

CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

BOOSTING

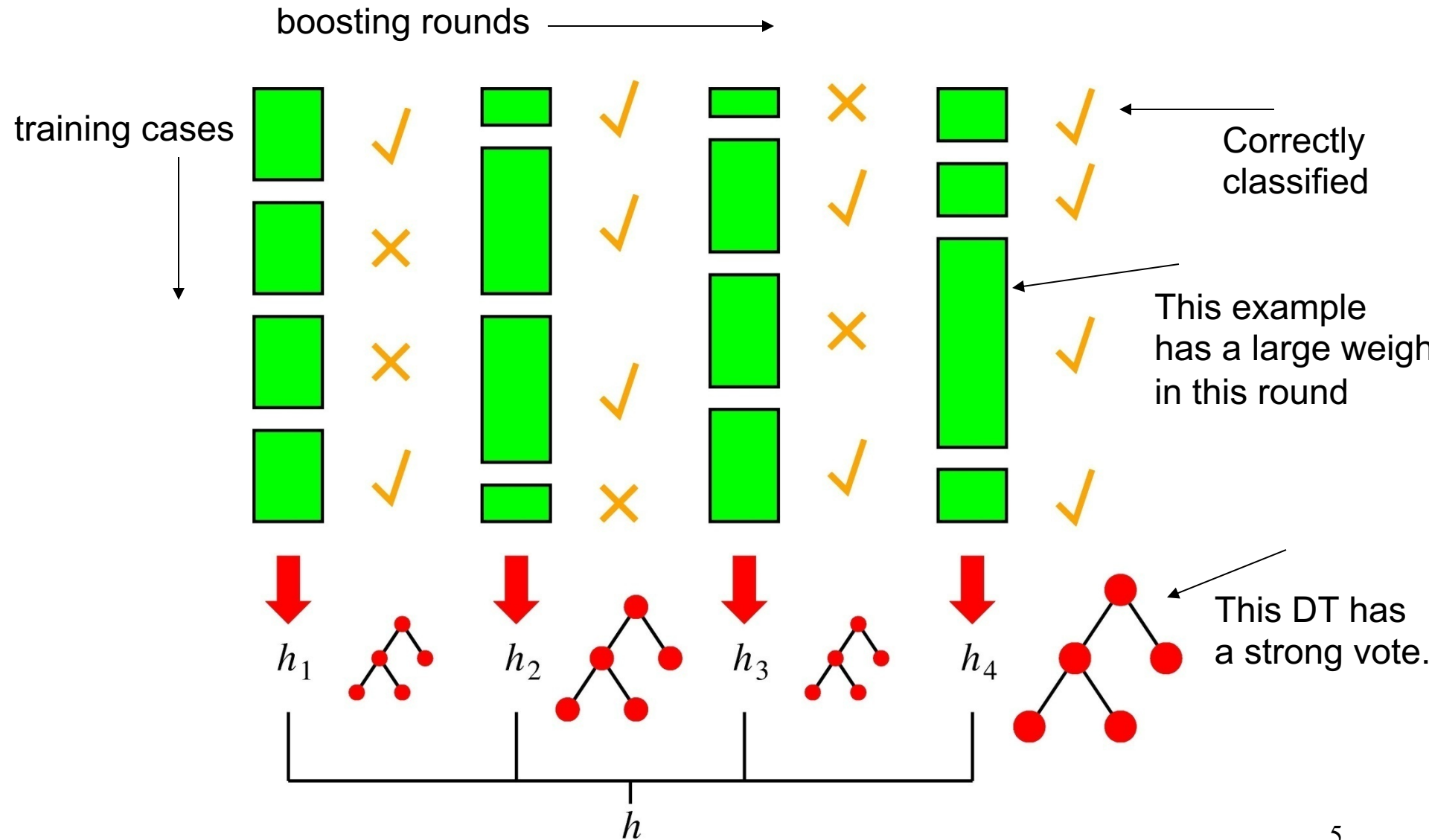
Boosting

- Train classifiers (e.g. decision trees) in a sequence.
- A new classifier should focus on those cases which were incorrectly classified in the last round.
- Combine the classifiers by letting them vote on the final prediction (like bagging).
- Each classifier is “weak” but the ensemble is “strong.”
- **AdaBoost** is a specific boosting method.

Boosting Intuition

- We adaptively weigh each data point.
- Data points which are wrongly classified get high weight (the algorithm will focus on them)
- Each boosting round learns a new (simple) classifier on the weighed dataset.
- These classifiers are weighed to combine them into a single powerful classifier.
- Classifiers that obtain low training error rate have high weight.
- We stop by using monitoring a hold out set (cross-validation).

Boosting in a Picture



Boosting: Adaboost

- Binary classification problem.
- Combining multiple “base” classifiers to come up with a “good” classifier.
- Base classifiers have to be “weak learners”, accuracy $> 50\%$
- Base classifiers are trained on a weighted training dataset.
- Boosting involves sequentially learning α_m and $y_m(x)$.

Adaboost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \dots, N$.
2. For $m = 1, \dots, M$:
 - (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \quad (14.15)$$

where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

- (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}} \quad (14.16)$$

and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\} \quad (14.17)$$

Adaboost (contd..)

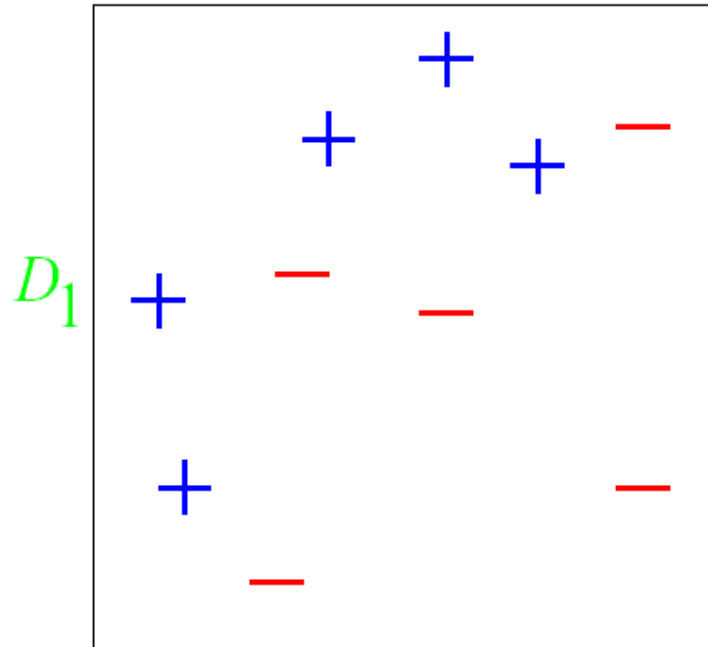
(c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(y_m(\mathbf{x}_n) \neq t_n) \}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right).$$

And in animation

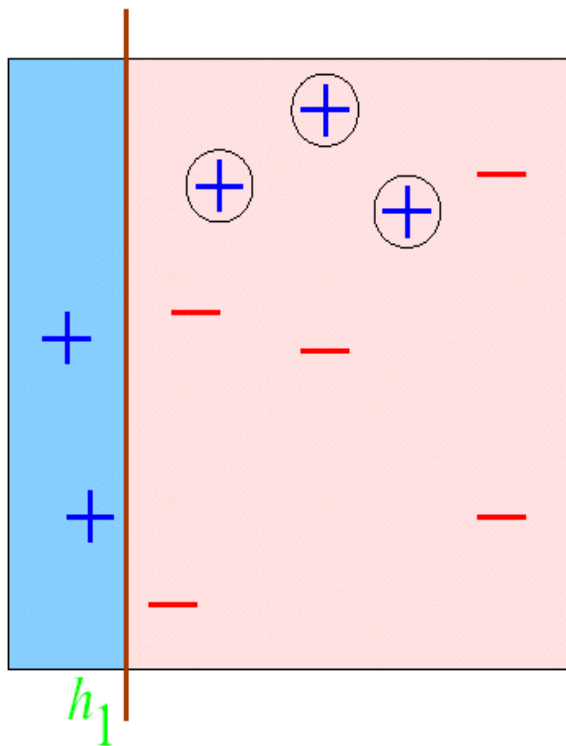


Original training set: equal weights to all training samples

AdaBoost example

ϵ = error rate of classifier
 α = weight of classifier

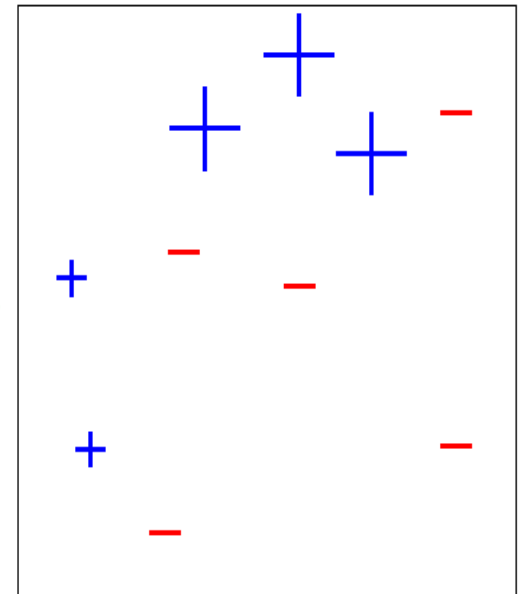
ROUND 1



$\epsilon_1 = 0.30$
 $\alpha_1 = 0.42$

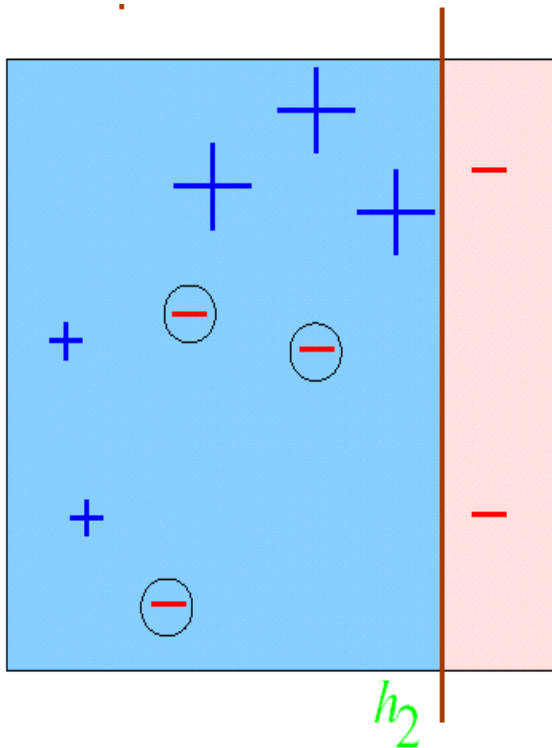


D_2



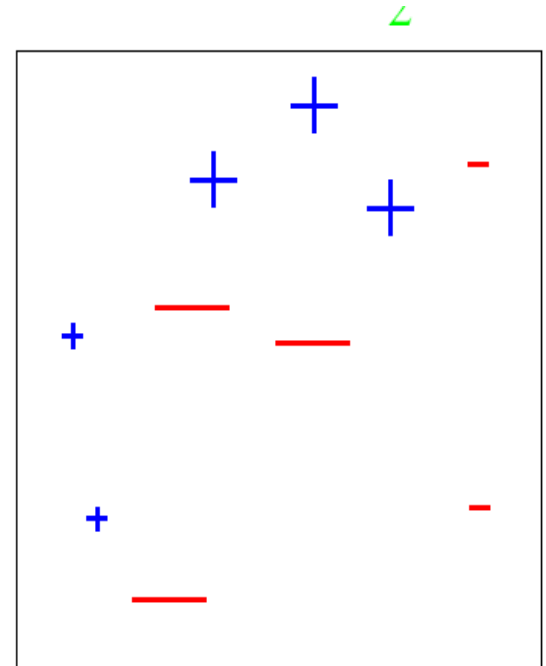
AdaBoost example

ROUND 2



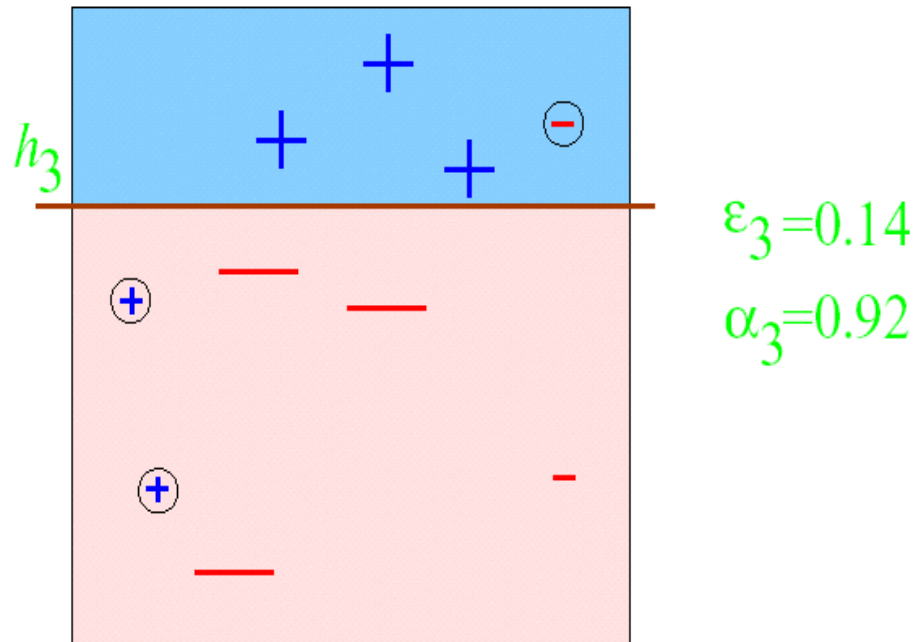
$$\begin{aligned}\epsilon_2 &= 0.21 \\ \alpha_2 &= 0.65\end{aligned}$$

D_3

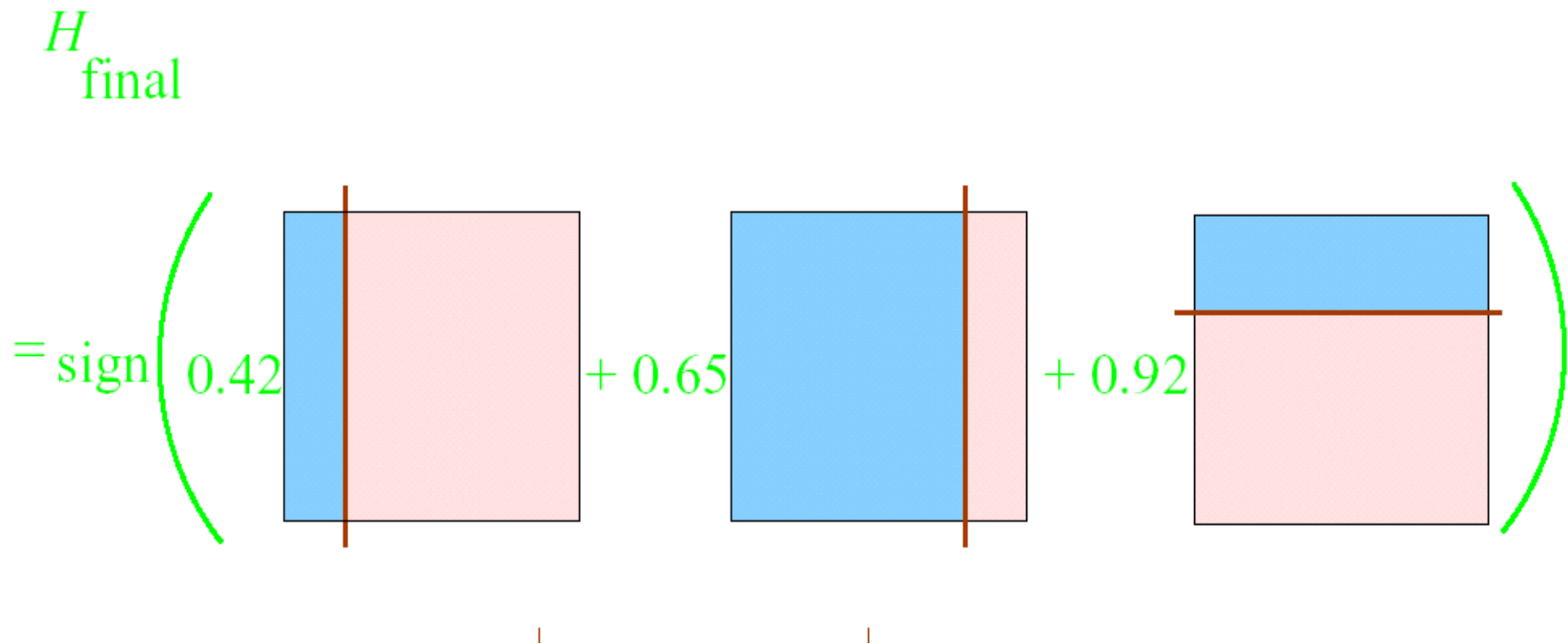


AdaBoost example

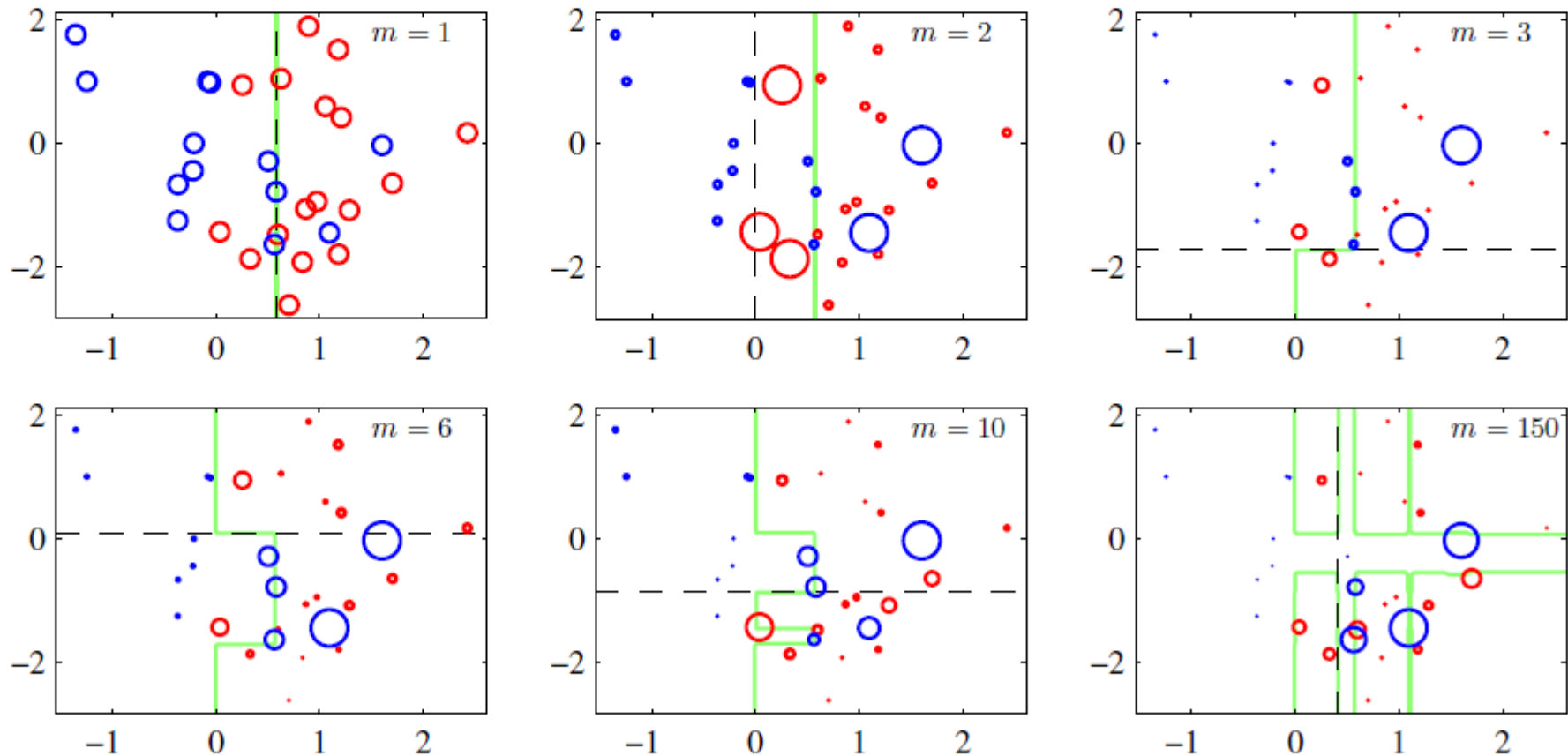
ROUND 3



AdaBoost example



Adaboost illustration



Adaboost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \dots, N$.
2. For $m = 1, \dots, M$:
 - (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) \quad (14.15)$$

where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

- (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}} \quad (14.16)$$

and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\} \quad (14.17)$$

Adaboost (contd..)

(c) Update the data weighting coefficients

$$w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(y_m(\mathbf{x}_n) \neq t_n) \}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right).$$

Adaboost - Observations

- ϵ_m : weighted error $\in [0, 0.5)$
- $\alpha_m \geq 0$
- w_i^{m+1} is higher than w_i^m by a factor $(1 - \epsilon_m) / \epsilon_m$, when i is misclassified.

Adaboost - derivation

- Consider the error function:

$$E = \sum_{n=1}^N \exp \{-t_n f_m(\mathbf{x}_n)\}$$

- Where

$$f_m(\mathbf{x}) = \frac{1}{2} \sum_{l=1}^m \alpha_l y_l(\mathbf{x})$$

- Goal: Minimize E w.r.t. α_l and $y_l(x)$, sequentially.

Adaboost - derivation

- Minimize w.r.t. α_m

$$\begin{aligned} E &= \sum_{n=1}^N \exp \left\{ -t_n f_{m-1}(\mathbf{x}_n) - \frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n) \right\} \\ &= \sum_{n=1}^N w_n^{(m)} \exp \left\{ -\frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n) \right\} \end{aligned}$$

- Let τ_m be the set of datapoints correctly classified by y_m .

$$\begin{aligned} E &= e^{-\alpha_m/2} \sum_{n \in \tau_m} w_n^{(m)} + e^{\alpha_m/2} \sum_{n \in \mathcal{M}_m} w_n^{(m)} \\ &= (e^{\alpha_m/2} - e^{-\alpha_m/2}) \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n) + e^{-\alpha_m/2} \sum_{n=1}^N w_n^{(m)}. \end{aligned}$$

Adaboost - derivation

- Minimizing w.r.t. y_m and α_m , we get the updates 2(a) and 2(b).
- We can see that:

$$w_n^{(m+1)} = w_n^{(m)} \exp \left\{ -\frac{1}{2} t_n \alpha_m y_m(\mathbf{x}_n) \right\}.$$

$$t_n y_m(\mathbf{x}_n) = 1 - 2I(y_m(\mathbf{x}_n) \neq t_n)$$

- Using:
- We get:

$$w_n^{(m+1)} = w_n^{(m)} \exp(-\alpha_m/2) \exp \{ \alpha_m I(y_m(\mathbf{x}_n) \neq t_n) \}.$$

Adaboost

1. Initialize the data weighting coefficients $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \dots, N$.
2. For $m = 1, \dots, M$:
 - (a) Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)$$

where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals 1 when $y_m(\mathbf{x}_n) \neq t_n$ and 0 otherwise.

- (b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}$$

and then use these to evaluate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}.$$

Adaboost (contd..)

(c) Update the data weighting coefficients

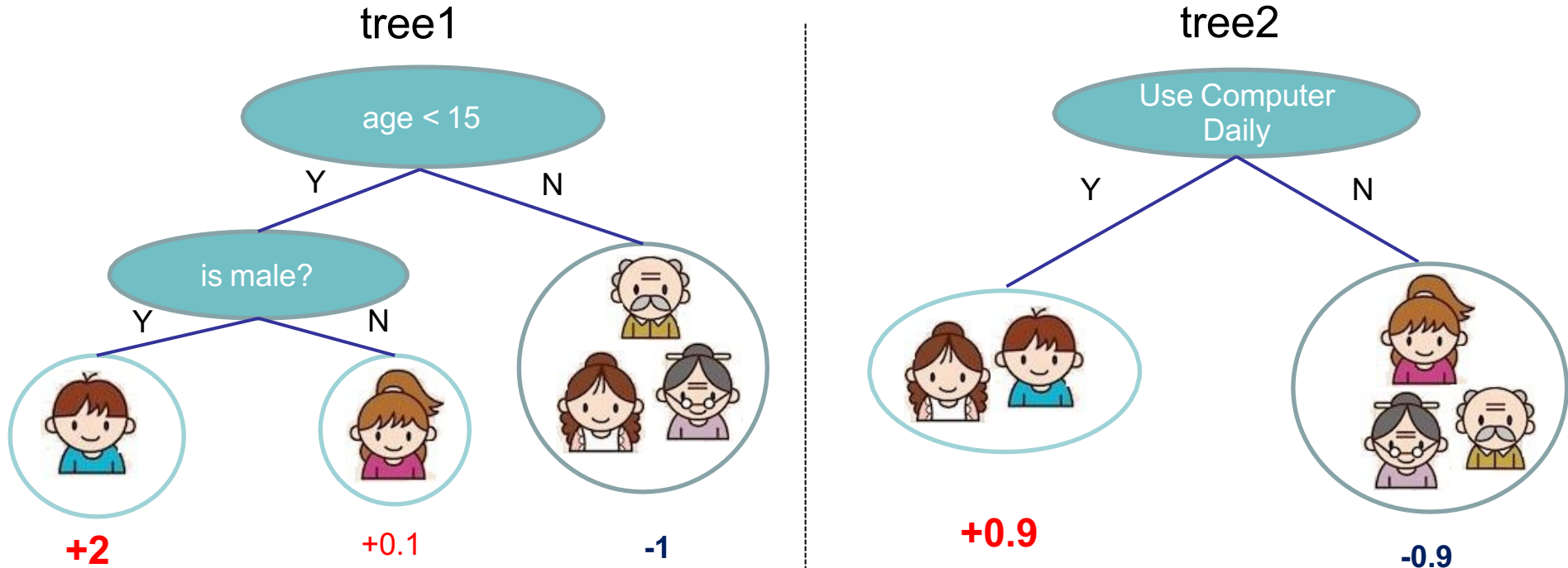
$$w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(y_m(\mathbf{x}_n) \neq t_n) \}$$

3. Make predictions using the final model, which is given by

$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right).$$

XGBOOST

Regression Tree Ensemble



$$f(\text{boy icon}) = 2 + 0.9 = 2.9$$

$$f(\text{old man icon}) = -1 - 0.9 = -1.9$$

Prediction of is sum of scores predicted by each of the tree

Tree Ensemble methods

- Very widely used, look for GBM, random forest...
 - Almost half of data mining competition are won by using some variants of tree ensemble methods
- Invariant to scaling of inputs, so you do not need to do careful features normalization.
- Learn higher order interaction between features.
- Can be scalable, and are used in Industry

Put into context: Model and Parameters

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

Space of functions containing all Regression trees

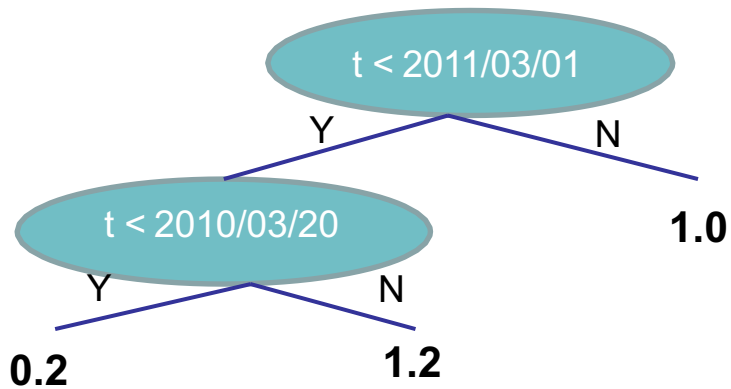
Think: regression tree is a function that maps the attributes to the score

- Parameters
 - Including structure of each tree, and the score in the leaf
 - Or simply use function as parameters
$$\Theta = \{f_1, f_2, \dots, f_K\}$$
 - Instead learning weights in \mathbf{R}^d , we are learning functions(trees)

Learning a tree on single variable

- How can we learn functions?
- Define objective (loss, regularization), and optimize it!!
- Example:
 - Consider regression tree on single input t (time)
 - I want to predict whether I like romantic music at time t

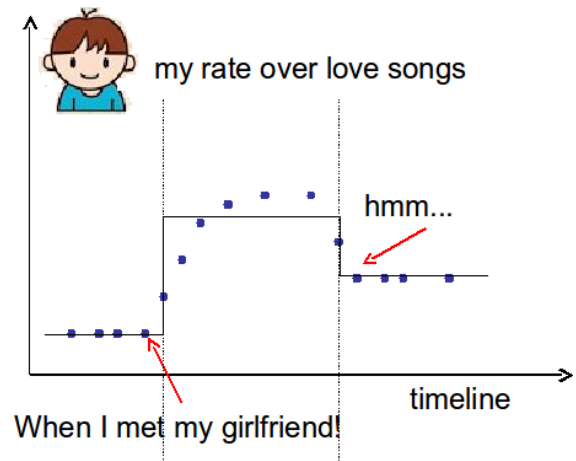
The model is regression tree that splits on time



Equivalently

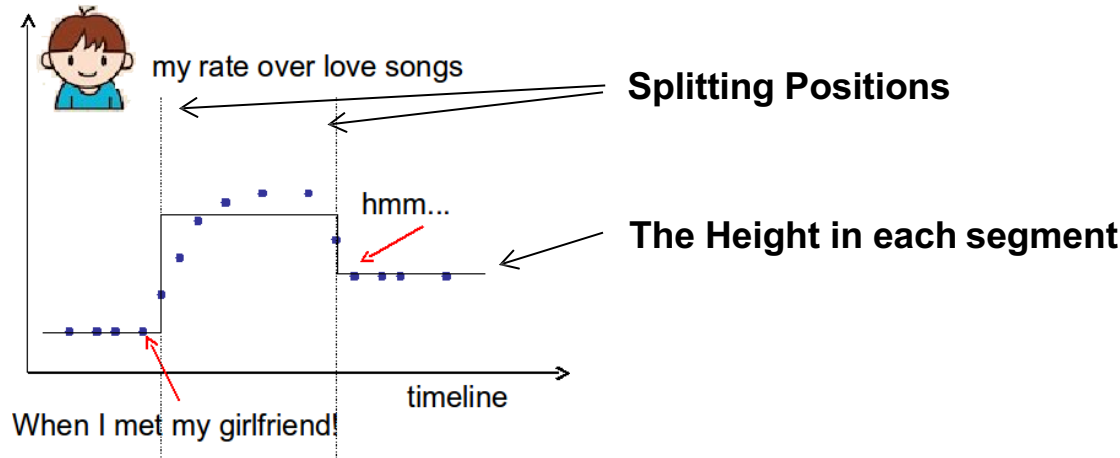


Piecewise step function over time



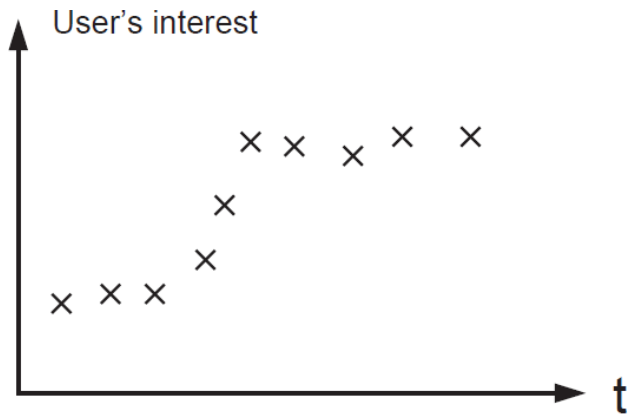
Learning a step function

- Things we need to learn

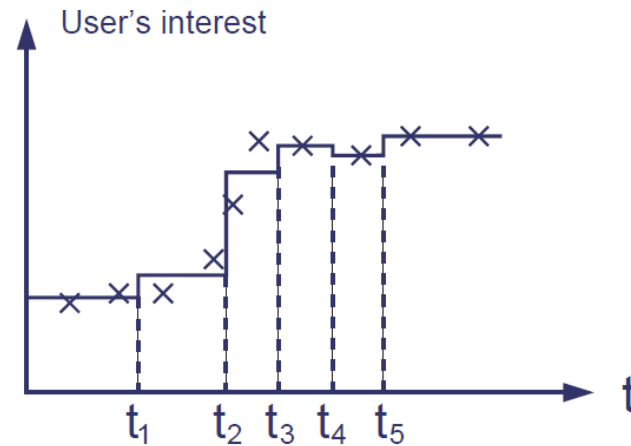


- Objective for single variable regression tree(step functions)
 - Training Loss: How will the function fit on the points?
 - Regularization: How do we define complexity of the function?
 - ◆ Number of splitting points, l_2 norm of the height in each segment?

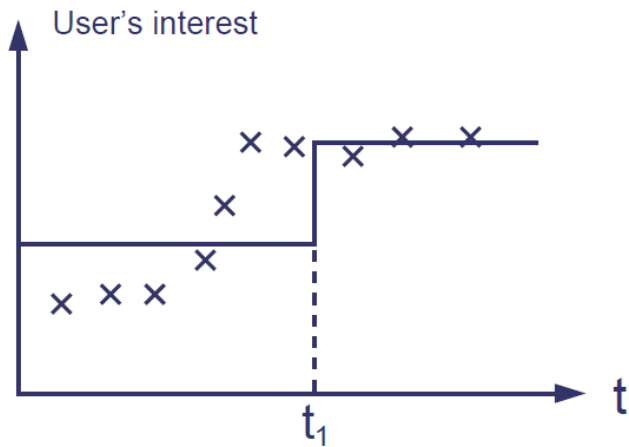
Learning step function (visually)



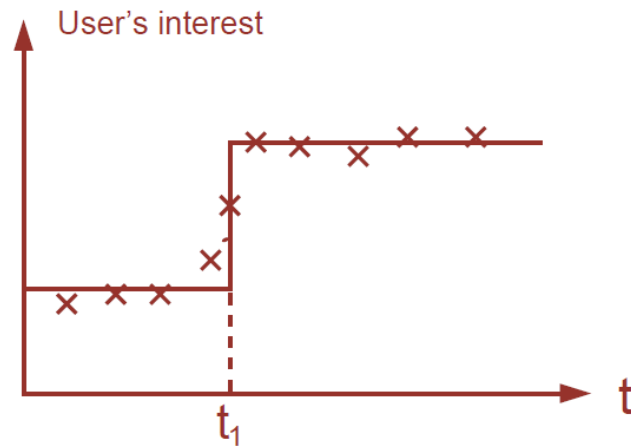
Observed user's interest on topic k against time t



Too many splits, $\Omega(f)$ is high



Wrong split point, $L(f)$ is high



Good balance of $\Omega(f)$ and $L(f)$

Coming back: Objective for Tree Ensemble

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

- Objective

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

Training loss

Complexity of the Trees

- Possible ways to define Ω ?
 - Number of nodes in the tree, depth
 - L2 norm of the leaf weights
 - ...detailed later

Objective vs Heuristic

- When you talk about (decision) trees, it is usually heuristics
 - Split by information gain
 - Prune the tree
 - Maximum depth
 - Smooth the leaf values
- Most heuristics maps well to objectives, taking the formal (objective) view let us know what we are learning
 - Information gain -> training loss
 - Pruning -> regularization defined by #nodes
 - Max depth -> constraint on the function space
 - Smoothing leaf values -> L2 regularization on leaf weights

Regression Tree is not just for regression!

- Regression tree ensemble defines how you make the prediction score, it can be used for
 - Classification, Regression, Ranking...
 -
- It all depends on how you define the objective function!
- So far we have learned:
 - Using Square loss $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
 - ♦ Will result in common gradient boosted machine
 - Using Logistic loss $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
 - ♦ Will results in LogitBoost

Outline

- Review of key concepts of supervised learning
- Regression Tree and Ensemble (What are we Learning)
- **Gradient Boosting (How do we Learn)**
- Summary

Take Home Message for this section

- Bias-variance tradeoff is everywhere
- The loss + regularization objective pattern applies for regression tree learning (function learning)
- We want **predictive** and **simple** functions
- This defines what we want to learn (objective, model).
- But how do we learn it?
 - Next section

So How do we Learn?

- 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 SGD, to find f (since they are trees, instead of just numerical vectors)
- Solution: **Additive Training (Boosting)**
 - Start from constant prediction, add a new function each time

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

New function

Model at training round t

Keep functions added in previous round

Additive Training

- How do we decide which f_t 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 $Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$
 - Seems still complicated except for the case of square loss
- 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 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

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

- *If you are not comfortable with this, think of square loss*

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

- Compare what we get to previous slide



Our New Goal

- Objective, with constants removed

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

- where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$
- Why spending so much effort to derive the objective, why not just grow trees ...
 - Theoretical benefit: know what we are learning, convergence
 - **Engineering** benefit, recall the elements of supervised learning
 - ♦ g_i and h_i comes from definition of loss function
 - ♦ The learning of function only depend on the objective via g_i and h_i
 - ♦ Think of how you can separate modules of your code when you are asked to implement boosted tree for both square loss and logistic loss

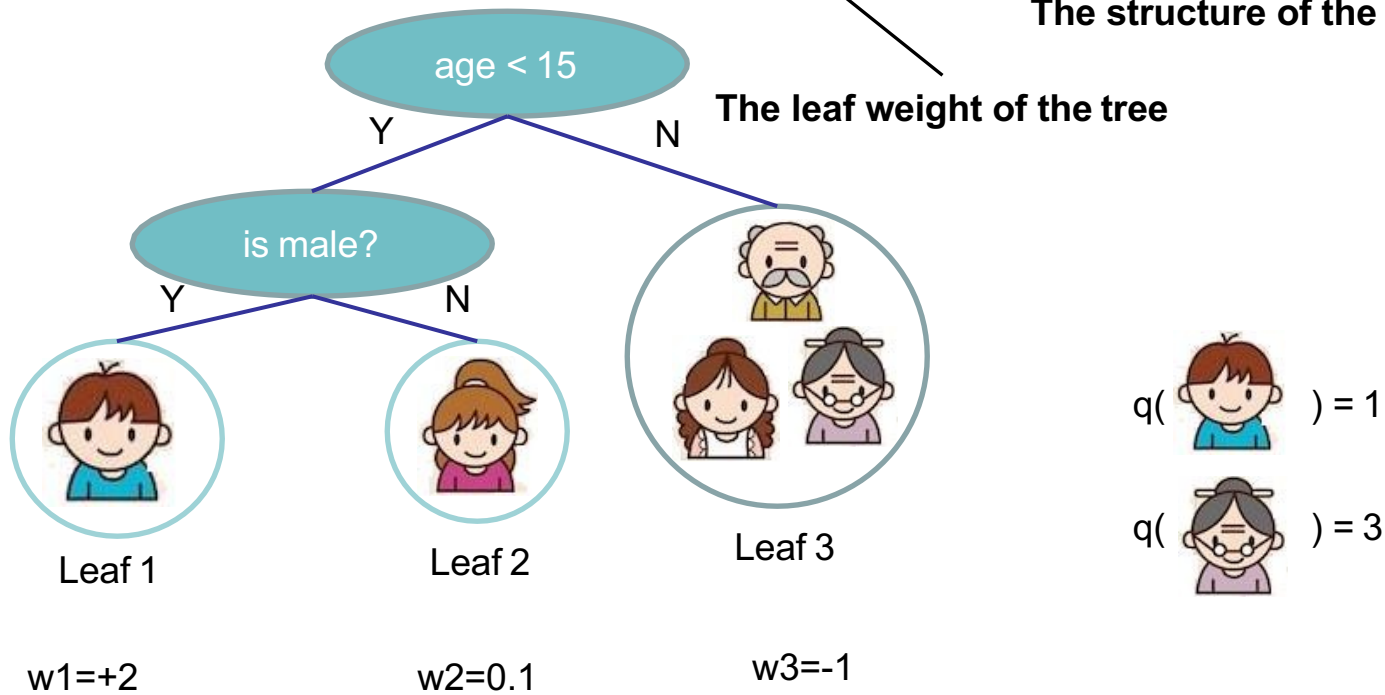
Refine the definition of tree

- We define tree by a vector of scores in leafs, and a leafindex mapping function that maps an instance to a leaf

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

The structure of the tree

The leaf weight of the tree



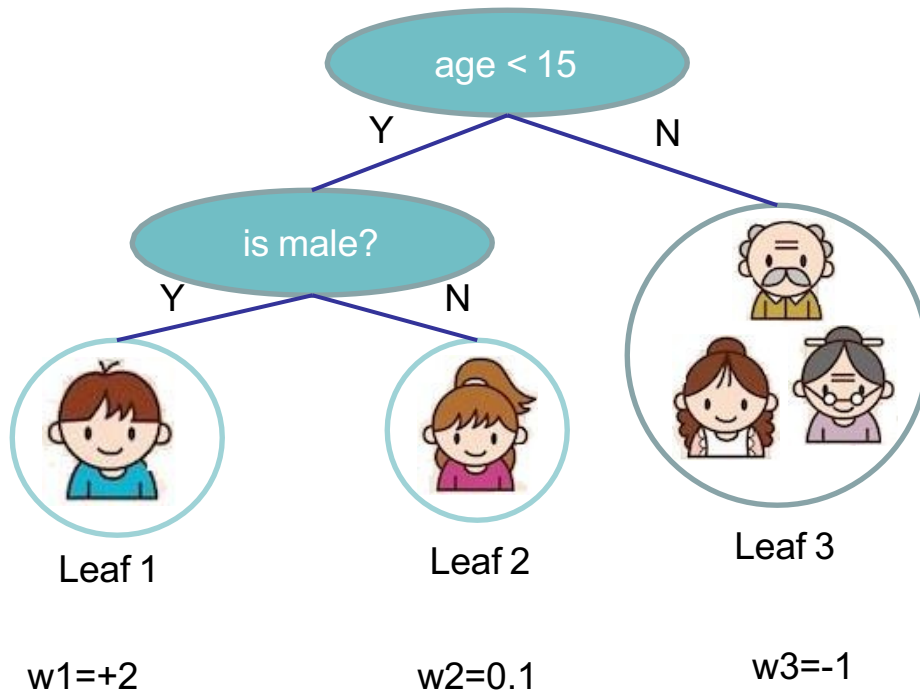
Define Complexity of a Tree

- Define complexity as (this is not the only possible definition)

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

Number of leaves

L2 norm of leaf scores



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

Revisit the Objectives

- Define the instance set in leaf j as $I_j = \{i | q(x_i) = j\}$
- Regroup the objective by each leaf

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

- This is sum of T independent quadratic functions

The Structure Score

- Two facts about single variable quadratic function

$$\operatorname{argmin}_x Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}, \quad H > 0 \quad \min_x Gx + \frac{1}{2}Hx^2 = -\frac{1}{2}\frac{G^2}{H}$$

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

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

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

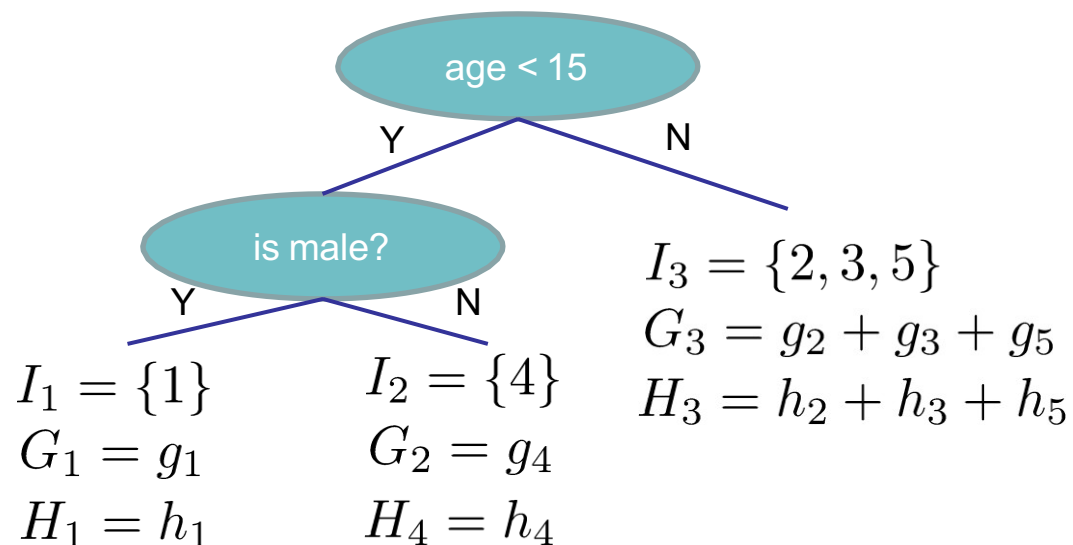
$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad \operatorname{Obj} = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

 This measures how good a tree structure is!

The Structure Score Calculation

Instance index gradient statistics

1		g_1, h_1
2		g_2, h_2
3		g_3, h_3
4		g_4, h_4
5		g_5, 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

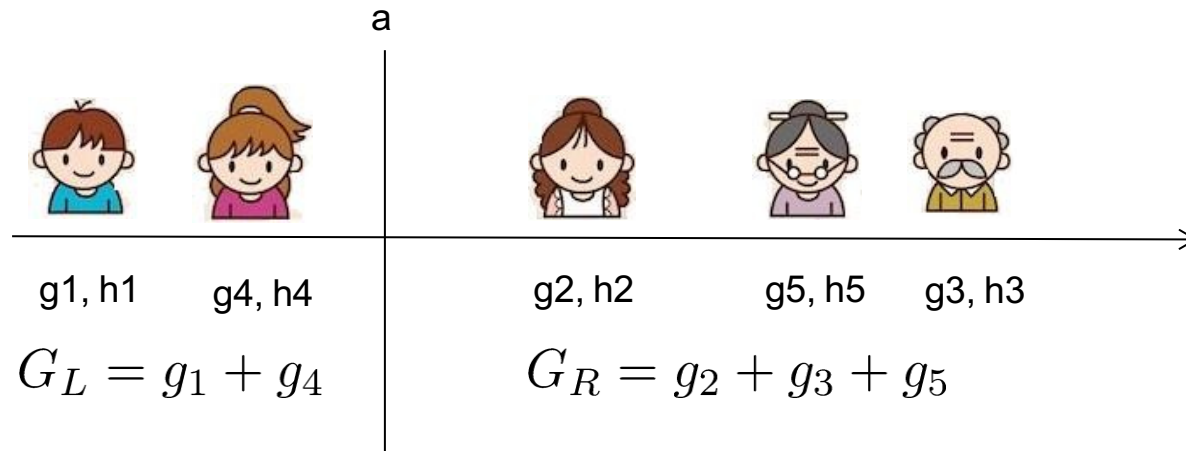
$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

the score of left child the score of right child the score of if we do not split The complexity cost by introducing additional leaf

- 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



- All we need is sum of g and h in each side, and calculate

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

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

An Algorithm for Split Finding

- For each node, enumerate over all features
 - For each feature, sorted the instances by feature value
 - Use a linear scan to decide the best split along that feature
 - Take the best split solution along all the features
- Time Complexity growing a tree of depth K
 - It is $O(n d K \log n)$: or each level, need $O(n \log n)$ time to sort
There are d features, and we need to do it for K level
 - This can be further optimized (e.g. use approximation or caching the sorted features)
 - Can scale to very large dataset

What about Categorical Variables?

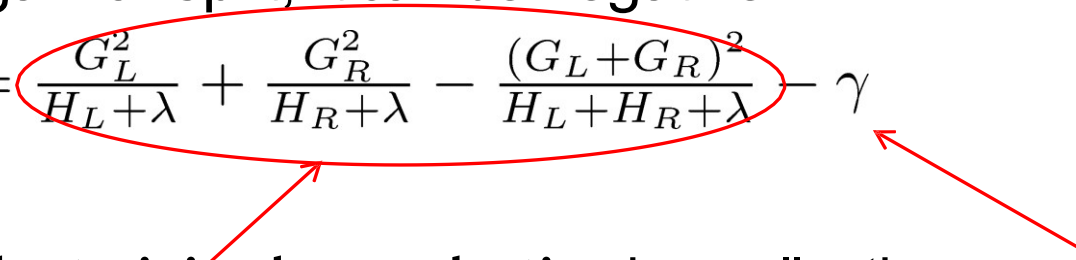
- Some tree learning algorithm handles categorical variable and continuous variable separately
 - We can easily use the scoring formula we derived to score split based on categorical variables.
- Actually it is not necessary to handle categorical separately.
 - We can encode the categorical variables into numerical vector using one-hot encoding. Allocate a *#categorical* length vector

$$z_j = \begin{cases} 1 & \text{if } x \text{ is in category } j \\ 0 & \text{otherwise} \end{cases}$$

- The vector will be sparse if there are lots of categories, the learning algorithm is preferred to handle sparse data

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-Pruning
 - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

Recap: Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

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

- Use the statistics to greedily grow a tree $f_t(x)$

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

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
 - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
 - ϵ is called step-size or shrinkage, usually set around 0.1
 - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting