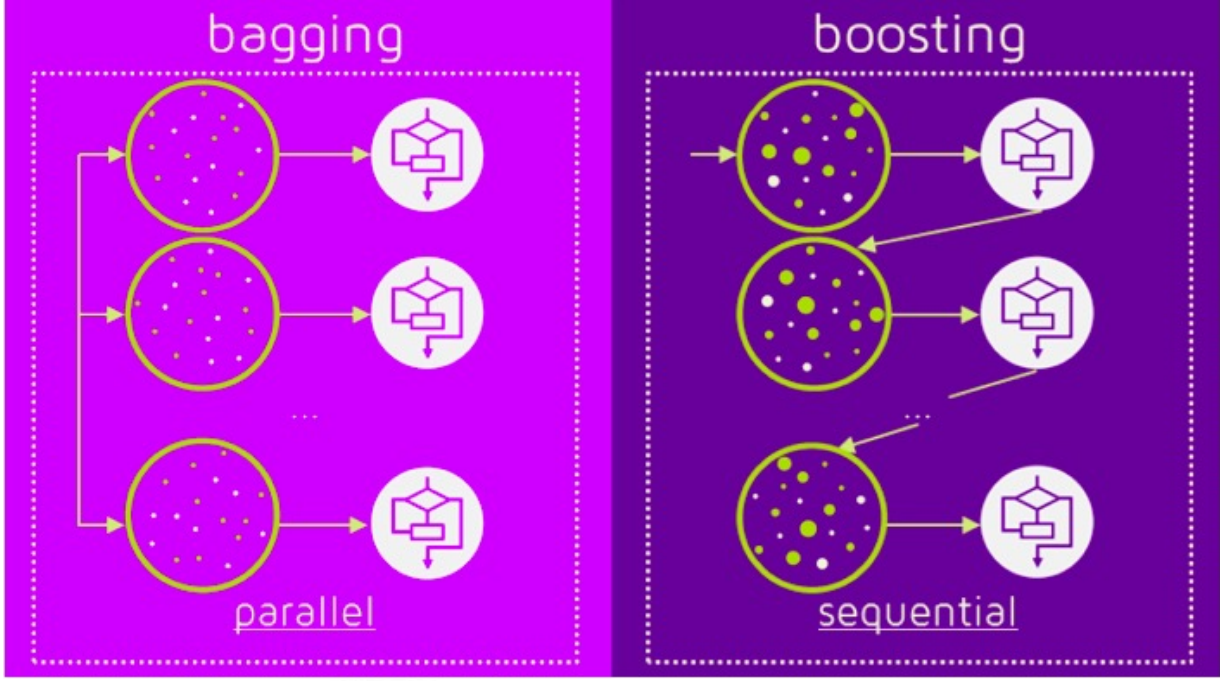
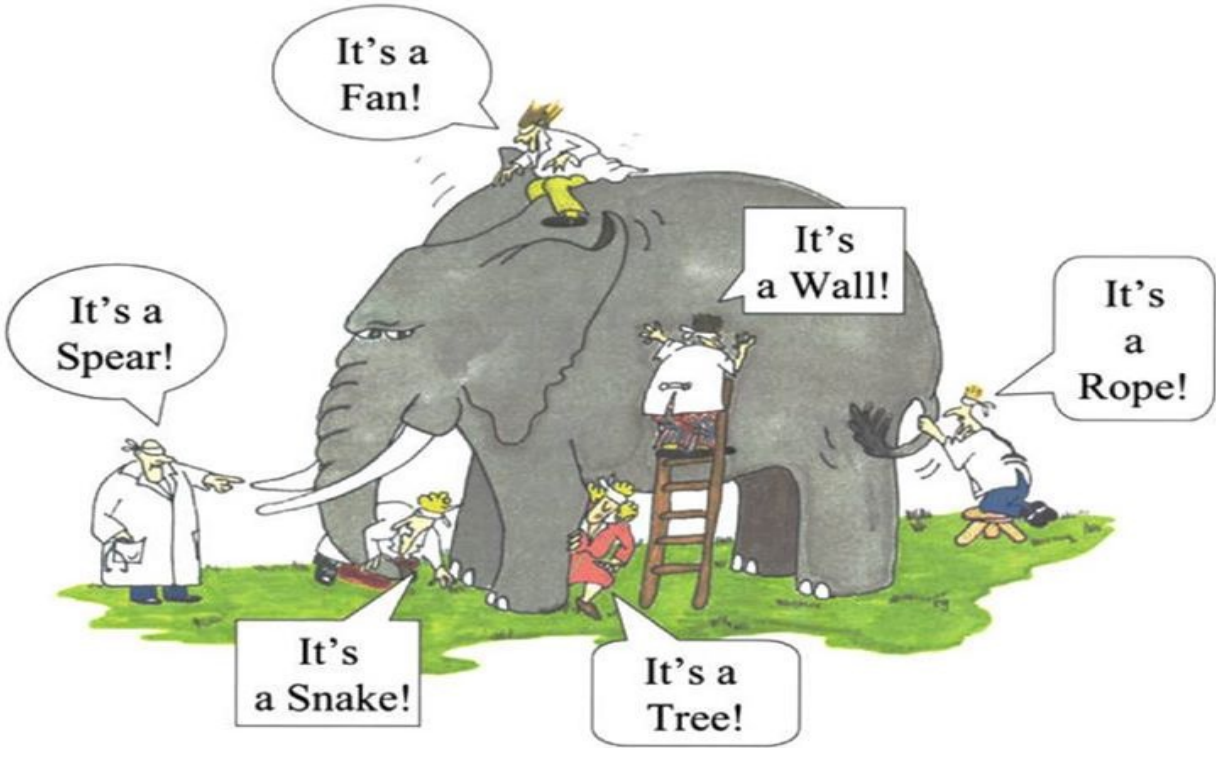
Ensemble Methods





Roadmap

- Basic Concepts
  - Collective Wisdom
  - Ensemble Classifiers
- Ensemble Methods
  - Bagging
  - Boosting
- Take-home messages

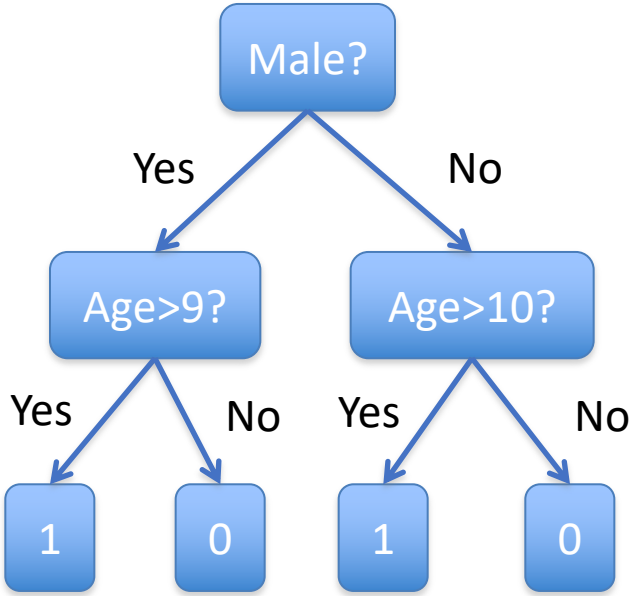
Recall that

Supervised Learning

- **Goal:** learn predictor h(x)
  - High accuracy (low error)
  - Using training data $\{(x_1,y_1),\ldots,(x_n,y_n)\}$

Supervised Learning

A specific classifier



Person	Age	Male?	Height > 55”
Alice	14	0	1
Bob	10	1	1
Carol	13	0	1
Dave	8	1	0
Erin	11	0	0
Frank	9	1	1
Gena	8	0	0

$$x = \begin{bmatrix} age \\ 1_{[gender=male]} \end{bmatrix} \qquad y = \begin{cases} 1 & height > 55" \\ 0 & height \leq 55" \end{cases}$$

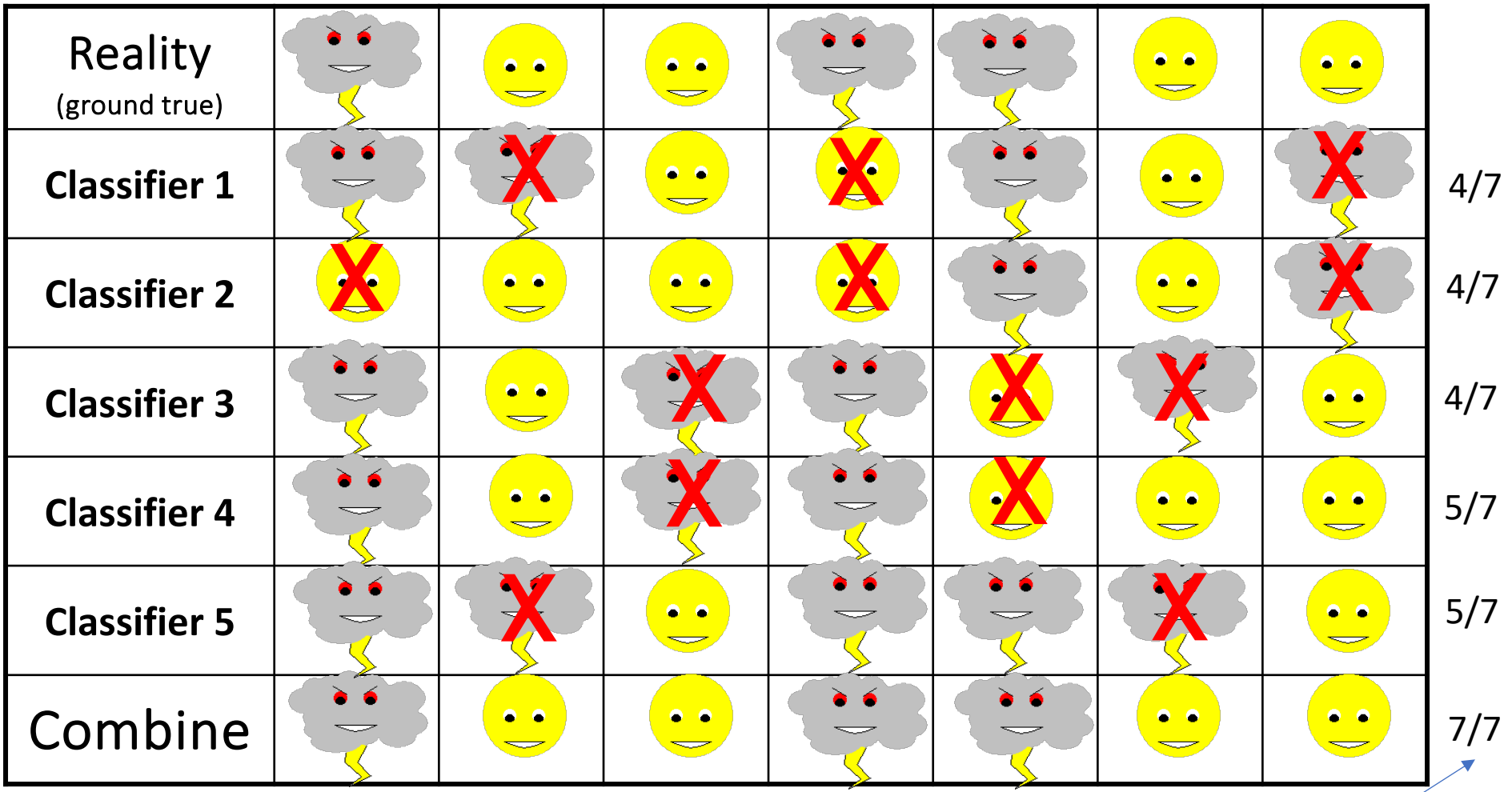
Every classifier behaves differently

- Performance
  - None of the classifiers is perfect
  - Complementary Effect
    - Examples which are not correctly classified by one classifier may be correctly classified by the other classifiers
- Potential Improvements?
  - Utilize the complementary property

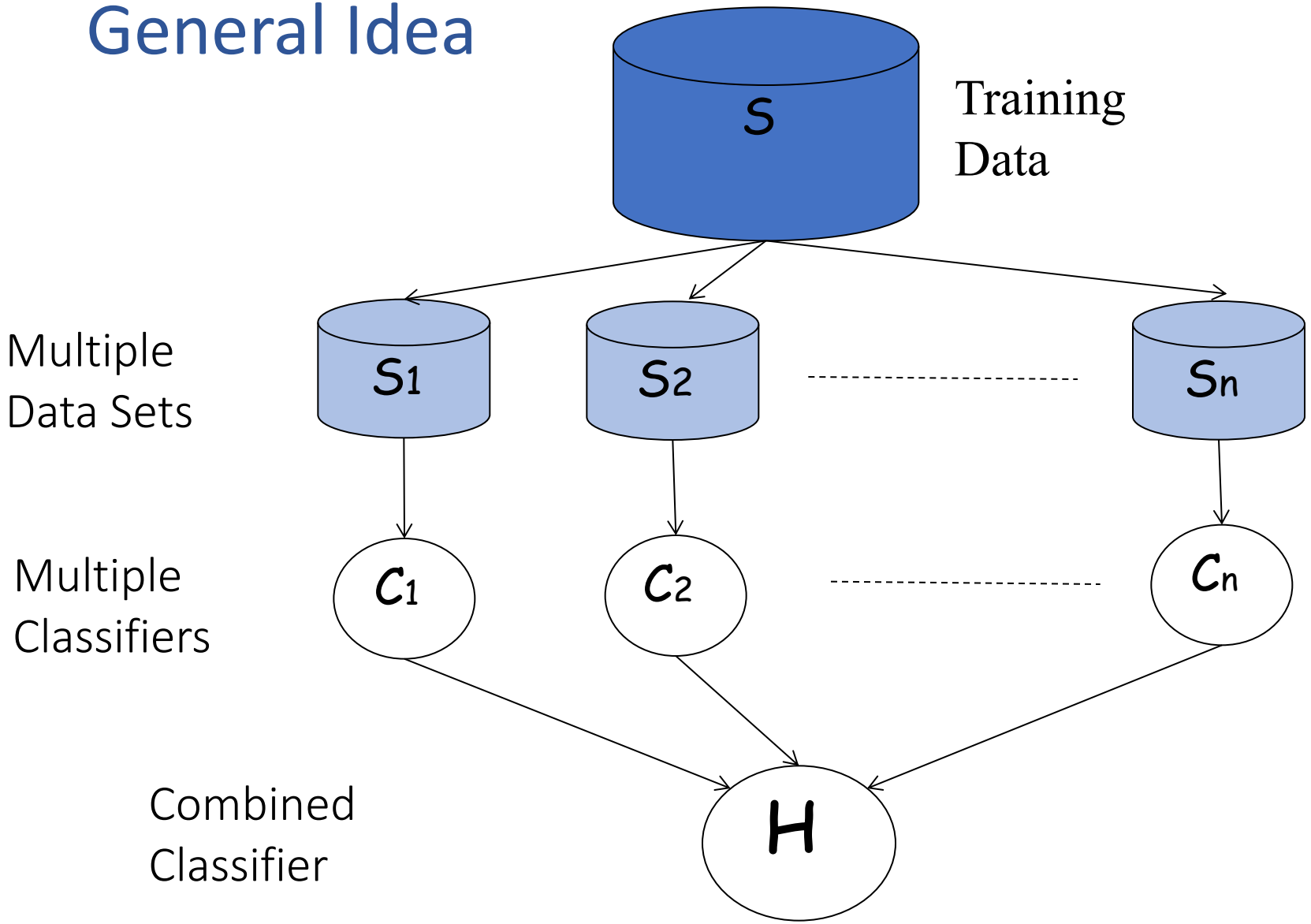
Ensembles of Classifiers
Achieving Collective Wisdom

- Idea
  - Combine the classifiers to improve the performance
- Ensembles of Classifiers
  - Combine the classification results from different classifiers to produce the final output
    - Unweighted voting
    - Weighted voting

Example: Weather Forecast



General Idea



Building Ensemble Classifiers

- Basic idea:
  Build different "experts", and let them vote
- Advantages:
  Improve predictive performance
  Other types of classifiers can be directly included
  Easy to implement
  No too much *parameter tuning*
- Disadvantages:
  The combined classifier is not so transparent (black box; low interpretability)
  Not a compact representation

Why do they work?

- Suppose there are 25 base classifiers
- Each classifier has error rate $\varepsilon = 0.35$
- Assume independence among classifiers
- Probability that the ensemble classifier makes a wrong prediction:

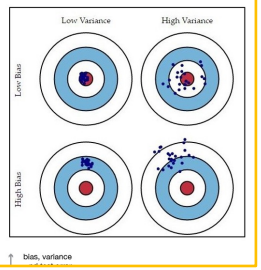
$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{\mathrm{i}} (1-\varepsilon)^{25-i} = 0.06$$

Ensemble Methods

- Ensemble methods that minimize variance
  - Bagging
  - Random Forests
- Ensemble methods that minimize bias
  - Boosting
  - Ensemble Selection

Bias and variance and their trade-off

- The bias is the error which results from missing a target.
- For example, if an estimated mean is 3, but the actual population value is 3.5, then the bias value is 0.5.
- The variance is the error which results from random noise.
- When the variance of a model is high, this model is considered unstable.



Concept and Assumption of Ensemble Methods

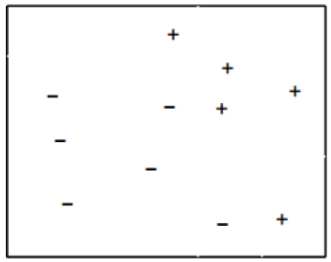
- Instead of training different models on same data, train same model multiple times on different data sets, and "combine" these "different" models
- We can use some simple/weak model as the base model
- How do we get multiple training data sets (in practice, we only have one data set, not true distribution, at training time)?

Bagging

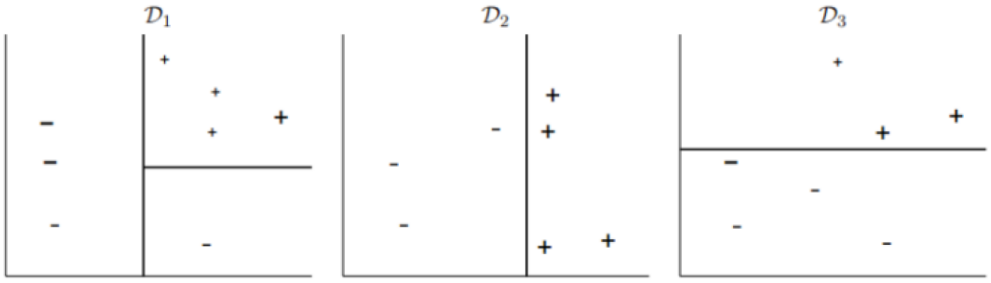
- Bagging stands for Bootstrap Aggregation
- Takes original data set *D* with *N* training examples
- Creates M copies $\{\tilde{D}_m\}_{m=1}^{M}$
  - Each $\tilde{D}_m$ is generated from *D* by sampling with replacement
  - Each data set $\tilde{D}_m$ has the same number of examples as in data set *D*
  - These data sets are reasonably different from each other
- Train models $h_1, \ldots, h_M$ using $\tilde{D}_1, \ldots, \tilde{D}_M$, respectively
- Use an averaged model $h = \frac{1}{M} \sum_{m=1}^{M} h_m$ as the final model
- Useful for models with high variance and noisy data

Bagging: An illustration

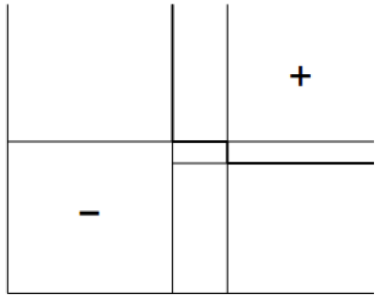
Original data:



3 models learned using three data sets:



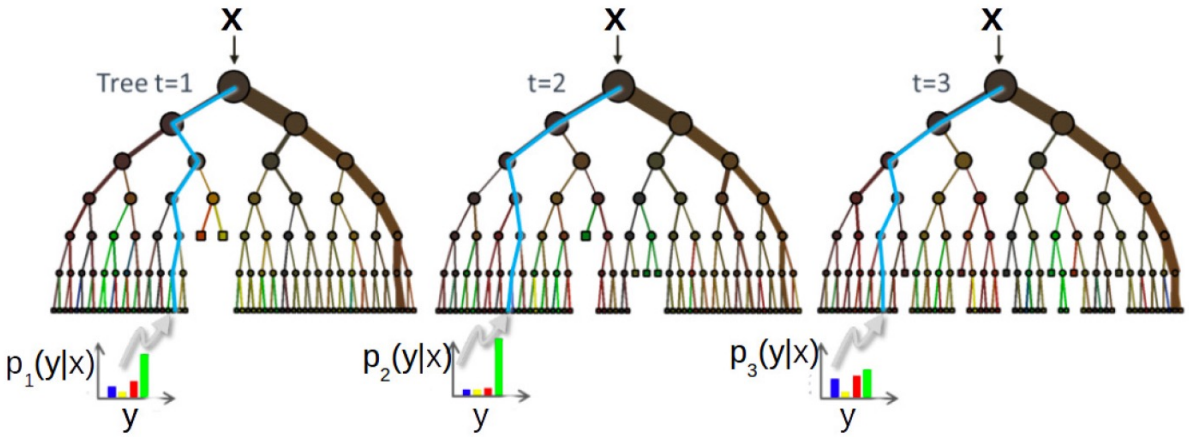
averaged model:



When does bagging works?

- Learning algorithm is unstable: if small changes to the training set cause large changes in the learned classifier.
- If the learning algorithm is unstable, then Bagging almost always improves performance
- Typically works for simple models like decision tree, regression tree, linear regression, SVMs, etc.

Random Forest (RF)



- An ensemble of decision tree (DT) classifiers
- Uses bagging on features (each DT will use a random set of features)
  - Given a total of D features, each DT uses $\sqrt{D}$ randomly chosen features
  - Randomly chosen features make the different trees uncorrelated
- All DTs usually have the same depth
- Each DT will split the training data differently at the leaves
- Prediction for a test example votes on/averages predictions from all the DTs

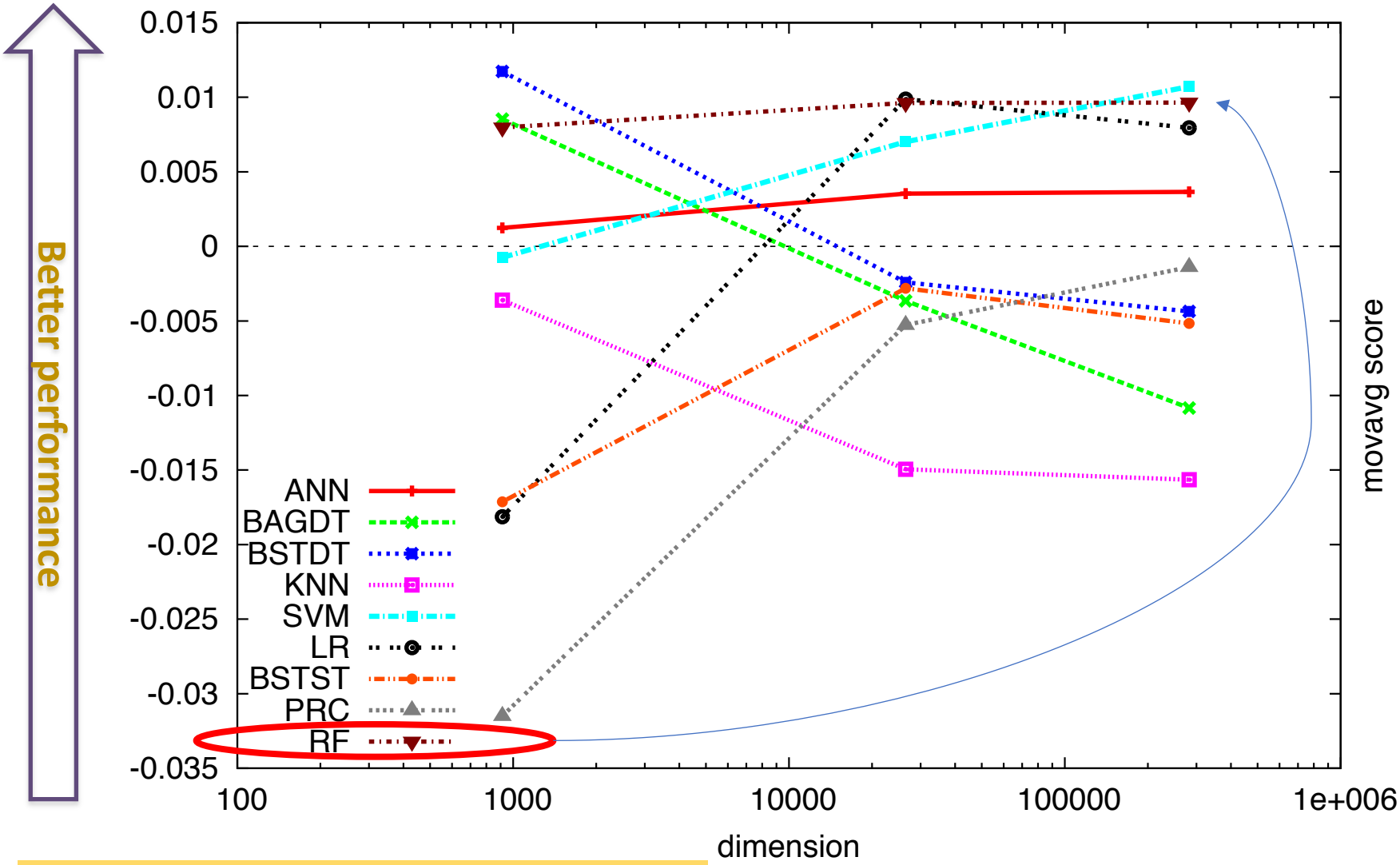
Random Forests

- **Goal:** reduce variance
  - Bagging can only do so much
  - Resampling training data asymptotes

- **Random Forests:** sample data & features as well!
  - Sample S'
  - Train DT
    - At each node, sample features
  - Average predictions

Further de-correlates trees

"Random Forests – Random Features" [Leo Breiman, 1997]
https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf



Average performance over many datasets
Random Forests perform the best!

"An Empirical Evaluation of Supervised Learning in High Dimensions"
Caruana, Karampatziakis & Yessenalina, ICML 2008

Let's focus on yet another idea -

Boosting

Basic idea

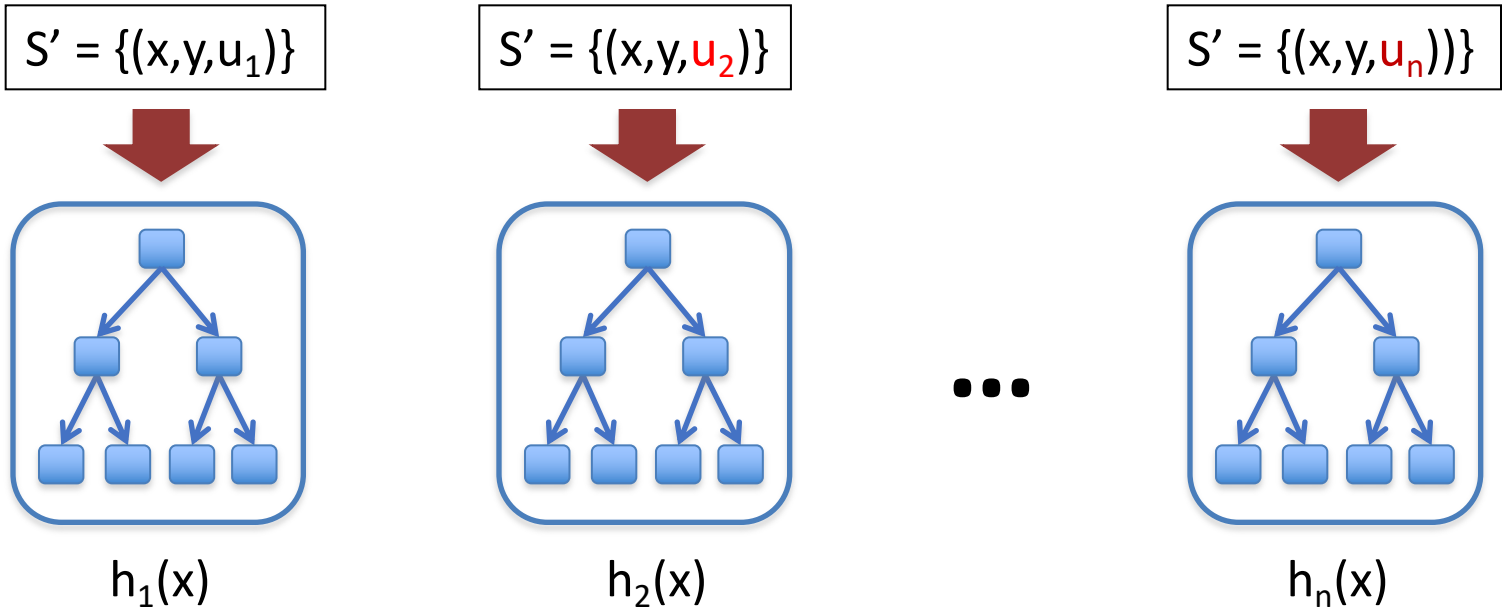
- Take a weak learning algorithm
- Only requirement: Should be slightly better than random
- Turn it into an awesome one by making it focus on difficult cases

Most boosting algorithms follow these steps:

1. Train a weak model on some 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
4. Re-train the model using "importance weighted" training examples
5. Go back to step 2

Boosting (AdaBoost)

$$h(x) = a_1h_1(x) + a_2h_2(x) + \ldots + a_nh_n(x)$$



u – weighting on data points
a – weight of linear combination

Stop when validation performance plateaus

https://towardsdatascience.com/a-comprehensive-mathematical-approach-to-understand-adaboost-f185104edced

AdaBoost (Adaptive Boosting): Algorithm

- Given: Training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ with $y_n \in \{-1, +1\}$, $\forall n$
- Initialize weight of each example $(\boldsymbol{x}_n, y_n)$: $D_1(n) = 1/N$, $\forall n$
- For round $t = 1 : T$
  - Learn a weak $h_t(\boldsymbol{x}) \rightarrow \{-1, +1\}$ using training data weighted as per D_t
  - Compute the weighted fraction of errors of h_t on this training data

$$\epsilon_t = \sum_{n=1}^{N} D_t(n) \mathbb{1}[h_t(\boldsymbol{x}_n) \neq y_n]$$

  - Set "importance" of h_t: $\alpha_t = \frac{1}{2} \log(\frac{1-\epsilon_t}{\epsilon_t})$(gets larger as ϵ_t gets smaller)
  - Update the weight of each example

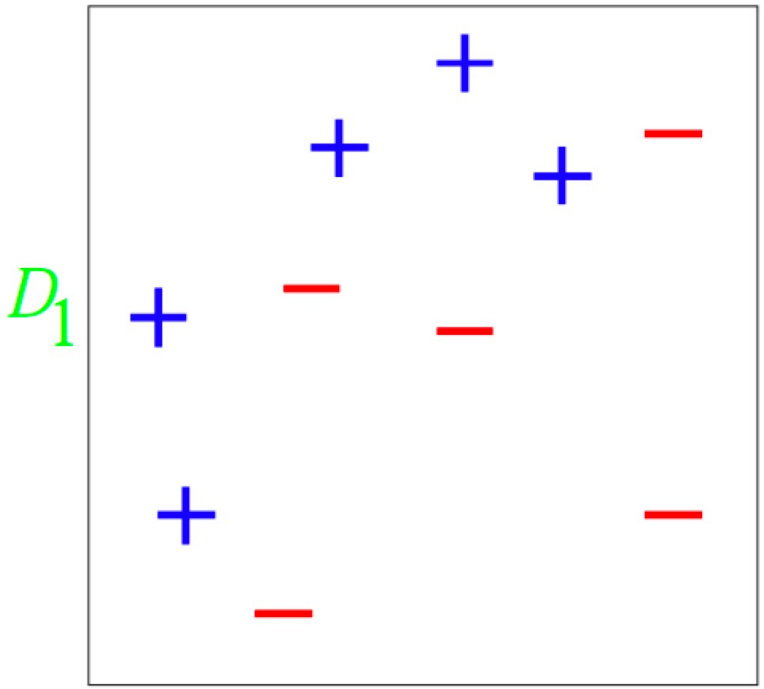
$$\begin{aligned} D_{t+1}(n) &\propto \begin{cases} D_t(n) \times \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) = y_n \quad \text{(correct prediction: decrease weight)} \\ D_t(n) \times \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) \neq y_n \quad \text{(incorrect prediction: increase weight)} \end{cases} \\ &= D_t(n) \exp(-\alpha_t y_n h_t(\boldsymbol{x}_n)) \end{aligned}$$

  - Normalize D_{t+1} so that it sums to 1: $D_{t+1}(n) = \frac{D_{t+1}(n)}{\sum_{m=1}^{N} D_{t+1}(m)}$
- Output the "boosted" final hypothesis $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$

AdaBoost: An Example

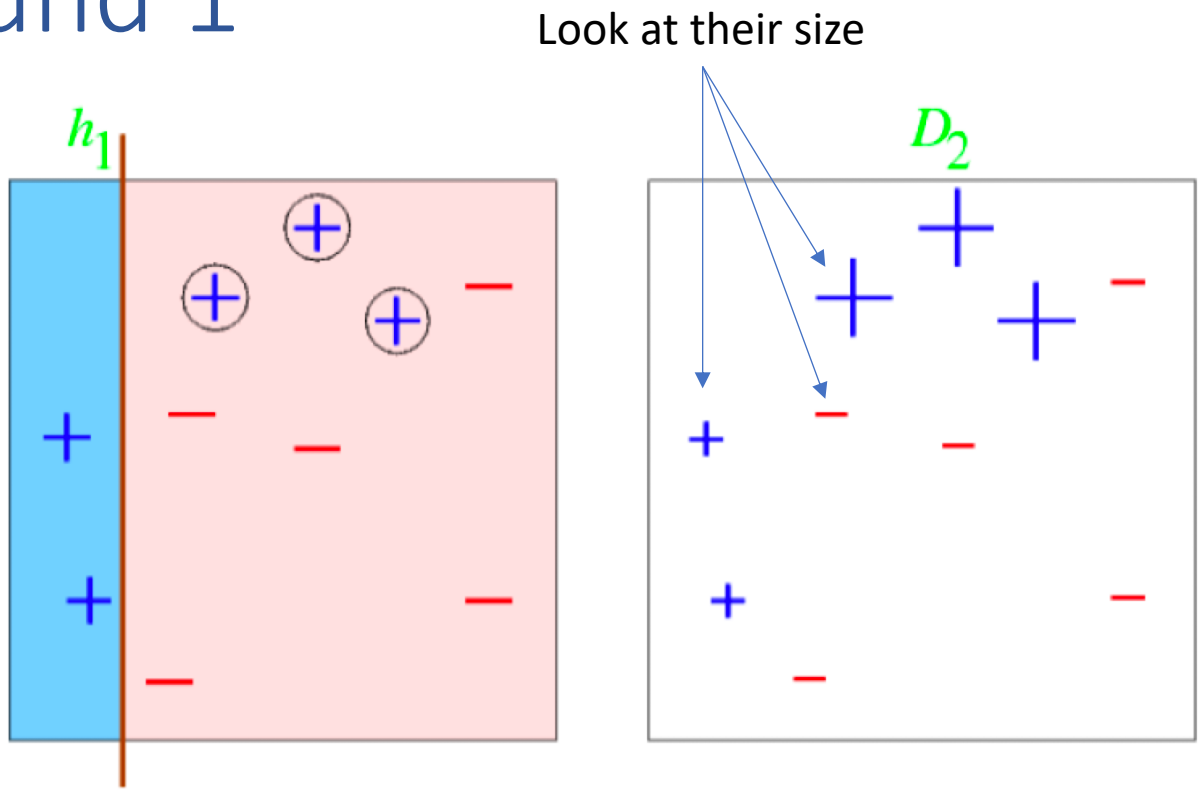
Consider binary classification with 10 training examples

Initial weight distribution D_1 is uniform (each point has equal weight $= 1/10$)



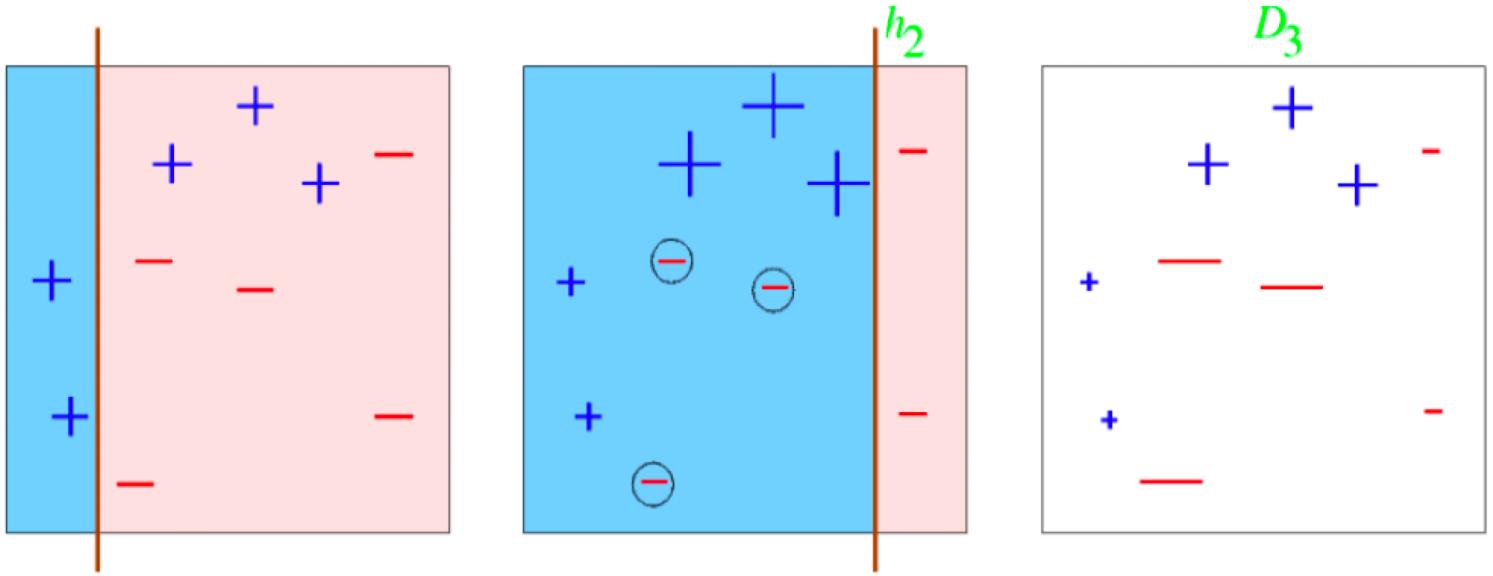
Each of our weak classifers will be an **axis-parallel linear classifier**

After Round 1



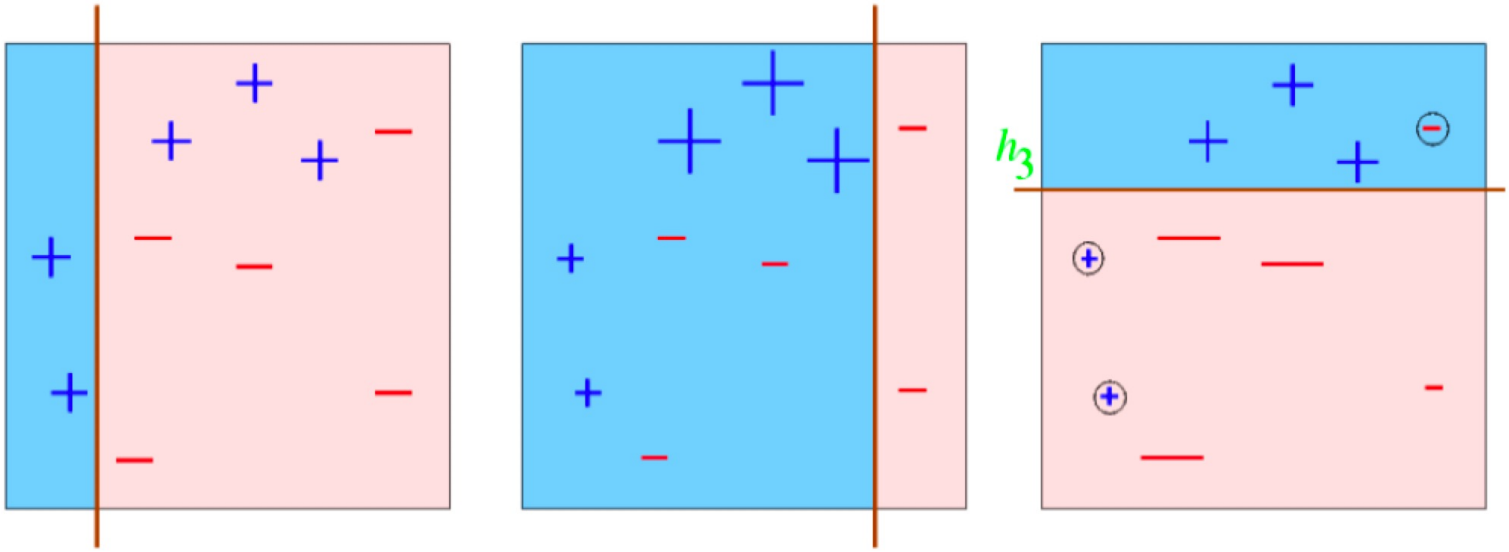
- Error rate of h_1: $\epsilon_1 = 0.3$; weight of h_1: $\alpha_1 = \frac{1}{2}\ln((1-\epsilon_1)/\epsilon_1) = 0.42$
- Each misclassified point upweighted (weight multiplied by $\exp(\alpha_2)$)
- Each correctly classified point downweighted (weight multiplied by $\exp(-\alpha_2)$)

After Round 2



- Error rate of h_2: $\epsilon_2 = 0.21$; weight of h_2: $\alpha_2 = \frac{1}{2}\ln((1-\epsilon_2)/\epsilon_2) = 0.65$
- Each misclassified point upweighted (weight multiplied by $\exp(\alpha_2)$)
- Each correctly classified point downweighted (weight multiplied by $\exp(-\alpha_2)$)

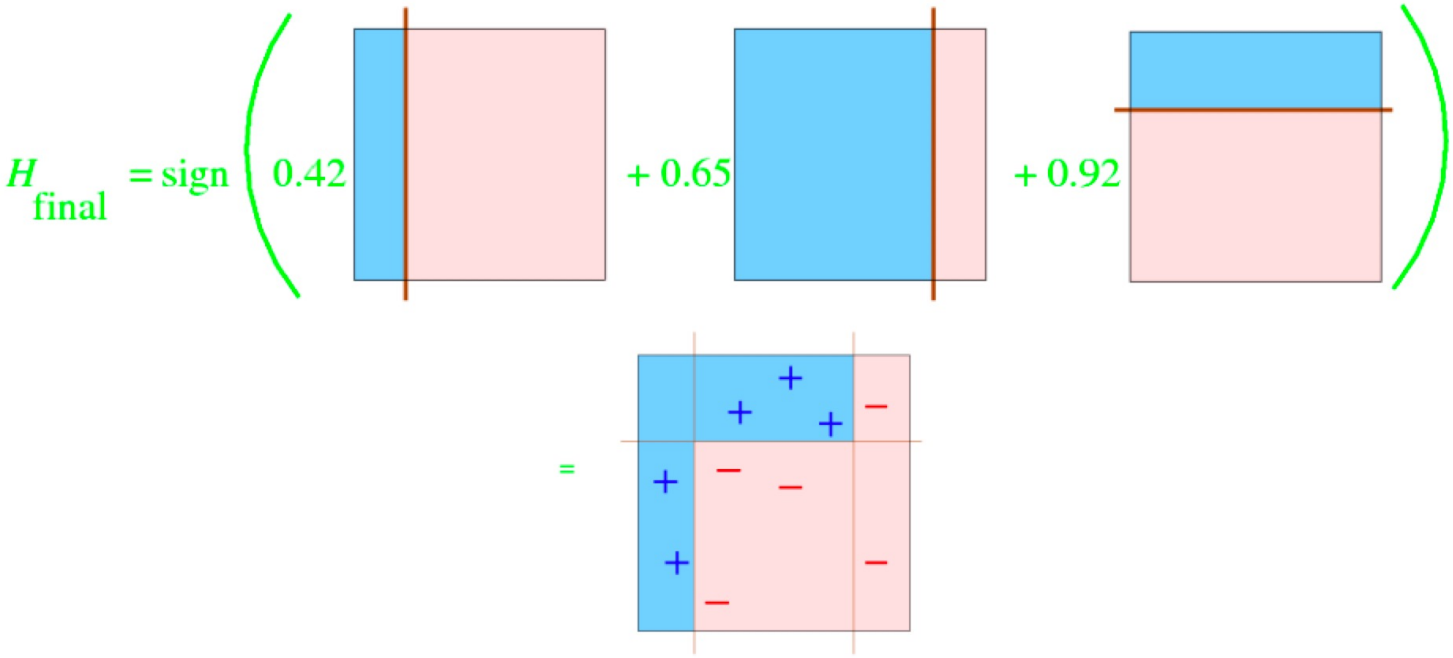
After Round 3



- Error rate of h_3: $\epsilon_3 = 0.14$; weight of h_3: $\alpha_3 = \frac{1}{2}\ln((1-\epsilon_3)/\epsilon_3) = 0.92$
- Suppose we decide to stop after round 3
- Our ensemble now consists of 3 classifiers: h_1, h_2, h_3

Finally, we have

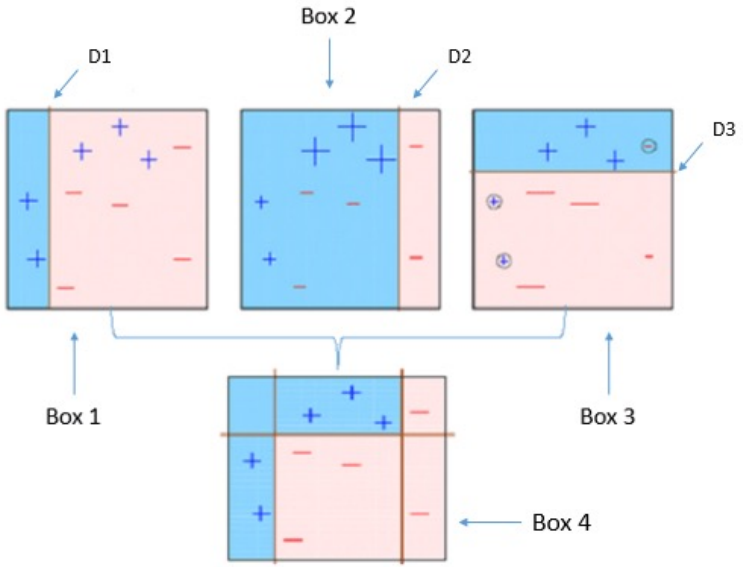
- Final classifier is a weighted linear combination of all the classifiers
- Classifier h_i gets a weight α_i

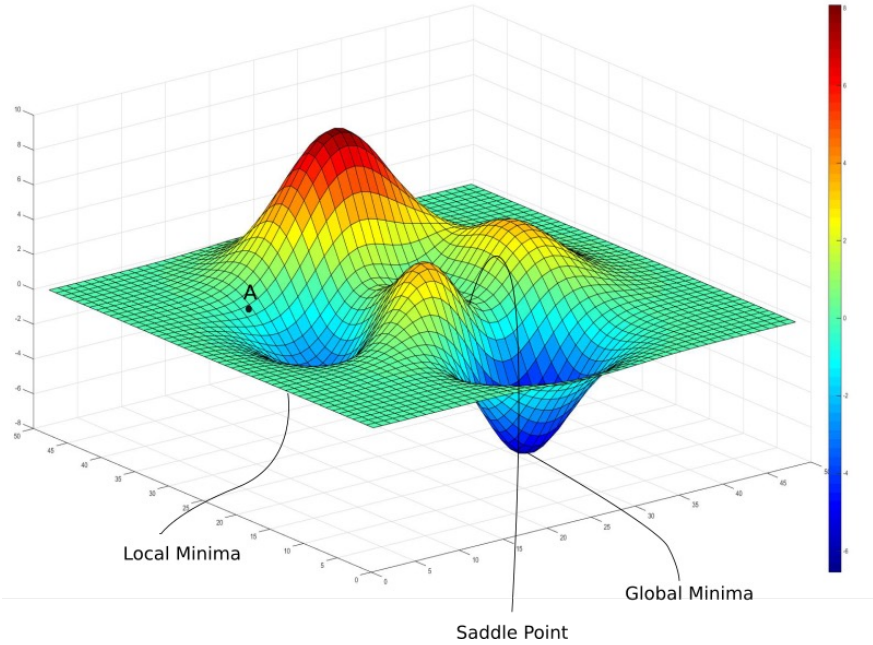


- Multiple weak, linear classifiers **combined** to give a strong, nonlinear classifier

What is XGBoost?

- Stands for:
  - e**X**treme **G**radient **B**oosting.
- XGBoost is a powerful iterative learning algorithm based on gradient boosting.
- Robust and highly versatile, with custom objective loss function compatibility.
- Similar in spirit to AdaBoost but adopting an optimization approach for tree boosting (learning tree ensembles)





Why use XGBoost?

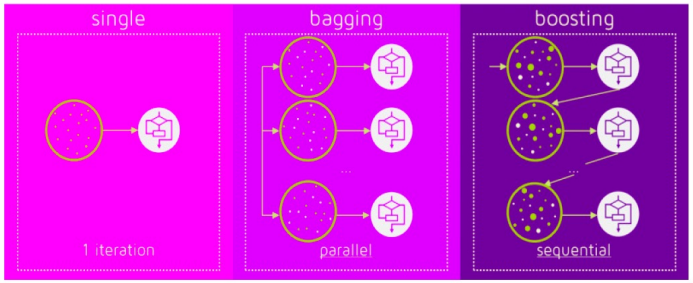
- All of the advantages of gradient boosting (a kind of learning tree ensembles), plus more.
- Frequent Kaggle data competition champion.
- Utilizes CPU Parallel Processing by default.
- Two main reasons for use:
  1. Low Runtime
  2. High Model Performance

Remarks

- Competitor-LightGBM (Microsoft)
  - Very similar, not as mature and feature rich
  - Slightly faster than XGBoost – much faster when it was published
- Hardly find an example where Random Forest outperforms XGBoost
- XGBoost Github: https://github.com/dmlc/xgboost
- XGBoost documentation: http://xgboost.readthedocs.io

Bagging vs Boosting

- No clear winner; usually depends on the data
- Bagging is computationally more efficient than boosting (note that bagging can train the M models in parallel, boosting can't)



- Both reduce variance (and overfitting) by combining different models
  - The resulting model has higher stability as compared to the individual ones
- Bagging usually can't reduce the bias, boosting can (note that in boosting, the training error steadily decreases)
- Bagging usually performs better than boosting if we don't have a high bias and only want to reduce variance (i.e., if we are overfitting)

Take-home Messages

- Ensemble methods are effective approaches to boost the performance of ML/DM system.
- One particular advantage is that they typically do not involve too many parameters and too involved tuning.
- Think about the following:
  - Would an ensemble of sophisticated ML/DM models be useful?
  - Slide 16 of Regularization notes: “It provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks (concept of ensemble methods).”

Acknowledgement

- Slides/Materials of
  - Yisong Yue's Presentation at University of Luebeck
  - P. Rai, IIT
  - Zhuowen Tu, UCLA
- Photos from Internet