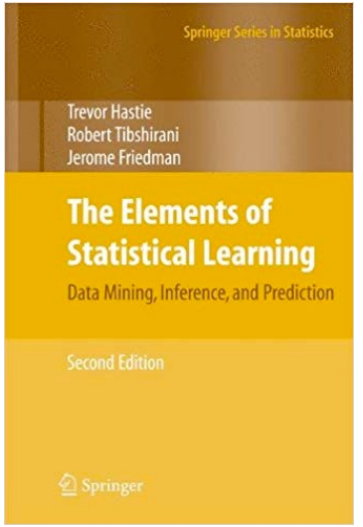
Course material at:
https://github.com/lyeskhalil/mlbootcamp2022



Model Engineering

Tuesday, Lecture 1

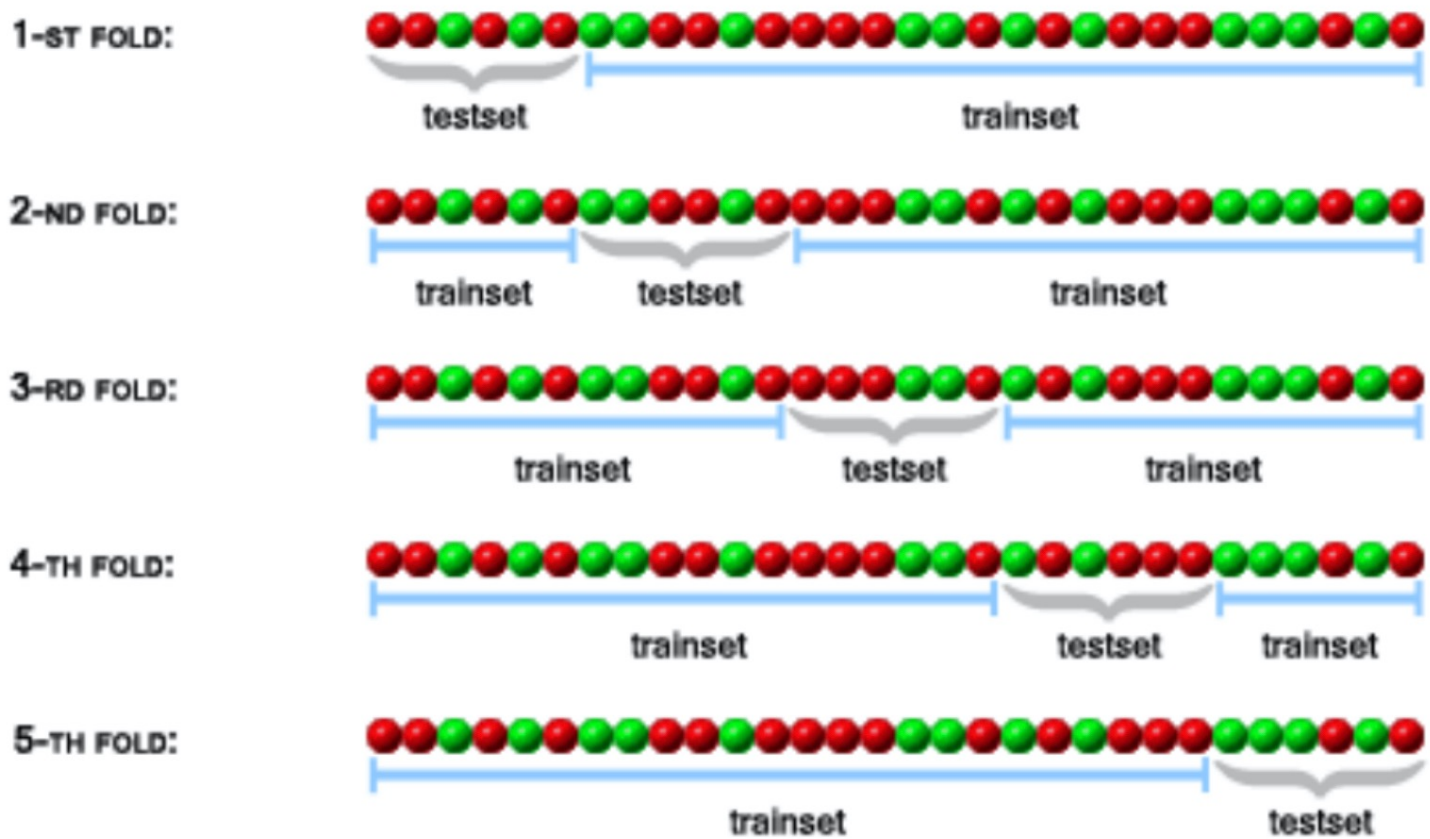
FASE ML Bootcamp

Based on material from:
Elements of Statistical Learning by Hastie, Tibshirani, Friedman,
Theory and Applications of Boosting by Schapire
XGBoost slides:
https://speakerd.s3.amazonaws.com/presentations/5c6dab45648344208185d2b1ab4fdc95/XGBoost-Newest.pdf

kaggle

Example: one run of *5-fold* cross validation

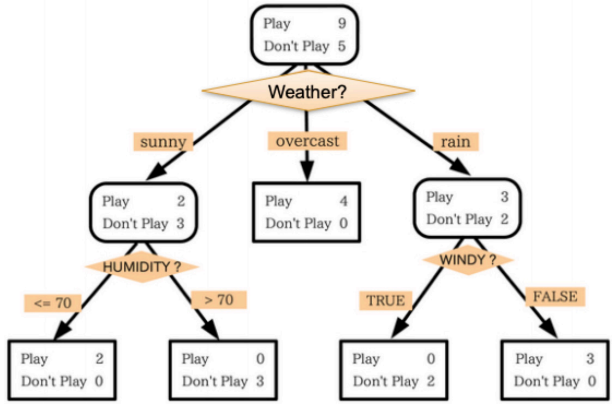
You should do a **few runs** and **compute the average**
(e.g., error rates if that's your evaluation metrics)



Cross validation

1. Divide your data into n parts
2. Hold 1 part as “test set” or “hold out set”
3. Train classifier on remaining n-1 parts “training set”
4. Compute test error on test set
5. Repeat above steps n times, once for each n-th part
6. Compute the average test error over all n folds
   (i.e., cross-validation test error)

Model Combination



Any single model may have **high variance** in its predictions

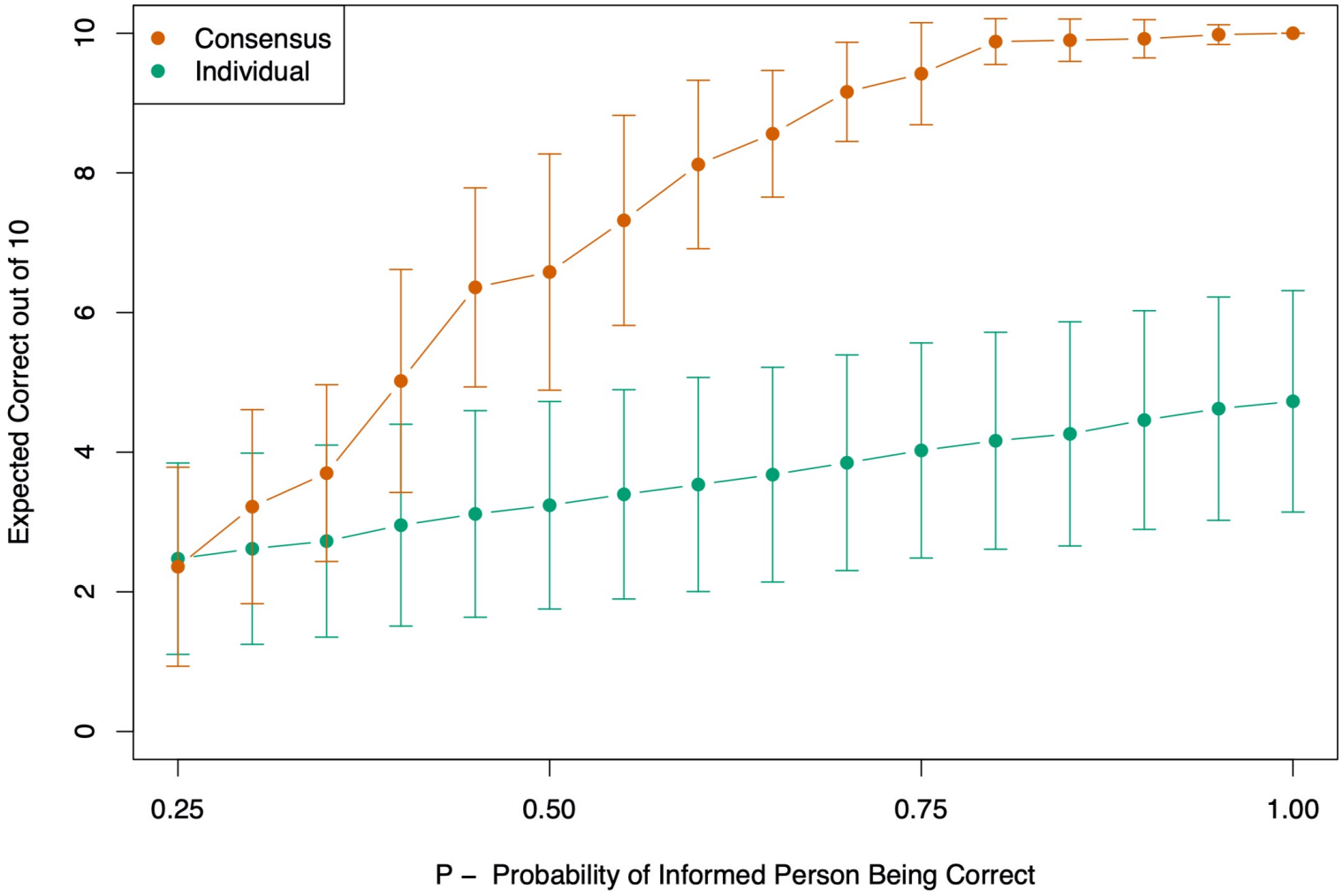
- Decision tree: predictions can be brittle, small perturbation in data can lead to misclassification

Creating more stable models:

- Come up with a completely new model
- **Combine (unstable) models intelligently!**

Simulated academy awards voting

- **50 members** vote in **10 categories**, each with **4 nominations**. For any category, **only 15 voters have some knowledge**, represented by their probability of selecting the “correct” candidate in that category (so P = 0.25 means they have no knowledge).
- For each category, the **15 experts are chosen at random** from the 50.
- Results show the **expected number of correct decisions** (based on 50 simulations) for the consensus, as well as for the individuals. The error bars indicate one standard deviation.
- We see, for example, that **if the 15 informed for a category have a 50% chance of selecting the correct candidate, the consensus doubles the expected performance of an individual.**



Elements of Statistical Learning, Figure 8.11

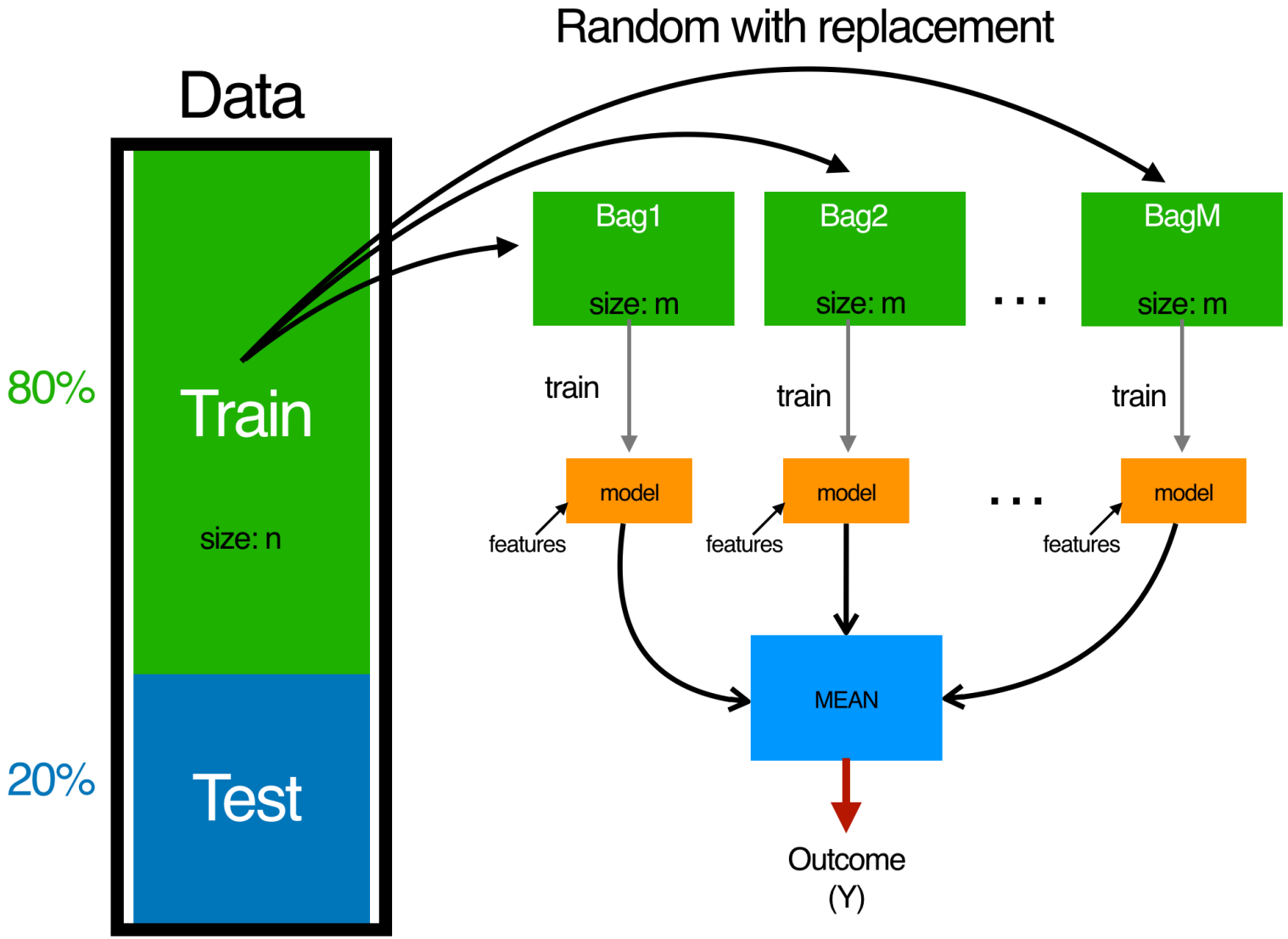
Bootstrap Sampling

Data: $S = \{(x_i, y_i)\}_{i = 1,\ldots,n}$

Bootstrap Sample: subsample of S, drawn **with replacement**

https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

Bootstrap **Aggregating (Bagging)**

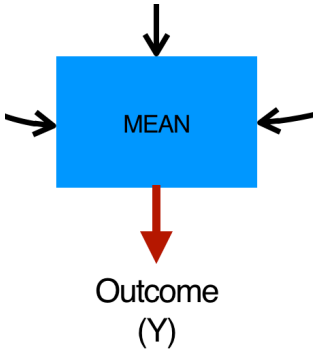


http://www.ub.edu/stat/docencia/EADB/bagging.html

Bagging

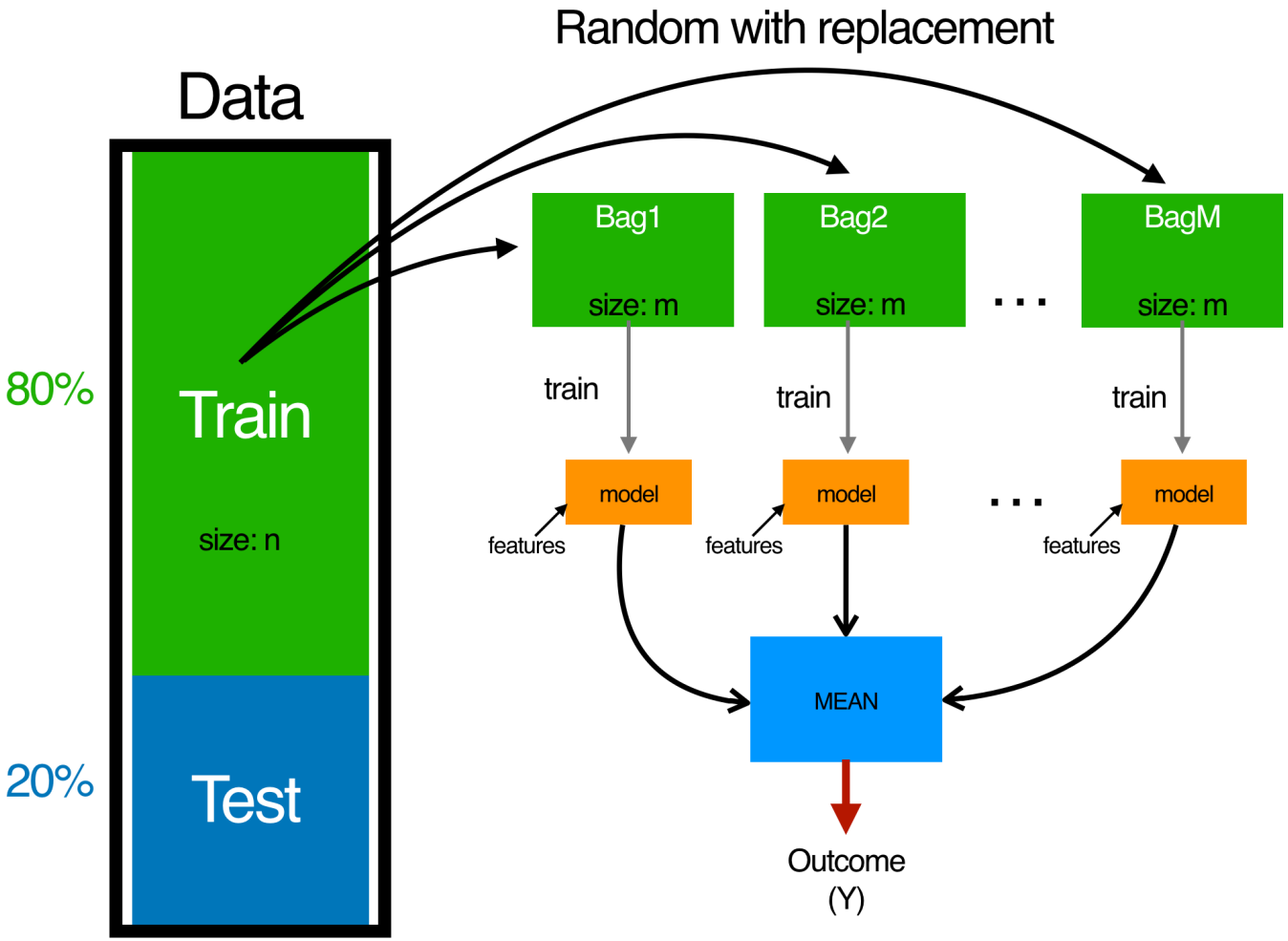
Aggregation:

- **Majority vote!**



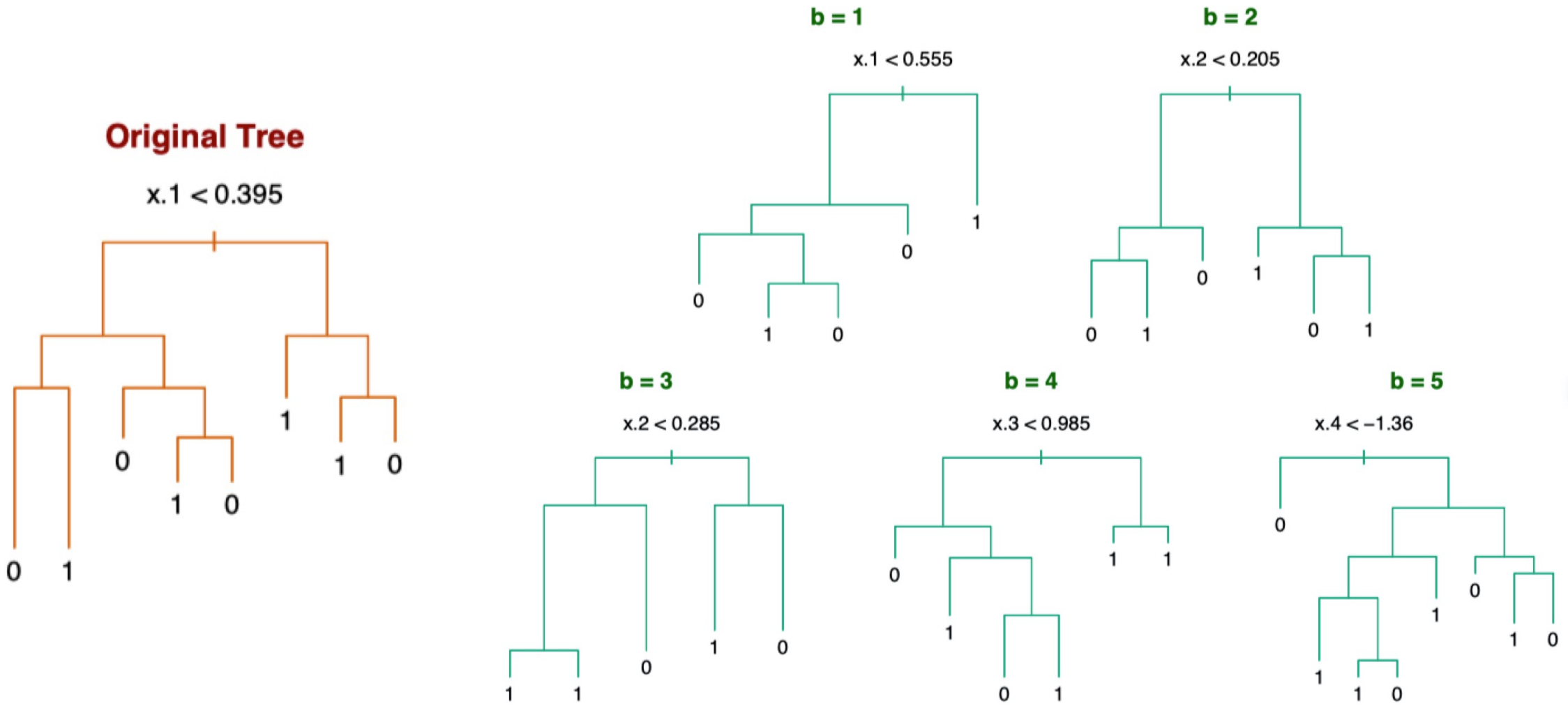
Benefits:

- Even if single model overfits, on average they will not



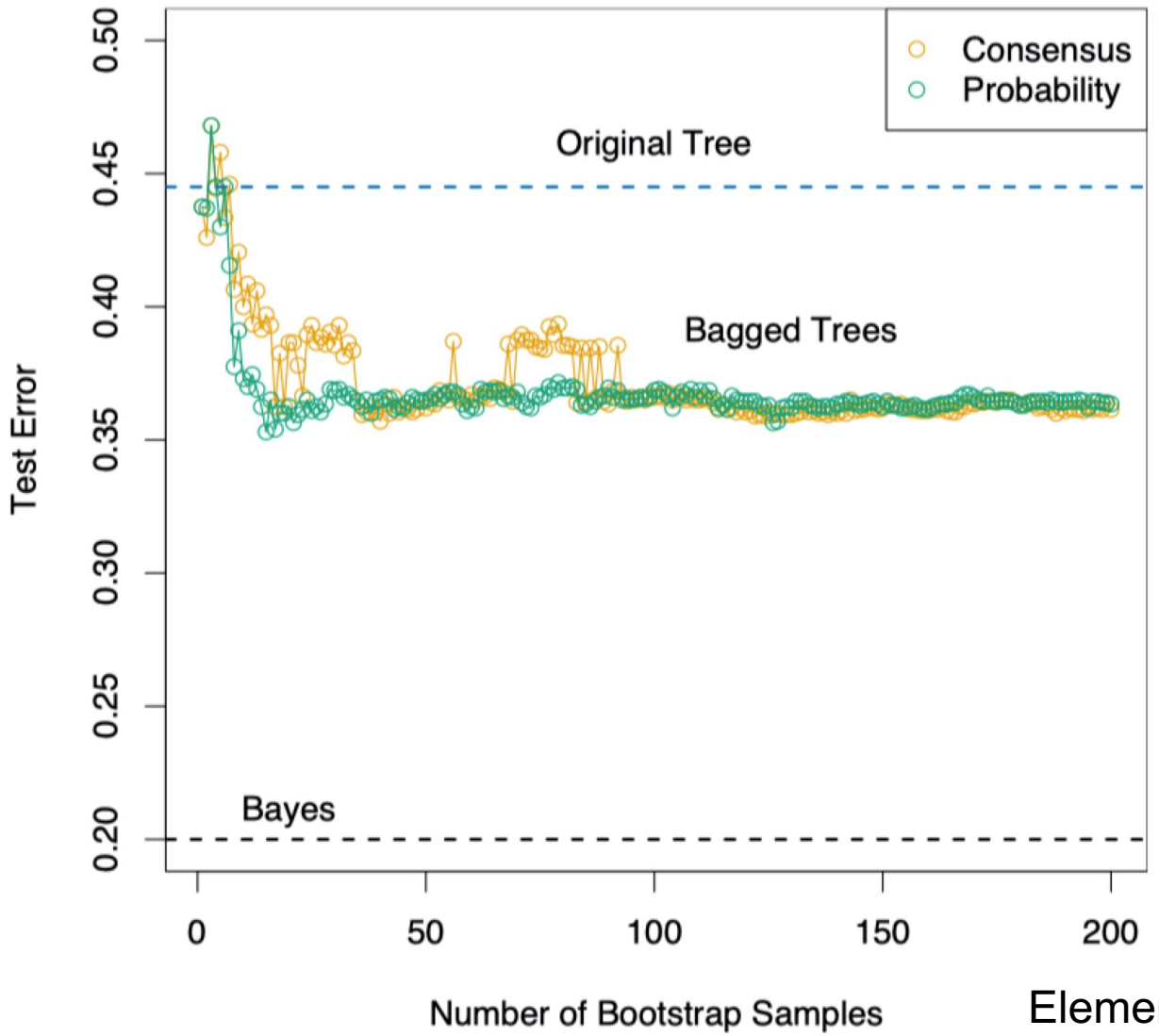
http://www.ub.edu/stat/docencia/EADB/bagging.html

Bagging with Decision Trees



Elements of Statistical Learning, Chapter 8.7

Predicting with bagged decision trees



Consensus: proportion of trees that predict a given class

Probability:

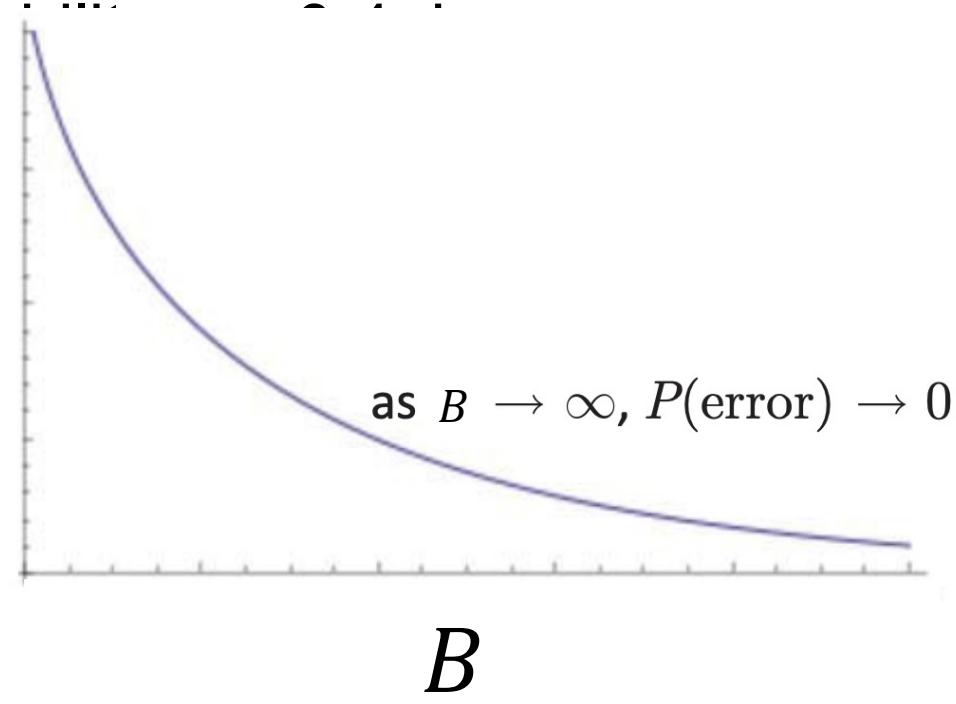
- For b-th tree, look at the leaf corresponding to test sample
- What proportion of the training data had the same label as test sample?
- Average those proportions out

Prediction: in both cases, predict class with highest score

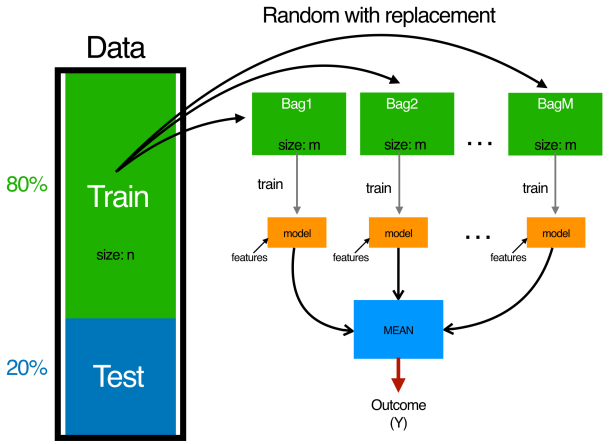
Elements of Statistical Learning, Chapter 8.7

Why does bagging work?

- Single test data point, **x**, with true label 1
- **B independent** classifiers, $f^b(x)$
- Each classifier misclassifies **x** with
$$Prob\left(f^b(x)\right)$$
- The bagged classifier predicts as:
$$f^{bag}(x) = \text{argmax}_{k=0}$$
- $B_0 = \sum_{b=1,\ldots,B} I\{f^b(x) = 0\}$ is the n
- B_0 is **binomial** random variable
- Probability of misclassification:
$$Prob\left(f^{bag}(x) = 0\right) = P(B_0 \geq B/2)$$
- As $B \to \infty$, $Prob\left(f^{bag}(x) = 0\right) \to 0$



When should use Bagging?



High-variance classifiers, e.g., decision trees.

Bagging reduces variance by providing an alternative approach to **regularization**.

Even if each of the learned classifiers are individually overfit, they are likely to **overfit to different things**.

Through voting, we can overcome a significant portion of this overfitting.

In practice, bagging tends to **reduce variance** and **increase bias**.

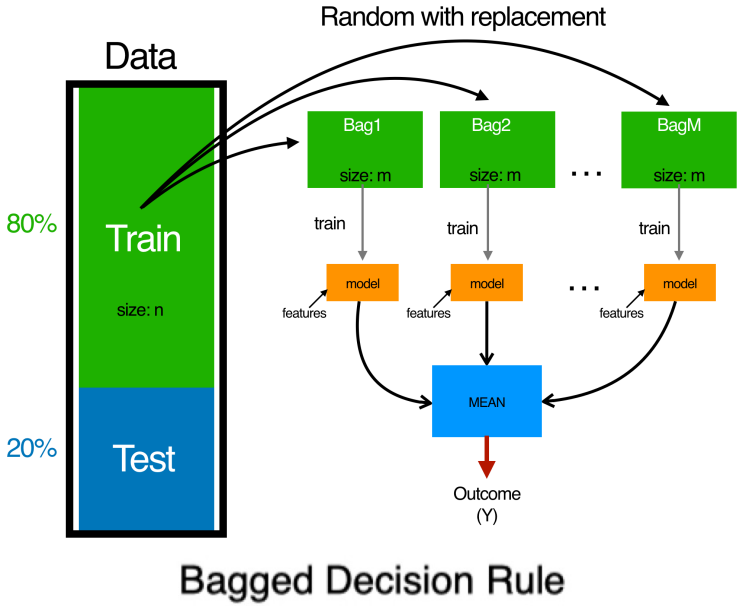
Final words on Bagging

Advantages

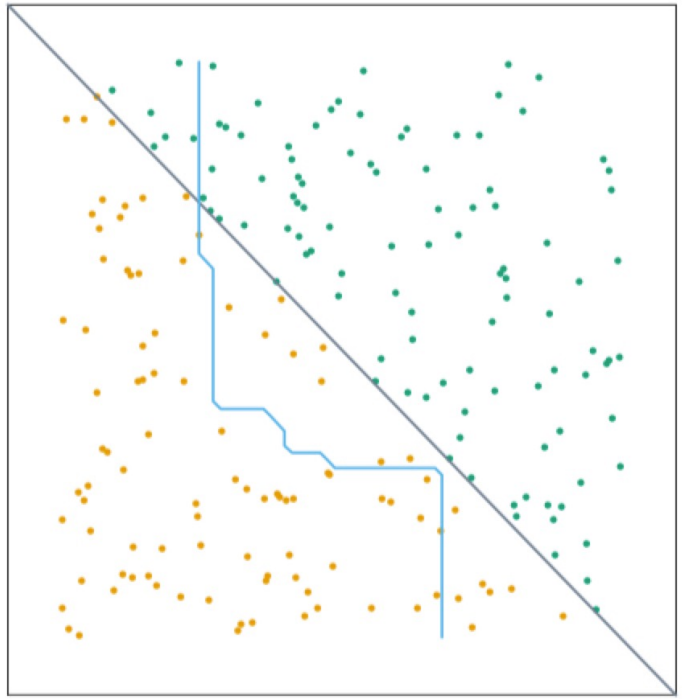
- Reduces model variance
- Trivial to implement and use

Caveats

- **Destroys any interpretability** in the base model
- May not capture simple patterns

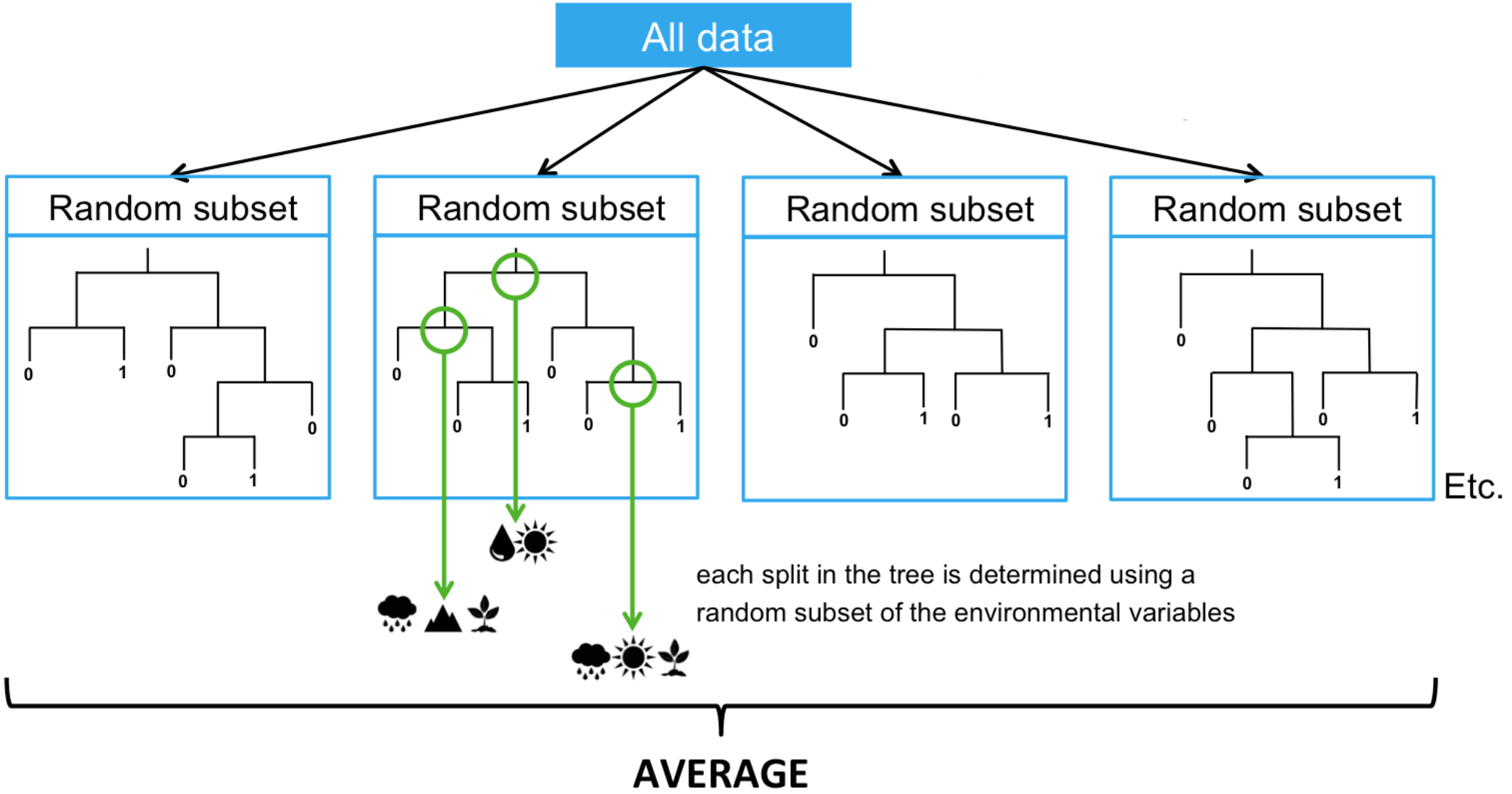


Bagged Decision Rule



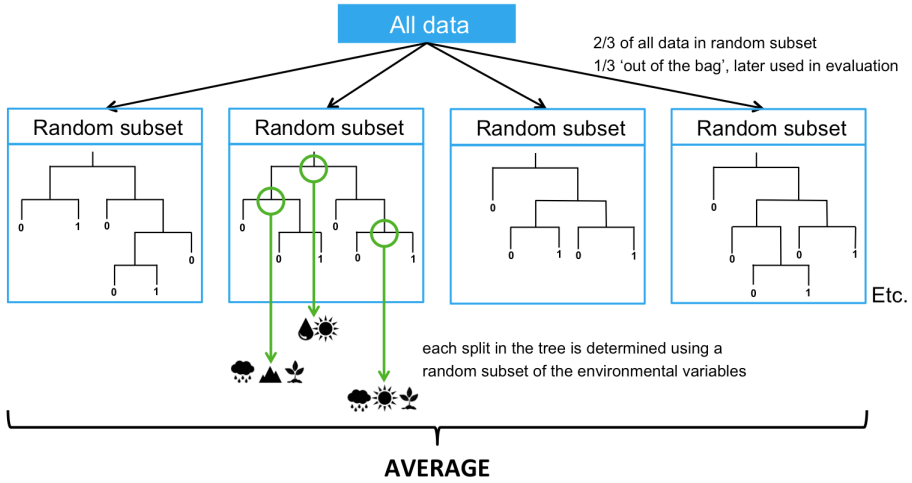
Random Forest

Similar to Bagging Decision Trees **except**:



https://support.bccvl.org.au/support/solutions/articles/6000083217-random-forest

Random Forest



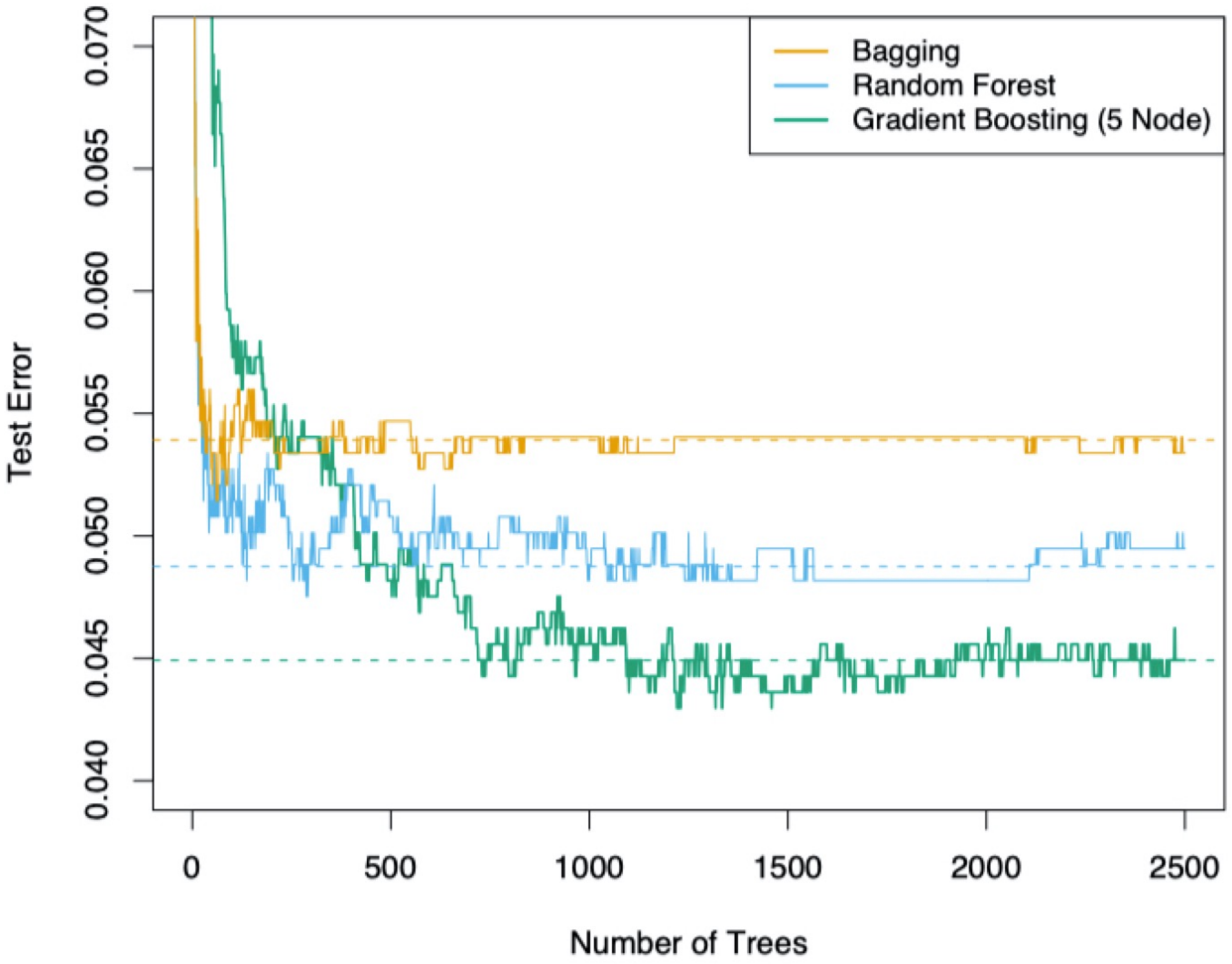
- One of the most **popular** models!
- Usually **more effective** than Bagging Decision Trees
- Typically, use **100s of trees** (hyper-parameter to be tuned!)
- For data with d features, num. random features used for splitting:
  - Classification: $\sqrt{\boldsymbol{d}}$
  - Regression: $\boldsymbol{d/3}$

Example: Spam classification

Dataset of 4000 emails: y = (spam, not spam)

x = % of words equal to 'business', 'address', etc.;

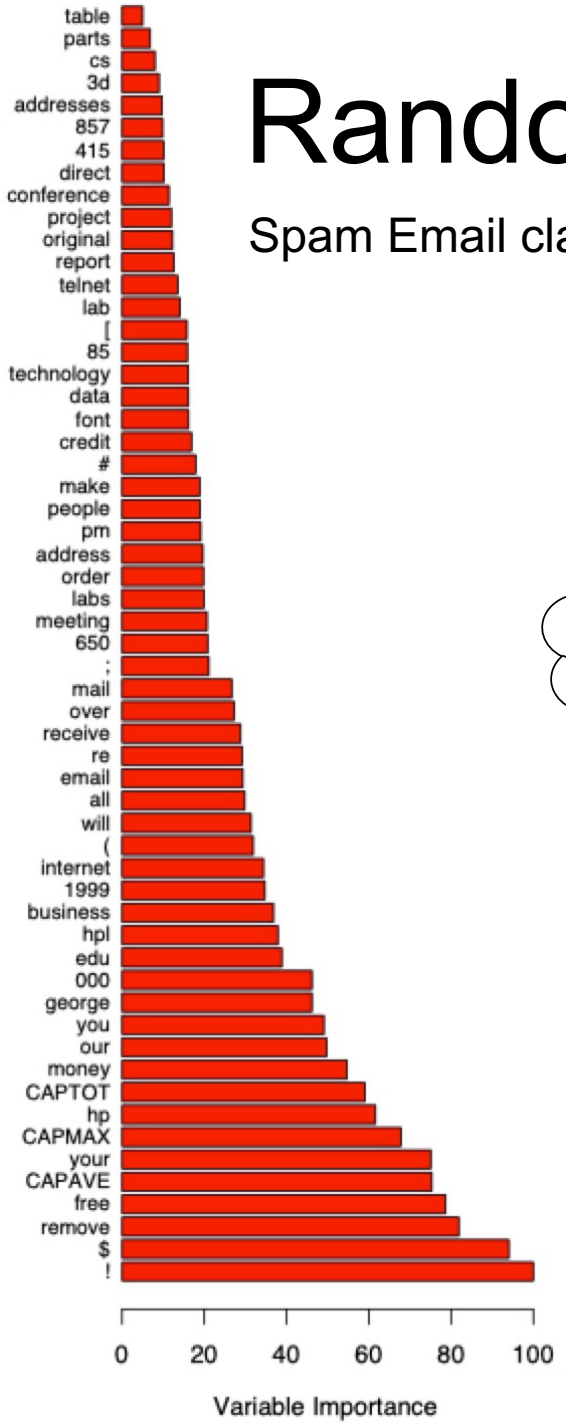
average length of uninterrupted sequences of capital letters, …



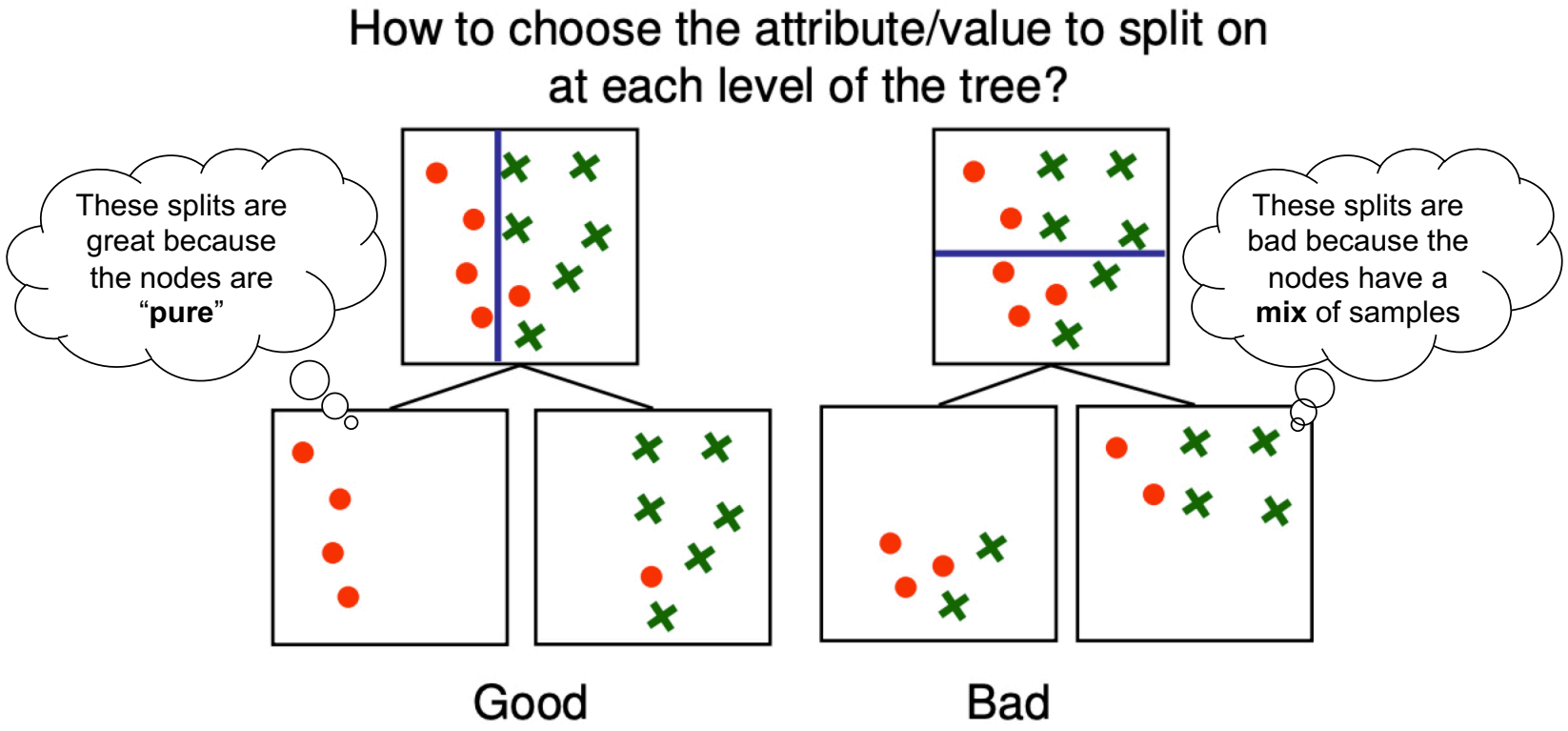
Ignore this for now

Random Forest: Feature Importance

Spam Email classification dataset



words



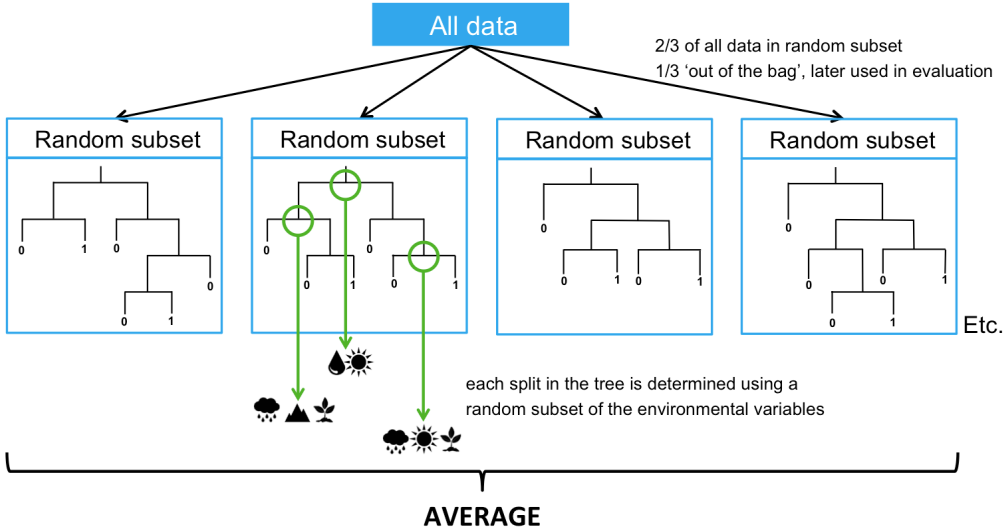
Final words on RF

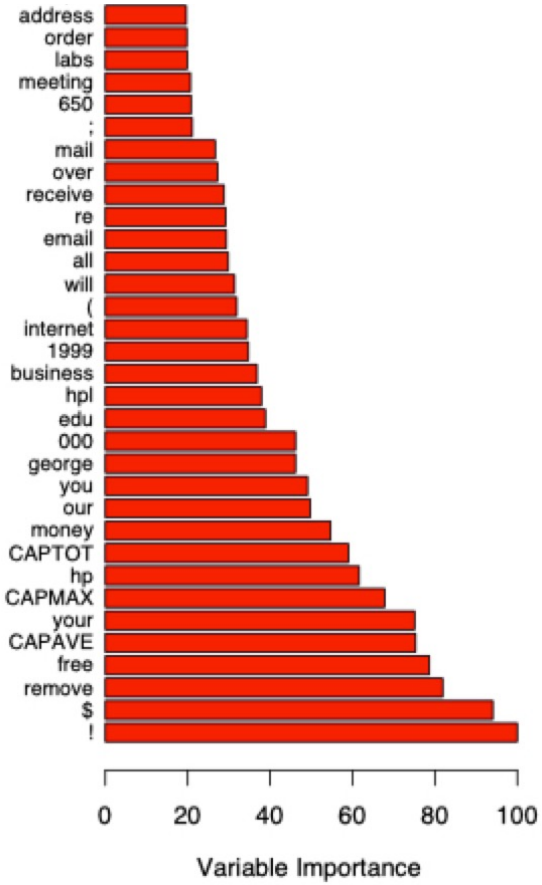
Advantages

- Feed it **data as is** without preprocessing
- **Fast training** because of feature sampling!
- Easy **parallel** training and prediction

Caveats

- **Not suitable for sparse data (why?)**





Boosting: “How May I Help You?”

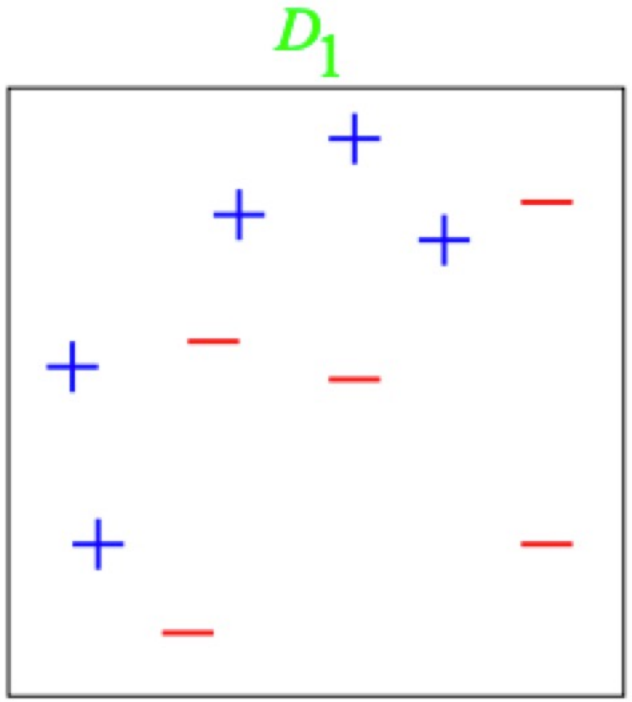
- goal: automatically categorize type of call requested by phone customer (Collect, CallingCard, PersonToPerson, etc.)
  - `yes I'd like to place a collect call long distance please` (Collect)
  - `operator I need to make a call but I need to bill it to my office` (ThirdNumber)
  - `yes I'd like to place a call on my master card please` (CallingCard)
  - `I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill` (BillingCredit)

Boosting: “How May I Help You?”

- observation:
  - easy to find “rules of thumb” that are “often” correct
    - e.g.: “IF ‘card’ occurs in utterance THEN predict ‘CallingCard’ ”
  - hard to find single highly accurate prediction rule

- goal: automatically categorize type of call requested by phone customer (Collect, CallingCard, PersonToPerson, etc.)
  - `yes I’d like to place a collect call long distance please` (Collect)
  - `operator I need to make a call but I need to bill it to my office` (ThirdNumber)
  - `yes I’d like to place a call on my master card please` (CallingCard)
  - `I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill` (BillingCredit)

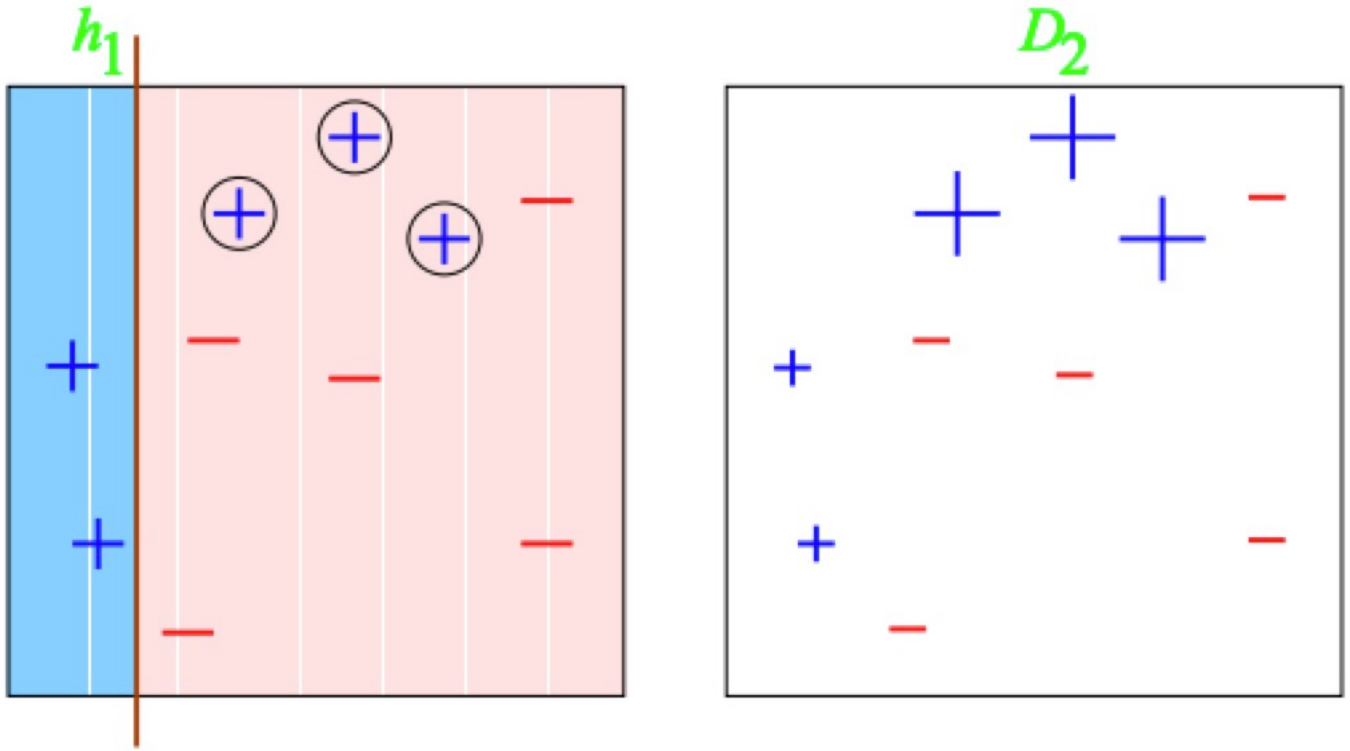
Boosting: A Toy Example



“rule of thumb” classifiers = vertical or horizontal half-planes

Boosting: A Toy Example

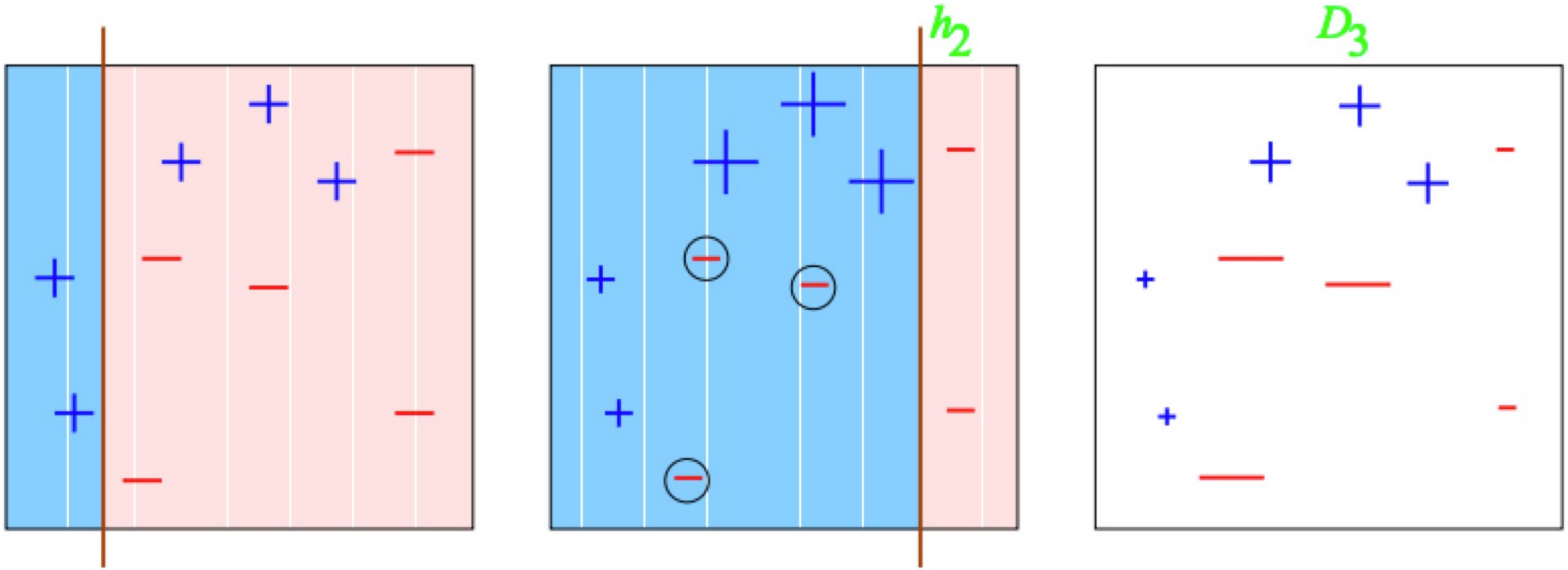
Size of the data point: penalty for misclassifying the point



“rule of thumb” classifiers = vertical or horizontal half-planes

Boosting: A Toy Example

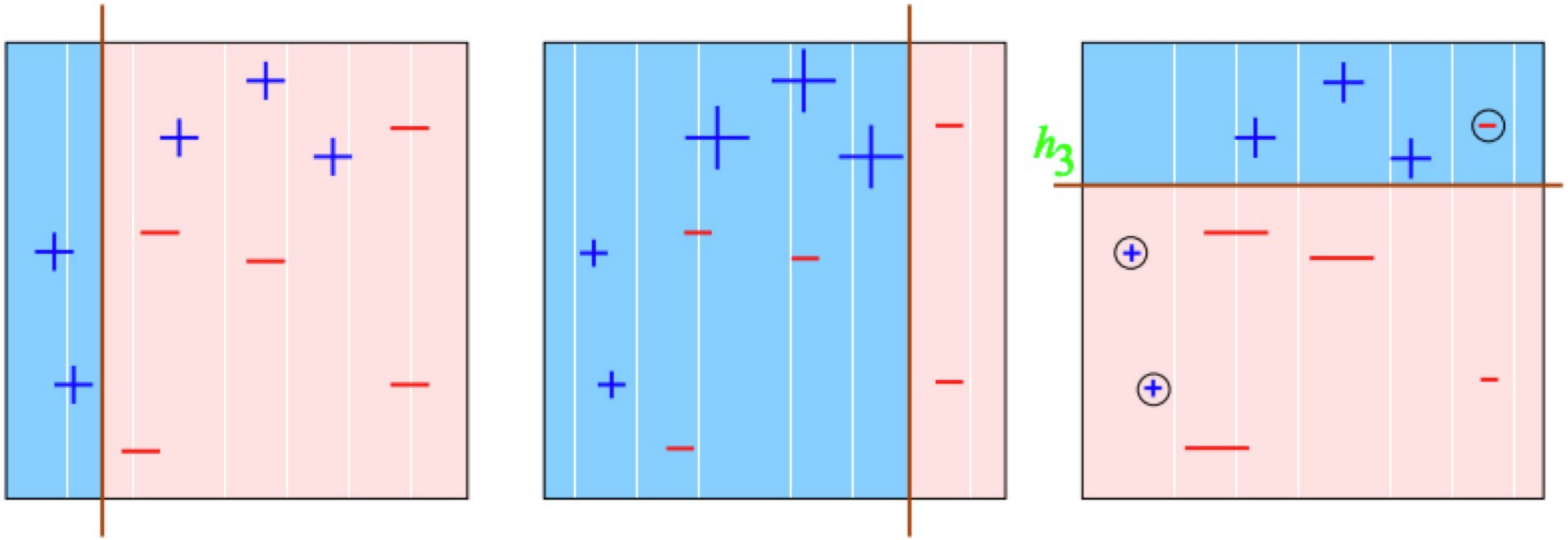
Size of the data point: penalty for misclassifying the point



“rule of thumb” classifiers = vertical or horizontal half-planes

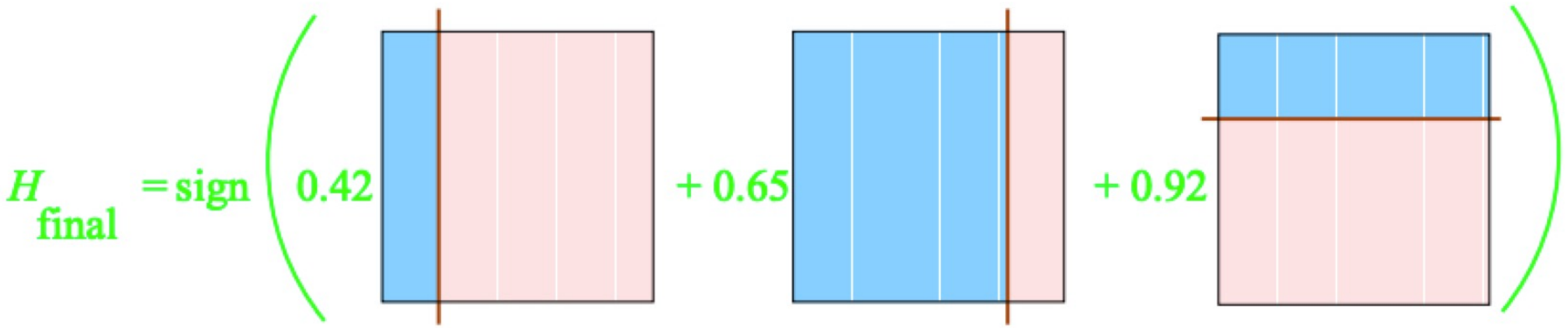
Boosting: A Toy Example

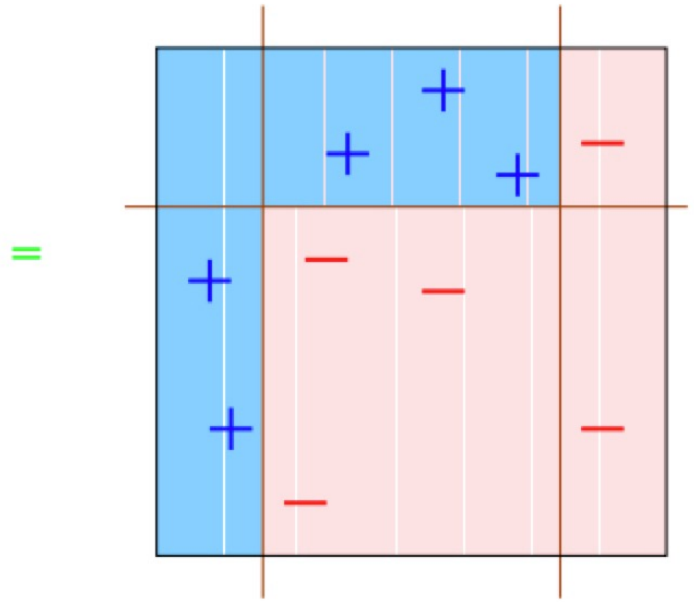
Size of the data point: weight, penalty for misclassifying the point

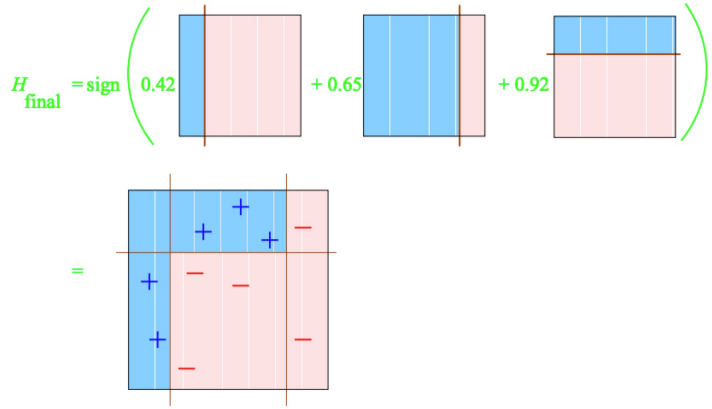


“rule of thumb” classifiers = vertical or horizontal half-planes

Boosting: A Toy Example







Boosting in a Nutshell

A general method of **converting rough rules** of thumb **into highly accurate prediction** rule.

Technically:

- assume given “weak” learning algorithm that can consistently find classifiers (“rules of thumb”) at least **slightly better than random**, say, accuracy ≥ 55%
- given sufficient data, a boosting algorithm can provably **construct single classifier with very high accuracy**

Xgboost: A scalable tree boosting system

T Chen, C Guestrin - Proceedings of the 22nd acm sigkdd international …, 2016 - dl.acm.org

Tree boosting is a highly effective and widely used machine learning method. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is used widely by data scientists to achieve state-of-the-art results on many machine learning …

☆ Cited by 3057 Related articles All 14 versions

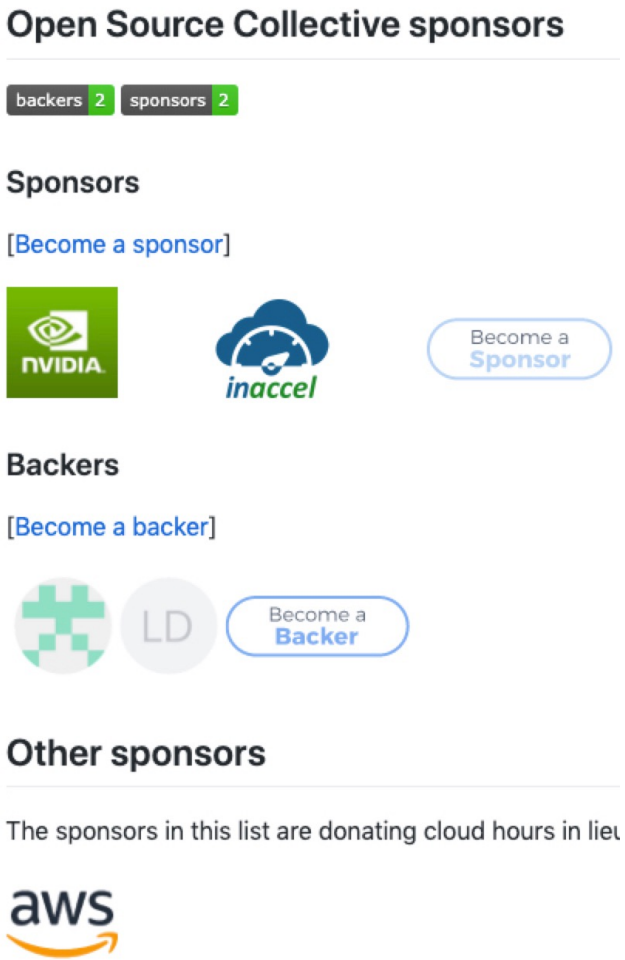
dmlc

XGBoost eXtreme Gradient Boosting

Very efficient, extensive **tree boosting** code by Tianqi Chen (UW)

A standard tool in data science applications

Easy to parallelize and run on billion-scale problems

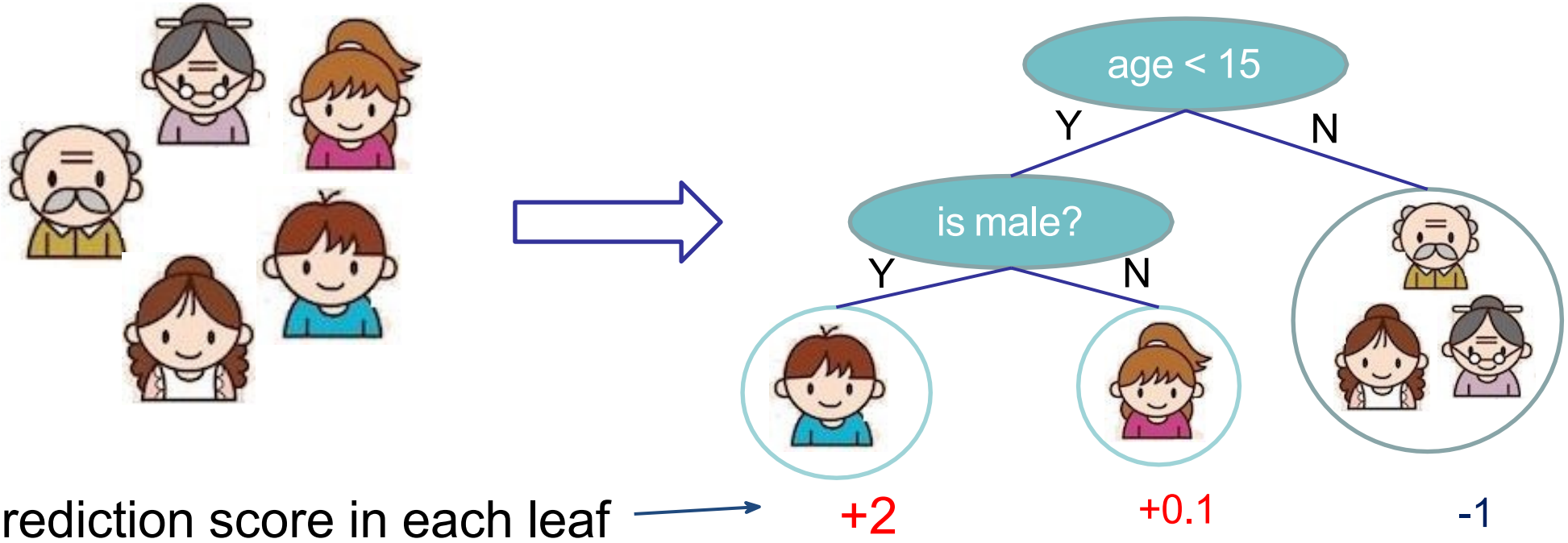


Regression Tree

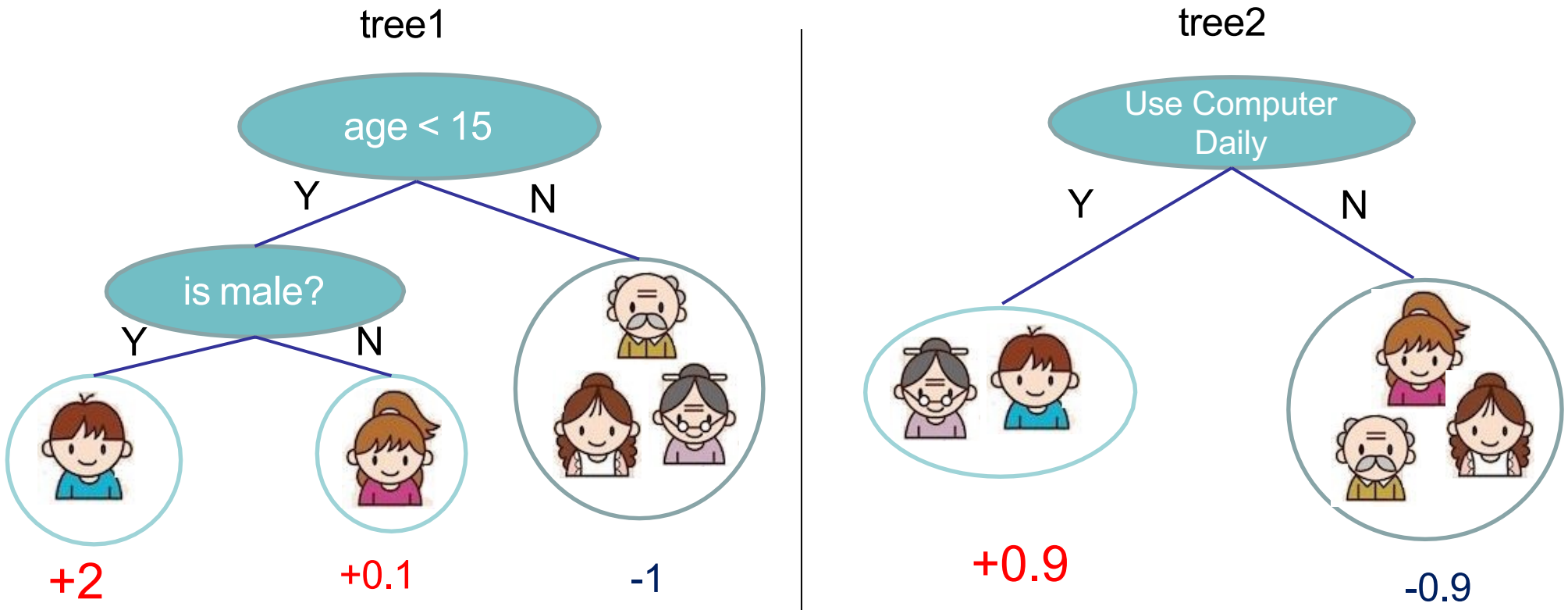
- Regression tree (also known as CART)
- This is what it would look like for a commercial system

Input: age, gender, occupation, …

Does the person like computer games



When Trees forms a Forest (Tree Ensembles)



f() = 2 + 0.9= 2.9

f()= -1 - 0.9= -1.9

Trade off in Learning

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

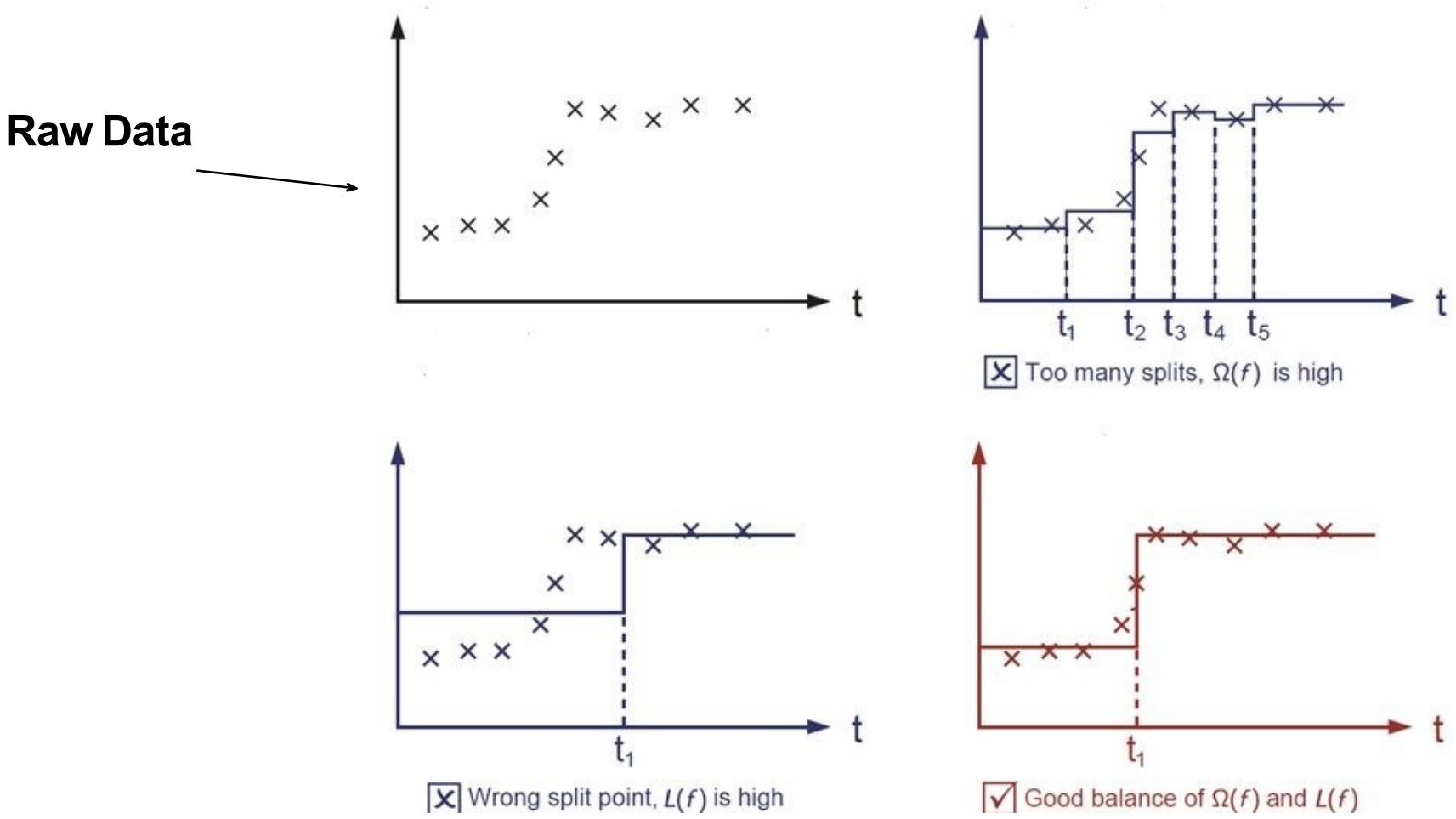
Training Loss measures how well model fit on training data

Regularization, measures complexity of trees

- Optimizing training loss encourages **predictive** models
  - Fitting well in training data at least get you close to training data which is hopefully close to the underlying distribution
- Optimizing regularization encourages **simple** models
  - Simpler models tends to have smaller variance in future predictions, making prediction **stable**

Why do we need regularization

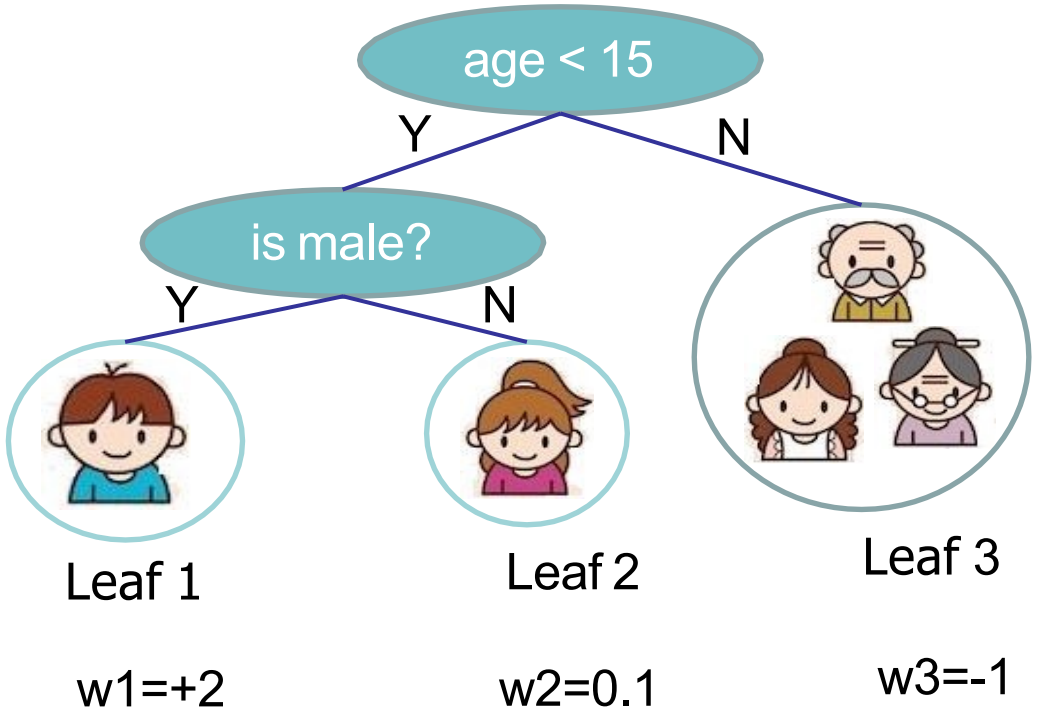
Consider the example of learning tree on a single variable t

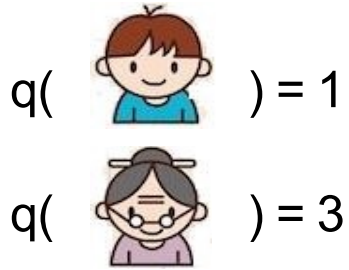


Define Complexity of a Tree

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$

The leaf weight of the tree **The structure of the tree**

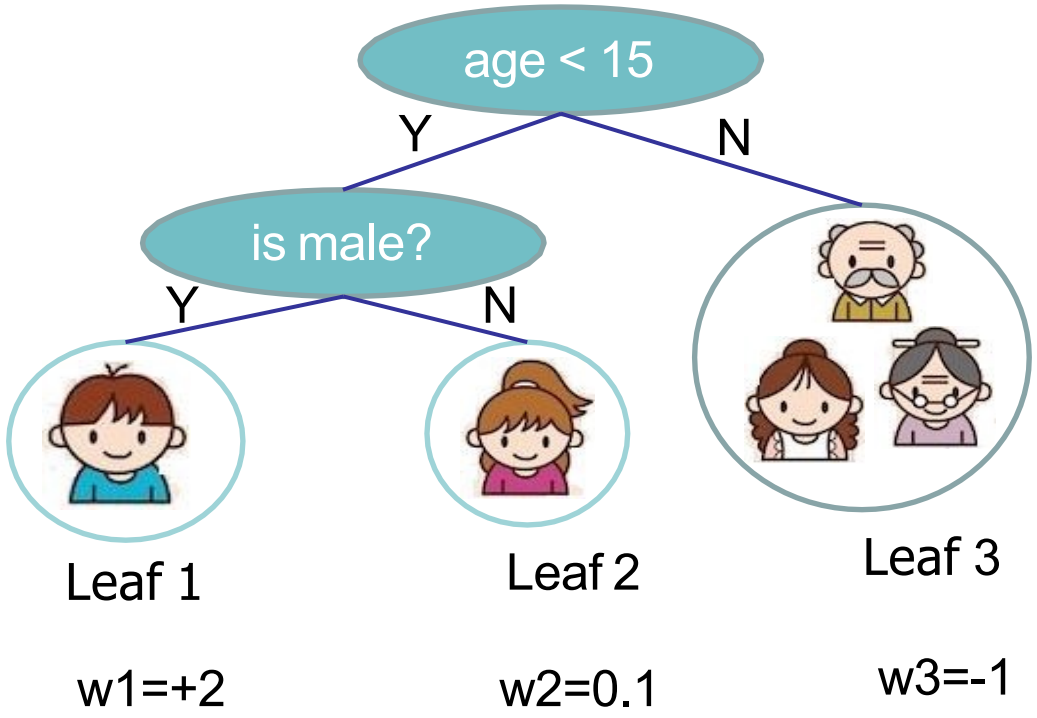




Define Complexity of a Tree (cont')

Objective in XGBoost $\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

Number of leaves **L2 norm of leaf scores**



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

How can we learn tree ensembles

- Objective: $\sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$
- We can not use methods such as gradient descent.
- Solution: **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

New function

Model at training round t

Keep functions added in previous round

Additive Training

- How do we decide which f to add: Optimize the objective!
- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

This is what we need to decide in round t

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant \end{aligned}$$

Goal: find f_t to minimize this

- Consider square loss

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))\right)^2 + \Omega(f_t) + const \\ &= \sum_{i=1}^{n} \left[2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + const \end{aligned}$$

This is usually called residual from previous round

Taylor Expansion Approximation of Loss

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

- *In terms of square loss*

$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial^2_{\hat{y}^{(t-1)}} (y_i - \hat{y}^{(t-1)})^2 = 2$$

Our New Goal

- Objective, with constants removed

$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

- Define the instance set in leaf j as
  - Regroup the objective by leaf

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w^2_{q(x_i)} \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

  - This is sum of T independent quadratic function

The Structure Score

- Two facts about single variable quadratic function

$$argmin_x \; Gx + \tfrac{1}{2}Hx^2 = -\tfrac{G}{H}, \; H > 0 \qquad \min_x \; Gx + \tfrac{1}{2}Hx^2 = -\tfrac{1}{2}\tfrac{G^2}{H}$$

- Let us define $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \tfrac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[G_j w_j + \tfrac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

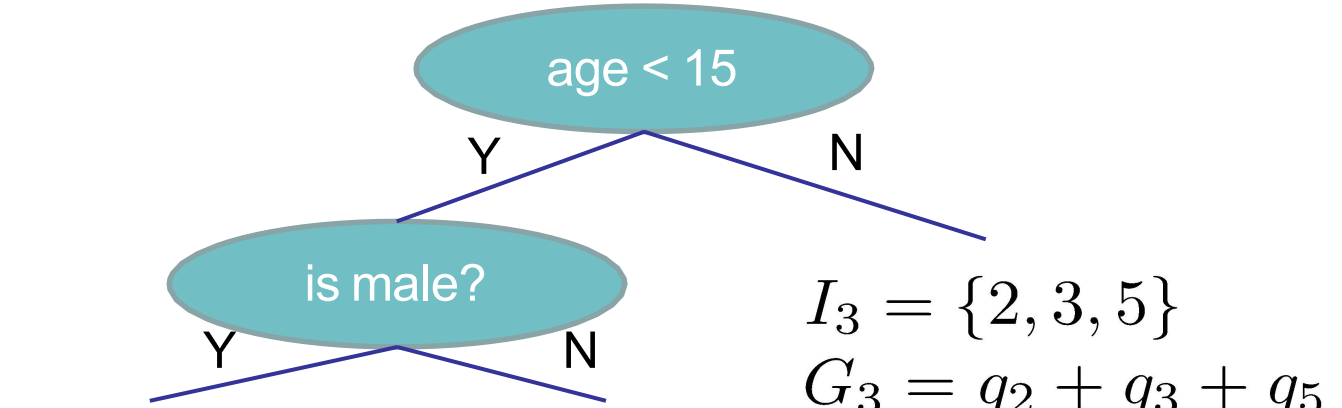
- Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

$$w_j^* = -\frac{G_j}{H_j + \lambda} \qquad Obj = -\tfrac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

This measures how good a tree structure is!

The Structure Score Calculation

Instance index		gradient statistics
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



$$I_1 = \{1\}$$
$$G_1 = g_1$$
$$H_1 = h_1$$

$$I_2 = \{4\}$$
$$G_2 = g_4$$
$$H_4 = h_4$$

$$I_3 = \{2, 3, 5\}$$
$$G_3 = g_2 + g_3 + g_5$$
$$H_3 = h_2 + h_3 + h_5$$

$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Searching Algorithm for Single Tree

- Enumerate the possible tree structures q
- Calculate the structure score for the q, using the scoring eq.

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

- But… there can be infinite possible tree structures..

Greedy Learning of the Tree

- In practice, we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

The complexity cost by introducing additional leaf

the score of left child

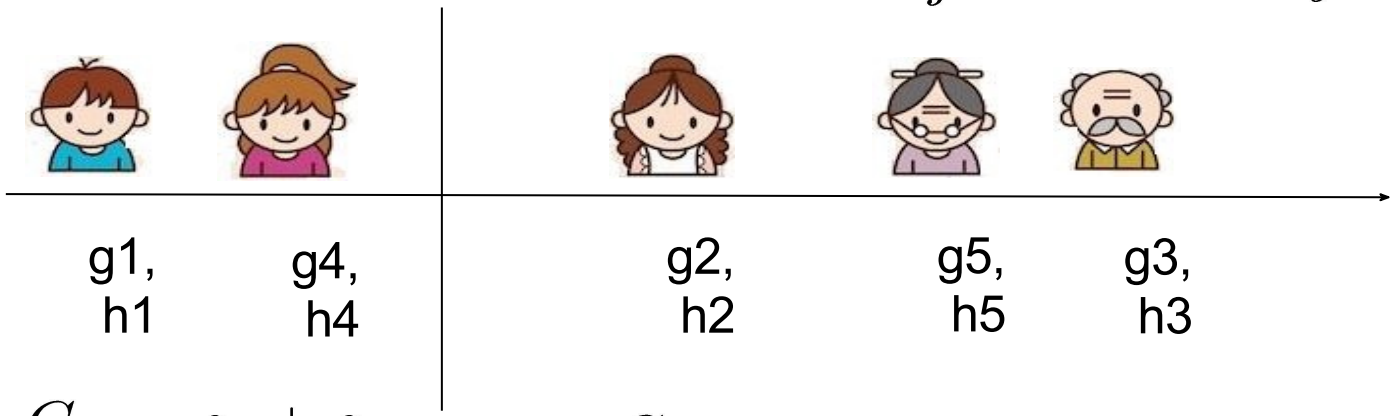
the score of right child

the score of if we do not split

  - Remaining question: how do we find the best split?

Efficient Finding of the Best Split

- What is the gain of a split rule $x_j < a$? Say x_j is age



$G_L = g_1 + g_4$ $\quad G_R = g_2 + g_3 + g_5$

- All we need is sum of g and h in each side, and calculate
- Left to right linear scan over sorted instance is enough to decide the best split along the feature

Pruning and Regularization

- Recall the gain of split, it can be negative!

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

  - When **the training loss reduction** is smaller than **regularization**
  - Trade-off between simplicity and predictiveness
- Pre-stopping
  - Stop split if the best split have negative gain
  - But maybe a split can benefit future splits..
- Post-Prunning
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

XGBoost Model Recap

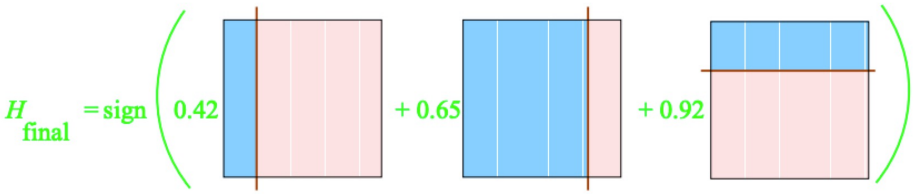
- A regularized objective for better generalization
- Additive solution for generic objective function
- Structure score to search over structures.
- Why take all the pain in deriving the algorithm
  - Know your model
  - Clear definitions in algorithm offers clear and extendible modules in software

Final words on Boosting

dmlc XGBoost eXtreme Gradient Boosting

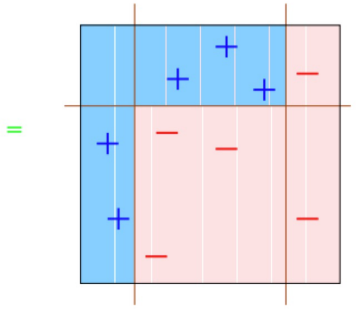
Advantages

- **Fast** and **simple**
- **Flexible**: can work with any weak learner
- Handles **various feature types**
- Strong **theoretical foundations**

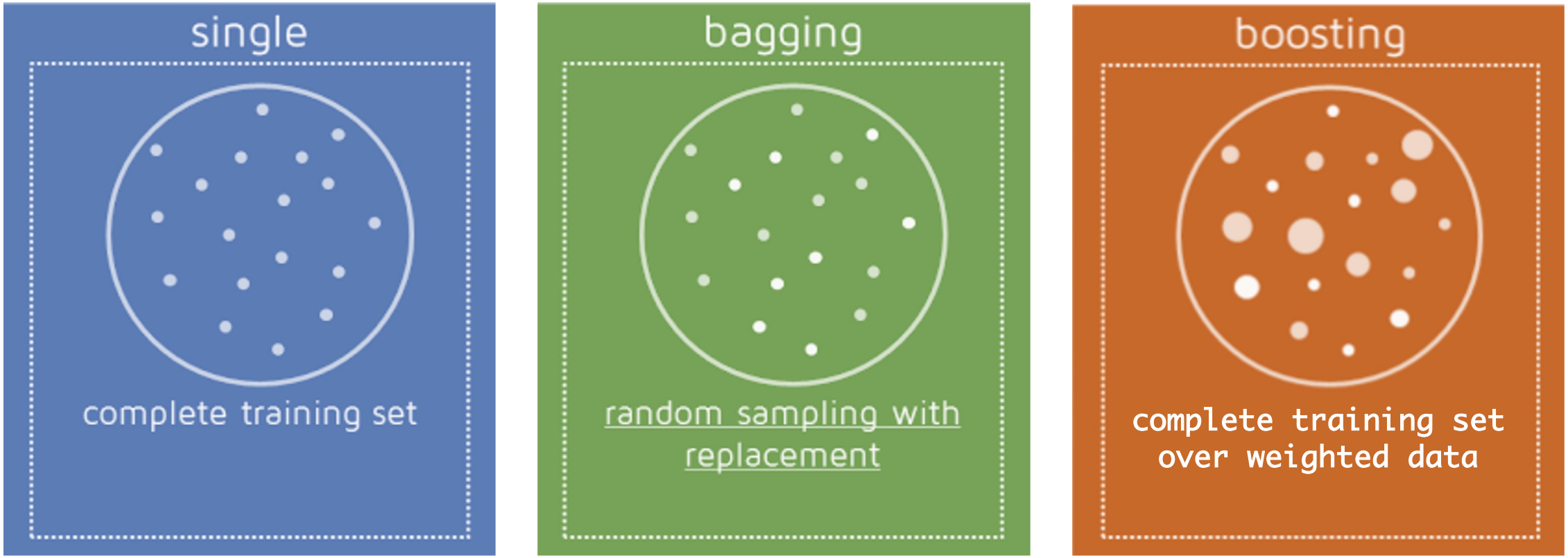


Caveats

- **Overfits** if weak learner is too complex

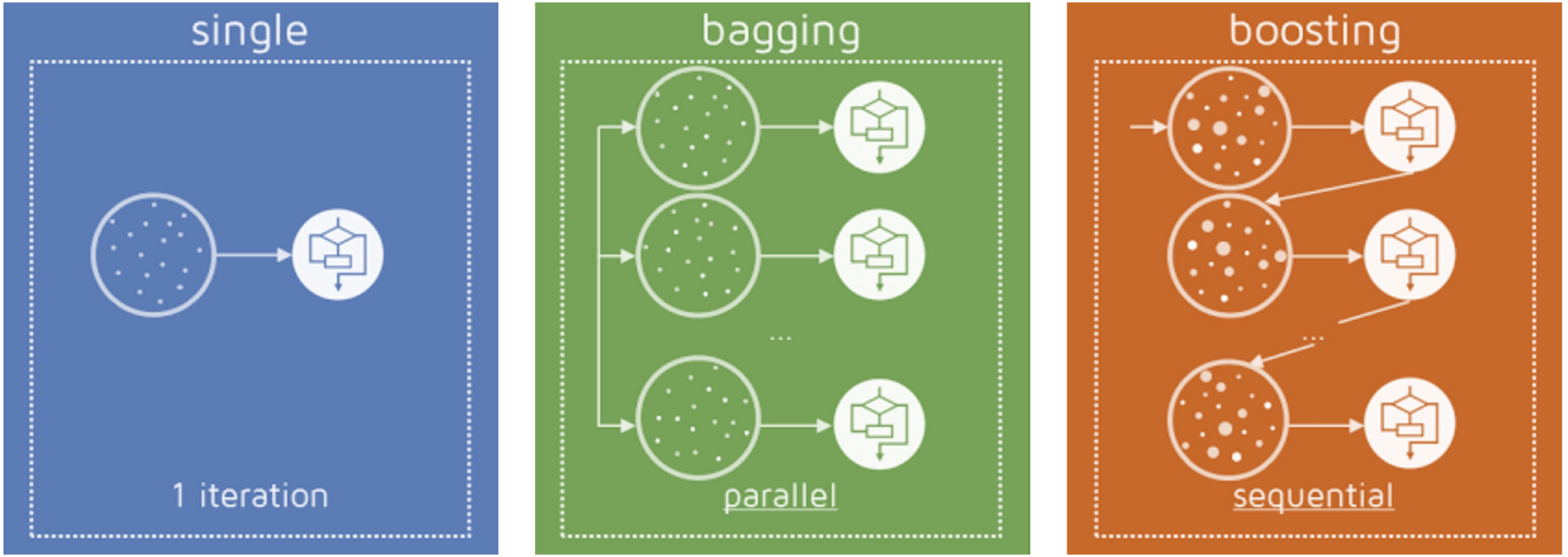


Comparing Ensembles: Data



Based on https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Comparing Ensembles: Training



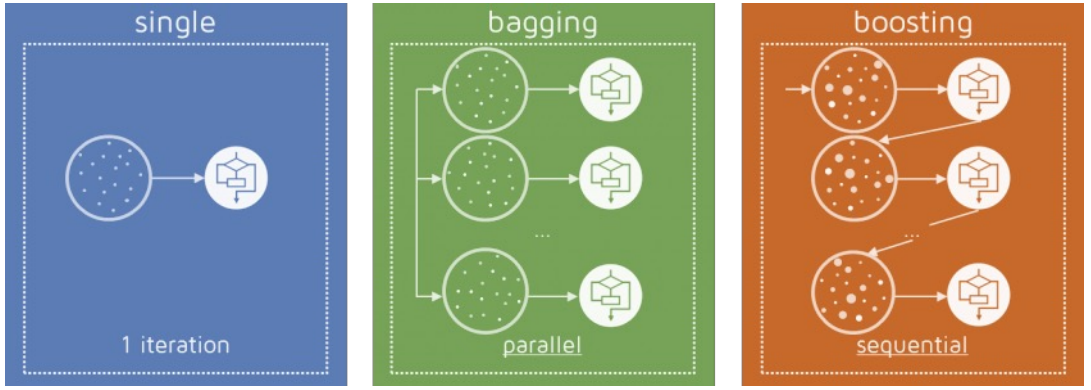
Based on https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Boosting vs Bagging

- Bagging is typically faster, but may get a smaller error reduction (not by much).
- Bagging works well with “reasonable” classifiers.
- Boosting works with very simple classifiers.
  - E.g., Boostexter - text classification using decision stumps based on single words.
- Boosting may have a problem if a lot of the data is mislabeled, because it will focus on those examples a lot, leading to overfitting.

Slide from Edith Law at U Waterloo

Recap



- Ensemble methods are a **must-try**

- Bagging: average models trained on bootstrap samples
- Good bagging: **Random Forests**

- Boosting: sequence of models that correct for remaining mistakes
- Great boosting: **xgboost**