Machine Learning in a Nutshell

15-488 Spring '20

Lecture 27:
Ensemble Methods

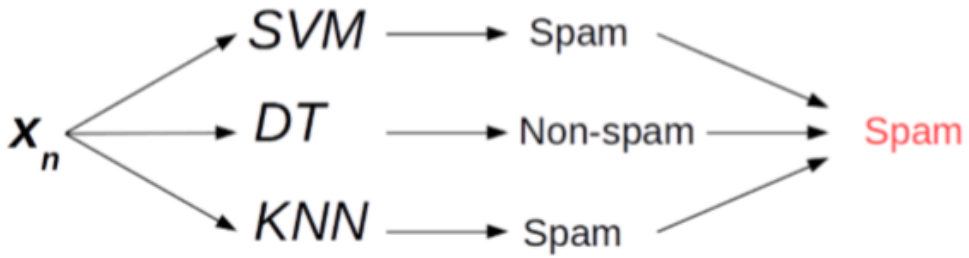
Teacher:
Gianni A. Di Caro

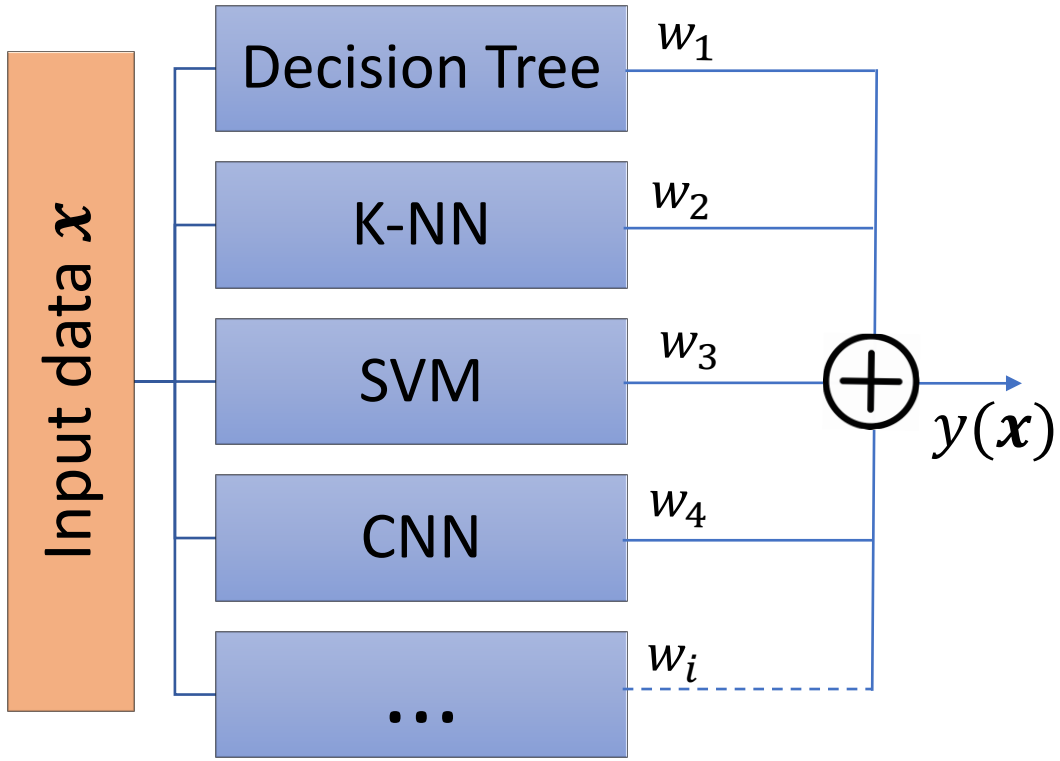
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Ensemble methods: combine different estimators

General Idea: combine the predictions of *several different estimators* in order to improve performance / robustness over a single estimator.

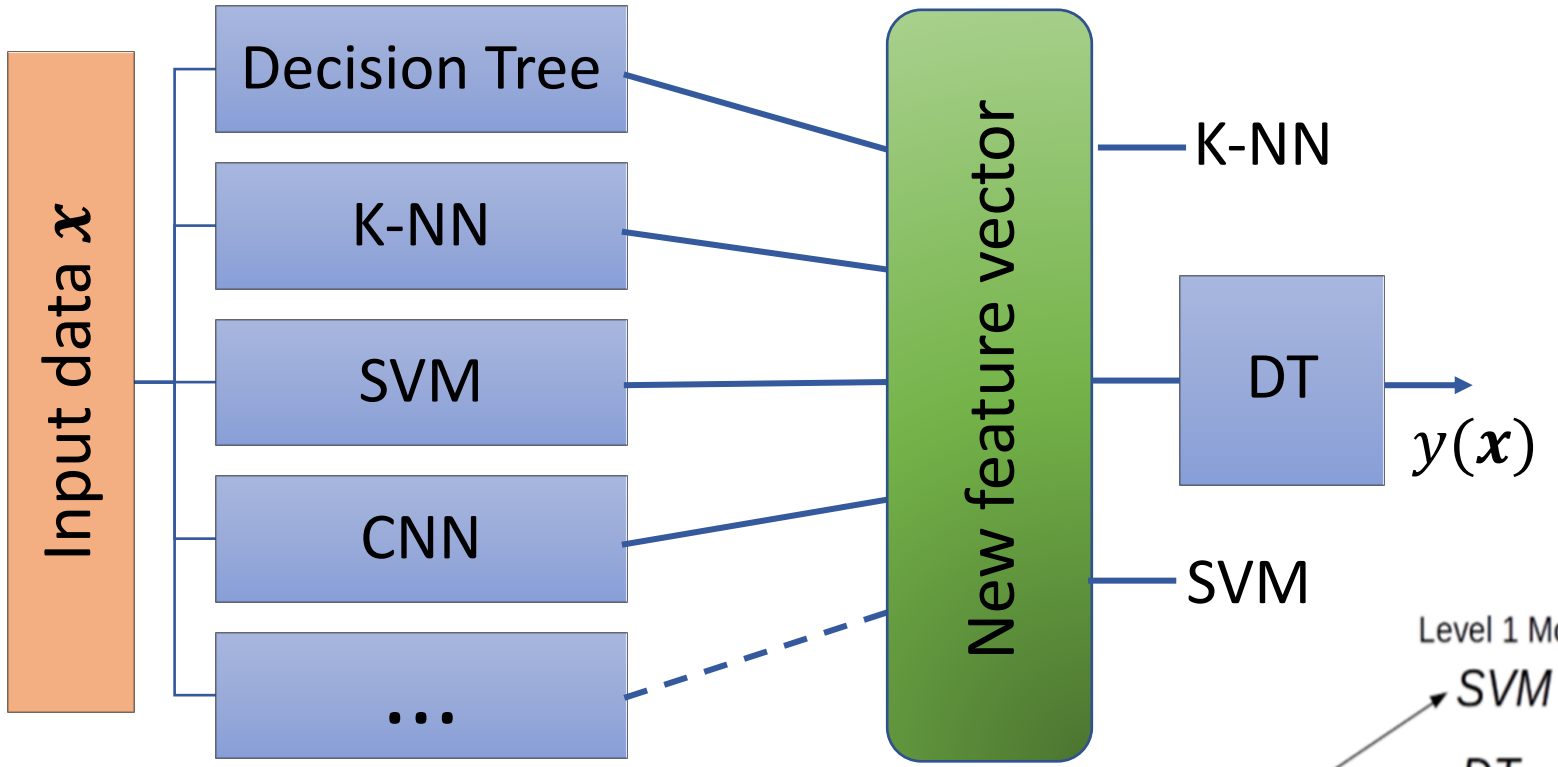
- **Voting / Averaging** of predictions of multiple trained models

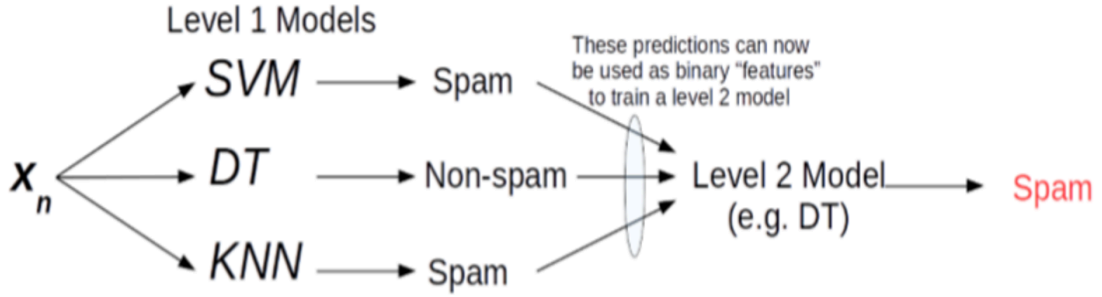




Ensemble methods: combine different estimators

❖ **Stacking**: use predictions of multiple models as *features* to train a new model, and use the new model to make predictions on new data

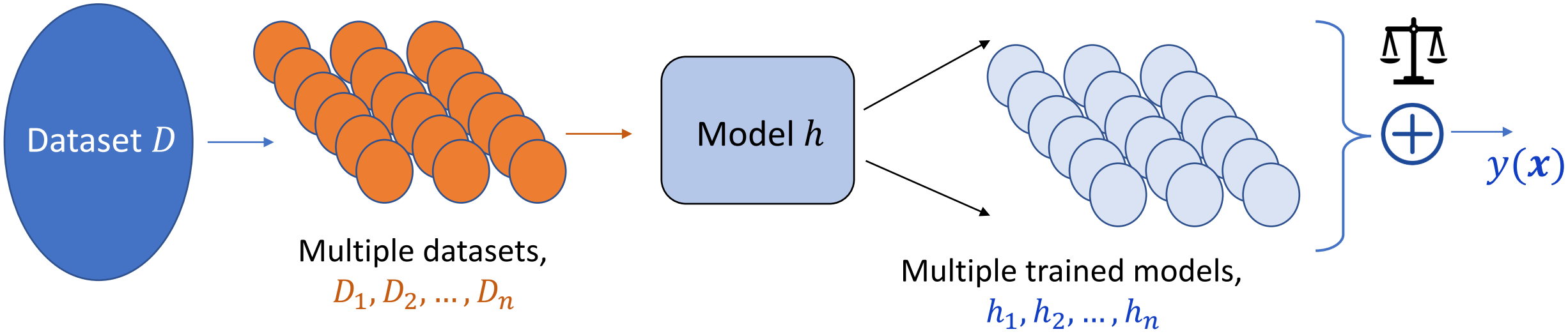




Ensembles: train same model multiple times on different datasets

Bagging & Boosting:

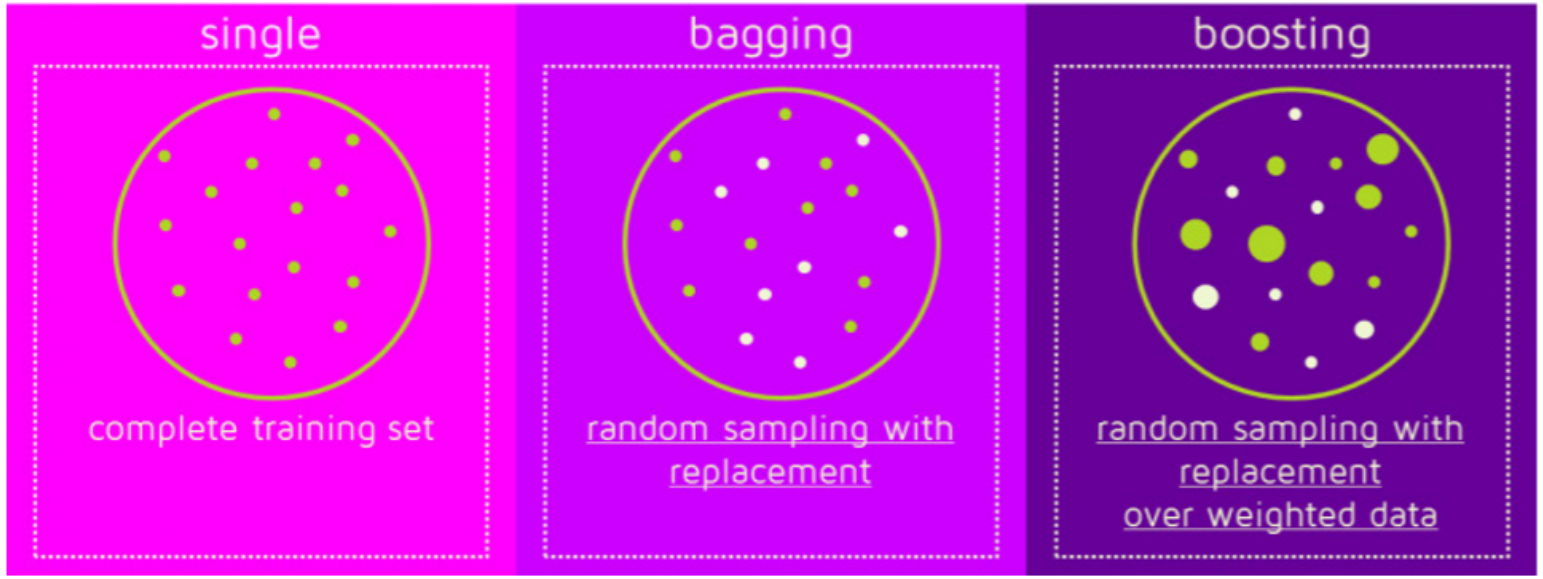
- From one single dataset, D
- ❖ Create multiple datasets, $D_1, D_2, \dots, D_n$
- Train the same model h (e.g., DT) on the different datasets
- ❖ Get n differently trained models, $h_1, h_2, \dots, h_n$ (i.e., n different models)
- ➢ Output prediction $y(\boldsymbol{x})$ is the weighted combination of the outputs of the n models



Bagging vs. Boosting

Differences:

- How to generate the n datasets
- How to choose the initial model
- Objectives



Bagging:

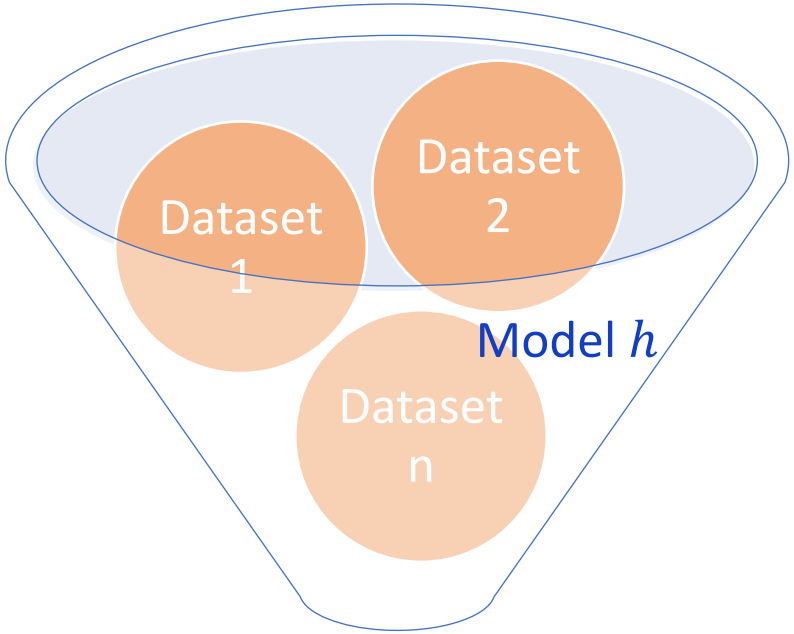
- ✓ Generate n independent datasets (models)
- ✓ Model h needs to be good enough to get a good performance
- ✓ Aim to variance reduction and robustness (using high complexity models)

Boosting:

- ✓ Generate and train models sequentially, with conditional dependencies
- ✓ Model h can/should be a weak model: fast to train, enough to be a little better than random
- ✓ Aim to boost the performance of a weak model, making it a strong model!

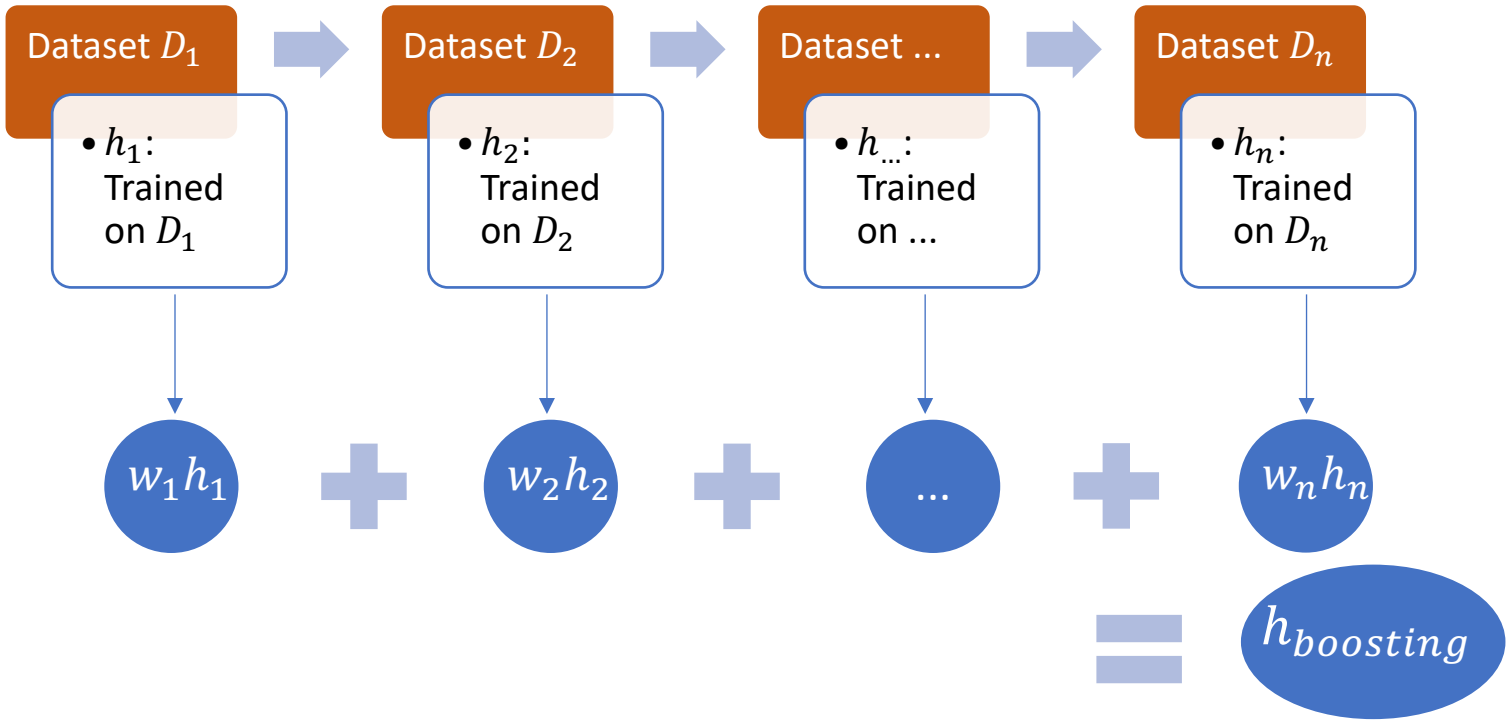
Bagging and Boosting: two ways of combining models by voting

Bagging: train the same model h on n independent training sets → n independently trained models, $h_1, h_2, \ldots, h_n$

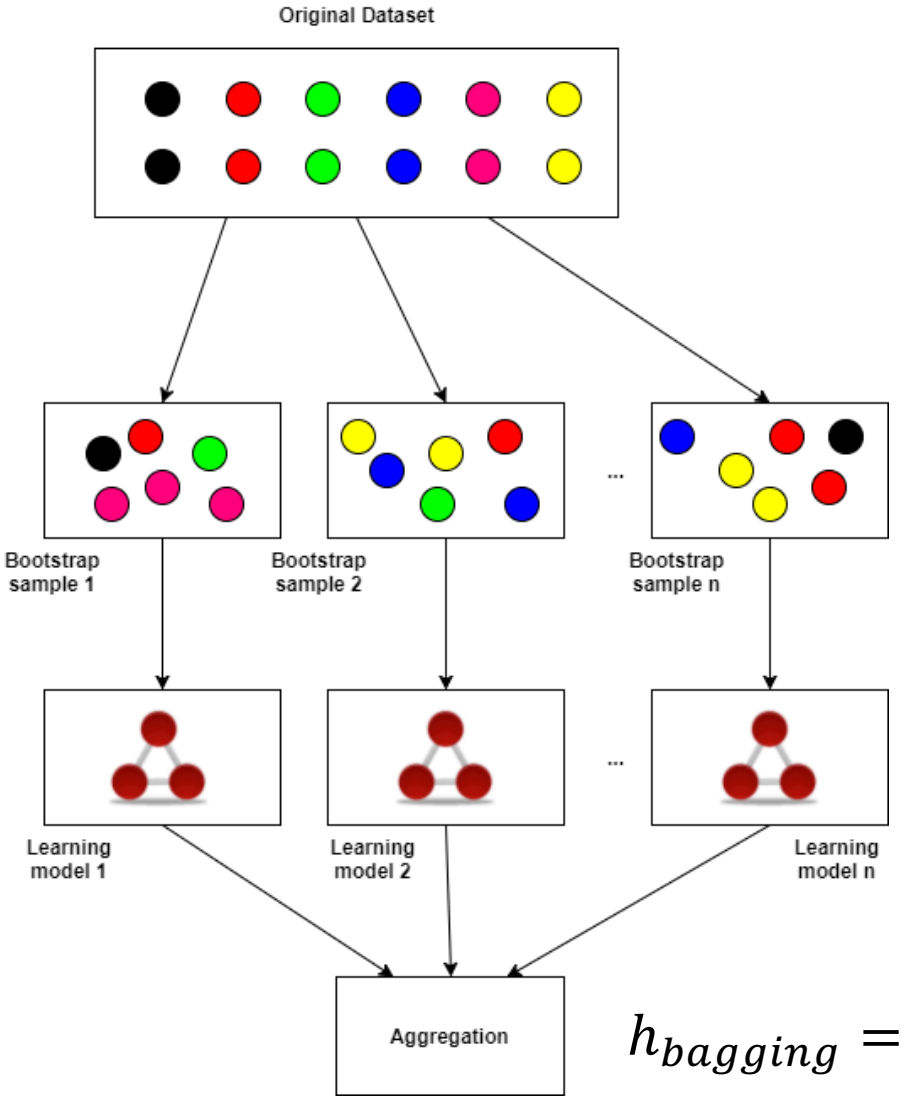


$$h_{bagging} = \frac{1}{n}\sum_{i=1}^{n} w_i h_i$$

Boosting: train the same model over a sequence of n training datasets, each generated conditionally to the previous model/dataset → n trained models each doing well in a different portion of the feature space

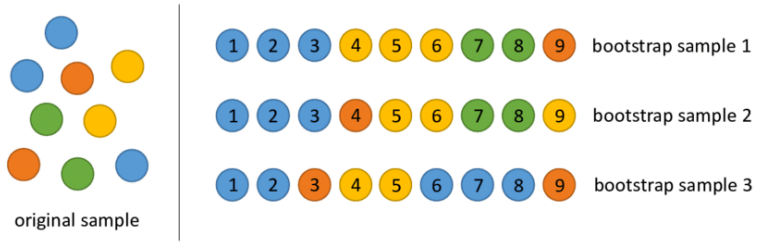


Bagging: Bootstrap Aggregation



Take original dataset D, with m examples

Create n **bootstrap** copies of size $\tilde{m} \leq m$ by re-sampling from D with replacement (sample one element from set D and *bring it back* to D for the next resampling)



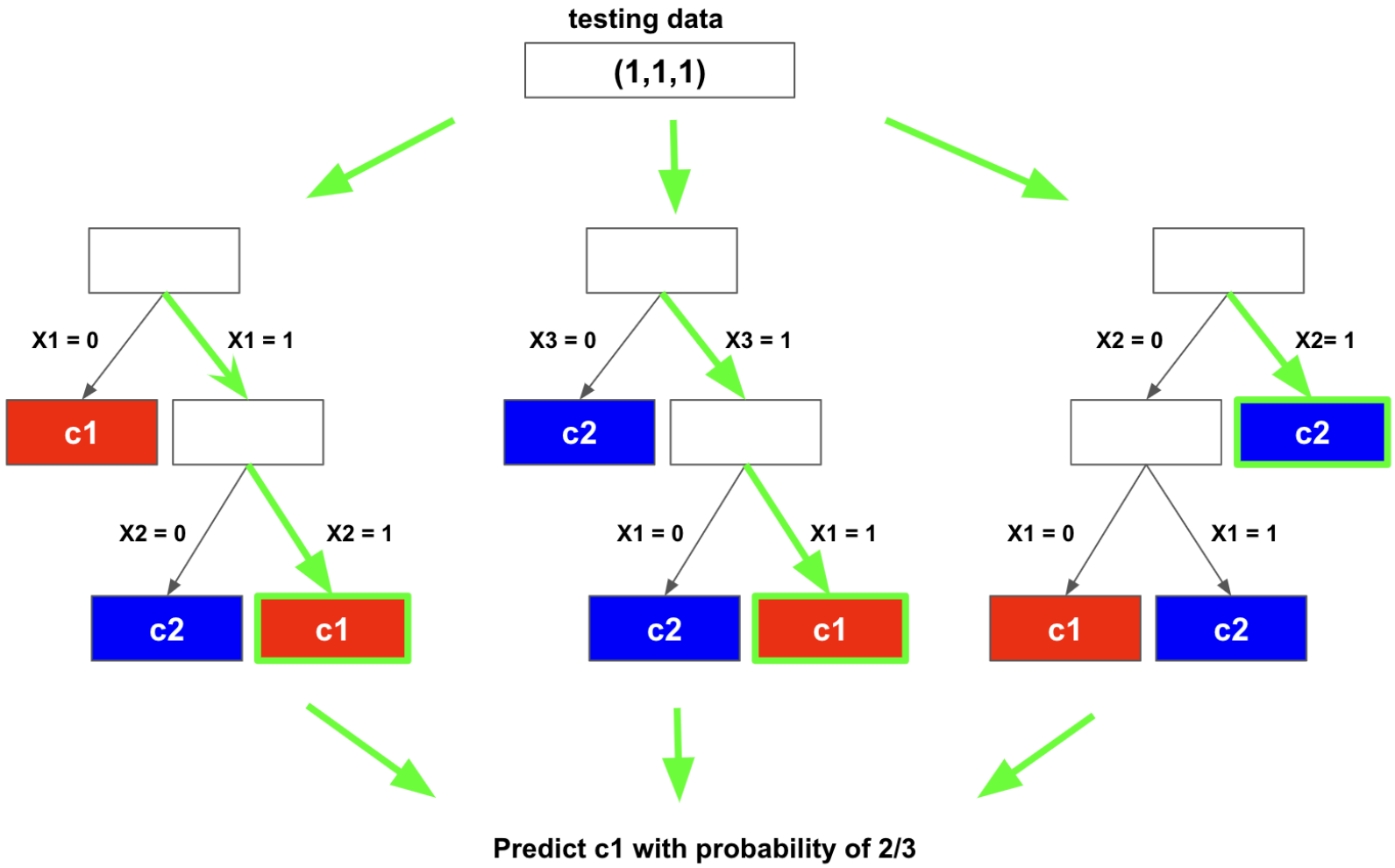
Train the n models independently

$$h_{bagging} = \frac{1}{n}\sum_{i=1}^{n} h_i$$

Average the answers from all n trained models

$$y(\boldsymbol{x}) \leftarrow h_{bagging}(\boldsymbol{x})$$

Bagging: Prediction (example classification)



Bagging: Bootstrap Aggregation

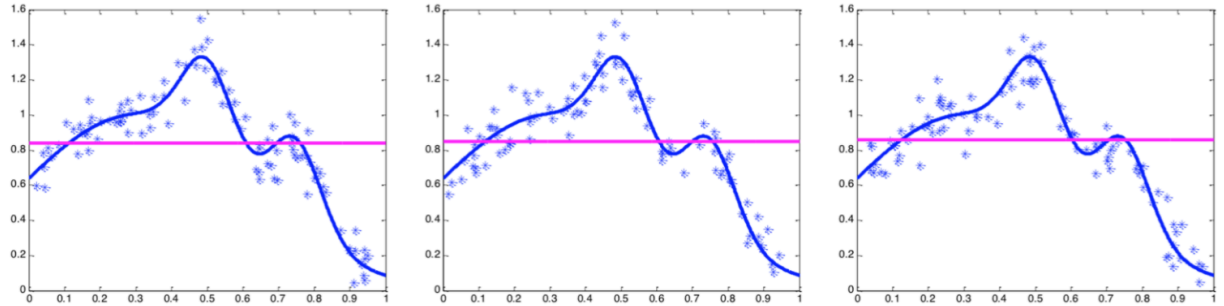
- Bagging is especially useful for models with high variance and noisy data
- High variance models = models whose prediction accuracies varies a lot across different data sets

❑ High variance models: potentially overfitting, high complexity models

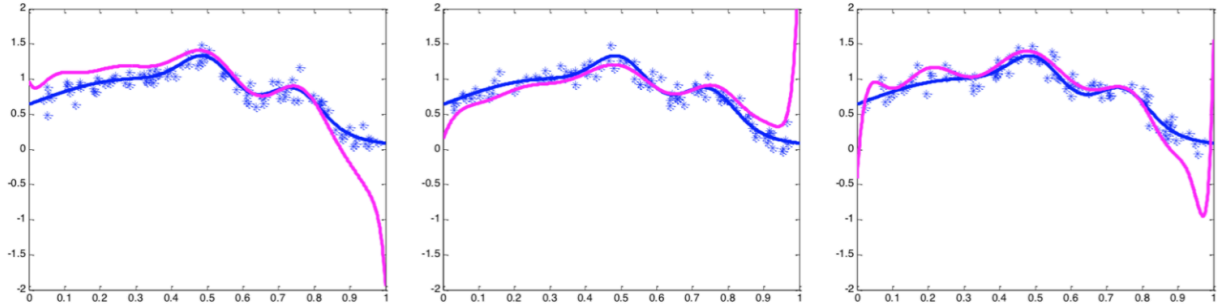
❑ High variance models: potentially very good on some subset of data, potentially very bad on much larger subsets of data
→ Averaging these behaviors will balance the overall answer

3 Independent training datasets

Large bias, Small variance – poor approximation but robust/stable

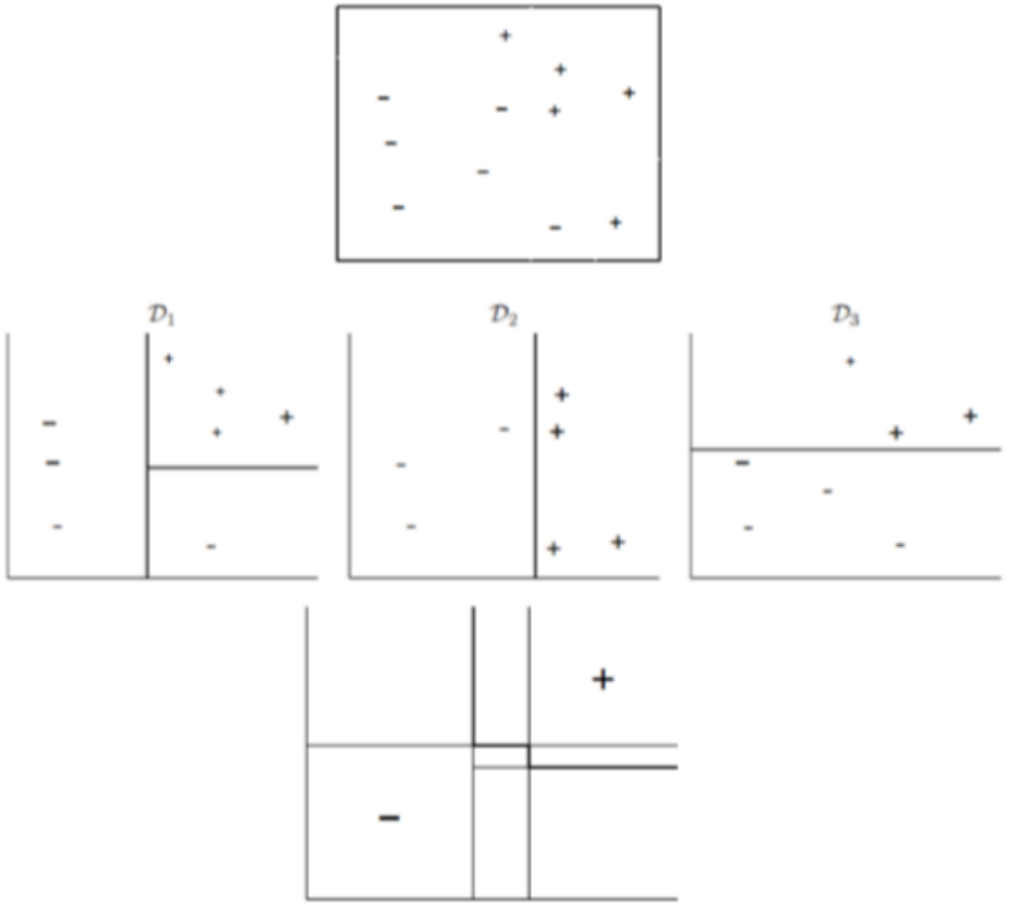


Small bias, Large variance – good approximation but unstable

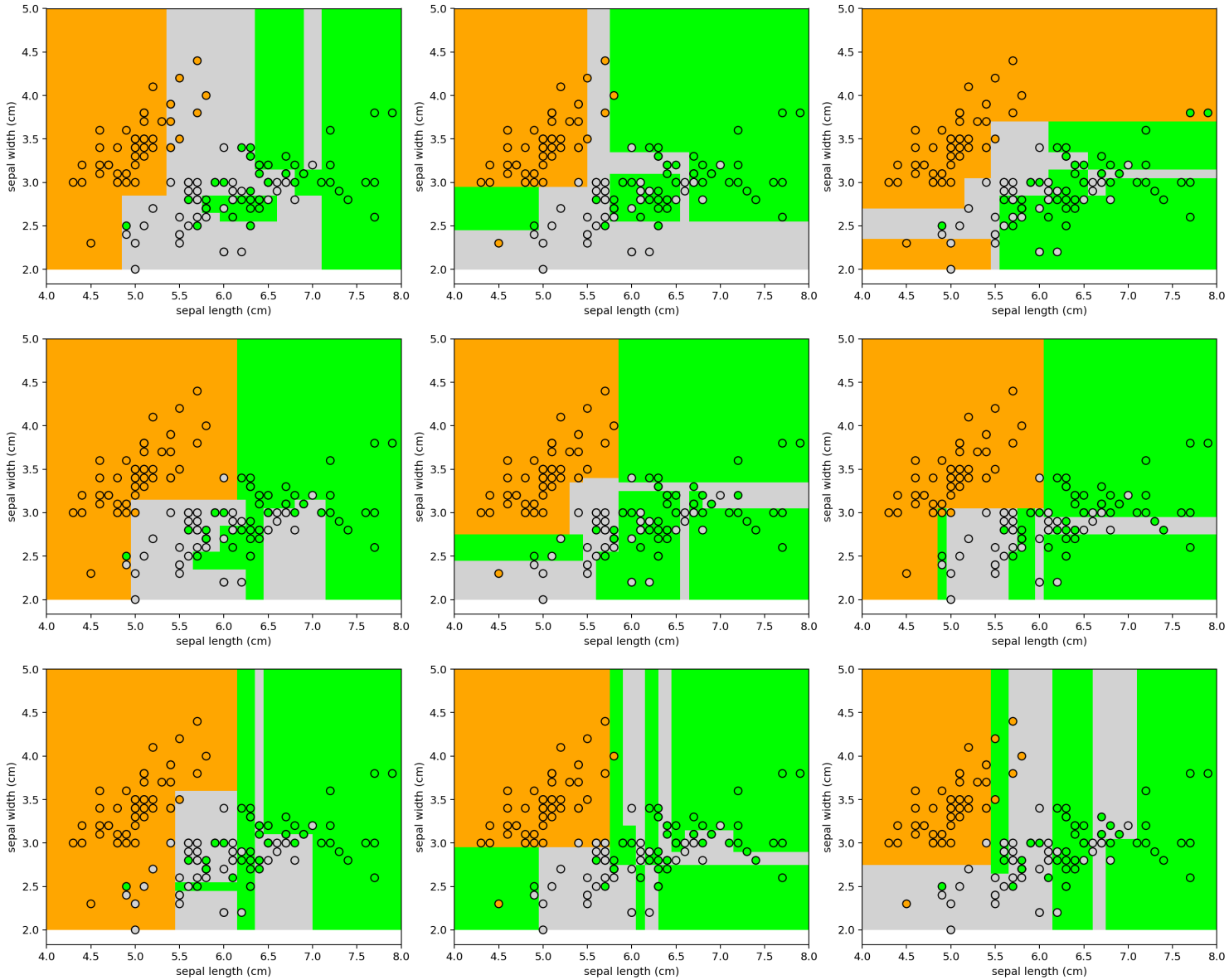


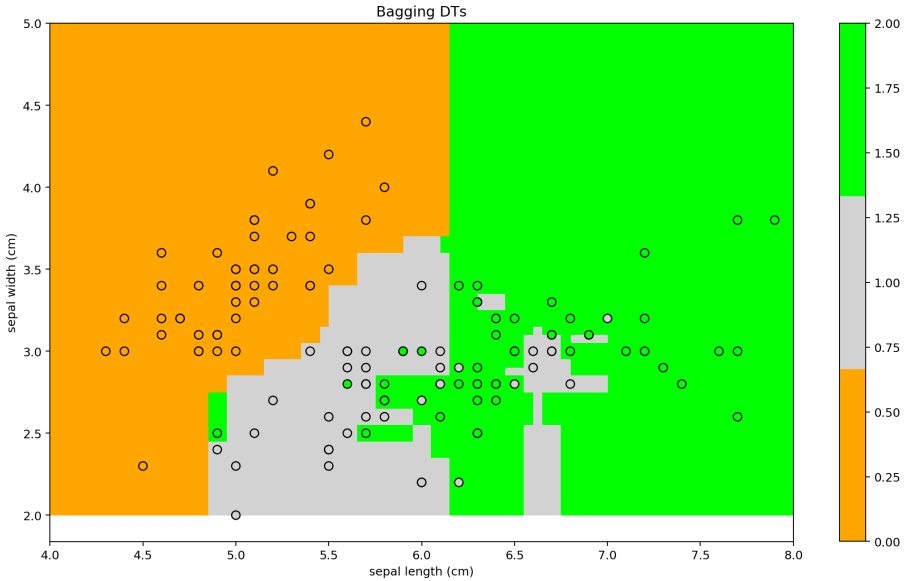
Bagging: illustration

Top: Original data, Middle: 3 models (from some model class) learned using three data sets chosen via bootstrapping, Bottom: averaged model

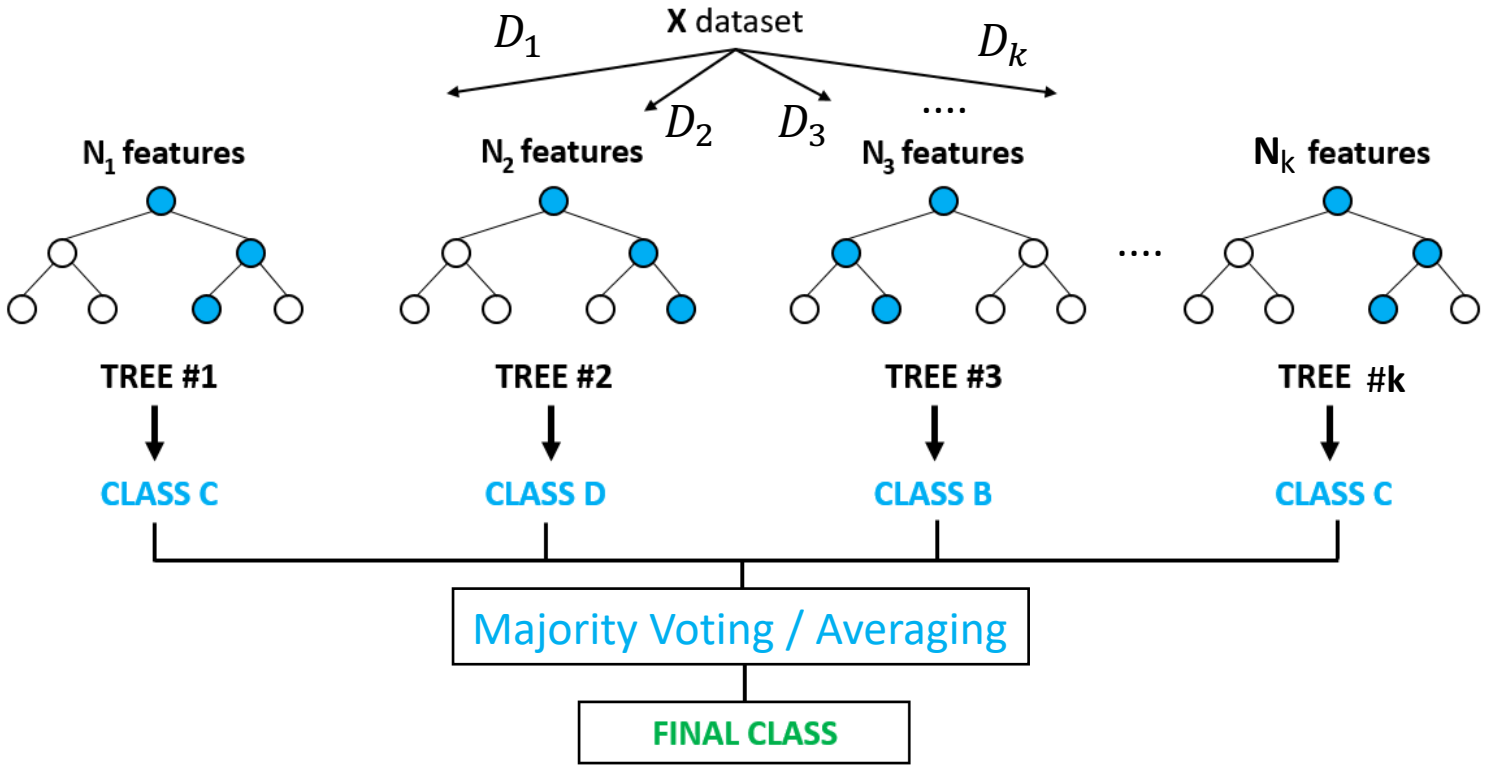


Bagging: illustration





Random Forest: Decision Tree ensemble + bagging on features



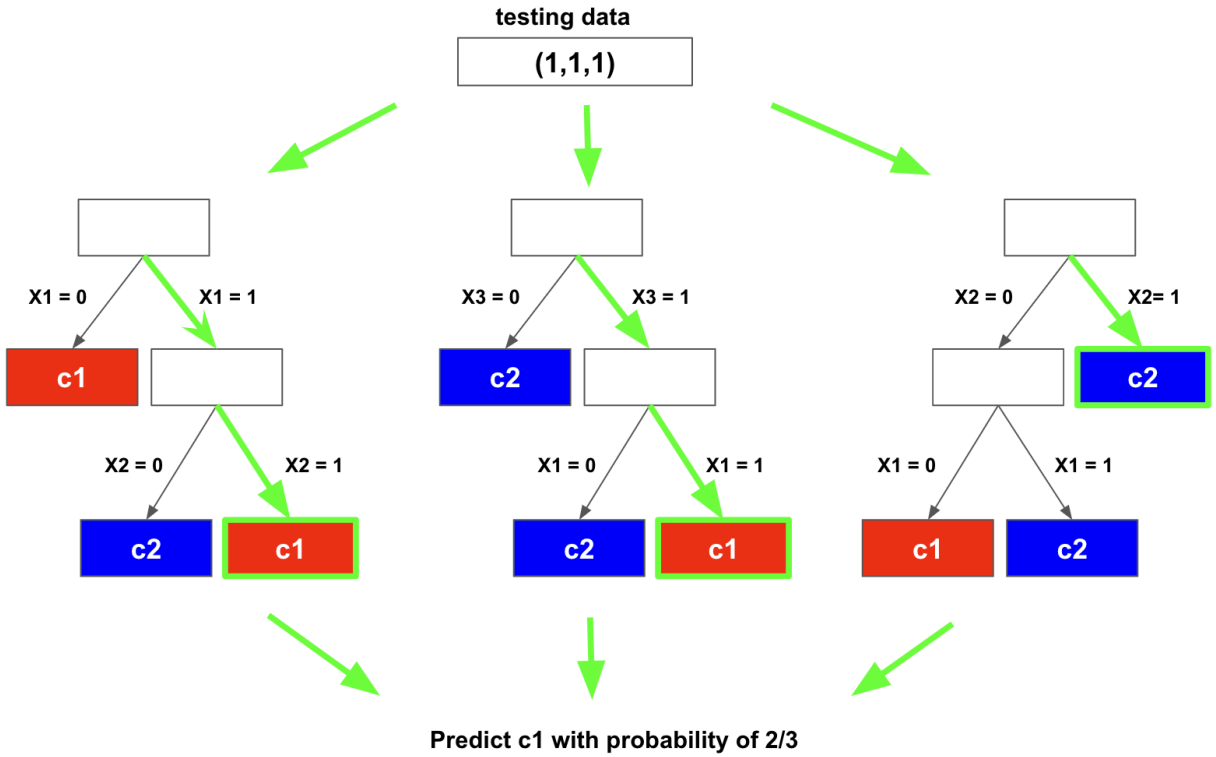
❖ Decision Tree ensemble + bagging on features: each DT uses a random set of features

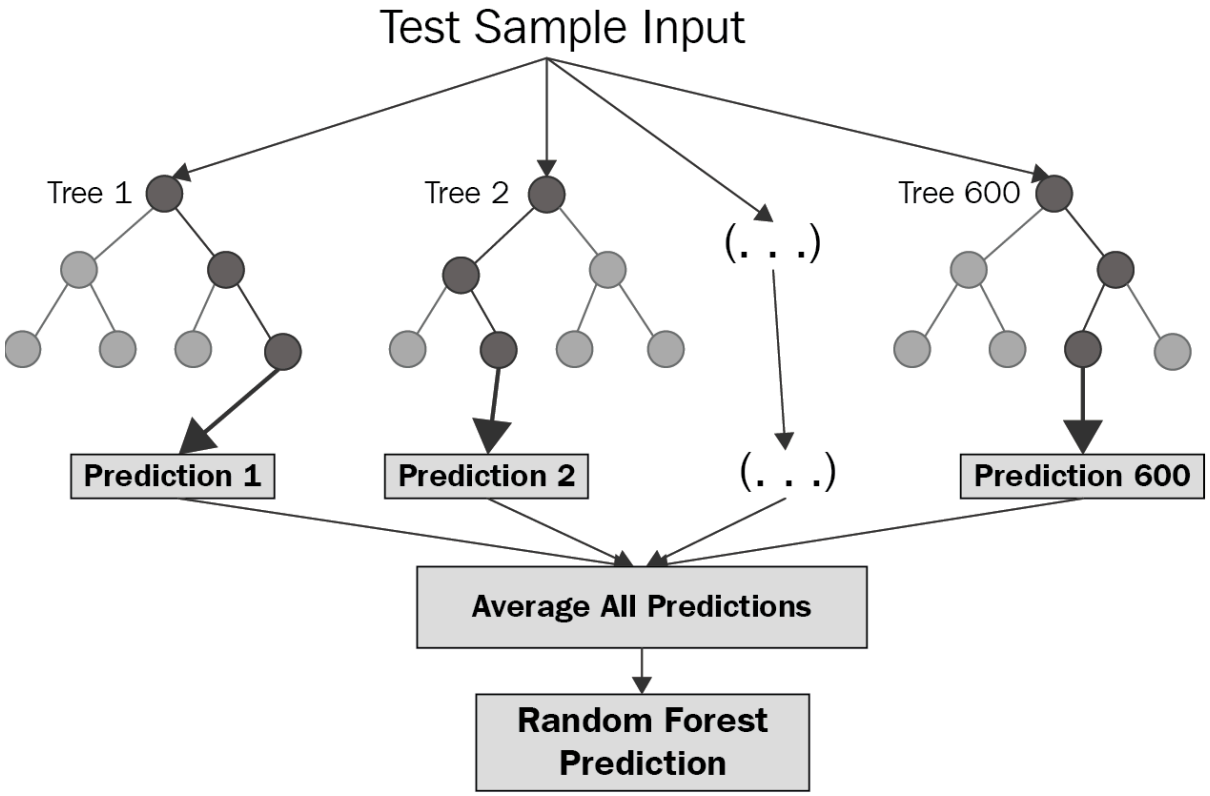
- E.g., given a total of D features, each decision tree uses $\sqrt{D}$ randomly chosen features
- Randomly chosen features make the trees uncorrelated

➢ All decision trees (usually) have the same depth (max depth parameter is set to a common max value)

❑ Prediction for a test examples is based on majority votes or averages from all DTs

Random Forest: Decision Tree ensemble + bagging on features





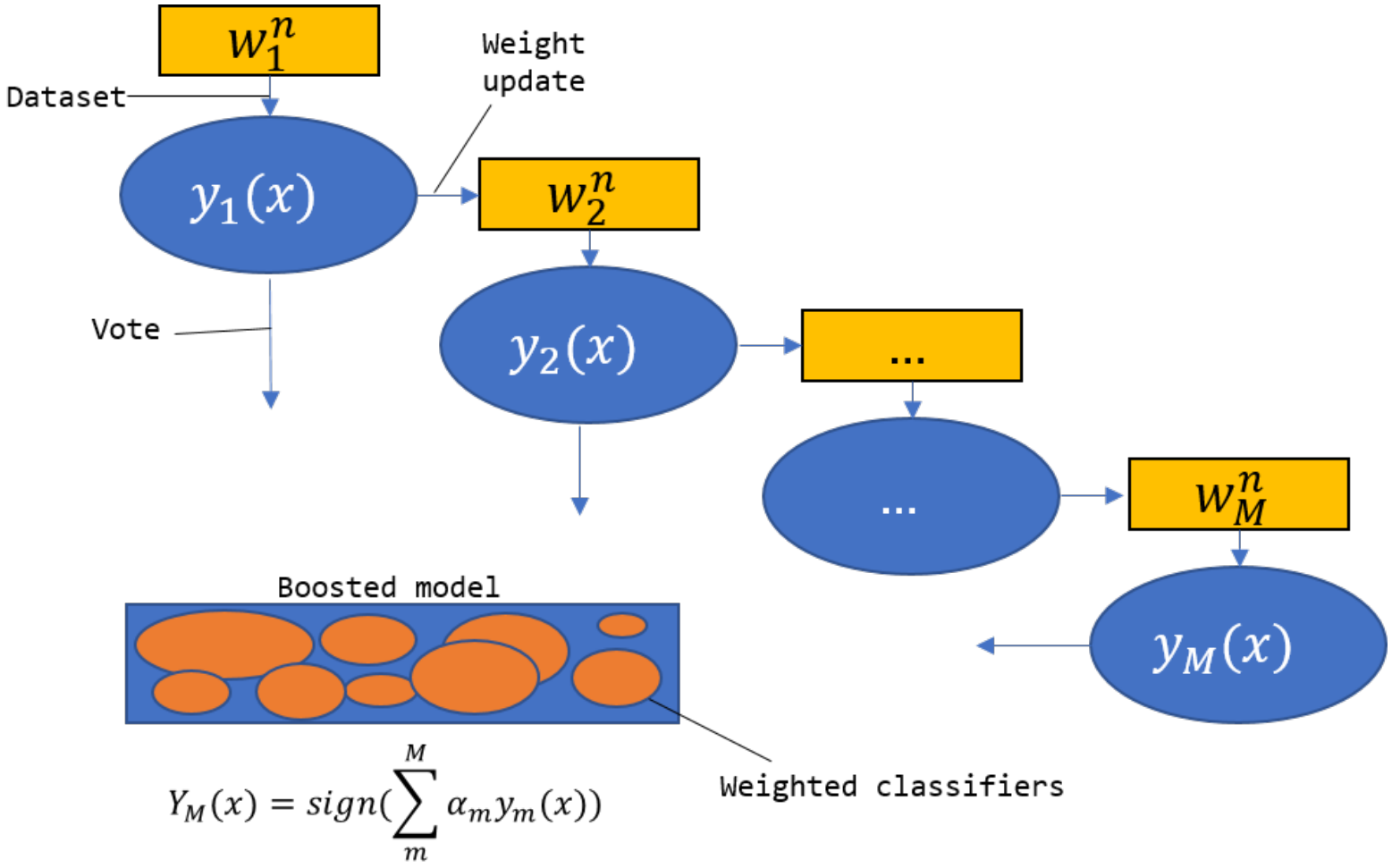
Boosting: turning a weak algorithm into an awesome one!

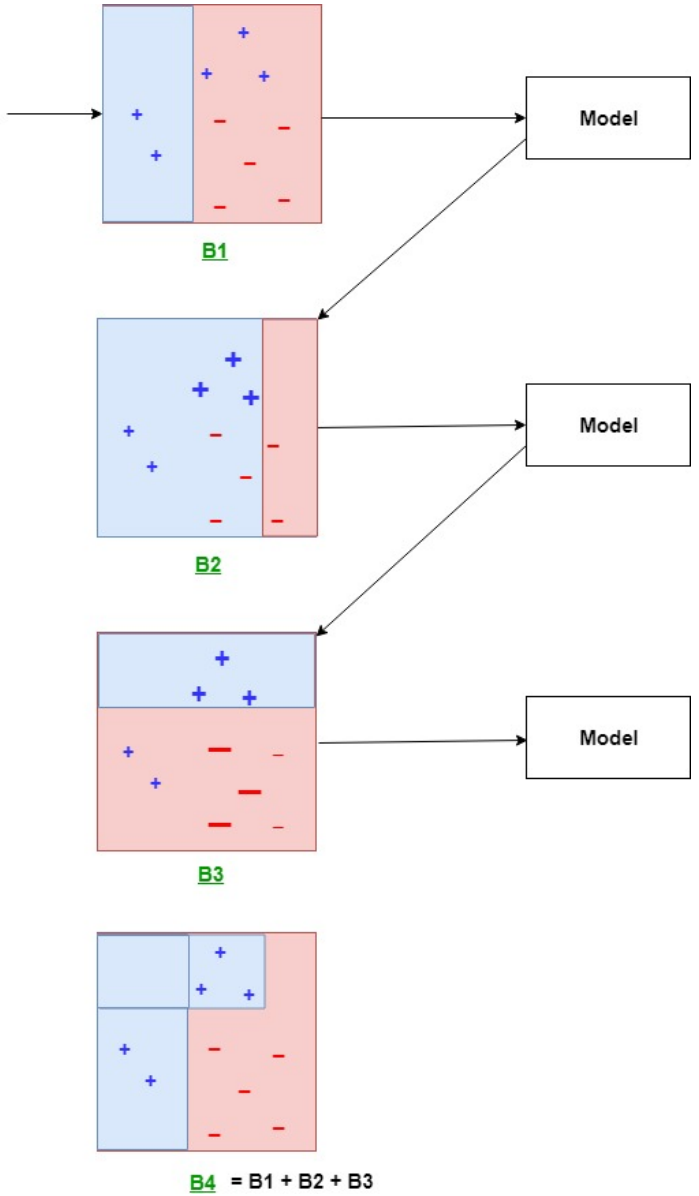
Sequential, Meta-learning algorithm:

- Take a *weak* learning algorithm (e.g., decision stump):
  - *One requirement*: Should be at least slightly *better than random*

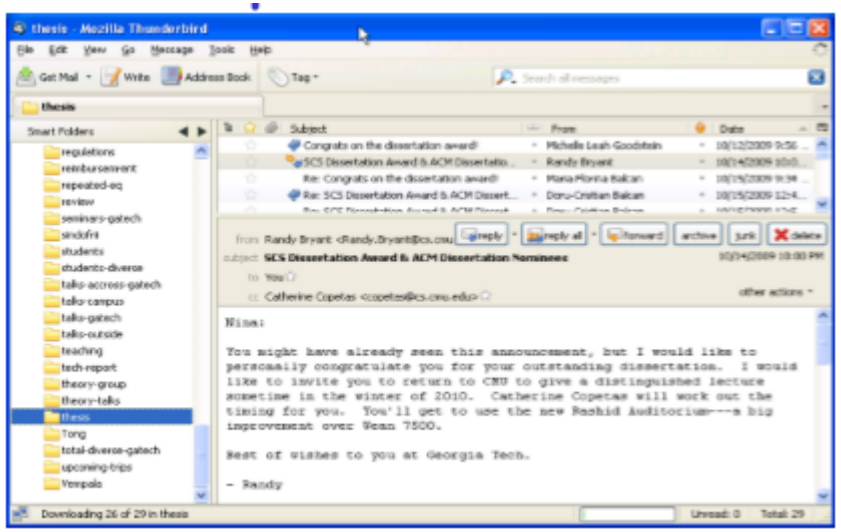
1. Train a weak model on some weighted training data
2. Compute the error of the model on each training example
3. Give higher importance to examples on which the model made mistakes
   → re-weight the examples
4. (Re)train the model using the importance weighted training examples
5. Got back to step 2, or stop based on some criterion (e.g., #iterations)
6. Combine all the trained models making each model voting on its output

Boosting: turning a weak algorithm into an awesome one!

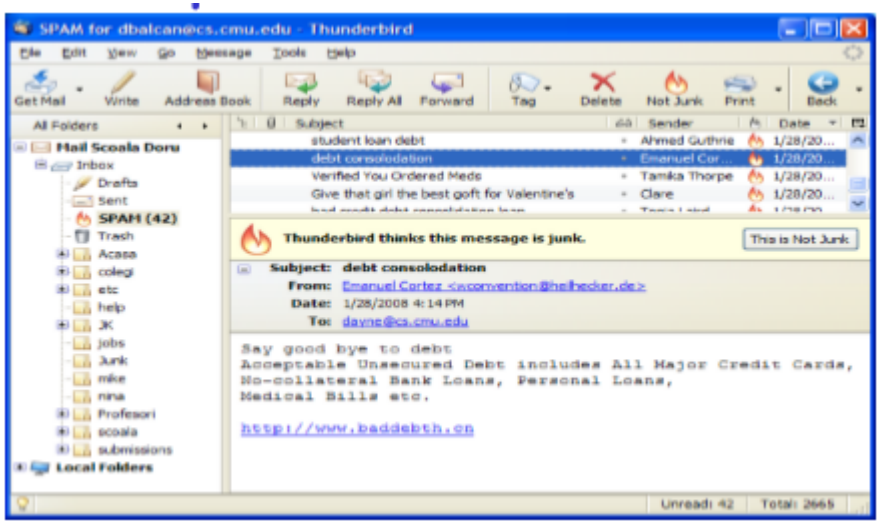




Why boost weak learners?



Not Spam



Spam

Key observation/motivation:

- ✓ Easy to find **rules of thumb** that are **often correct**.
  - E.g., If `buy now` in the message, then predict spam
  - E.g., If `say good-bye to debt` in the message, then predict spam

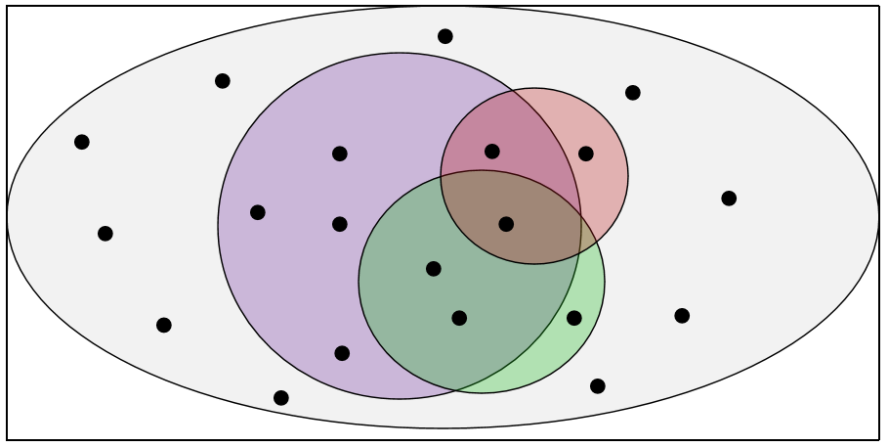
- ❖ Harder to find single rule that is always very highly accurate

Why boost weak learners?

❖ Boosting meta-procedure:

- Takes in an **algorithm for finding rules of thumb** (weak learner)
- Expected to be relatively *easy / cheap / fast*
- Produces a **highly accurate prediction**, by calling the weak learner repeatedly on cleverly chosen datasets

Emails



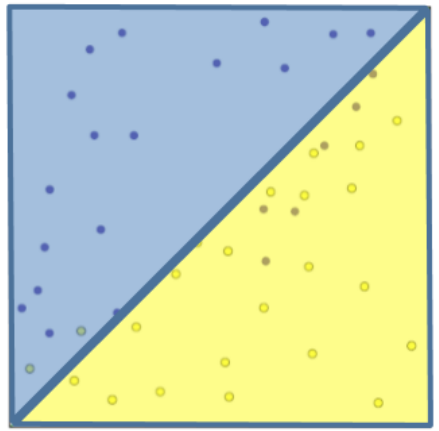
h_1
h_2
h_3
$\vdots$
h_n

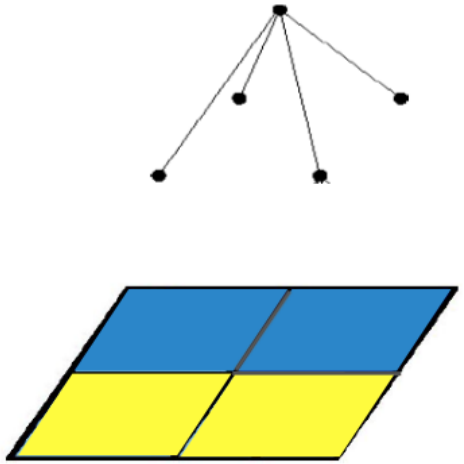
- apply weak learner to a subset of emails, obtain rule of thumb h_1
- apply to 2nd subset of emails, obtain rule of thumb h_2
- apply to 3rd subset of emails, obtain h_3
- …
- repeat n times
- combine weak rules into a single highly accurate rule

Fighting the bias-variance trade-off

Simple (a.k.a. *weak*) learners

- Decision stumps (or shallow decision trees)
- Naïve Bayes
- Logistic regression
- …





✓ Are good: don't usually overfit

✓ Are bad: can't usually solve hard / complex learning problems

Can we turn a single weak learner into a strong one? NO!

But … Boosting makes use of an **ensemble of weak learners** to produce highly accurate predictions also for hard problems

Ensemble methods and Voting

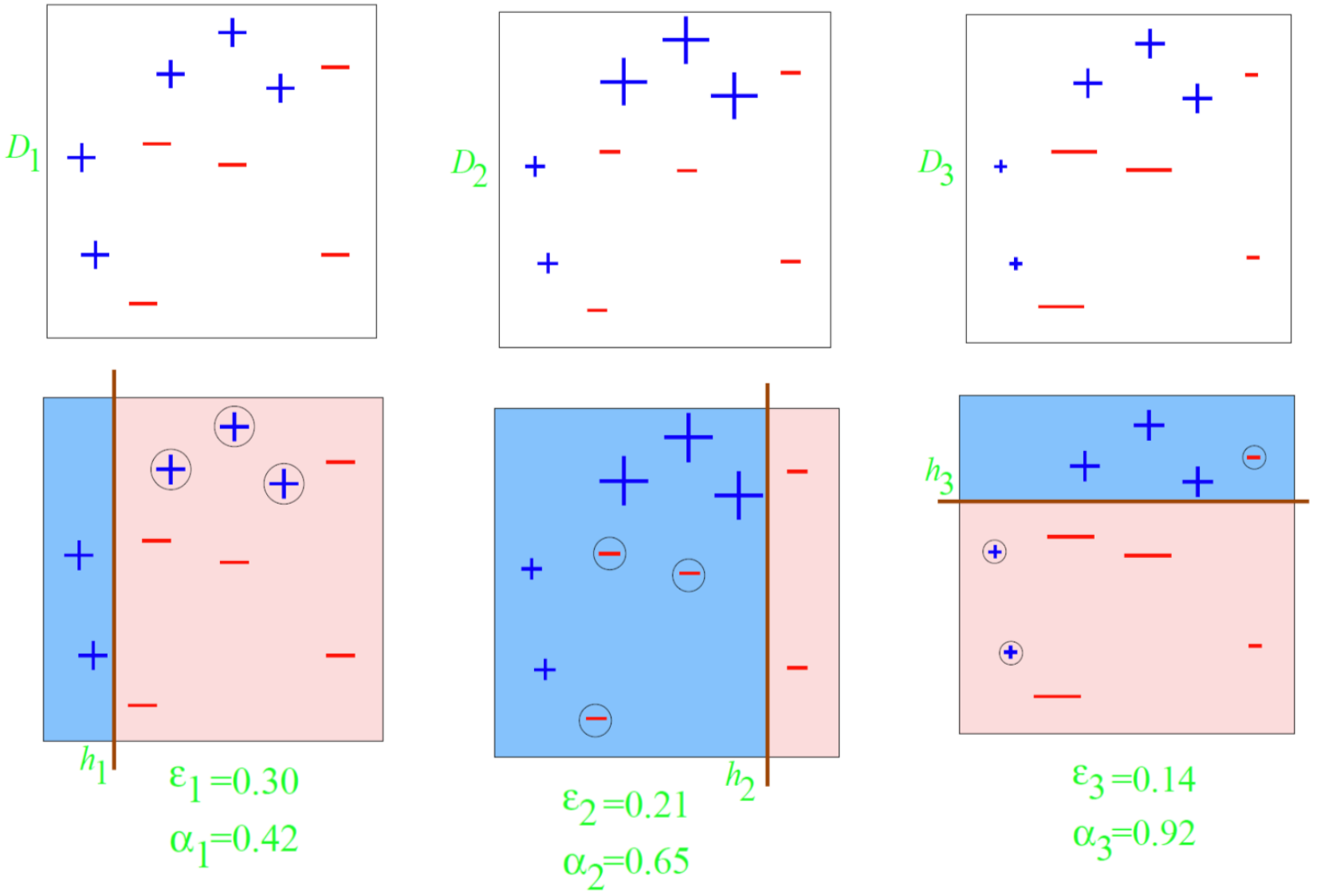
- ✓ Instead of learning a single (maybe complex) classifier learn an **ensemble of weak learners** (many weak learners) that **are good at different parts of the input space**

- ✓ Output class: Weighted ***vote*** of each classifier
  - Classifiers that are *most sure* will vote with more confidence
  - Being weak, each classifier is expected to be most sure about *a part of the input space*
  - *Combining outputs using confidence voting* will do, on average, better than using a single classifier

Questions:

1. How do we force classifiers to learn about different portions of the space?
2. How do we weight the votes of the classifiers?

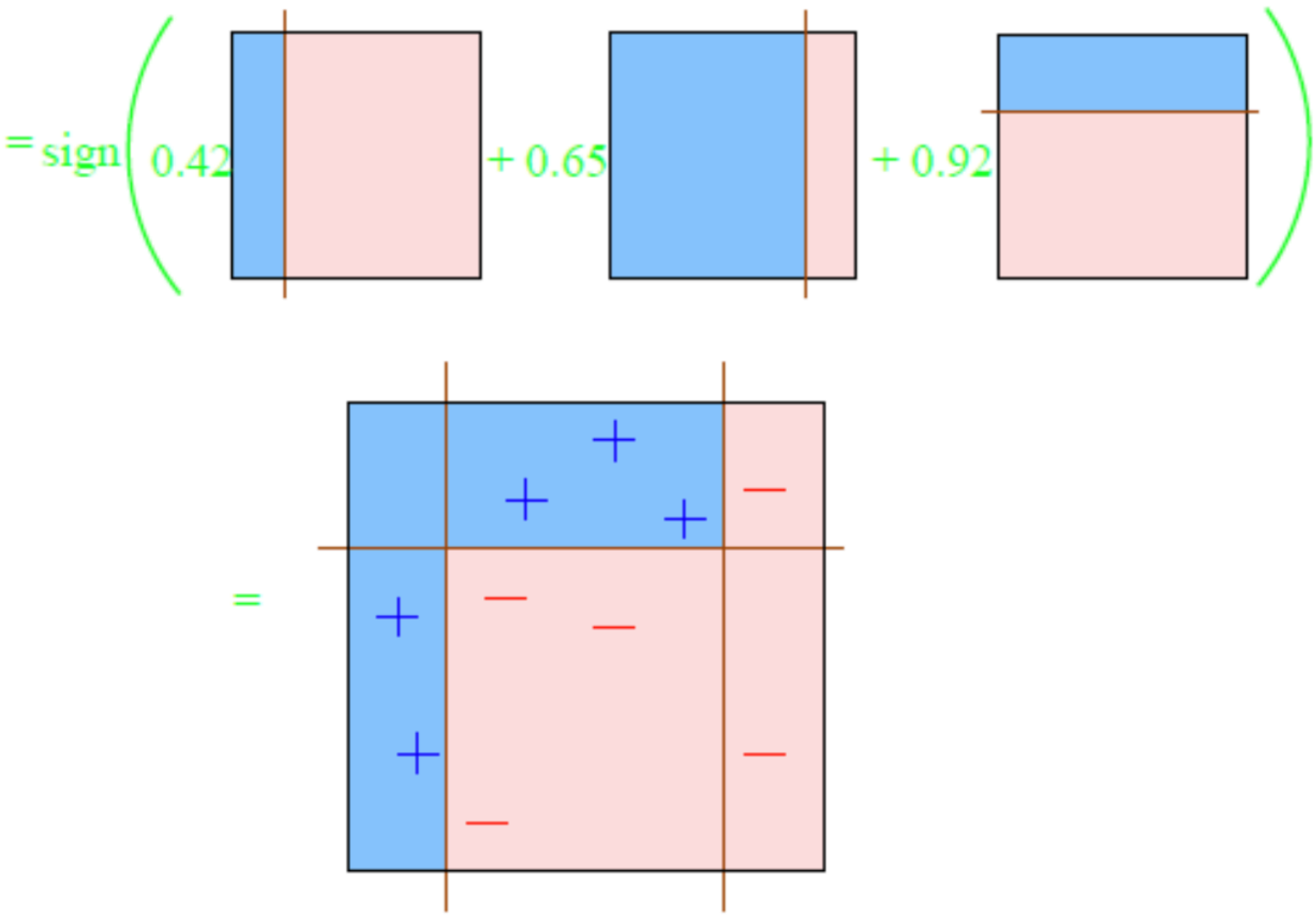
Example: decision stumps as weak learners



- Size of +,- indicates the weight of the sample
- ε_i is classification error of model h_i in D_i
- α_i is **voting weight** of h_i, based on error ε_i

Example: decision stumps

H_{final}



(Extras) Weak learning vs. Strong learning

- [Kearns & Valiant '88]: defined weak learning: being able to predict better than random guessing (error $\leq \frac{1}{2} - \gamma$) , consistently.

- Posed an open pb: "Does there exist a boosting algo that turns a weak learner into a strong learner (that can produce arbitrarily accurate hypotheses)?"

- Informally, given "weak" learning algo that can consistently find classifiers of error $\leq \frac{1}{2} - \gamma$, a boosting algo would provably construct a single classifier with error $\leq \epsilon$.

(Extras) Weak learning vs. Strong learning

Weak Learning =Strong Learning

Original Construction [Schapire '89]:

- poly-time boosting algo, exploits that we can learn a little on every distribution.

  - A modest booster obtained via calling the weak learning algorithm on 3 distributions.

$$\text{Error} = \beta < \frac{1}{2} - \gamma \rightarrow \text{error } 3\beta^2 - 2\beta^3$$

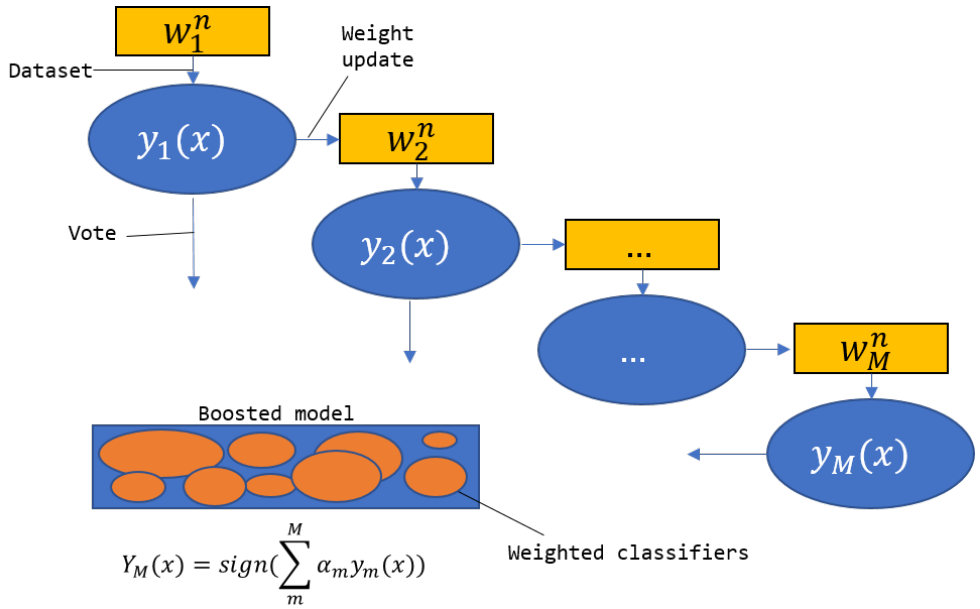
  - Then amplifies the modest boost of accuracy by running this somehow recursively.

- Cool conceptually and technically, not very practical.

Boosting (Schapire, ‘89)

Idea: given a weak learner, run it sequentially multiple times, each time on reweighted training data, then let the learned individual classifiers vote combining their models

- **On each iteration t:**
  - Compute a weight $D_t(i)$ for each training example i, based on how incorrectly it was classified in $t-1$
  - Learn a weak hypothesis h_t
  - Compute a voting weight α_t for the hypothesis



- **Final classifier:** $H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{n} \alpha_t \, h_t(\boldsymbol{x})\right)$

  (binary case)

Learning from weighted data

- **Consider a weighted dataset:**
  - $D(i)$ = weight of the i-th training example $(\boldsymbol{x}^i, \mathrm{y}^{\mathrm{i}})$
  - Interpretations:
    - i-th training example *counts* as $D(i)$ examples
    - If we would *resample* data, we would get more samples of *heavier* data points

- **In all calculation, whenever used, the $\boldsymbol{i}$-th training example counts as $\boldsymbol{D(i)}$ *examples***

(Extras) AdaBoost (Adaptive Boosting) [Freund & Schapire '95] (Godel Prize'03)

Given: a dataset $\{(\boldsymbol{x}_i, y_i)_{i=1}^m\}$, where $\boldsymbol{x}_i \in X,\ y_i \in Y = \{-1, +1\}$, a weak learner

Decision stump, NB

Initialize: $D_{1(i)} = \frac{1}{m}$ Initially equal weights

For $t = 1, \cdots, T$:

1. Train weak learner using distribution D_t
2. Get weak classifier h_t: $X \to \mathbb{R}$
3. Choose $\alpha_t \in \mathbb{R}$ How?
4. Update: $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$

Decrease weight if correct on example i: $y_i h_t(\boldsymbol{x}_i) = +1 > 0$

Increase weight if wrong on example i: $y_i h_t(\boldsymbol{x}_i) = -1 < 0$

where Z_t is a normalization factor: $Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$

Weight for all examples must sum to 1

(Extras) AdaBoost (Adaptive Boosting) [Freund & Schapire '95] (Godel Prize'03)

Given: a dataset $\{(\boldsymbol{x}_i, y_i)_{i=1}^m\}$, where $\boldsymbol{x}_i \in X,\ y_i \in Y = \{-1, +1\}$, a weak learner

Decision stump, NB

Initialize: $D_1(i) = \frac{1}{m}$ Initially equal weights

For $t = 1, \cdots, T$:

1. Train weak learner using distribution D_t
2. Get weak classifier h_t: $X \to \mathbb{R}$
3. Choose $\alpha_t \in \mathbb{R}$ How?
4. Update: $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$

Increase weight if wrong on example i: $y_i h_t(\boldsymbol{x}_i) = -1 < 0$

Decrease weight if correct on example i: $y_i h_t(\boldsymbol{x}_i) = +1 > 0$

Final classifier: (binary case) $H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t\, h_t(\boldsymbol{x})\right)$

(Extras) How to choose α_t?

- Weight update rule: $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$
- **Voting weight:** $\alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$ [Freund & Schapire '95]

ε_t = Probability of error in the weighted training set t → Weighted training error

$$\varepsilon_t = P_{i\sim D_{t(i)}}[h_t(\boldsymbol{x}_i) \neq y_i] = \sum_{i=1}^{m} D_t(i)\underbrace{\delta(h_t(\boldsymbol{x}_i) \neq y_i)}$$

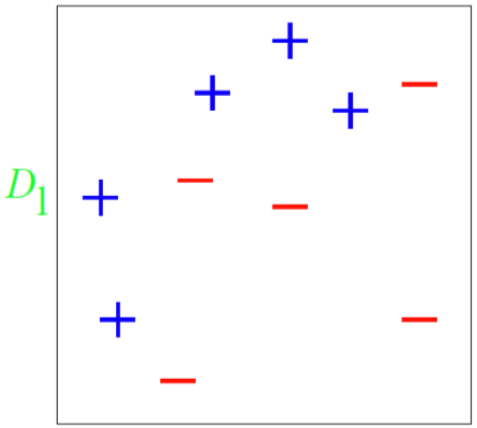
Does h_t get i-th point wrong? (1/0)

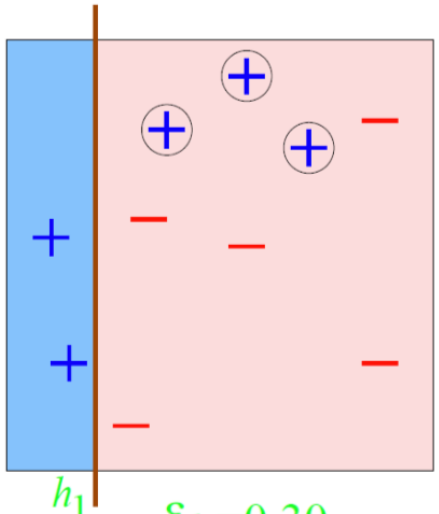
$\varepsilon_t = 0$ if h_t perfectly classifies all weighted data points → $\alpha_t = \infty$

$\varepsilon_t = 1$ if h_t perfectly wrong → $-h_t$ perfectly right → $\alpha_t = -\infty$

$\varepsilon_t = 0.5 \rightarrow \alpha_t = 0$

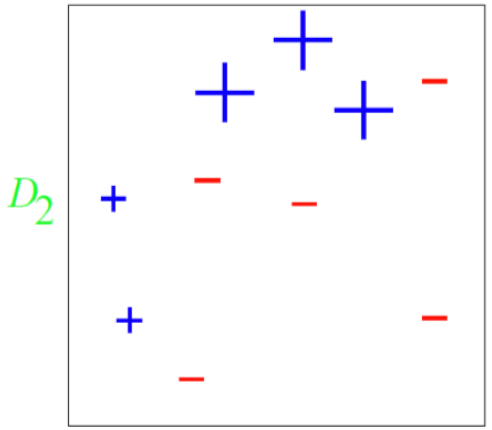
Example: decision stumps

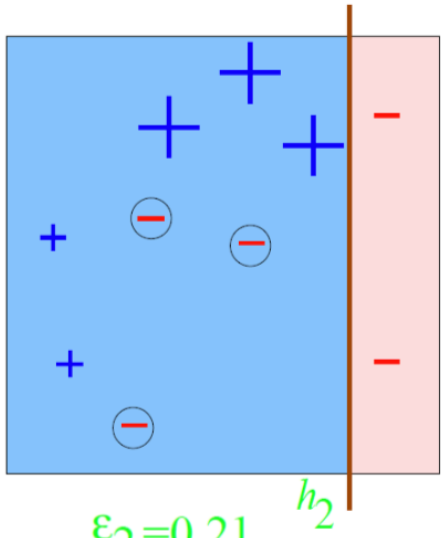




$\varepsilon_1 = 0.30$

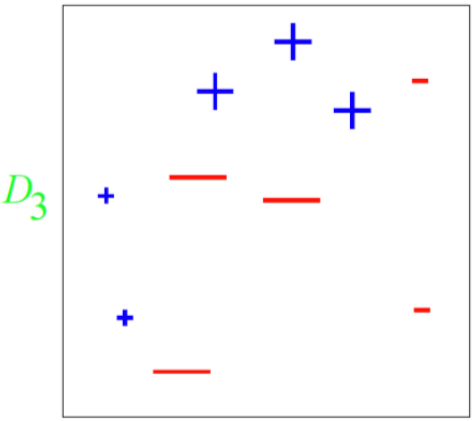
$\alpha_1 = 0.42$

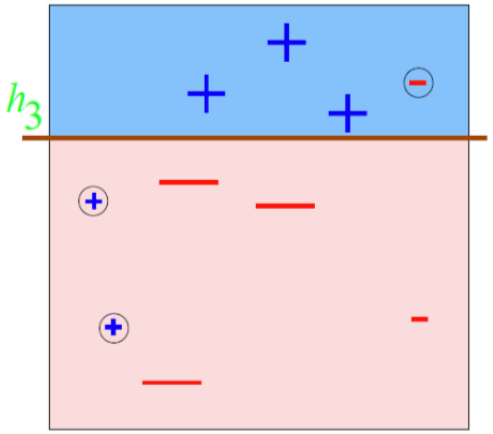




$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$



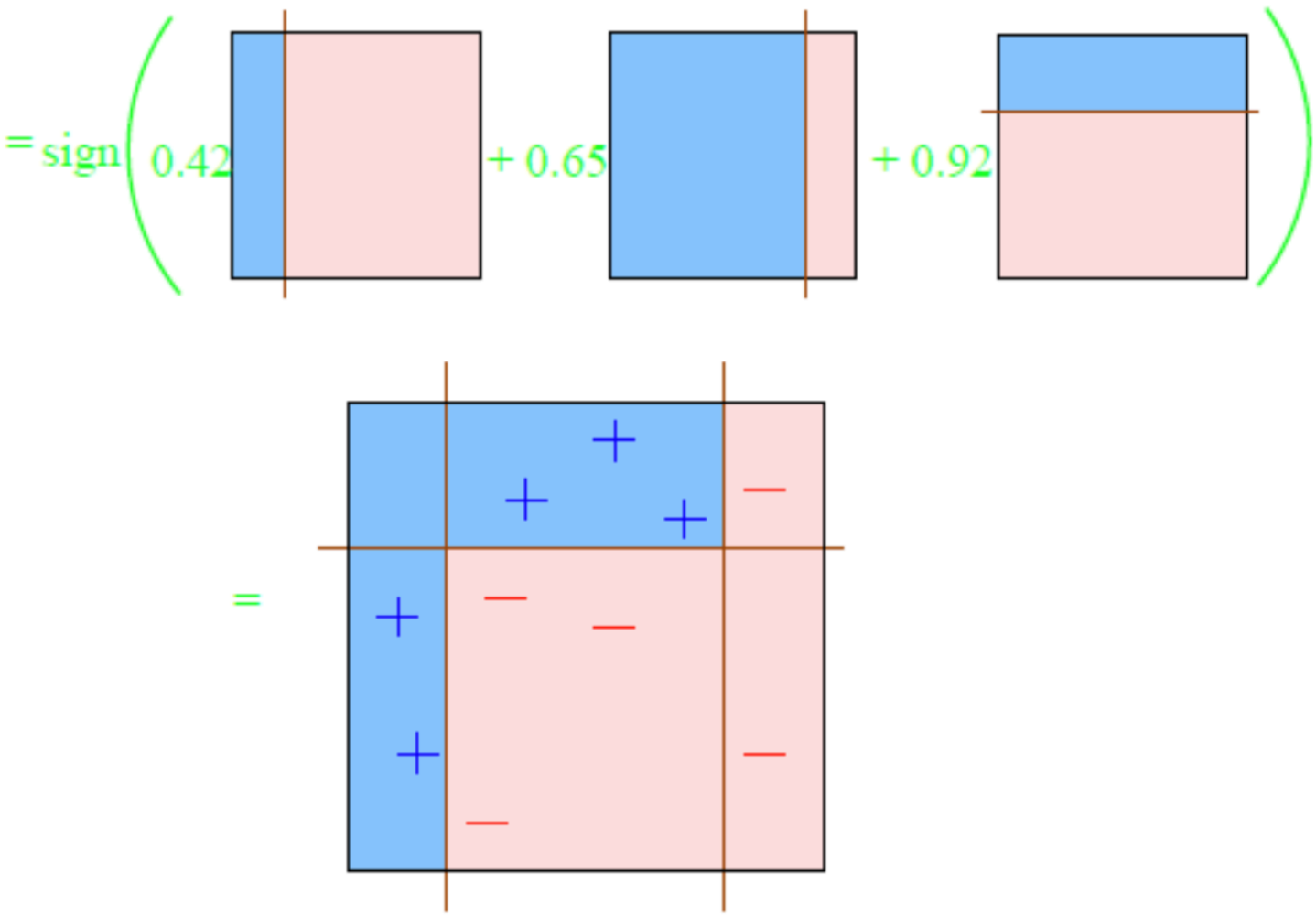


$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

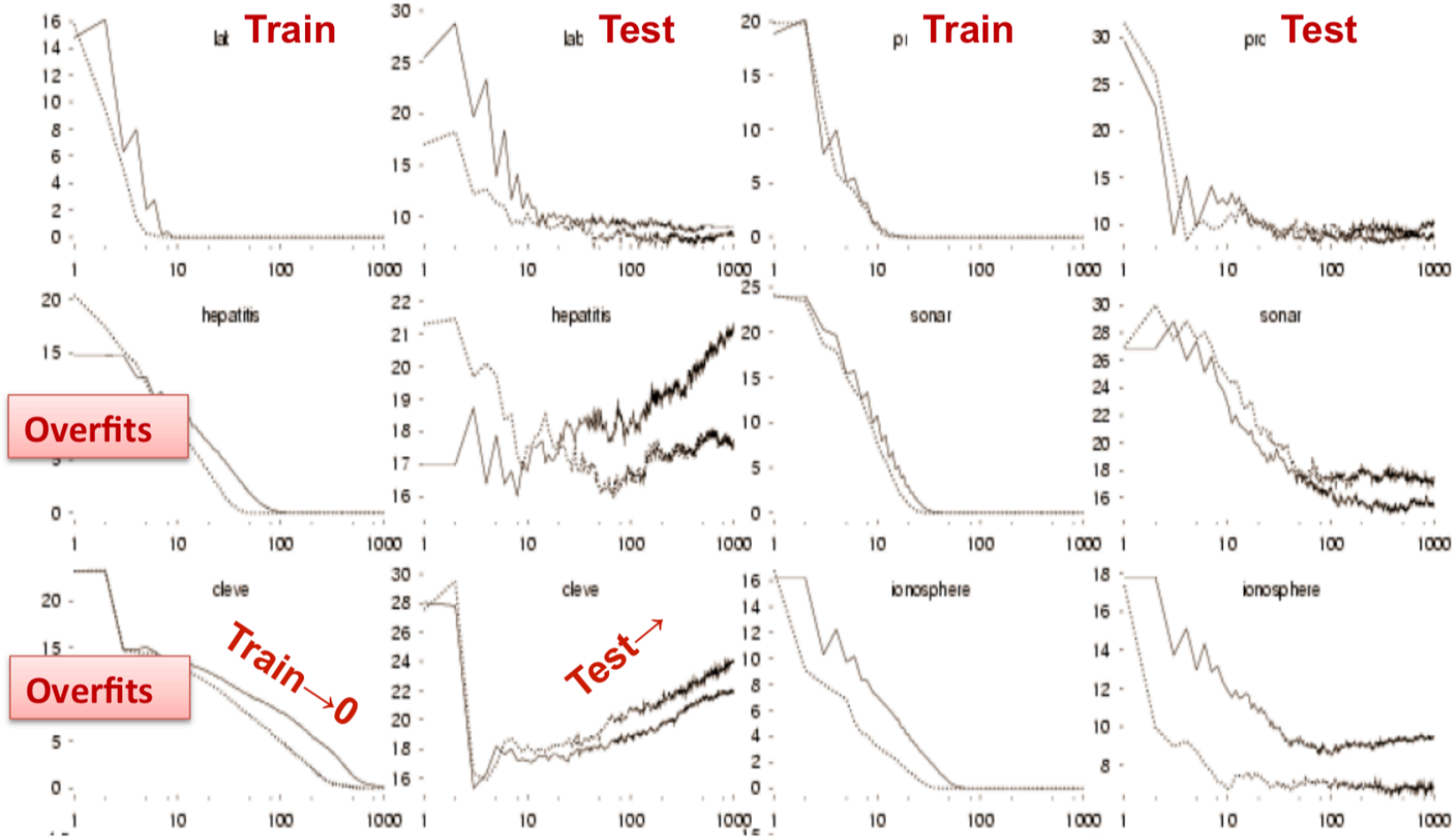
Example: decision stumps

H_{final}



Boosting can overfit

AdaBoost and AdaBoost.MH on Train (left) and Test (right) data from Irvine repository. [Schapire and Singer, ML 1999]



Boosting can overfit if margin between classes is too small (high label noise) or weak learners are too complex.

Summary

- Ensemble methods, various ways of making ensembles
- Bagging: creating multiple datasets by bootstrapping, averaging the classifiers
- Boosting: combine weak classifiers to obtain a very strong classifier
- Sequential meta-algorithm, adaptively changing the datasets
- Weak classifier: slightly better than random on training data
- Theoretical results for strong classifier resulting from Boosting (zero error on training)
- AdaBoost algorithm: adapting weights, voting weights, exponential error decrease
- Popular implementations of Boosting: Decision stumps as weak classifiers, very easy to implement, very effective
- Robust to overfit in general, but can still overfit if class margins are tight