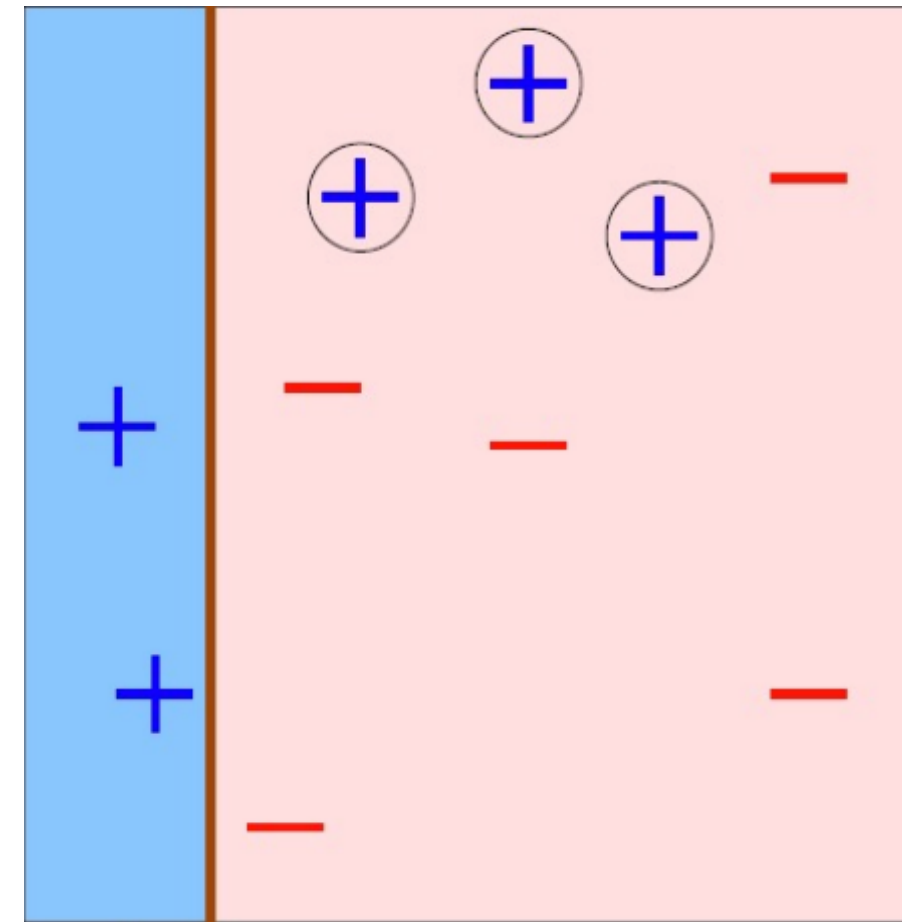
0.92
 h_3

+



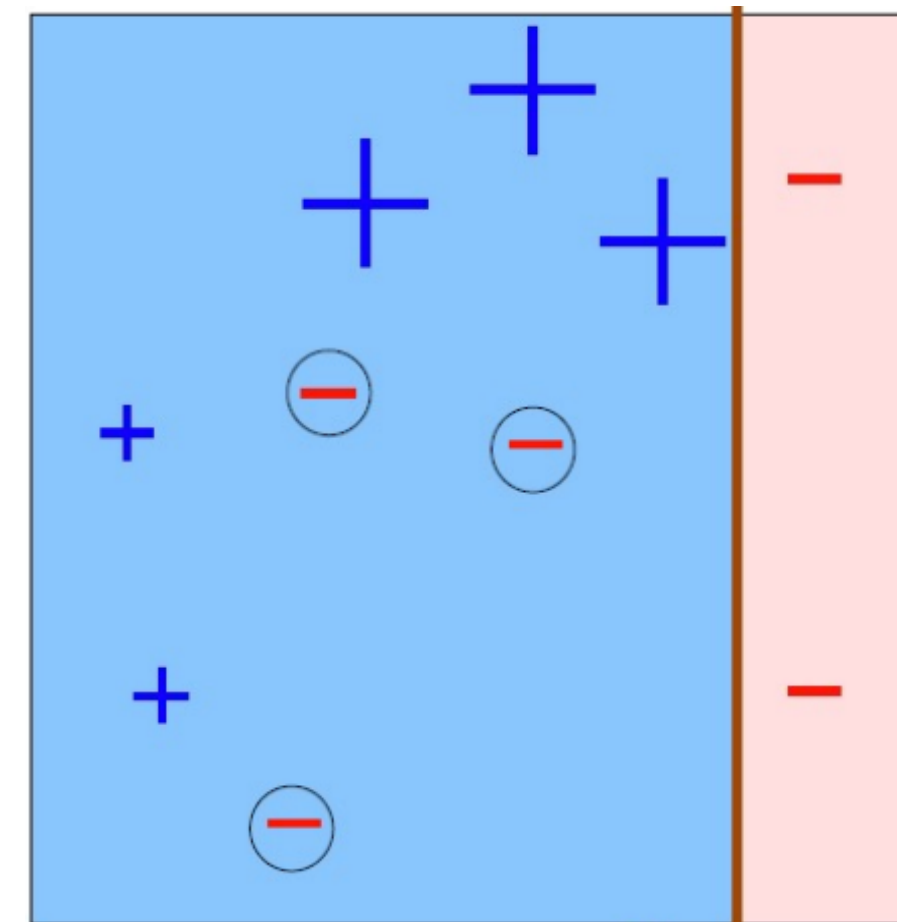
AdaBoost example

$0.42 h_1$

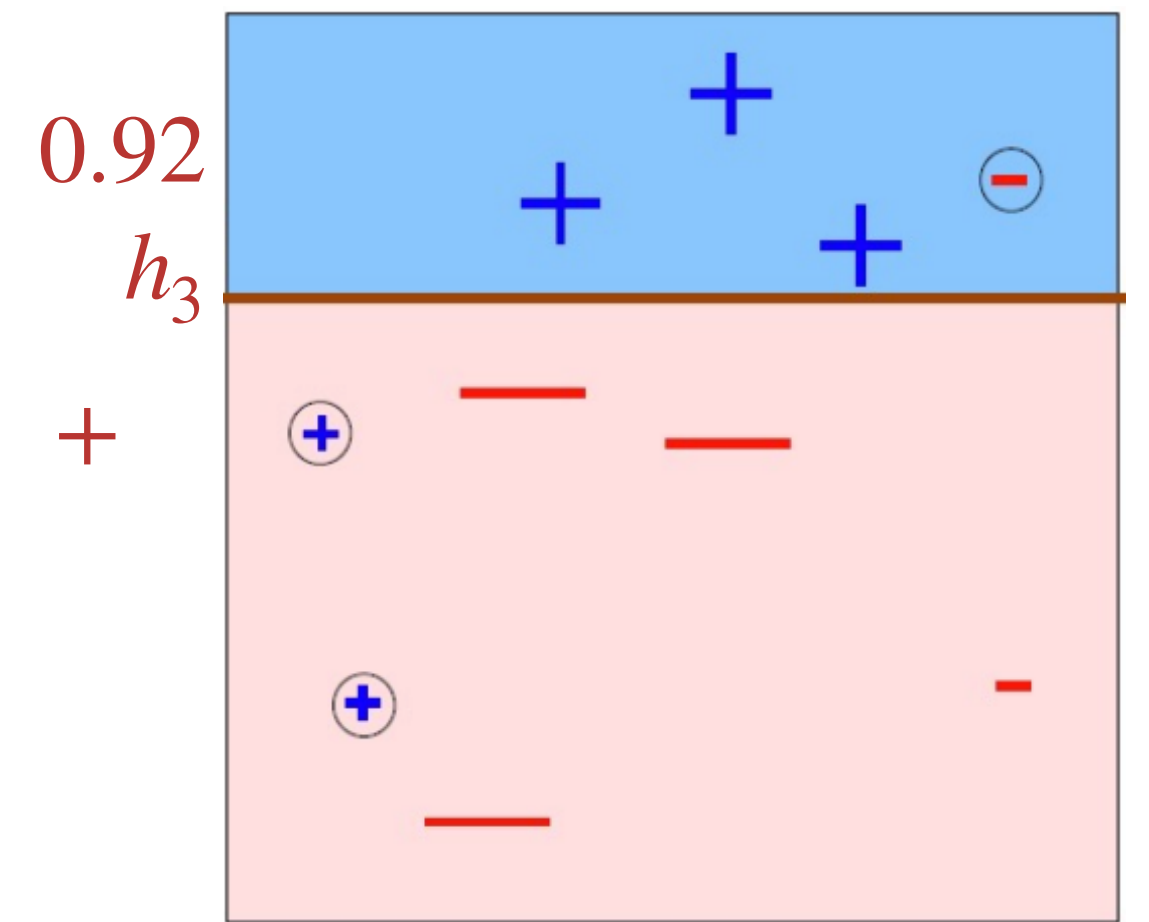


+

$0.65 h_2$

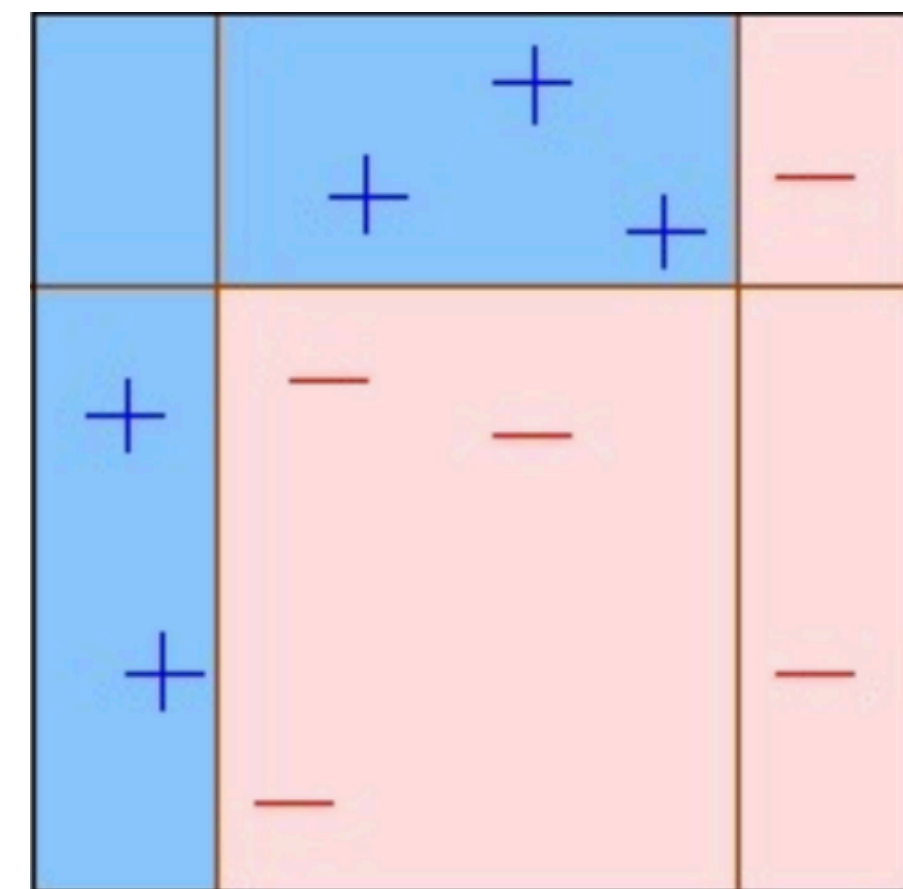


$0.92 h_3$

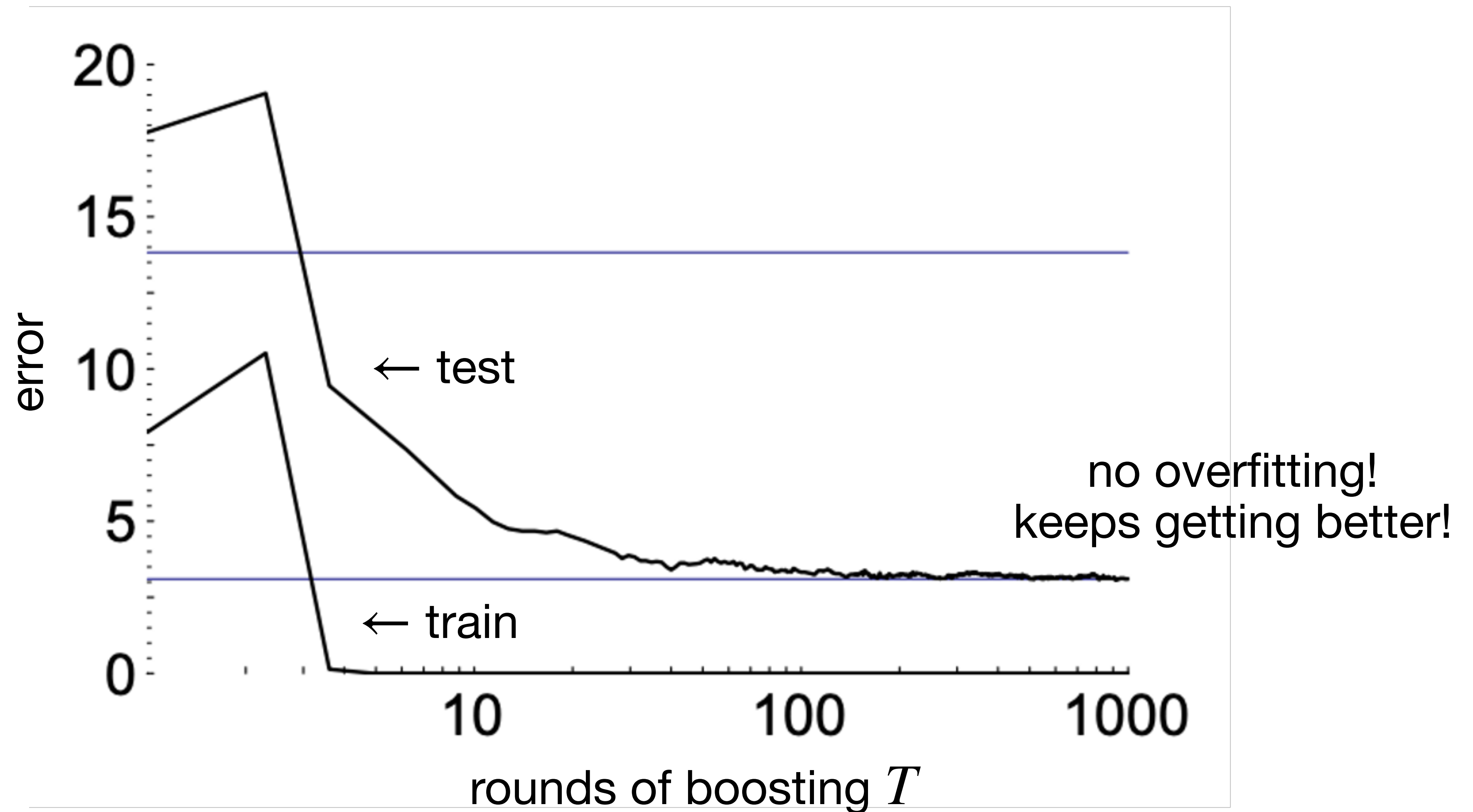


+

=



Another AdaBoost example



Source: <http://rob.schapire.net/papers/msri.pdf>

AdaBoost analysis

- Suppose we fix $\mathcal{H}$ and train until all margins are positive
- With probability $1 - \delta$,

$$\text{true } \mathbf{error} \text{ of final vote} \leq \text{boosting } \mathbf{loss} + O\left(\frac{A + \sqrt{\ln 1/\delta}}{\sqrt{N}}\right)$$

where A bounds weights: $\sum_t \alpha_t \leq A$

- Constant in big-O depends on complexity of hypothesis class of weak learners
 - ▶ bound **increases** with lower failure probability δ , higher loss, bigger weights, more complex weak learners
 - ▶ bound **decreases** with bigger training set N
 - ▶ **no dependence** on number of rounds T

[Bartlett & Mendelson 2002, Thm 7, Thm 12] — they actually prove something stronger, and many extensions

Boosting vs. bagging

- Both boosting and bagging learn a vote over an ensemble of hypotheses from base hypothesis class
- But their purpose is different:
 - ▶ bagging tries to capture uncertainty (posterior over classifiers, predictive distribution) without necessarily beating the base hypothesis class
 - ▶ boosting tries to beat the base hypothesis class (weak learners), doesn't help much with estimating uncertainty since votes tend to be lopsided
 - ▶ bagging is easy to use for multi-class and regression; boosting was designed for binary classification (though generalizations have been proposed)

Boosting vs. bagging

- And their behavior is different
 - ▶ many small trees / weak hypotheses (boosting) vs. fewer large trees / stronger hypotheses (bagging)
 - ▶ fit different parts of target concept (boosting) vs. all of it (bagging)
 - ▶ limits overfitting by maximizing minimum margin (boosting) vs. by randomization and averaging (bagging)

Learning objectives: boosting

- Explain how to construct a feature matrix where each column corresponds to a classifier h_t
- Explain how a vote over linear classifiers can lead to a nonlinear decision boundary
- Implement and explain AdaBoost
 - ▶ how $e^{-\text{margin}}$ loss function approximately maximizes minimum margin
 - ▶ what is coordinate descent
 - ▶ repeated calls to weak learner with weighted dataset
- Explain the generalization bound for AdaBoost

Learning objectives: bagging

- Distinguish between bagging, column subsampling, and random forests
- Implement bagging for an arbitrary base learner
- Implement column subsampling for an arbitrary base learner (classifier or regressor)
- Implement random forests
- Contrast out-of-bag error with cross-validation error
- Differentiate boosting from bagging
- Compare weighted and unweighted votes of classifiers