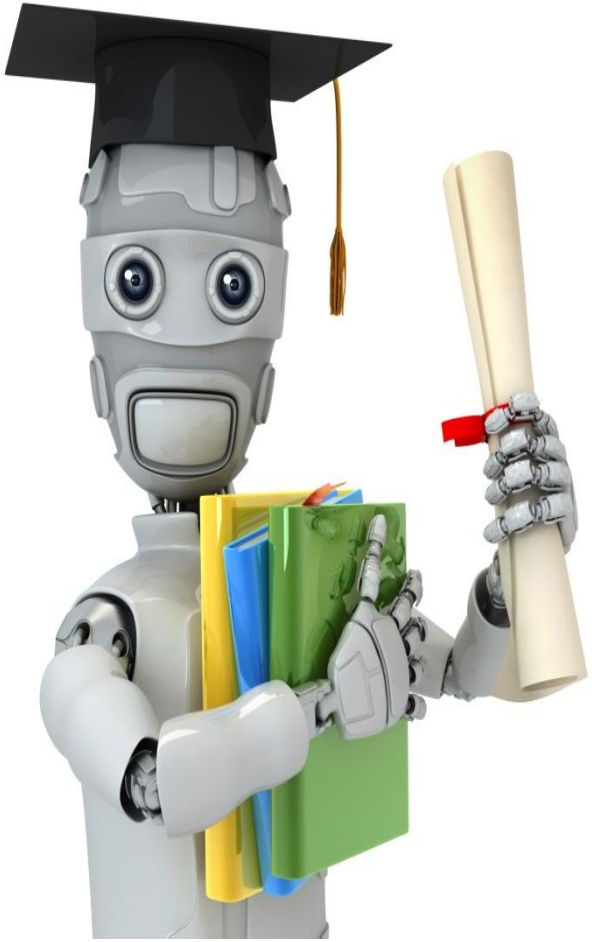


Tree-Based Methods

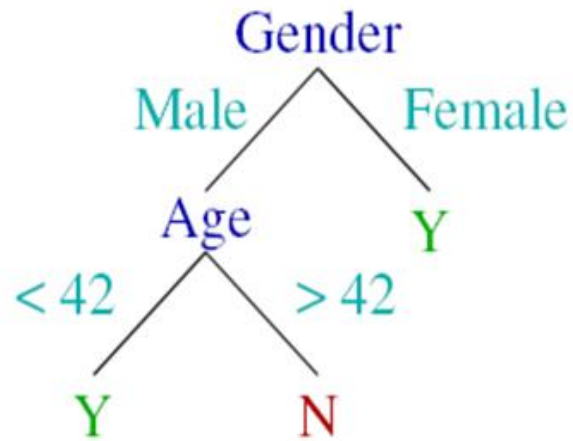
Decision Tree

Machine Learning

Main idea

- Decision tree: a flow-chart like tree structure
  - Each internal node represents a test
  - Training instances are split at each internal node
  - Branch represents an outcome of a test
  - Leaf nodes represent class label or class distribution

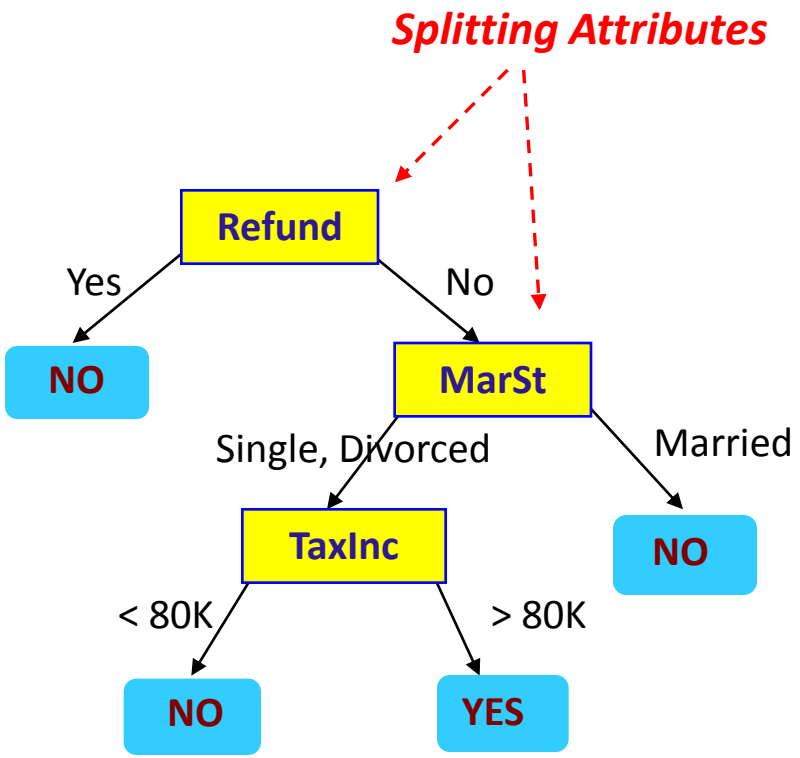
- Greedy algorithm



Example of a Decision Tree

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	*categorical*	*continuous*	*class*
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

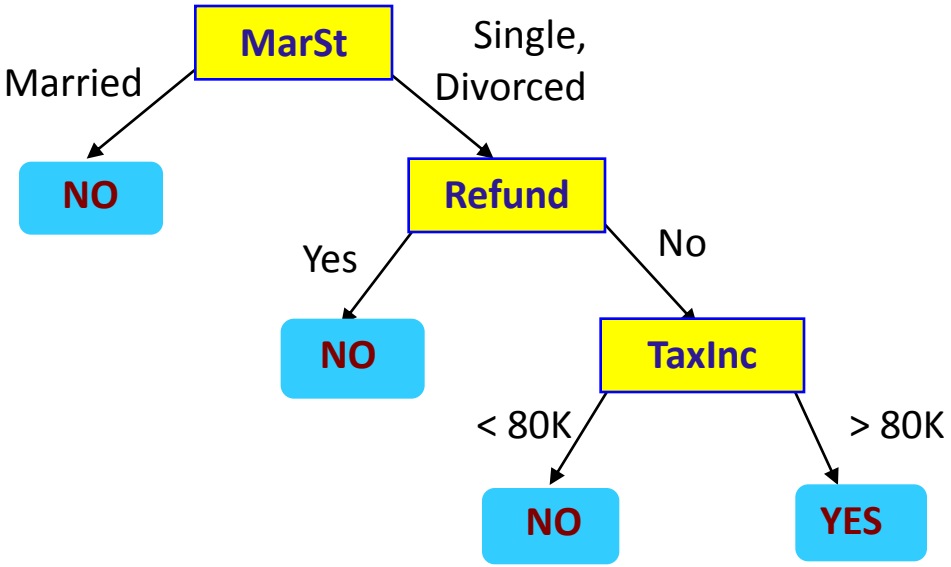
Training Data



Model: Decision Tree

Another Example of Decision Tree

Tid	Refund (categorical)	Marital Status (categorical)	Taxable Income (continuous)	Cheat (class)
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



There could be more than one tree that fits the same data!

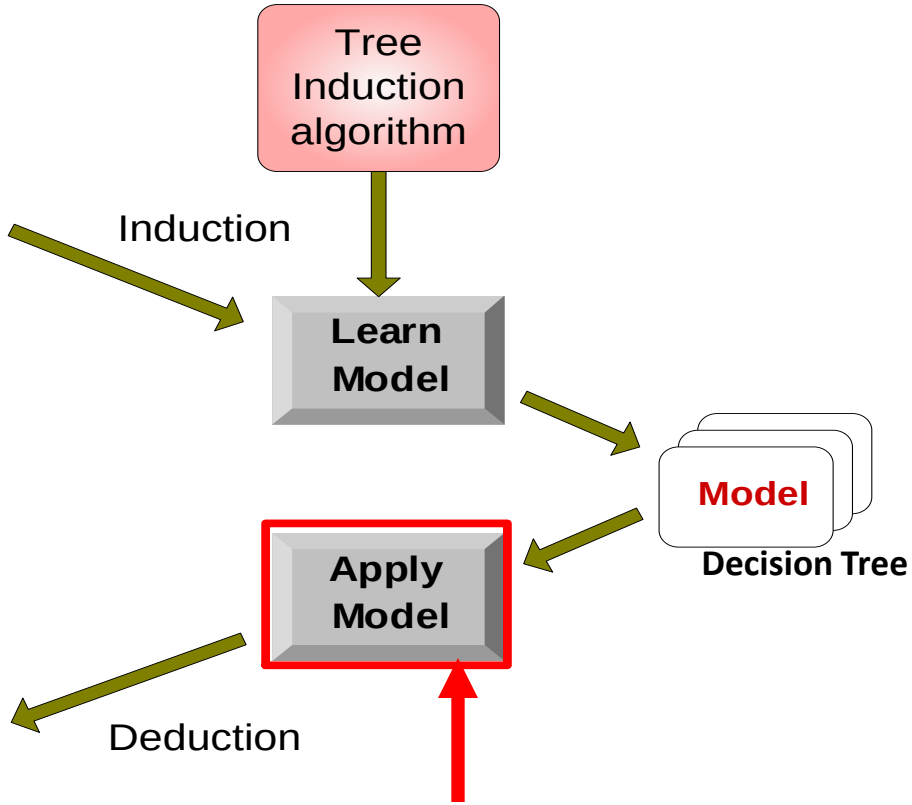
Decision Tree Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

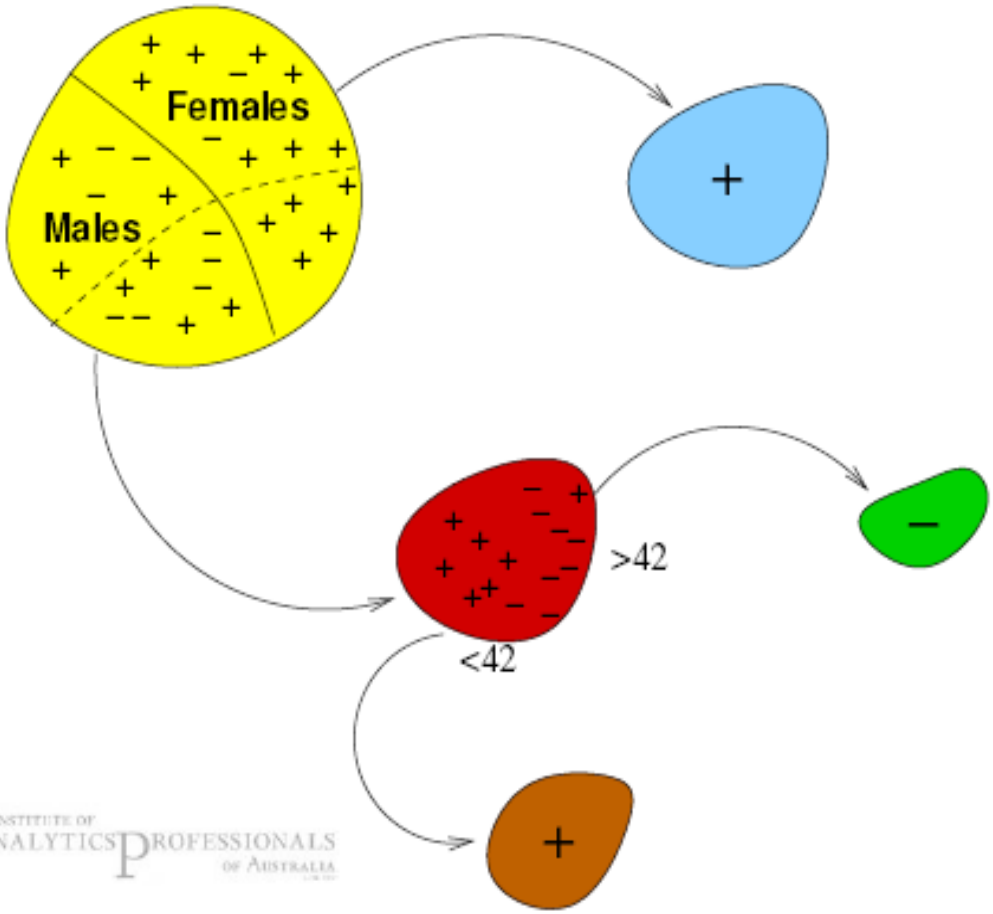
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

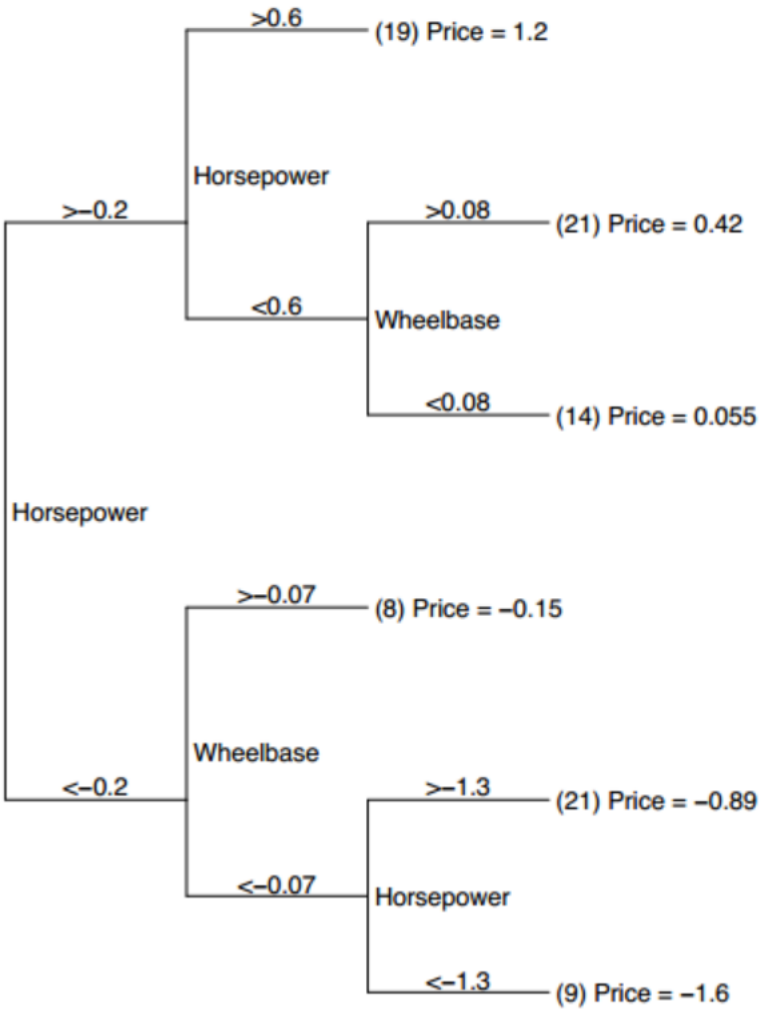


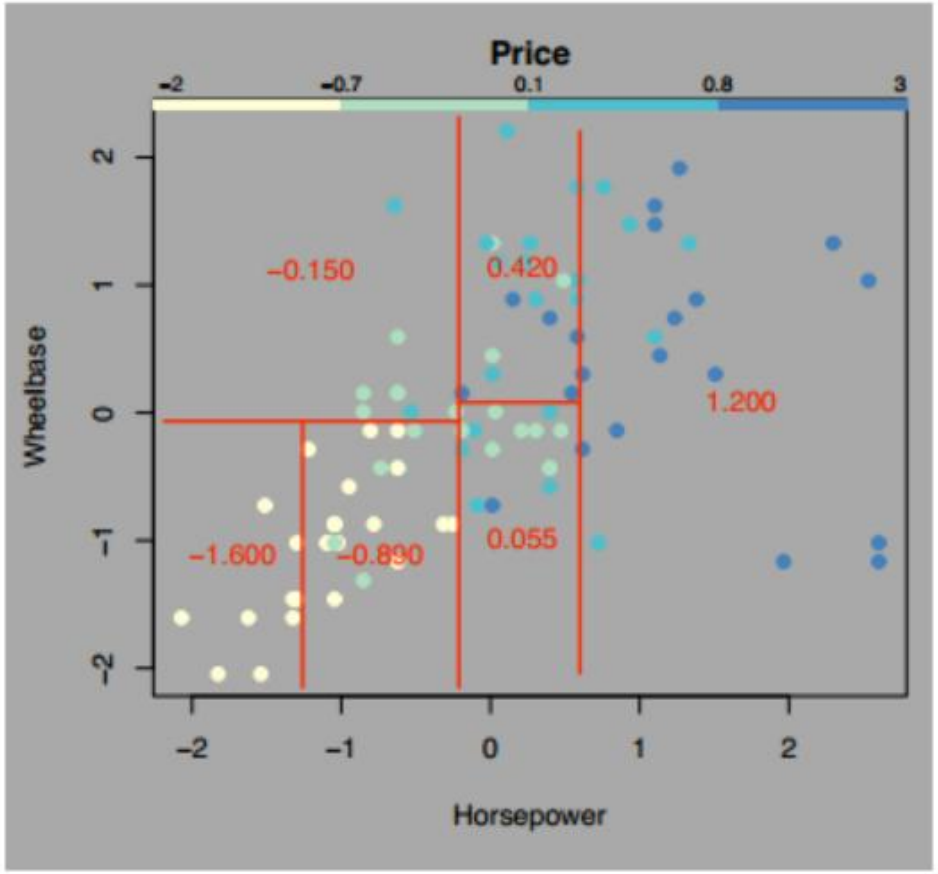
Tree Construction: Divide and Conquer

- Decision tree induction is an example of a recursive partitioning algorithm: divide and conquer.
- At start, all the training examples are at the root
- Partition examples recursively based on selected attributes



Regression Tree





Decision tree representation

- Each internal node is a test:
  - Theoretically, a node can test multiple features
  - In most systems, a node tests exactly one feature
- Each branch corresponds to test results
  - A branch corresponds to an feature value or a range of feature values
- Each leaf node assigns
  - a class: classification tree
  - a real value: regression tree

What’s the best decision tree?

- “Best”: You need a bias (e.g., prefer the “smallest” tree): least depth? Fewest nodes? Which trees are the best predictors of unseen data?

- Occam's Razor: we prefer the simplest hypothesis that fits the data.

➔ Find a decision tree that is as small as possible and fits the data

Finding a smallest decision tree

- A decision tree can represent any discrete function of the inputs: $y=f(x_1, x_2, \ldots, x_n)$
  - How many functions are there assuming all the features are binary?

- The space of decision trees is too big for systemic search for a smallest decision tree.

- Solution: greedy algorithm

Basic algorithm: top-down induction

1. Find the “best” decision feature, A, and assign A as decision feature for node

2. For each value of A, create a new branch, and divide up training samples

3. Repeat the process 1-2 until the gain is small enough

Major issues

Q1: Choosing best attribute: what quality measure to use?

Q2: Determining when to stop splitting: avoid overfitting

Q3: Handling continuous features

Q1: What quality measure

- Information gain
- Gini index
- Variance

Information gain

- The expectation of any classification of samples

  $\mathrm{I}(s_1, s_2, \ldots, s_m) = -\sum_{i=1}^{m} P_i \log_2(P_i)$ (i = 1..m)

  S : training set m : the number of classes

  P_i : the proportion of samples in S whose category is c_i (S_i) $p_i = \frac{|S_i|}{|S|}$

- The entropy of subsets divided by A

  $\mathrm{E(A)} = \sum_{j=1}^{V} (s_{1j} + \cdots + s_{mj})/s * I(s_{1j}, \ldots, s_{mj})$

  A: the feature, has V different values

- Information gain

$$\mathrm{Gain(A)} = I(s_1, \ldots, s_m) - \mathrm{E(A)}$$

An example

S=[9+,5-]
E=0.940

Income

High → [3+,4-]
E=0.985

Low → [6+,1-]
E=0.592

S=[9+,5-]
E=0.940

PrevCust

Yes → [6+,2-]
E=0.811

No → [3+,3-]
E=1.00

InfoGain (S, Income)
=0.940-(7/14)*0.985-(7/14)*0.592
=0.151

InfoGain(S, Wind)
=0.940-(8/14)*0.811-(6/14)*1.0
=0.048

Choose the A with the max information gain

Gini Index

- The set T contains N category record ,and the Gini is:

  $\mathrm{gini(T)} = 1 - \sum_{j=1}^{N} P_j^2$

  P_j : the probability of category j

- After the first division, the set T is divided into m parts, N1, N2…Nm. The split gini is

  $$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \cdots + \frac{N_m}{N} gini(T_m)$$

- Choose the A with the min gini index

Examples for Computing GINI

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

  - Maximum (1 - 1/n_c) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

Examples for computing GINI

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Gini = 1 – P(C1)^2 – P(C2)^2 = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Gini = 1 – (1/6)^2 – (5/6)^2 = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – (2/6)^2 – (4/6)^2 = 0.444

Splitting Based on GINI

- Used in CART, SLIQ, SPRINT.
- When a node p is split into k partitions (children), the quality of split is computed as,

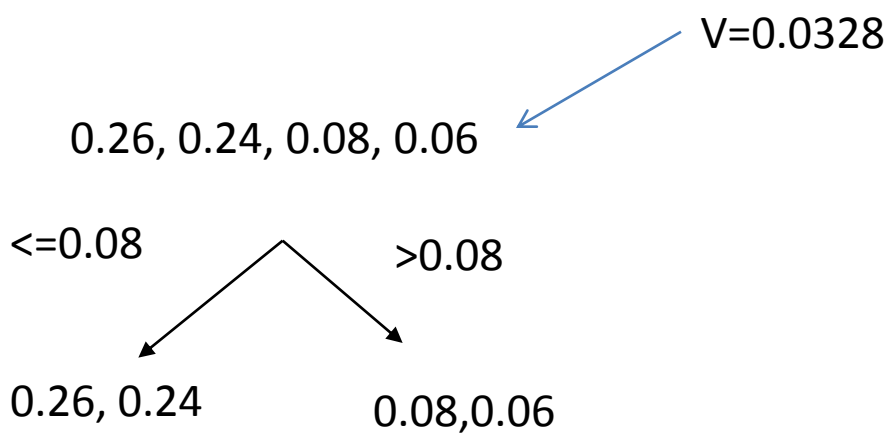
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at node p.

Variance (for continuous values in regression trees)

V=\sum_i (叶子i里的样本数/总样本数)×叶子i里的样本的方差



2/4 ×0.0001+2/4×0.0001=0.0001

Q2:How to avoiding overfitting

- Stop growing the tree earlier. E.g., stop when
  - InfoGain < threshold
  - Size of examples in a node < threshold
  - Depth of the tree > threshold
  - All the records in a leaf belong to the same class
  - All the records in a leaf node have similar attribute values
  - …

- Grow full tree, then post-prune

➔ Less practical due to expensive computational costs

Performance measure

- Accuracy:
  - on validation data
  - K-fold cross validation

- Misclassification cost: Sometimes more accuracy is desired for some classes than others.

- MDL(最小描述长度): size(tree) + errors(tree)

Q3: handling numeric attributes

- Continuous attribute ➔ discrete attribute
- Example
  - Original attribute: Temperature = 82.5
  - New attribute: (temperature > 72.3) = t, f

➔ Question: how to choose split points?

Choosing split points for a continuous attribute

- Sort the examples according to the values of the continuous attribute.

- For classification trees, Identify adjacent examples that differ in their target labels and attribute values ➔ a set of candidate split points. Calculate the gain for each split point and choose the one with the highest gain.

- For regression trees, just use each value as a split point and choose the one with the smallest varaince.

Summary of Major issues

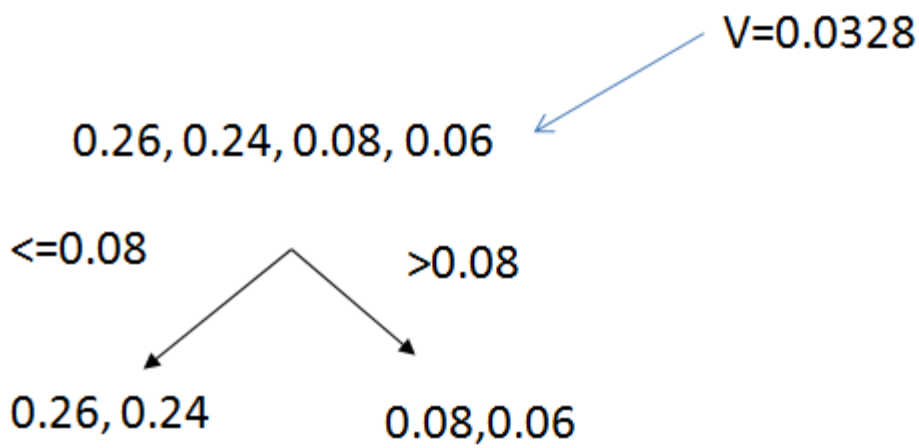
Q1: Choosing best attribute: different quality measures.

Q2: Determining when to stop splitting: stop earlier or post-pruning

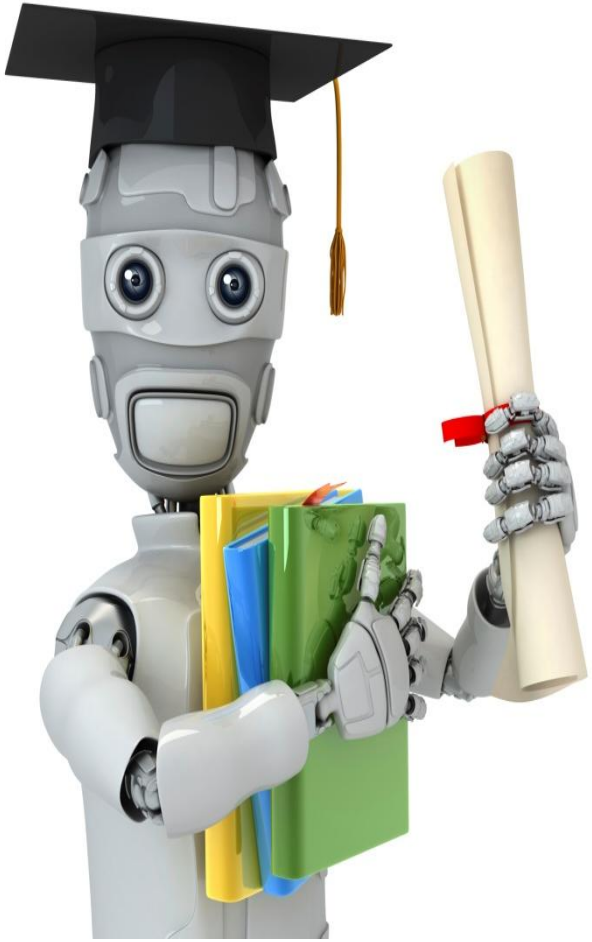
Q3: Handling continuous attributes: find the breakpoints

Highlights

Regression trees with continuous values are the most widely-used trees in ensemble methods for classification (e.g. Random forest, Gradient Boosted Decision Trees).



$2/4 \times 0.0001 + 2/4 \times 0.0001 = 0.0001$



Tree-Based Methods

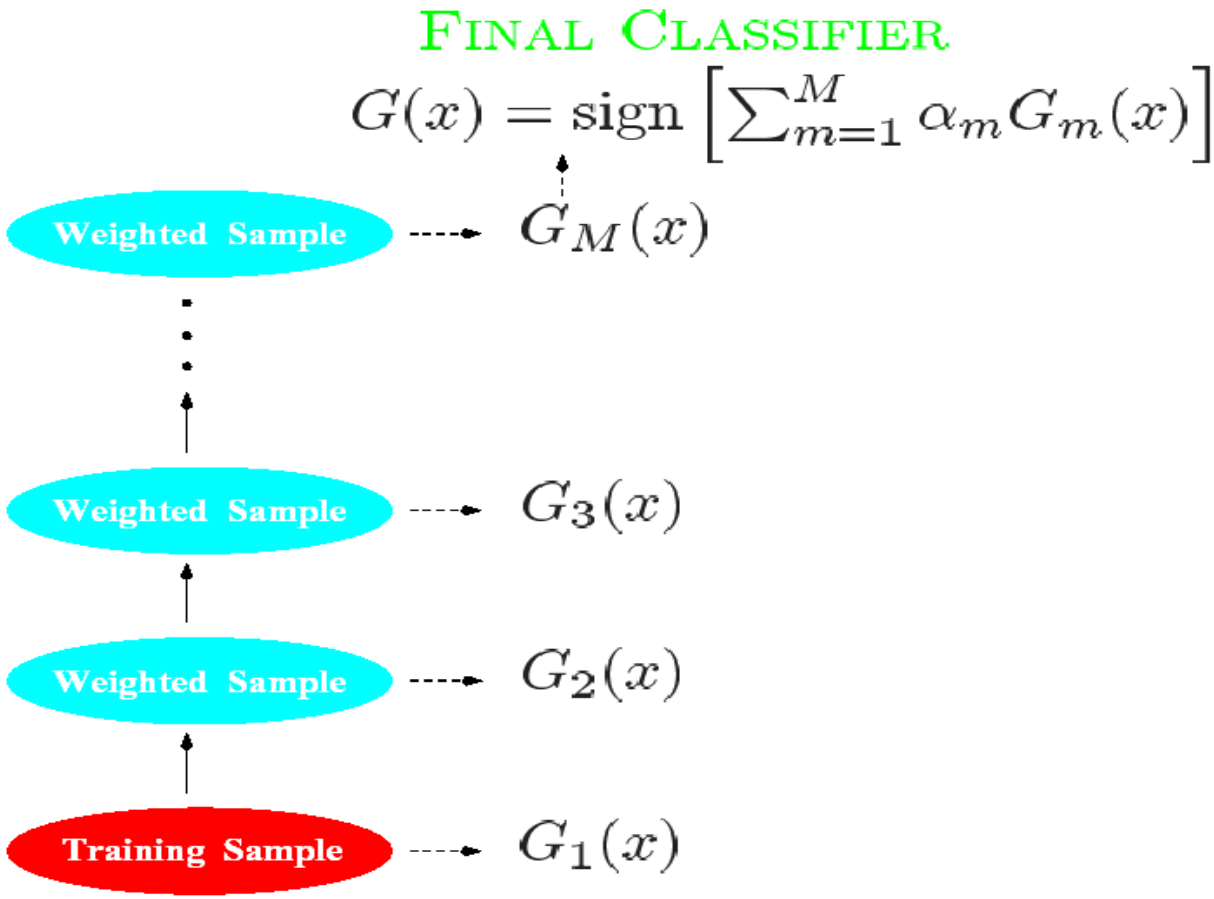
Boosting

Machine Learning

Boosting

- Boosting is considered to be one of the most significant developments in machine learning
- Finding many weak rules of thumb is easier than finding a single, highly prediction rule
- Key in combining the weak rules

Boosting (Algorithm)



Elements in Supervised Learning

- Notations: $x_i \in \mathbf{R}^d$ i-th training example
- **Model**: how to make prediction $\hat{y}_i$ given x_i
  - Linear model: $\hat{y}_i = \sum_j w_j x_{ij}$ (include linear/logistic regression)
  - The prediction score $\hat{y}_i$ can have different interpretations depending on the task
    - Linear regression: $\hat{y}_i$ is the predicted score
    - Logistic regression: $1/(1 + exp(-\hat{y}_i))$ is predicted the probability of the instance being positive
    - Others… for example in ranking $\hat{y}_i$ can be the rank score
- **Parameters**: the things we need to learn from data
  - Linear model: $\Theta = \{w_j | j = 1, \cdots, d\}$

Elements continued: Objective Function

- Objective function that is everywhere

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

Training Loss measures how well model fit on training data

Regularization, measures complexity of model

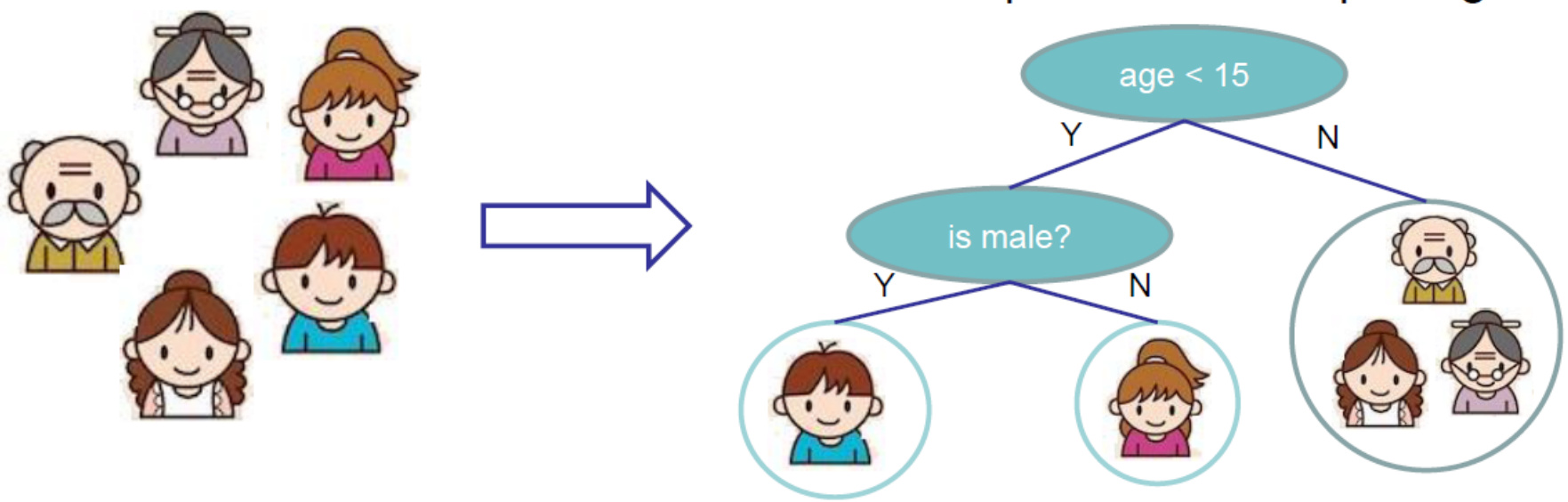
- Loss on training data: $L = \sum_{i=1}^{n} l(y_i, \hat{y}_i)$
  - Square loss: $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
  - Logistic loss: $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
- Regularization: how complicated the model is?
  - L2 norm: $\Omega(w) = \lambda \|w\|^2$
  - L1 norm (lasso): $\Omega(w) = \lambda \|w\|_1$

Regression Tree (CART)

- regression tree (also known as classification and regression tree):
  - Decision rules same as in decision tree
  - Contains one score in each leaf value

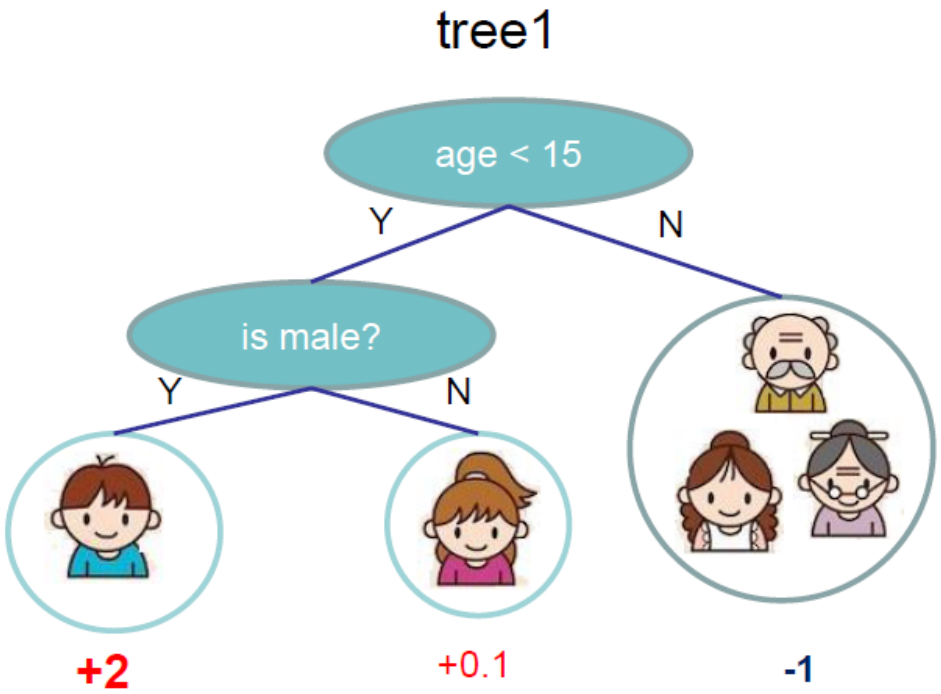
Input: age, gender, occupation, …

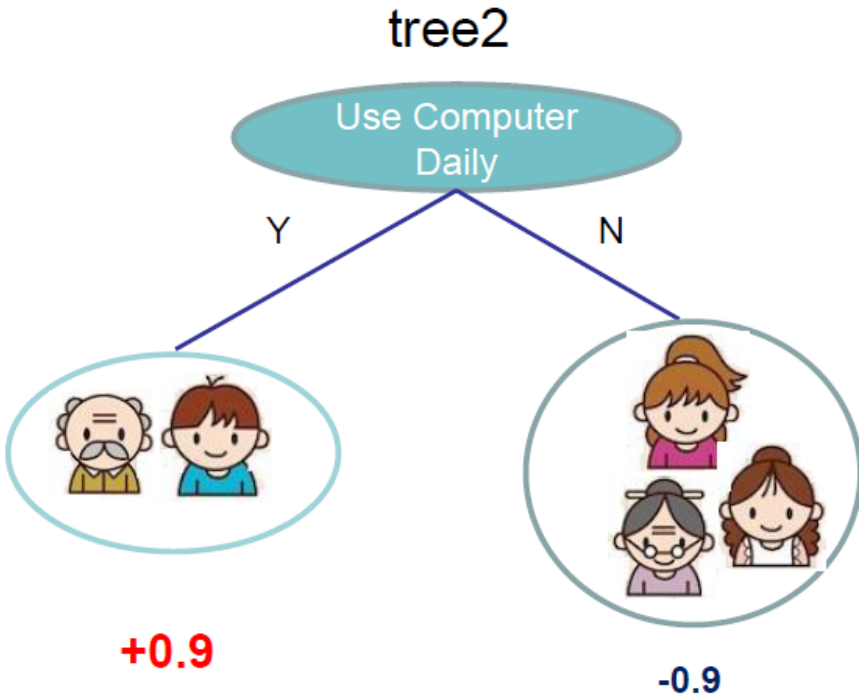
Does the person like computer games



prediction score in each leaf → +2 +0.1 -1

Regression Tree Ensemble





f() = 2 + 0.9= 2.9 f()= -1 + 0.9= -0.1

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

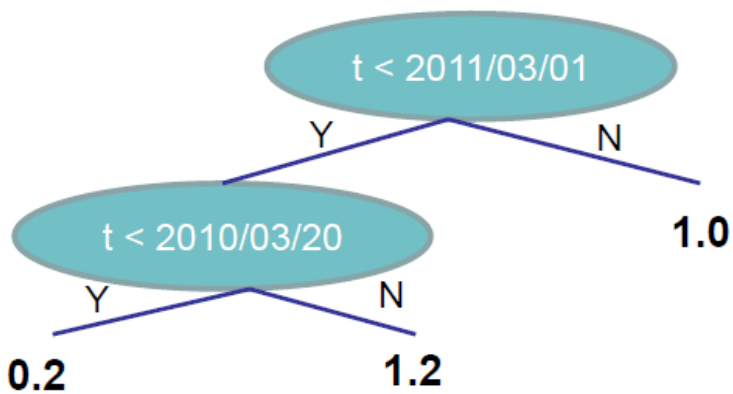
$$\Theta = \{f_1, f_2, \cdots, f_K\}$$

  - Instead learning weights in $\mathbf{R}^d$, we are learning functions(trees)

Learning a tree on single variable

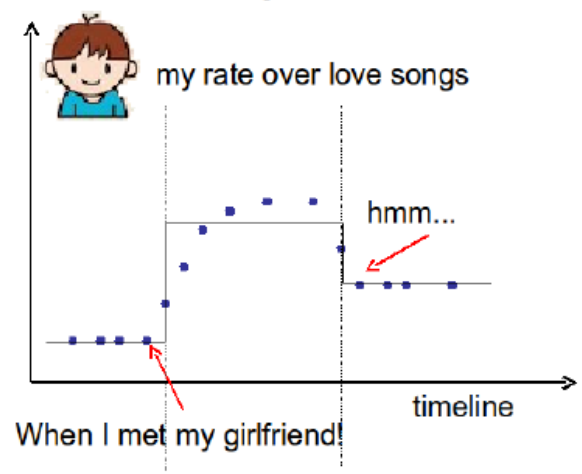
- How can we learn functions?
- Define objective (loss, regularization), and optimize it!!
- Example:
  - Consider regression tree on single input t (time)
  - I want to predict whether I like romantic music at time t

The model is regression tree that splits on time



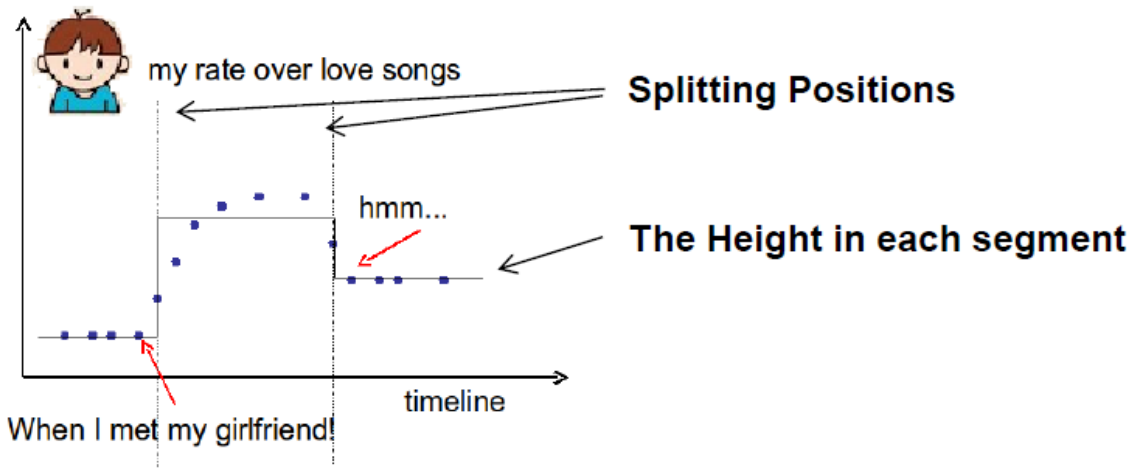
Equivalently

Piecewise step function over time



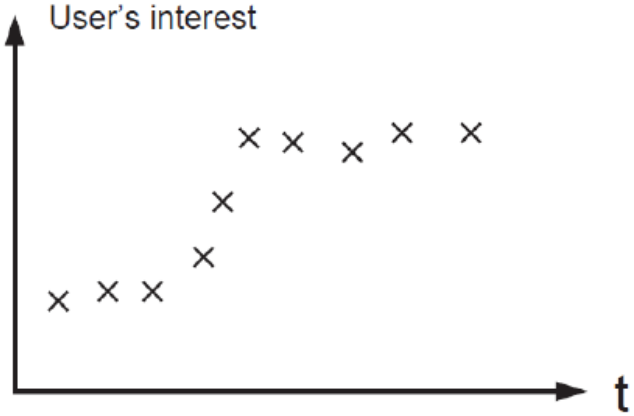
Learning a step function

- Things we need to learn

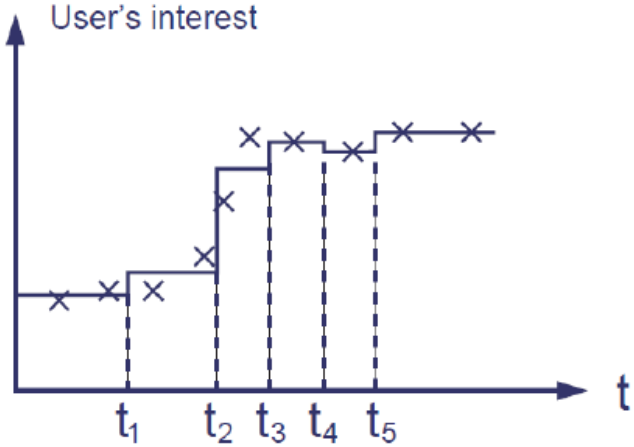


- Objective for single variable regression tree(step functions)
  - Training Loss: How will the function fit on the points?
  - Regularization: How do we define complexity of the function?
    - Number of splitting points, l2 norm of the height in each segment?

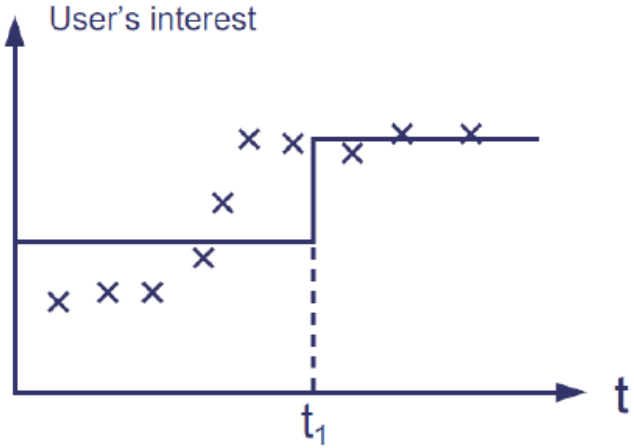
Learning step function (visually)



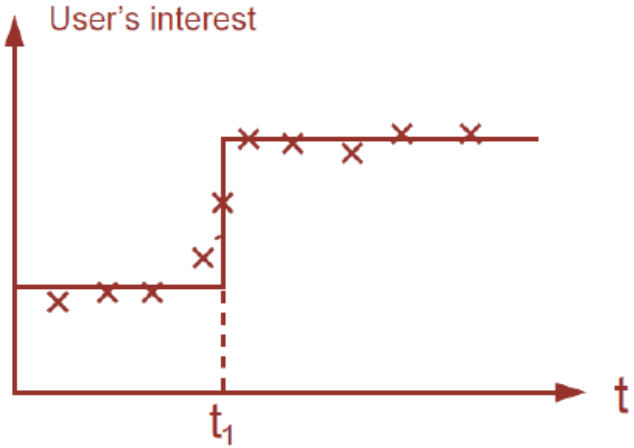
Observed user's interest on topic k against time t



☒ 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 results 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

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

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

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 $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - 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 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

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

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

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

- Why spending s much efforts 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

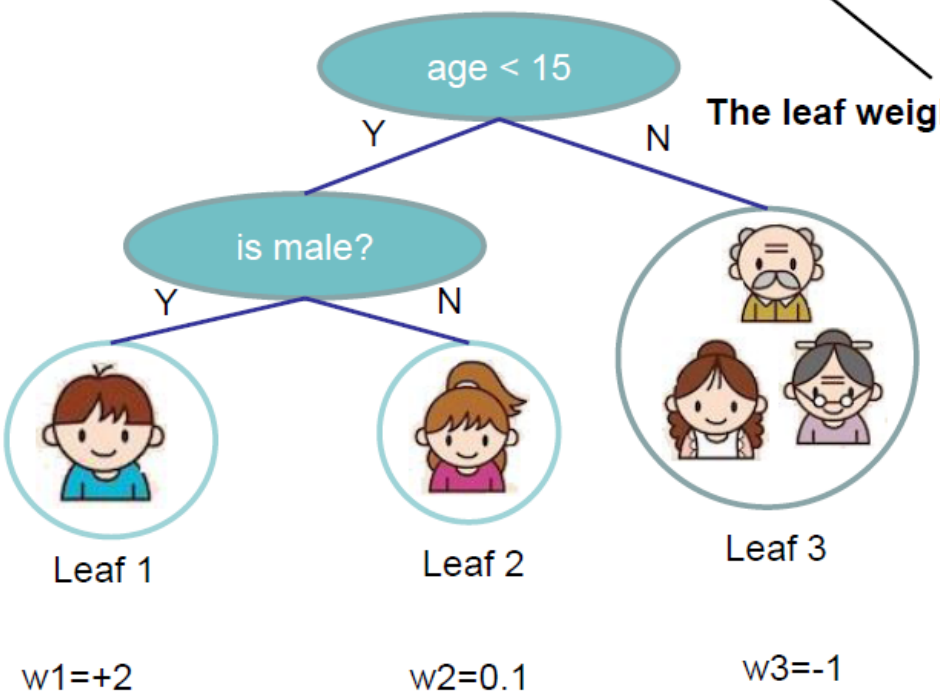
Refine the definition of tree

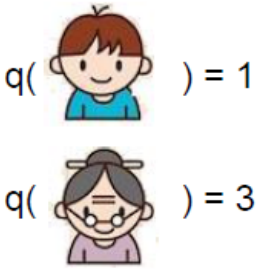
- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

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

The structure of the tree

The leaf weight of the tree



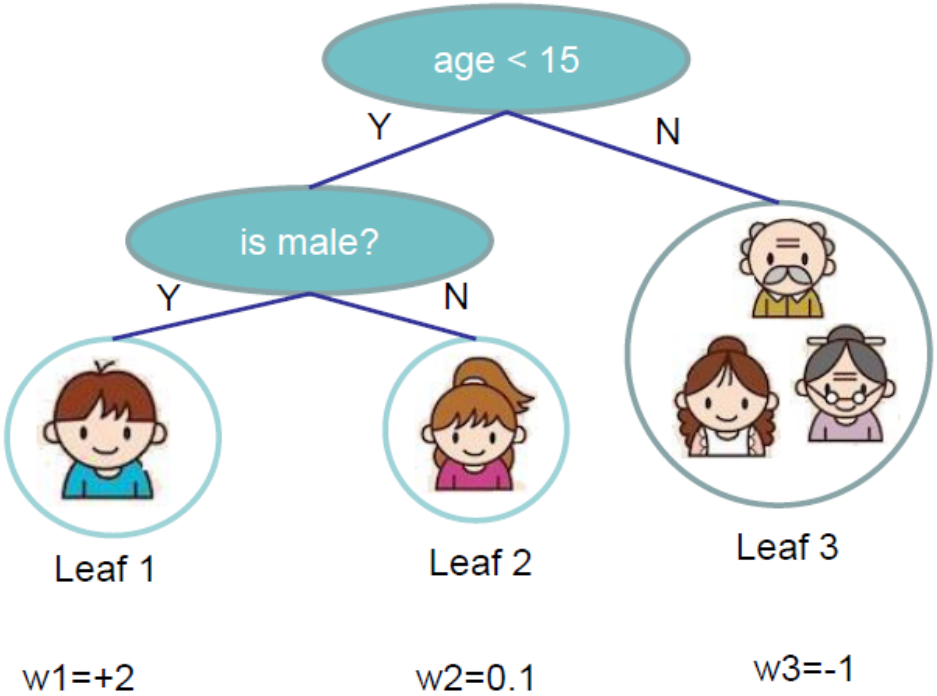


Define the Complexity of Tree

- 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

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

- This is sum of T independent quadratic functions

The Structure Score

- Two facts about single variable quadratic function

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

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

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

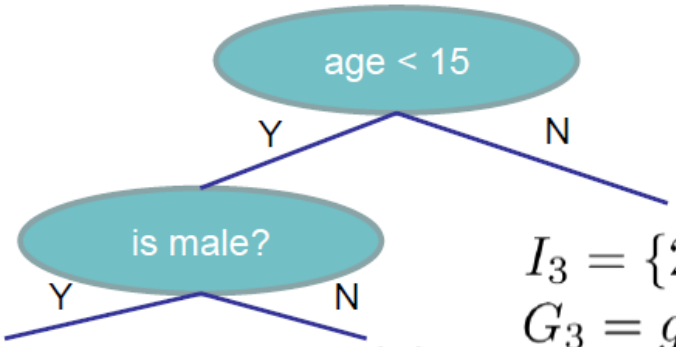
- Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

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

This measures how good a tree structure is!

The Structure Score Calculation

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



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

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

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

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

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

The complexity cost by introducing additional leaf

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

the score of left child

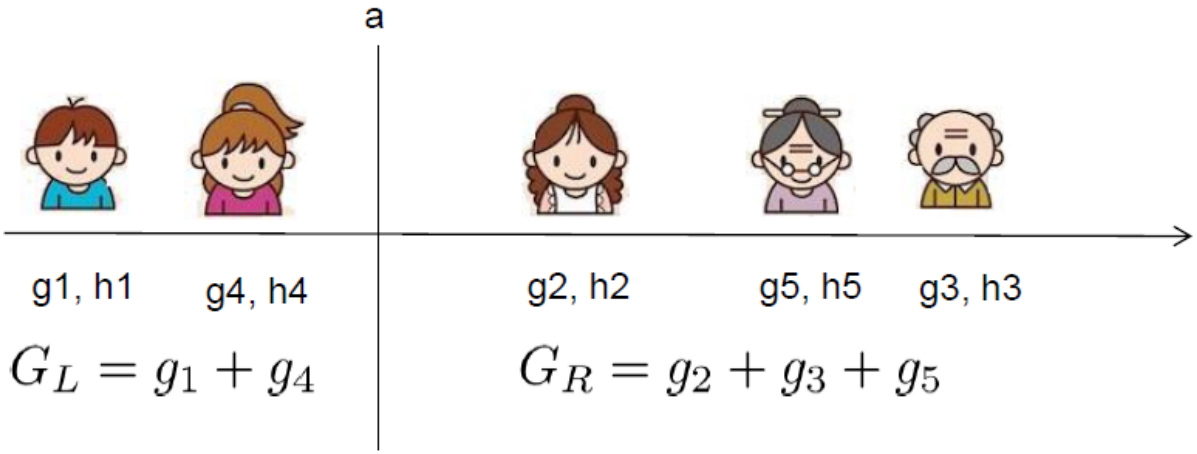
the score of right child

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



- 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

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

  - 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 predictivness

- 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

Recap: Boosted Tree Algorithm

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

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

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

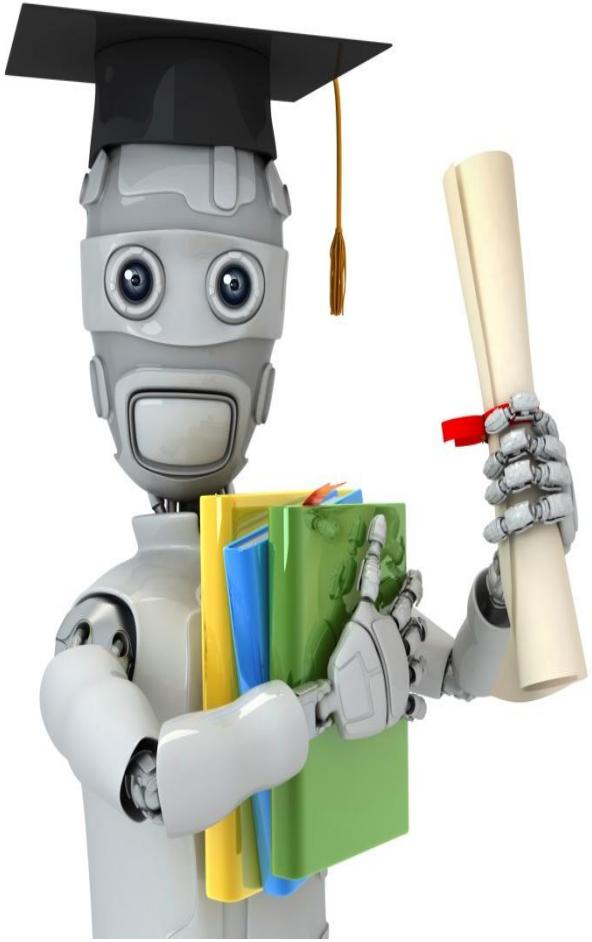
$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- 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

Reference

- Greedy function approximation a gradient boosting machine. *J.H. Friedman*
  - *First paper about gradient boosting*
- *Stochastic Gradient Boosting. J.H. Friedman*
  - *Introducing bagging trick to gradient boosting*
- *Elements of Statistical Learning. T. Hastie, R. Tibshirani and J.H. Friedman*
  - *Contains a chapter about gradient boosted boosting*
- Additive logistic regression a statistical view of boosting. *J.H. Friedman T. Hastie R. Tibshirani*
  - *Uses second-order statistics for tree splitting, which is closer to the view presented in this slide*
- Learning Nonlinear Functions Using Regularized Greedy Forest. *R. Johnson and T. Zhang*
  - *Proposes to do fully corrective step, as well as regularizing the tree complexity. The regularizing trick is closed related to the view present in this slide*

- Software implementing the model described in this slide: https://github.com/tqchen/xgboost



Tree-Based Methods

Bagging

Machine Learning

Bagging

Bagging or bootstrap aggregation averages a given procedure over many samples, to reduce its variance. Suppose $T(\boldsymbol{x})$ is a classifier, such as a tree, producing a predicted class label at input point $\boldsymbol{x}$. To bag T, we draw bootstrap samples $\{(\boldsymbol{x}_i^*, y_i^*)\}^1, \ldots, \{(\boldsymbol{x}_i^*, y_i^*)\}^B$ each of size n with replacement from the training data. Then

$$\hat{C}_{bag}(\boldsymbol{x}) = \text{Majority Vote} \left\{T^{*b}(\boldsymbol{x})\right\}_{b=1}^{B}$$

Bagging can dramatically reduce the variance of unstable procedures (like trees), leading to improved prediction.

Algorithm

Input:

- D, a set of d training tuples;
- K, the number of models in the ensemble;
- A learning scheme(e.g., decision tree algorithm, back-propagation, etc.)

Output: A composite model, M*.

- Method:

a) **for** i = 1 to k **do** //create k models:

b) create bootstrap sample, D_i, by sampling D with replacement;

c) use D_i to derive a model, M_i;

d) **Endfor**

e) **Majority voting by all of the models**

bootstrap

- Raw data $X = (X_1, \dots, X_n)$
- Create an artificial list of data by randomly picking elements from raw data. Repeat n times.
- Some elements will be picked more than once.

- To use the composite model on a tuple, *X*:

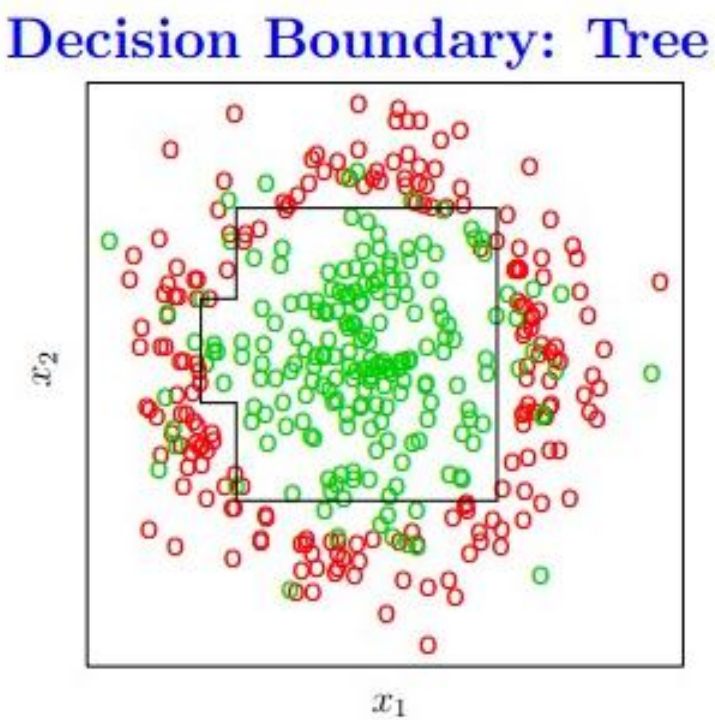
If classification **then**

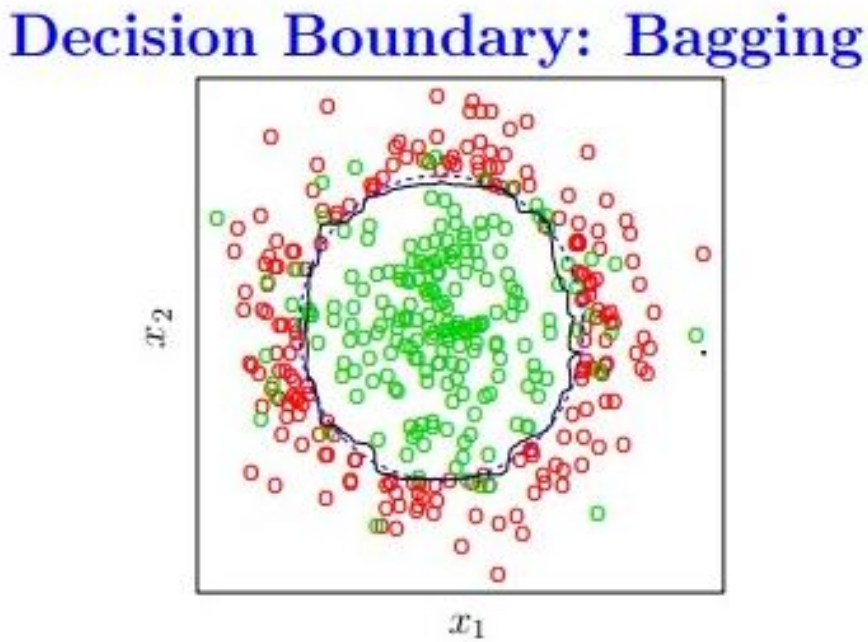
let each of the *k* models classify *X* and return the majority vote;

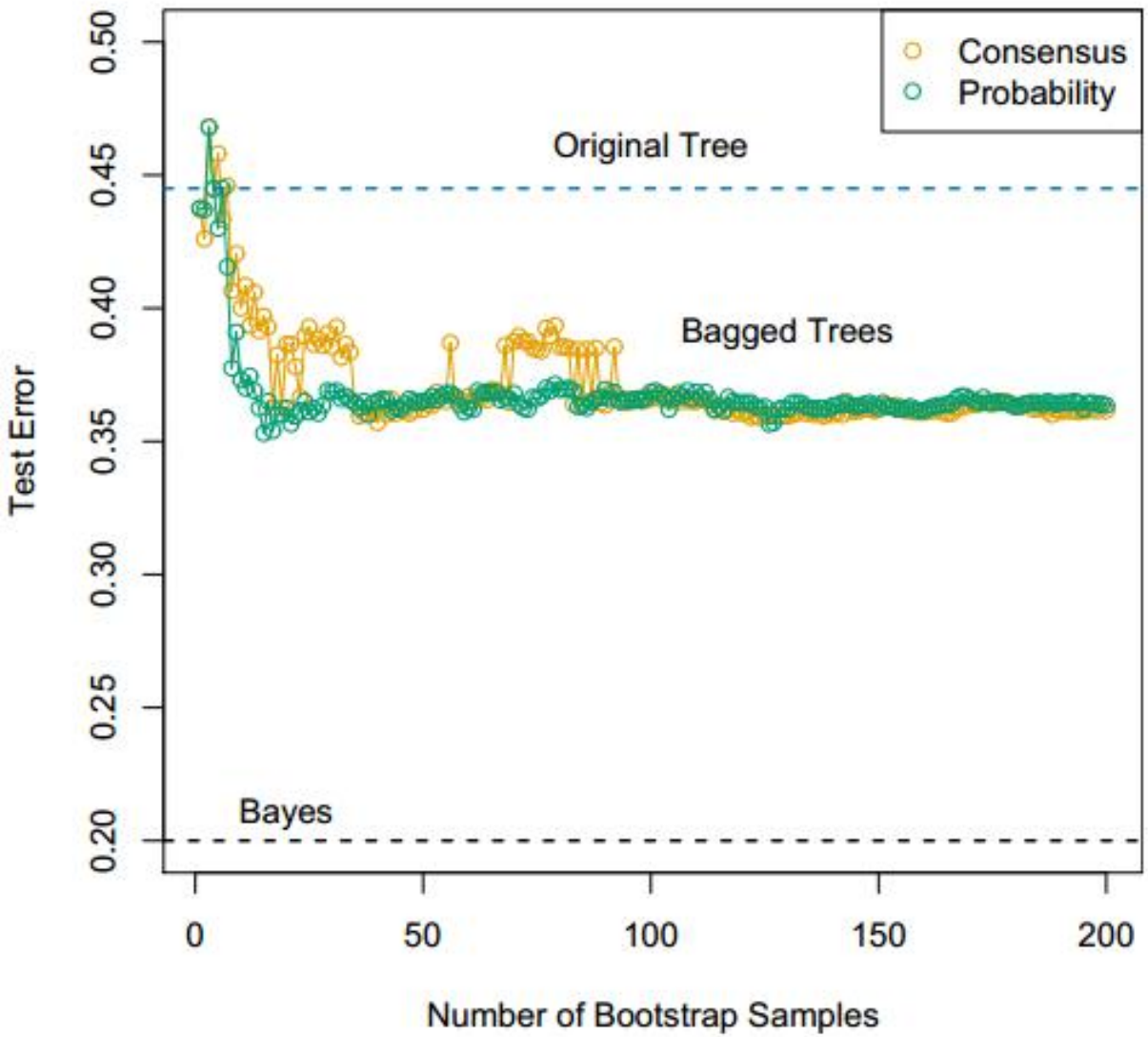
If regression **then**

let each of the *k* models predict a value for *X* and return the average predicted value

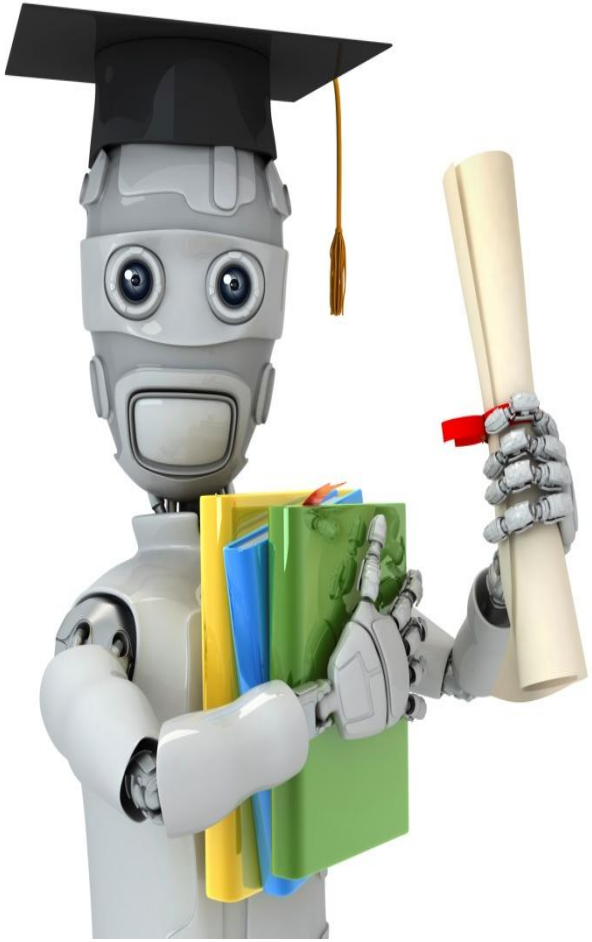
The contrast







- Error curves for the bagging example of Figure. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.



Machine Learning

Tree-Based Methods

Random Forests

Random Forests

- Random forests (Breiman, 2001) is a substantial modification of bagging that builds a large collection of de-correlated trees, and then averages them.
- On many problems the performance of random forests is very similar to boosting, and they are simpler to train and tune.
- As a consequence, random forests are popular, and are implemented in a variety of pack ages.

The Origin of RF

- since each tree generated in bagging is identically distributed(i.d.), the expectation of an average of B such trees is the same as the expectation of any one of them.
- This means the bias of bagged trees is the same as that of the individual trees, and the only hope of improvement is through variance reduction.
- This is in contrast to boosting, where the trees are grown in an adaptive way to remove bias, and hence are not i.d.

The Origin of RF

- The idea in random forests is to improve the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance too much. This is achieved in the tree-growing process through random selection of the input variables/features.
- Specifically, when growing a tree on a bootstrapped dataset:

  Before each split, select m <= p of the input variables at random as candidates for splitting.

- For classification, the default value for m is $\lfloor \sqrt{p} \rfloor$ and the minimum node size is one.
- For regression, the default value for m is $\lfloor p/3 \rfloor$ and the minimum node size is five.

- After B such trees $\{T(x;\Theta_b)\}_1^B$ are grown, the random forest (regression) predictor is

$$\hat{f}_{\mathrm{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b).$$

- Θb characterizes the bth random forest tree in terms of split variables, cutpoints at each node, and terminal-node values.
- Intuitively, reducing m will reduce the correlation between any pair of trees in the ensemble, and hence reduce the variance of the average.

The algorithm of RF

Random Forest for Regression or Classification

1. For b = 1 to B:

(a) Draw a bootstrap sample Z^* of size N from the training data.

(b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

   i. Select m variables at random from the p variables.

   ii. Pick the best variable/split-point among the m.

   iii. Split the node into two daughter nodes.

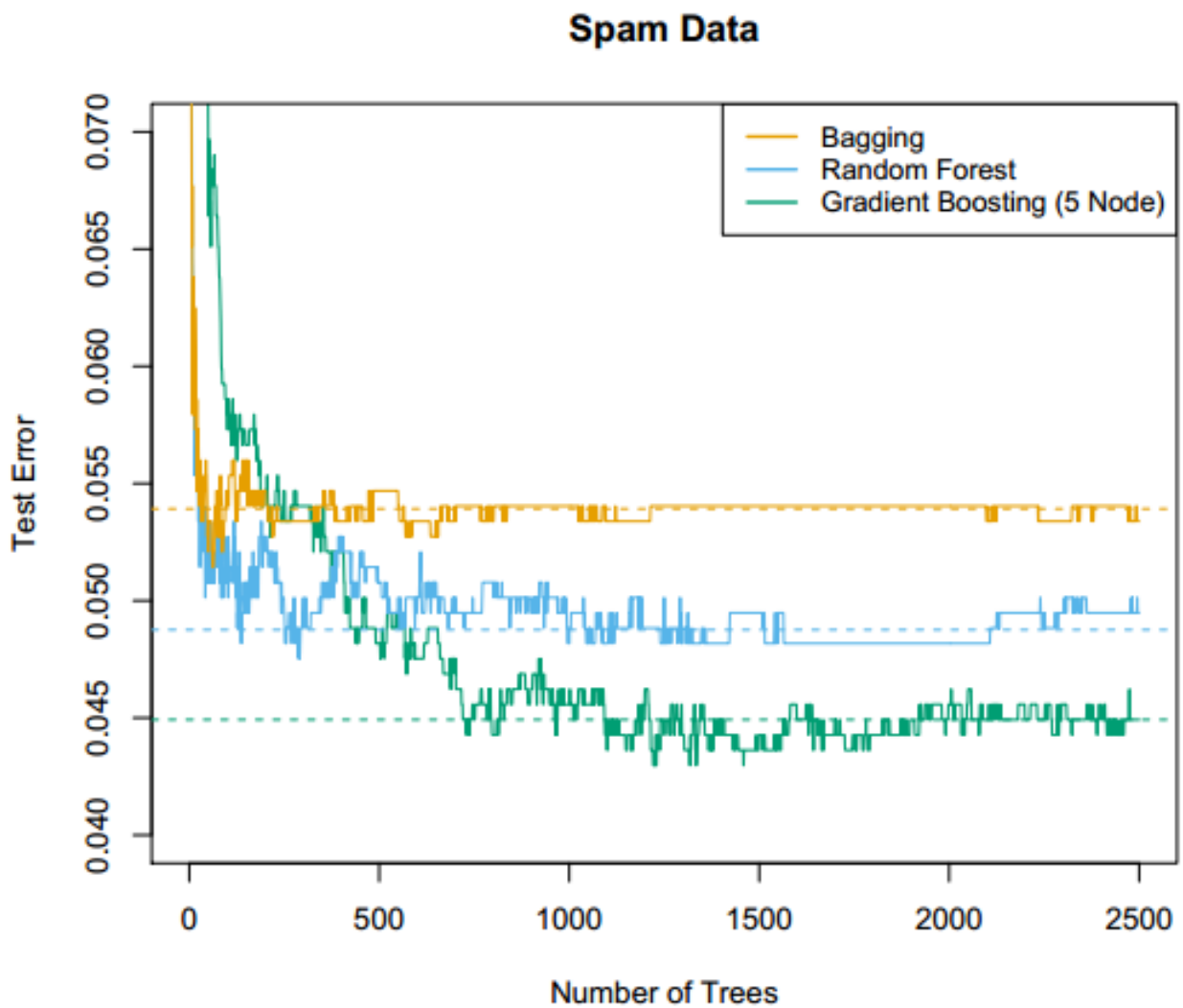
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

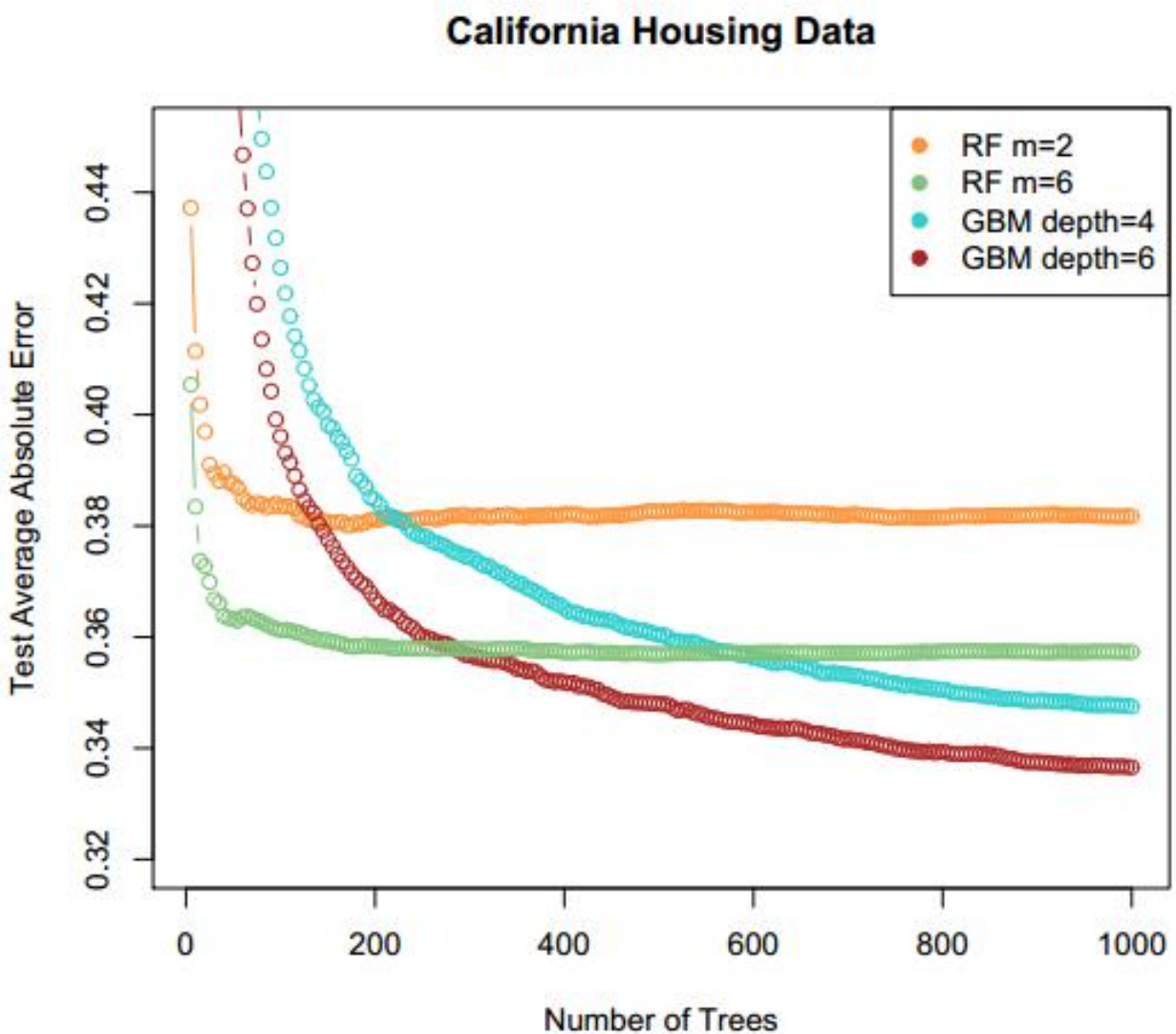
Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then

$$\hat{C}_{rf}^B(x) = majority\ vote\ \{\hat{C}_b(x)\}_1^B .$$



Bagging, random forest, and gradient boosting, applied to the spam data. For boosting, 5-node trees were used, and the number of trees were chosen by 10-fold cross-validation (2500 trees). Each “step” in the figure corresponds to a change in a single misclassification (in a test set of 1536).

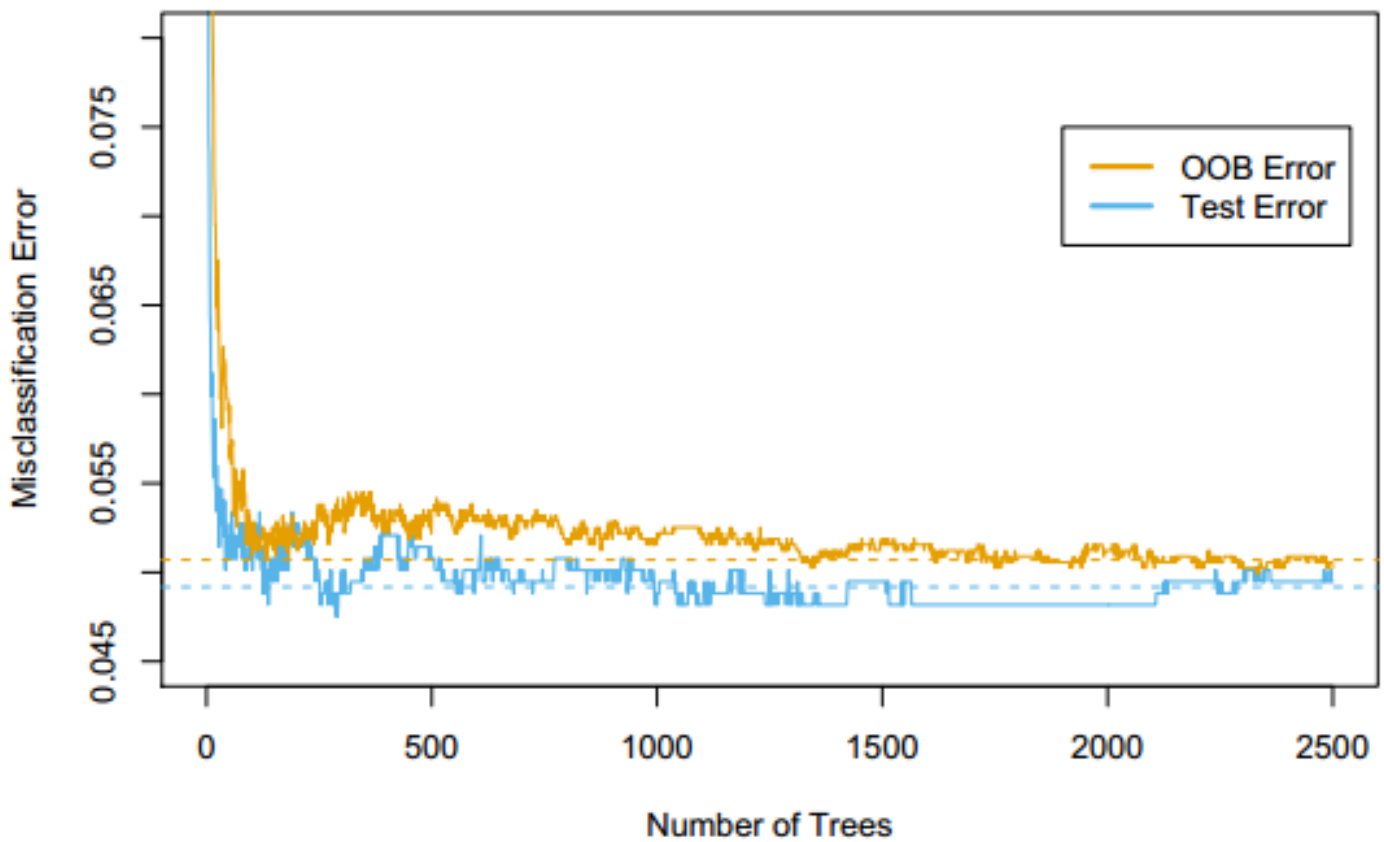


Random forests compared to gradient boosting on the California housing data. The curves represent mean absolute error on the test data as a function of the number of trees in the models. Two random forests are shown, with m = 2 and m = 6. The two gradient boosted models use a shrinkage parameter ν = 0.05 in (10.41), and have interaction depths of 4 and 6. The boosted models outperform random forests.

Out-of-Bag samples

- An important feature of random forests is its use of out-of-bag (oob) samples:

  For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear.



- oob error computed on the spam training data, compared to the test error computed on the test set.