Machine Learning

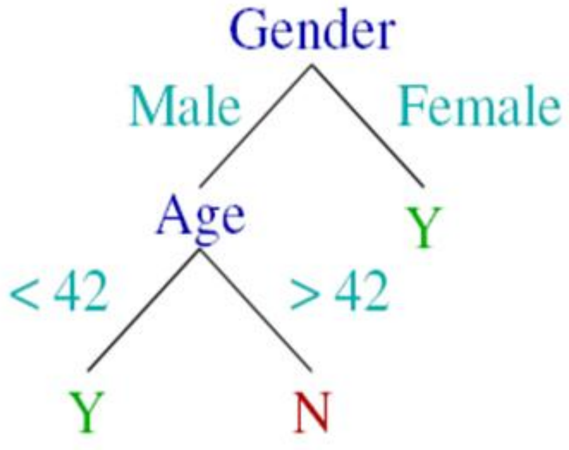
Tree-Based Methods

Decision Tree

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

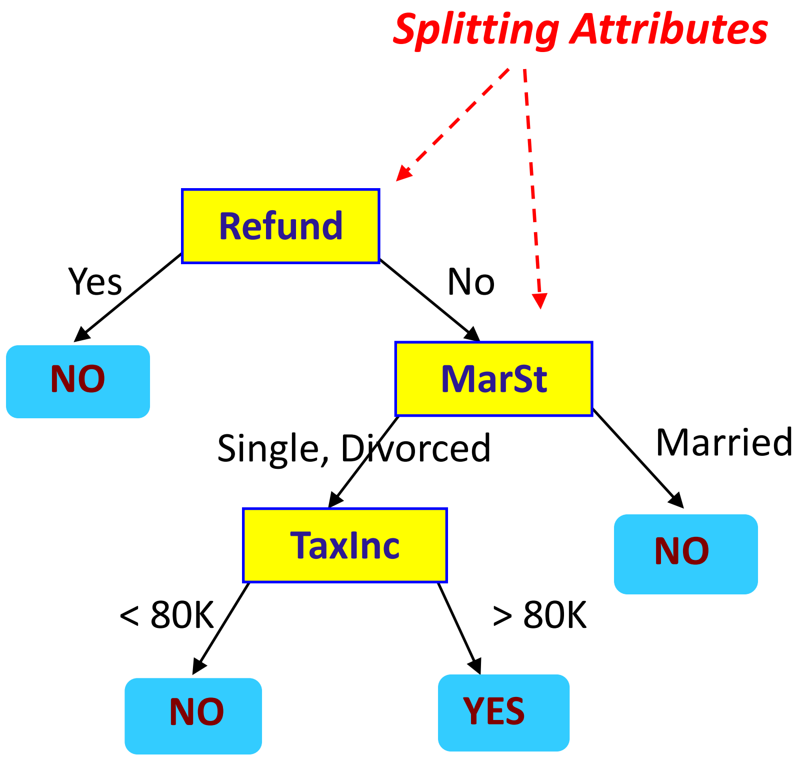


Example of a Decision Tree

categorical categorical continuous class

Tid	Refund	Marital Status	Taxable Income	Cheat
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



Splitting Attributes

Refund

Yes

No

NO

MarSt

Single, Divorced

Married

TaxInc

NO

< 80K

> 80K

NO

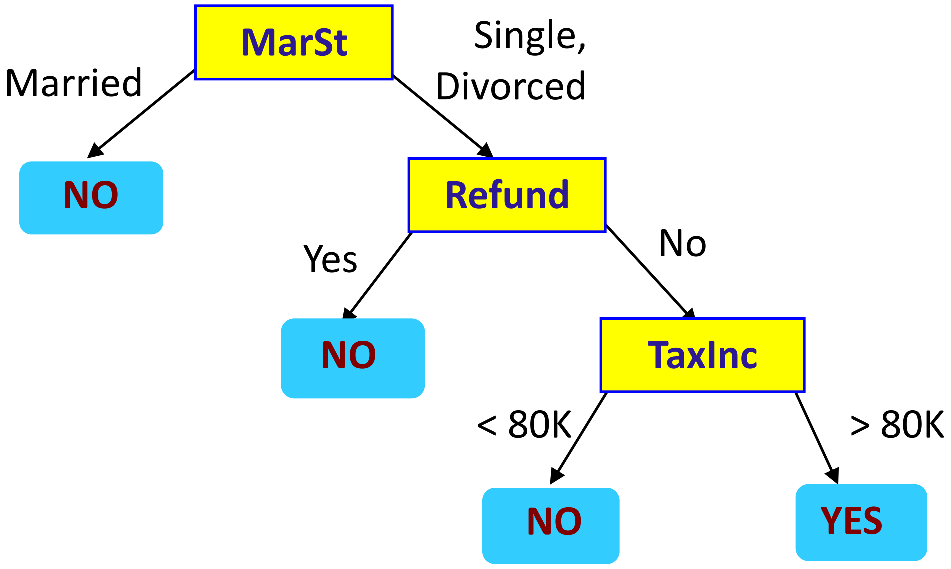
YES

Model: Decision Tree

Another Example of Decision Tree

categorical categorical continuous class

Tid	Refund	Marital Status	Taxable Income	Cheat
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

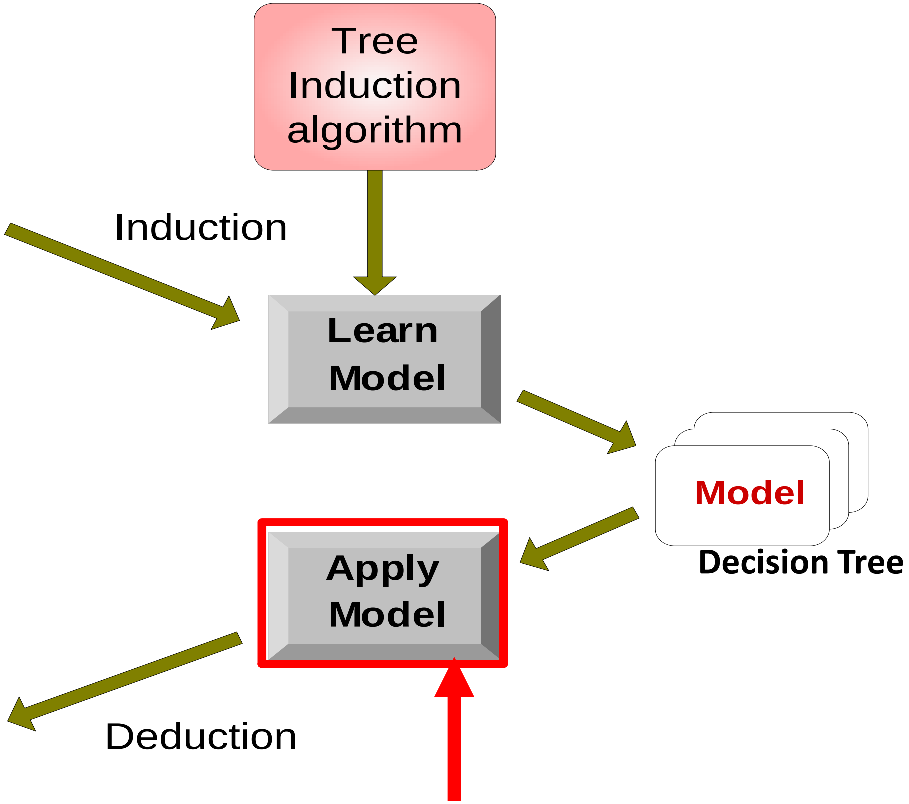


There could be more than one tree that fits the same data!

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



Tree Induction algorithm

Induction

Learn Model

Model

Decision Tree

Apply Model

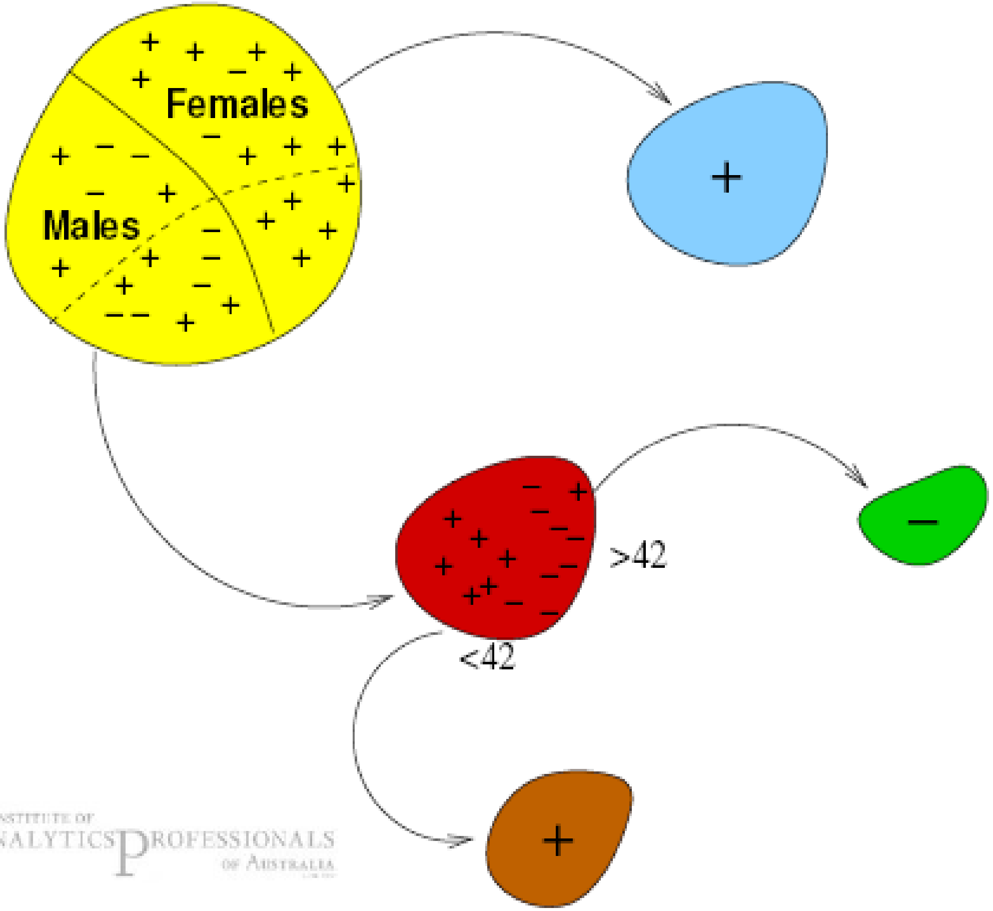
Deduction

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

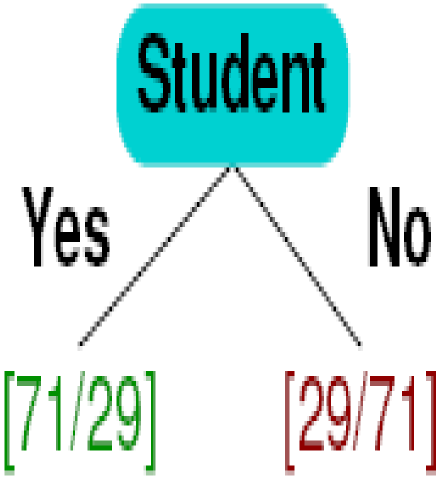


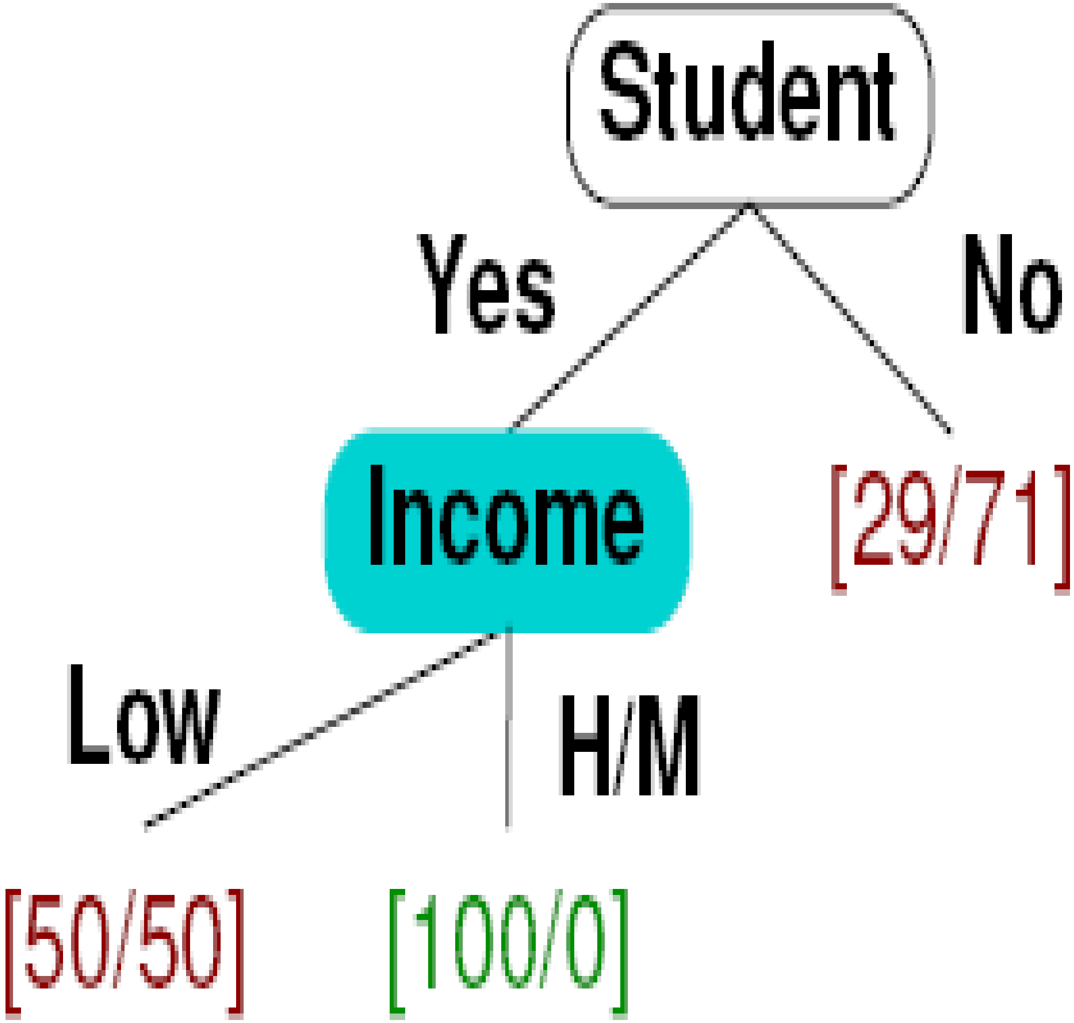
Training Dataset: Buys Computer?

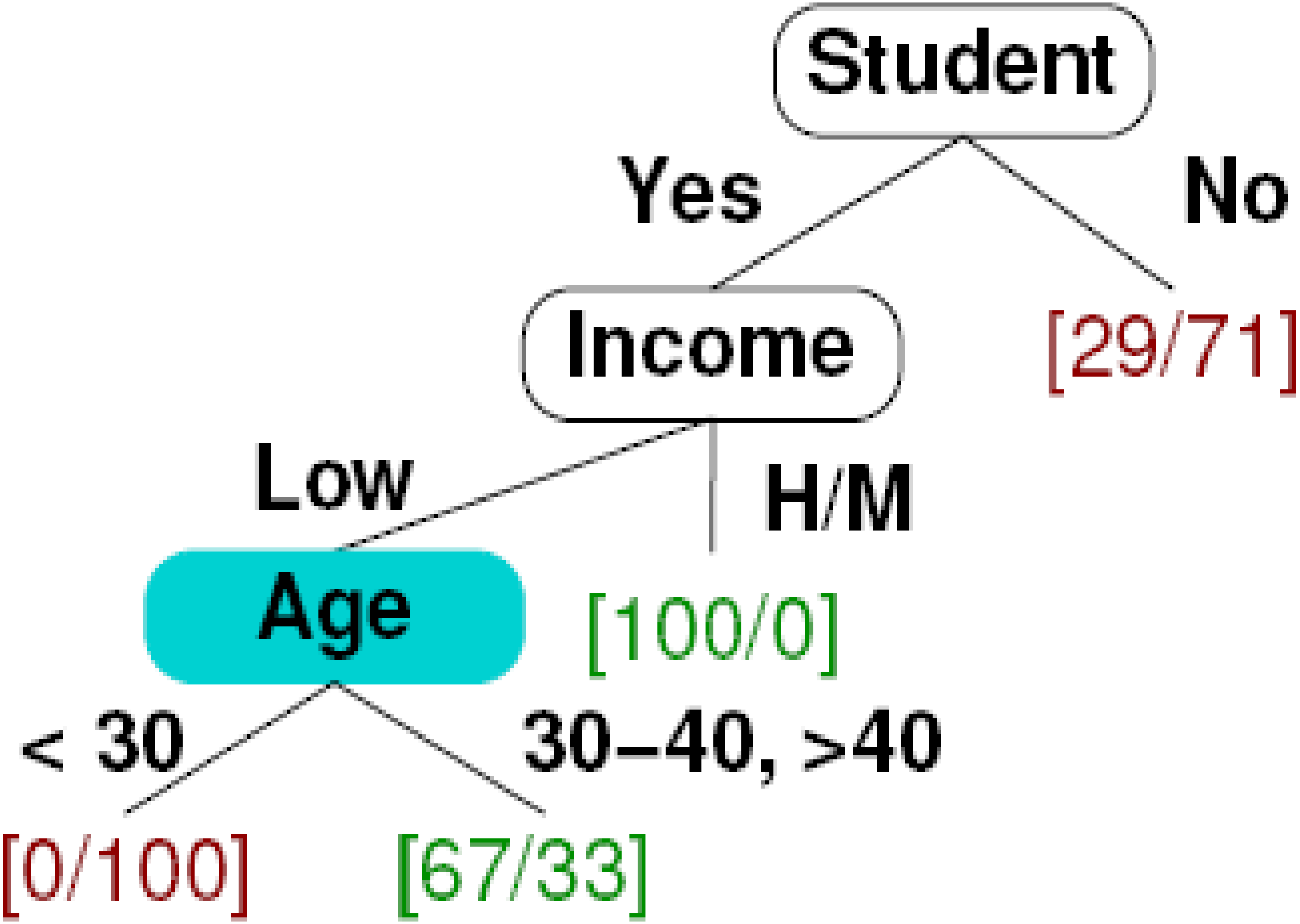
What rule would you “learn” to identify who buys a computer?

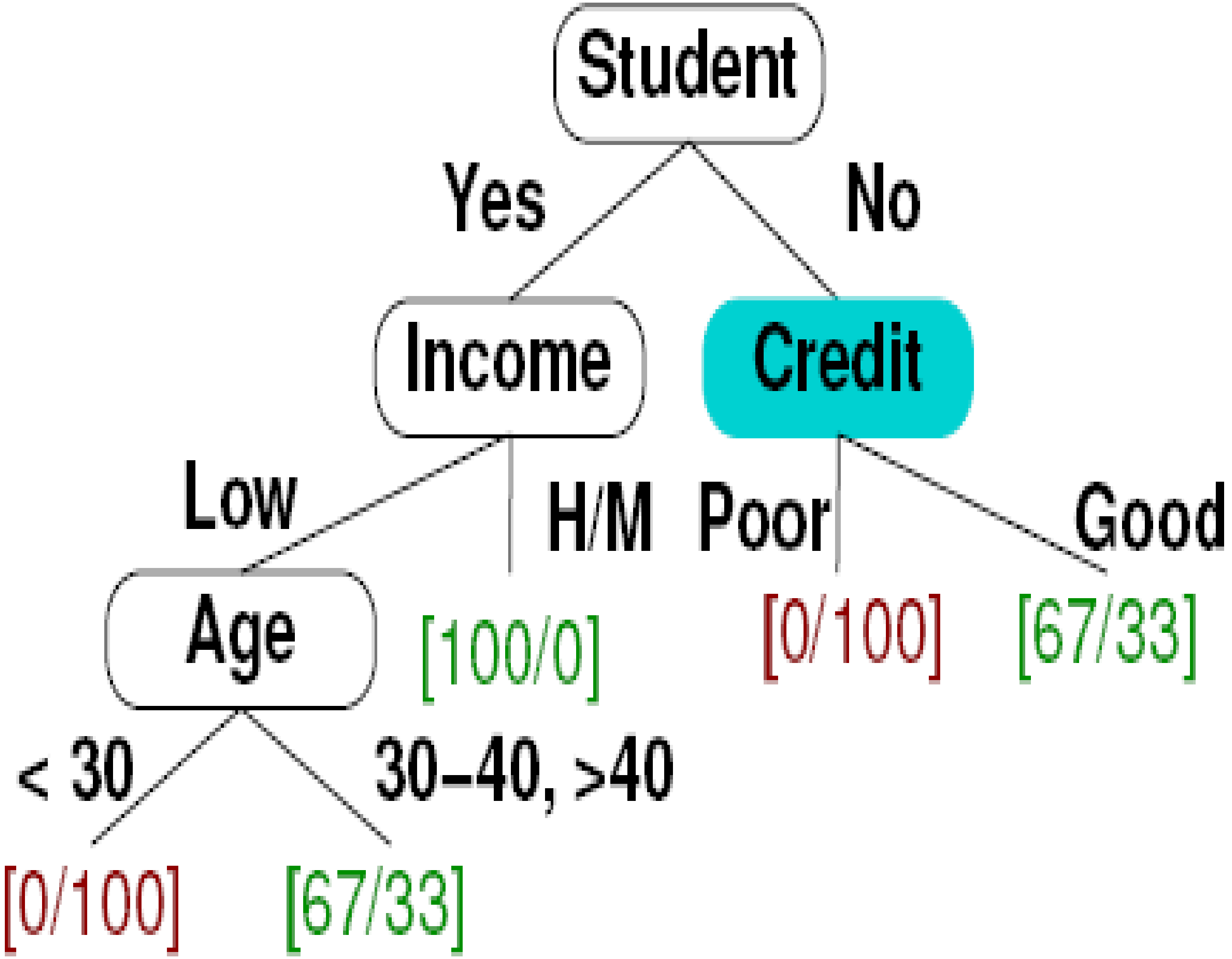
Age	Income	Student	Credit	Buys
< 30	High	No	Poor	No
< 30	High	No	Good	Yes
$30 - 40$	High	No	Poor	Yes
> 40	Medium	No	Poor	Yes
> 40	Low	Yes	Poor	Yes
> 40	Low	Yes	Good	No
$30 - 40$	Low	Yes	Good	Yes
< 30	Medium	No	Poor	No
< 30	Low	Yes	Poor	No
> 40	Medium	Yes	Poor	Yes
< 30	Medium	Yes	Good	Yes
$30 - 40$	Medium	No	Good	Yes
$30 - 40$	High	Yes	Poor	Yes
> 40	Medium	No	Good	No

Output: Decision Tree for Buys Computer




Student
Yes
No
Income
[29/71]
Low
H/M
[50/50]
[100/0]

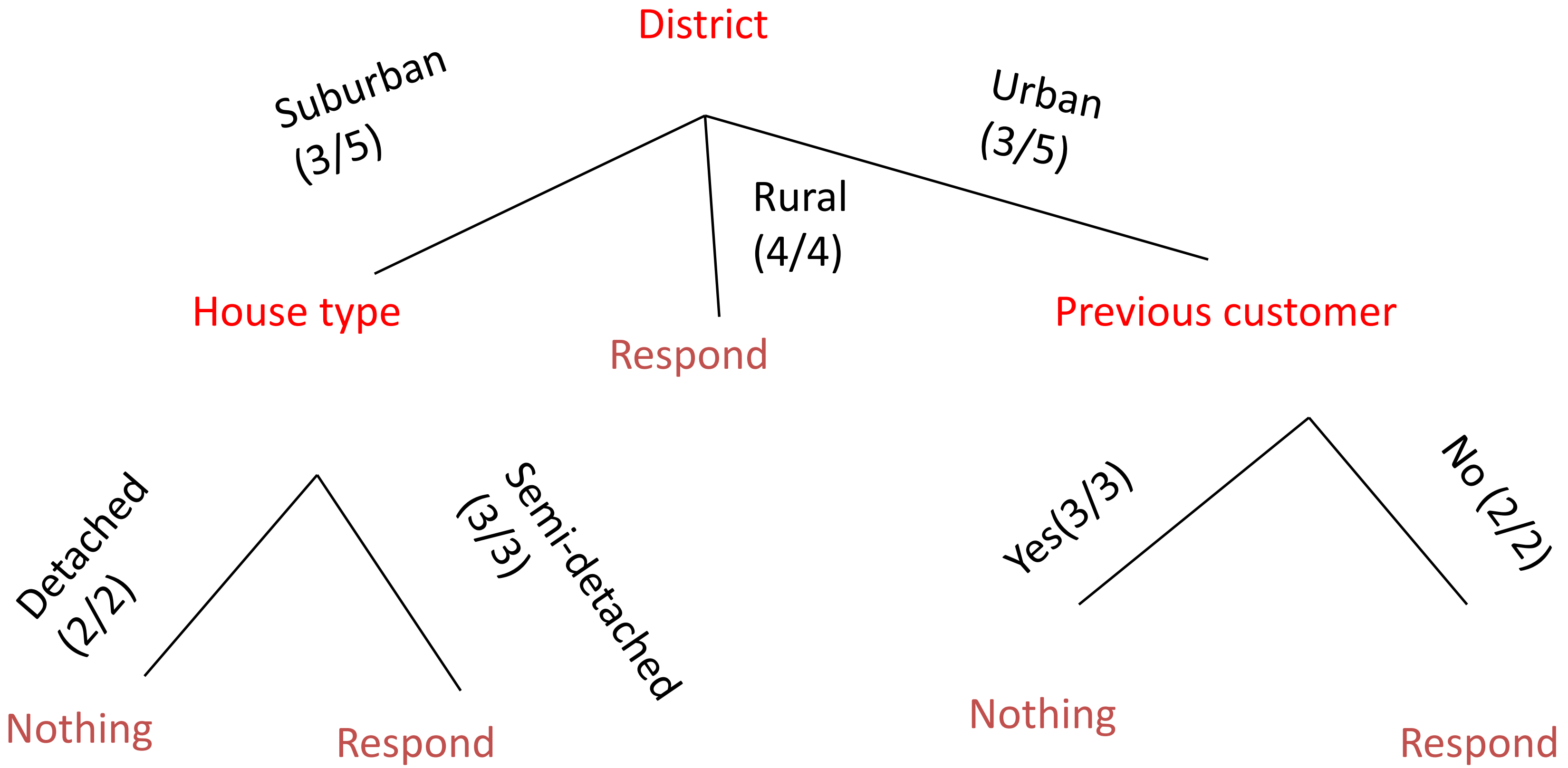

Student
Yes
No
Income
[29/71]
Low
H/M
Age
[100/0]
< 30
30–40, >40
[0/100]
[67/33]


Student
Yes
No
Income
Credit
Low
H/M
Poor
Good
Age
[100/0]
[0/100]
[67/33]
< 30
30–40, >40
[0/100]
[67/33]

A classification problem

District	House type	Income	Previous Customer	Outcome (target)
Suburban	Detached	High	No	Nothing
Suburban	Semi-detached	High	Yes	Respond
Rural	Semi-detached	Low	No	Respond
Urban	Detached	Low	Yes	Nothing
…				

Decision 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

Information gain

- The expectation of any classification of samples

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

  S : training set m : the number of classes

  P_i : the proportion of samples in S whose category is c_i

(Si) $p_i = \frac{|S_i|}{|S|}$

- The entropy of subsets divided by A

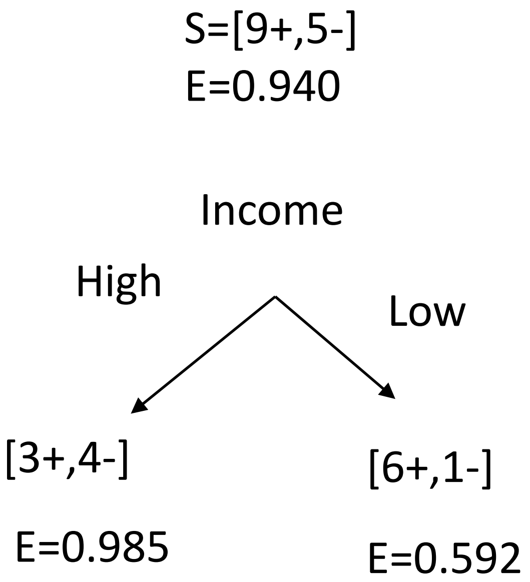
  $\mathrm{E(A)} = \sum_{j=1}^{V} (s_{1j} + \cdots + s_{mj})/s * I(s_{1j}, \dots, s_{mj})$

  A: the feature, has V different values

- Information gain

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

An example



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

Income

High

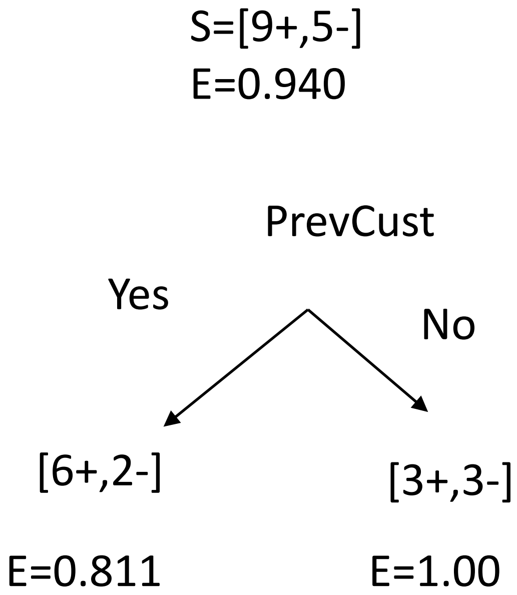
Low

[3+,4-]

E=0.985

[6+,1-]

E=0.592



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

PrevCust

Yes

No

[6+,2-]

E=0.811

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

  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 :

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

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

- 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.

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

➔ In practice, both are used. Some people claim that the latter works better than the former.

Post-pruning

- Split data into training and validation set
- Do until further pruning is harmful:
  - Evaluate impact on validation set of pruning each possible node (plus those below it)
  - Greedily remove the ones that don't improve the performance on validation set

➔Produces a smaller tree with best performance measure

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.
- 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.

Summary of Major issues

Q1: Choosing best attribute: different quality measures.

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

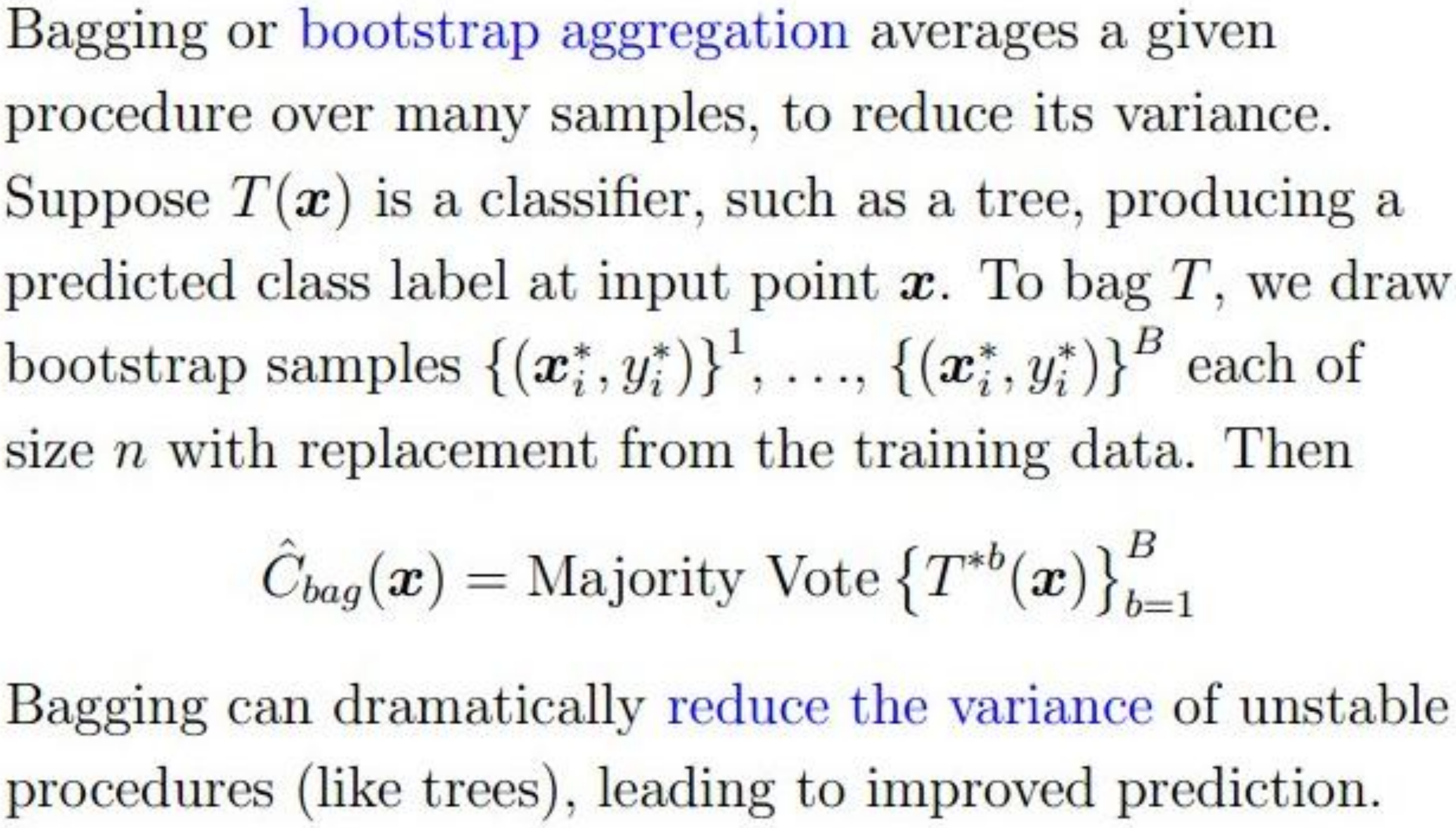
Q3: Handling continuous attributes: find the breakpoints

Machine Learning

Tree-Based Methods

Bagging

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
```

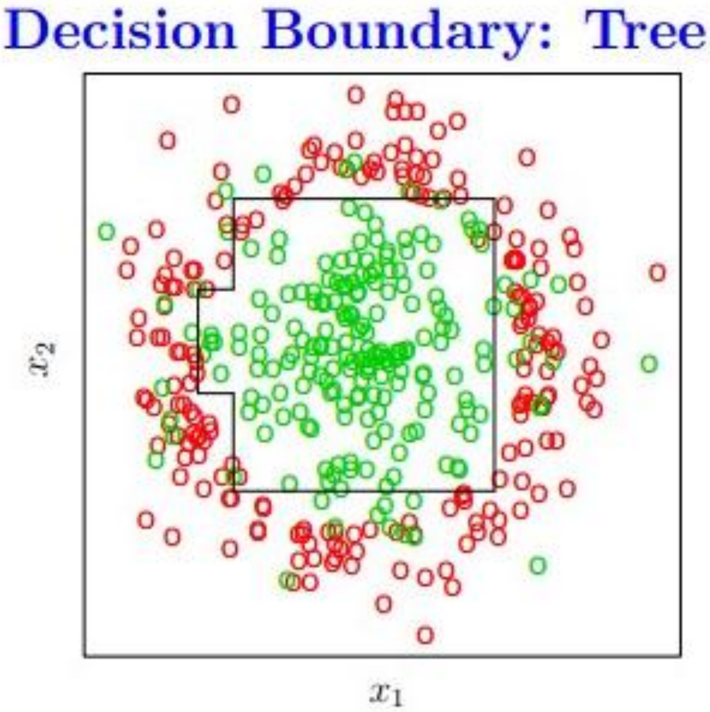
bootstrap

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

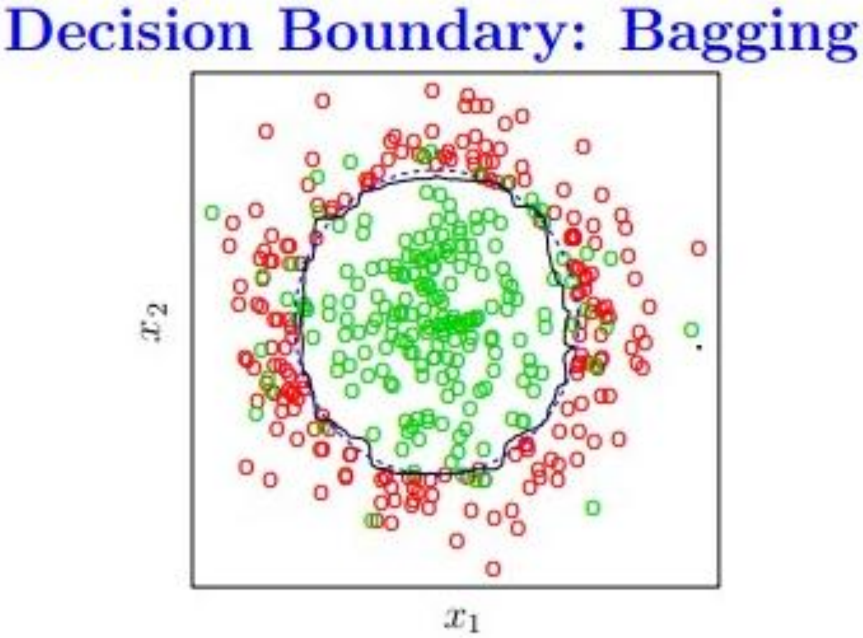
- To use the composite model on a tuple, *X*:

```
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
```

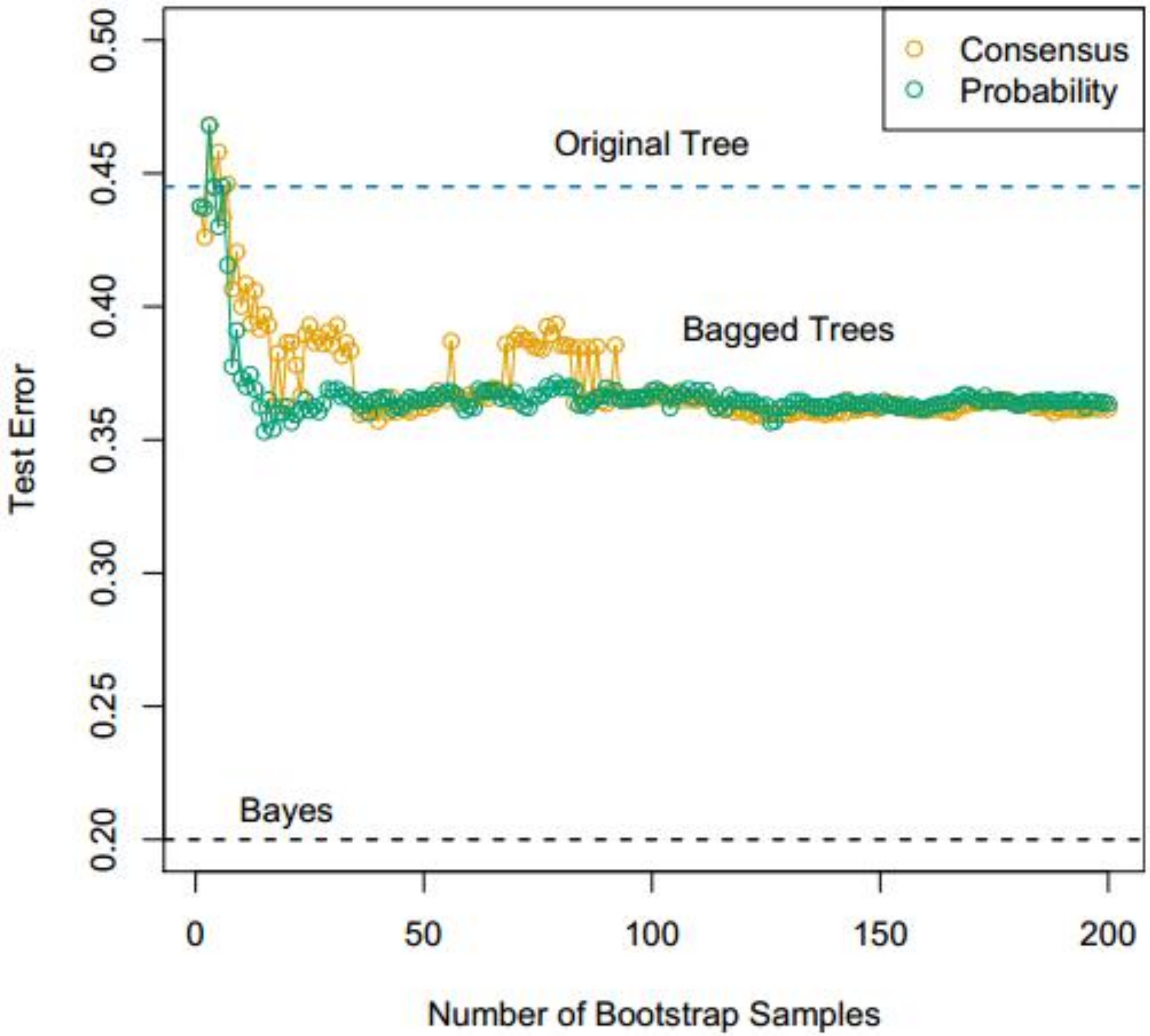
The contrast



Decision Boundary: Tree



Decision Boundary: Bagging



- 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.

Tree-Based Methods

Machine Learning

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.
- 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.

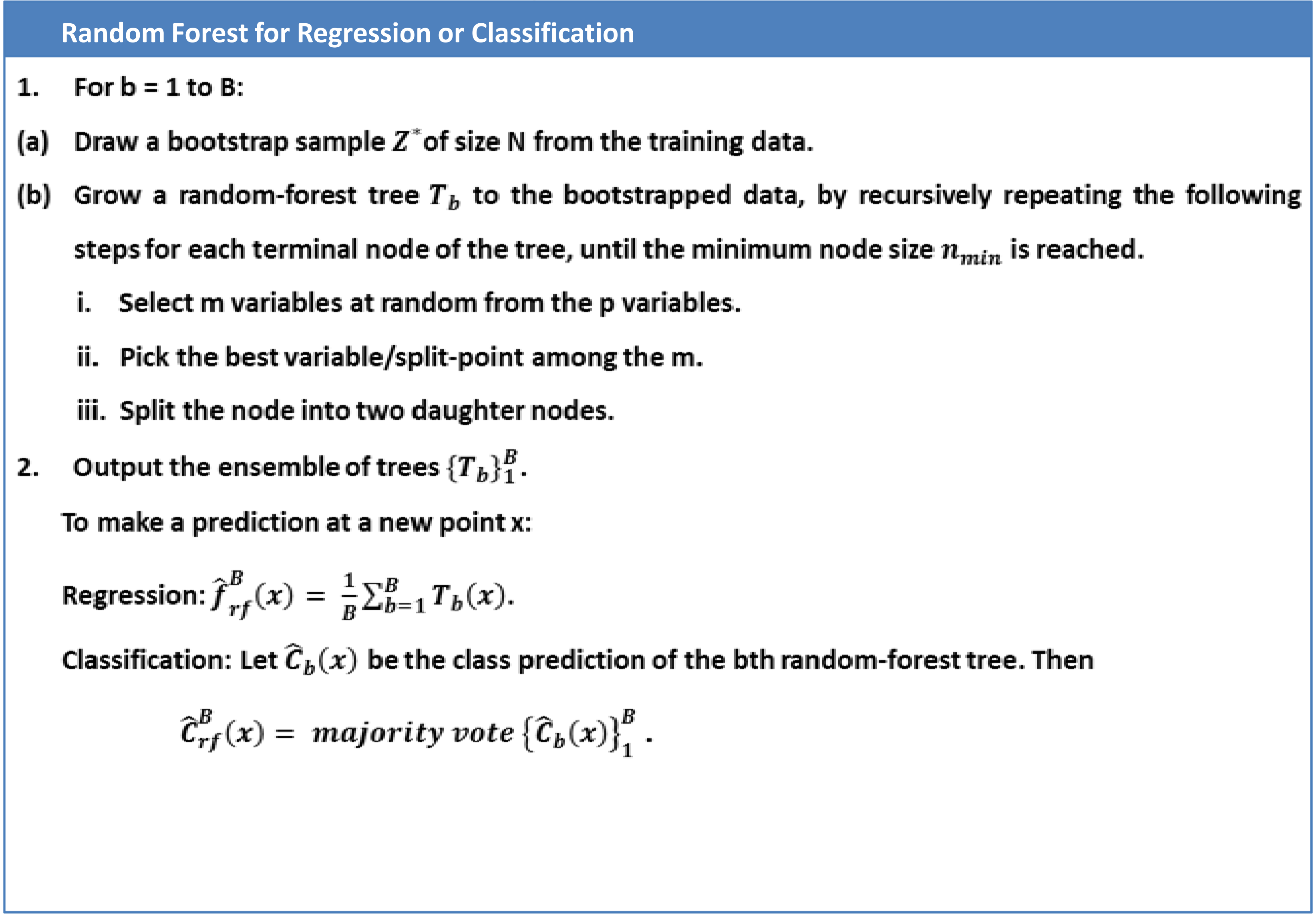
- For classification, the default value for m is $\lfloor\sqrt{p}\,\rfloor$ and the minimum node size is one.
- For regression, the default value for m is $\lfloor$p/3$\rfloor$ and the minimum node size is five.

- 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

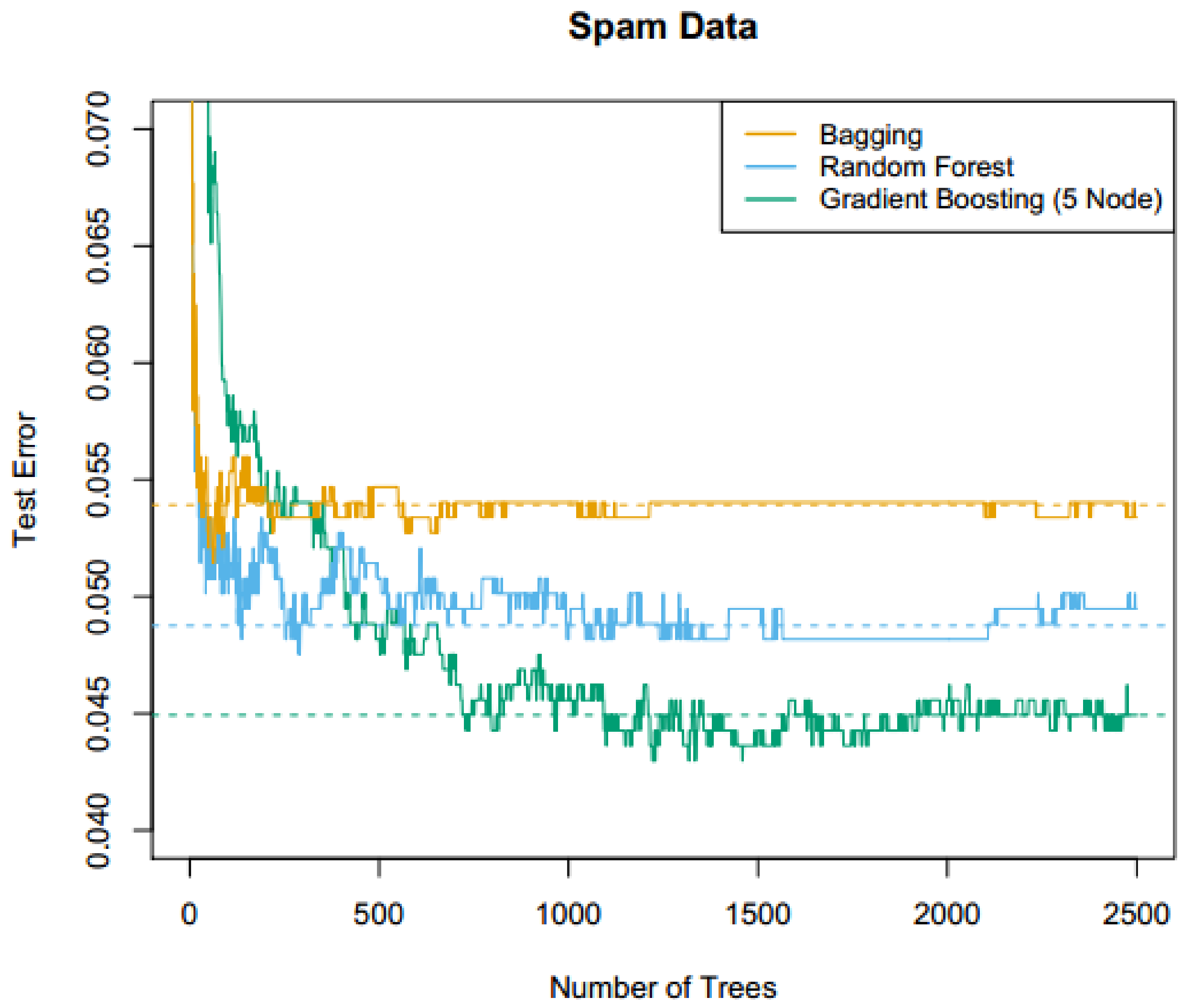
```
1.   For b = 1 to B:
(a)  Draw a bootstrap sample $Z^*$ of size N from the training data.
(b)  Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following
     steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
       i.   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:

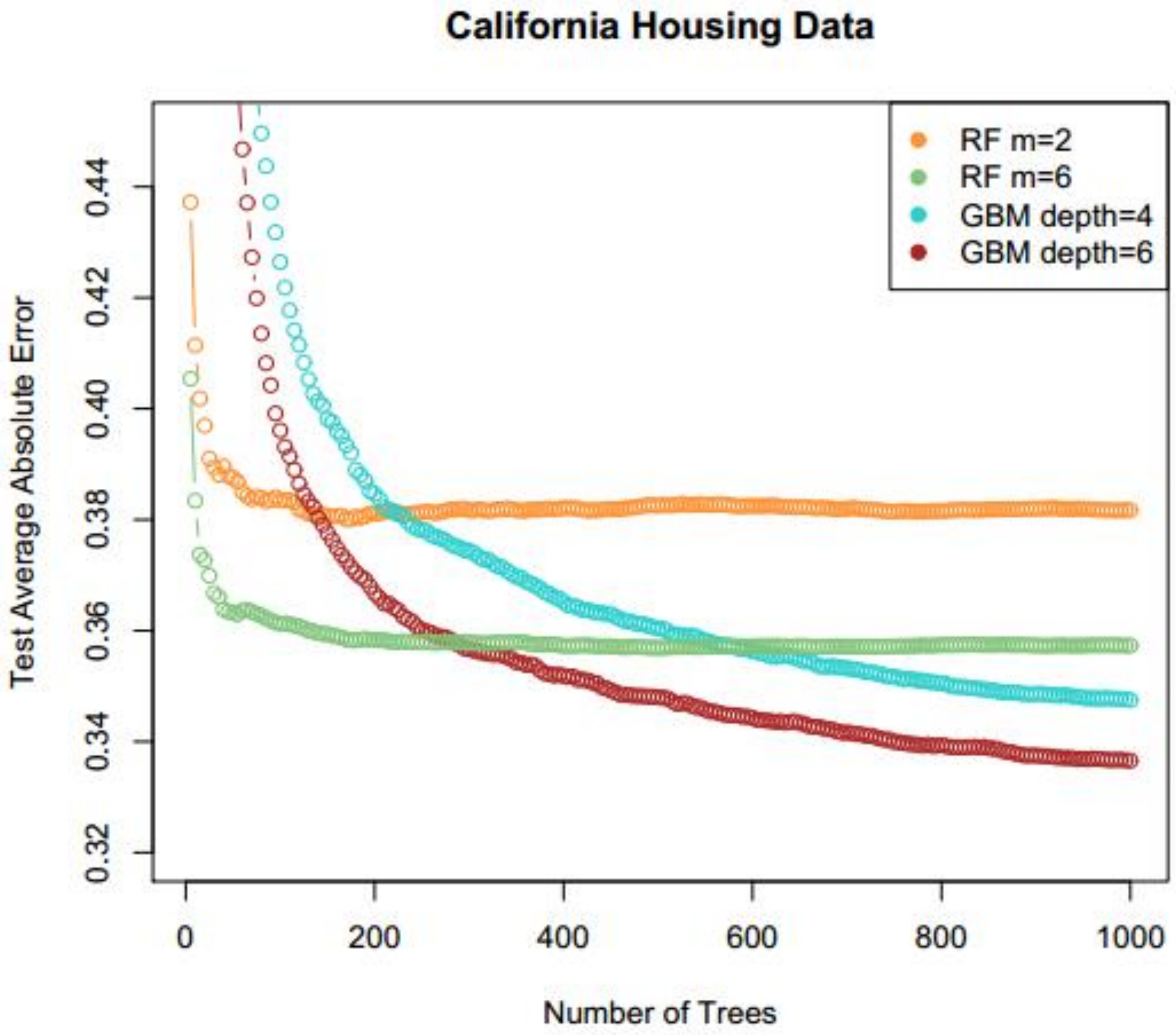
Regression: $\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

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).

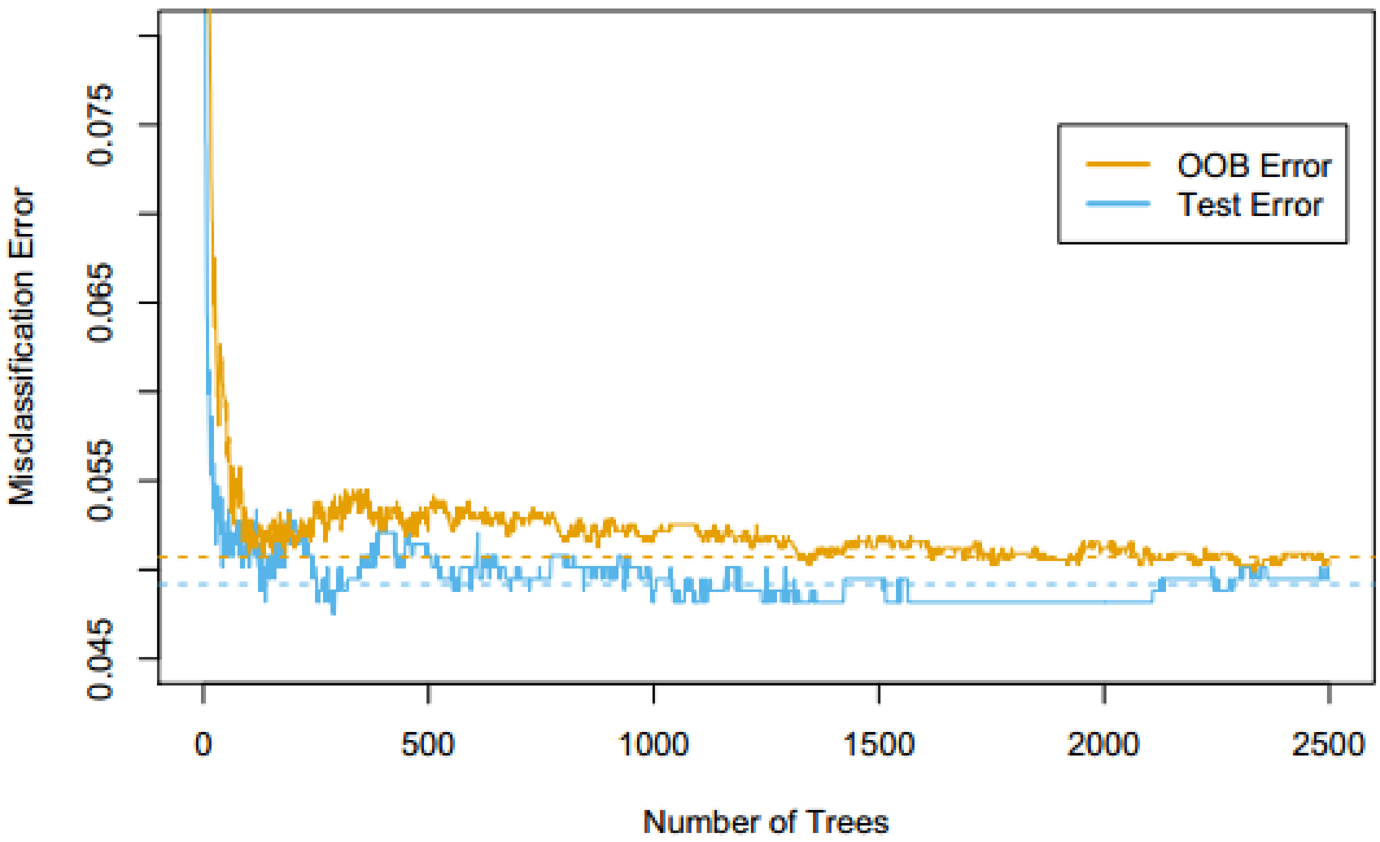


Random forests compared to gradient boosting on the California housing data. The curves represent mean absolute error on the test data as a function of the number of trees in the models. Two random forests are shown, with m = 2 and m = 6. The two gradient boosted models use a shrinkage parameter ν =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

Machine Learning

Tree-Based Methods

Boosting

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

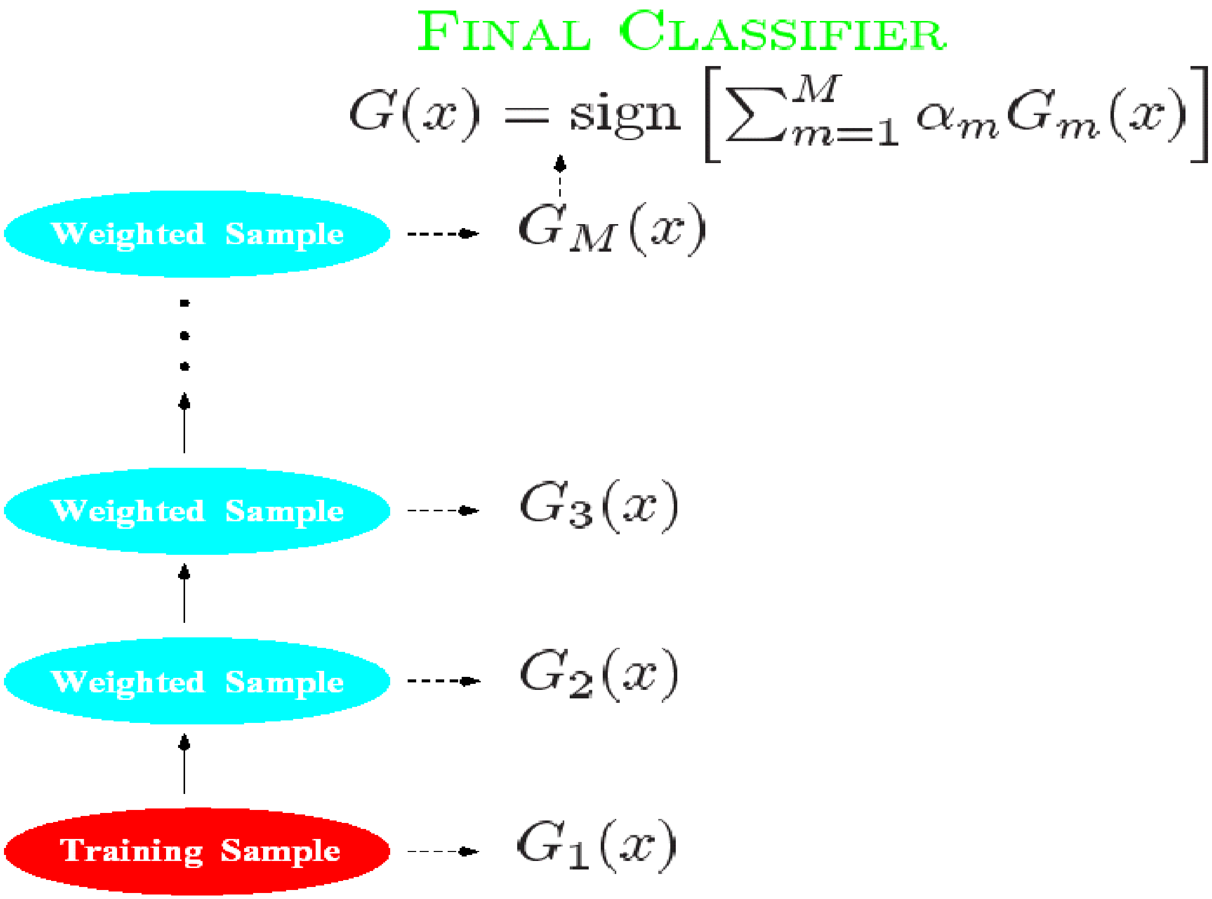
- Key in combining the weak rules

Adaboost

Boosting (Algorithm)

- $W(x)$ is the distribution of weights over the N training points $\sum W(x_i)=1$
- Initially assign uniform weights $W_0(x) = 1/N$ for all x, *step* $k=0$
- At each iteration k :
  - Find best weak classifier $C_k(x)$ using weights $W_k(x)$
  - With error rate ε_k and based on a loss function:
    - weight α_k the classifier C_k's weight in the final hypothesis
    - For each x_i , update weights based on ε_k to get $W_{k+1}(x_i)$
- $C_{FINAL}(x)$ =sign $[\sum \alpha_i C_i (x)]$

Boosting (Algorithm)



FINAL CLASSIFIER

$$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$

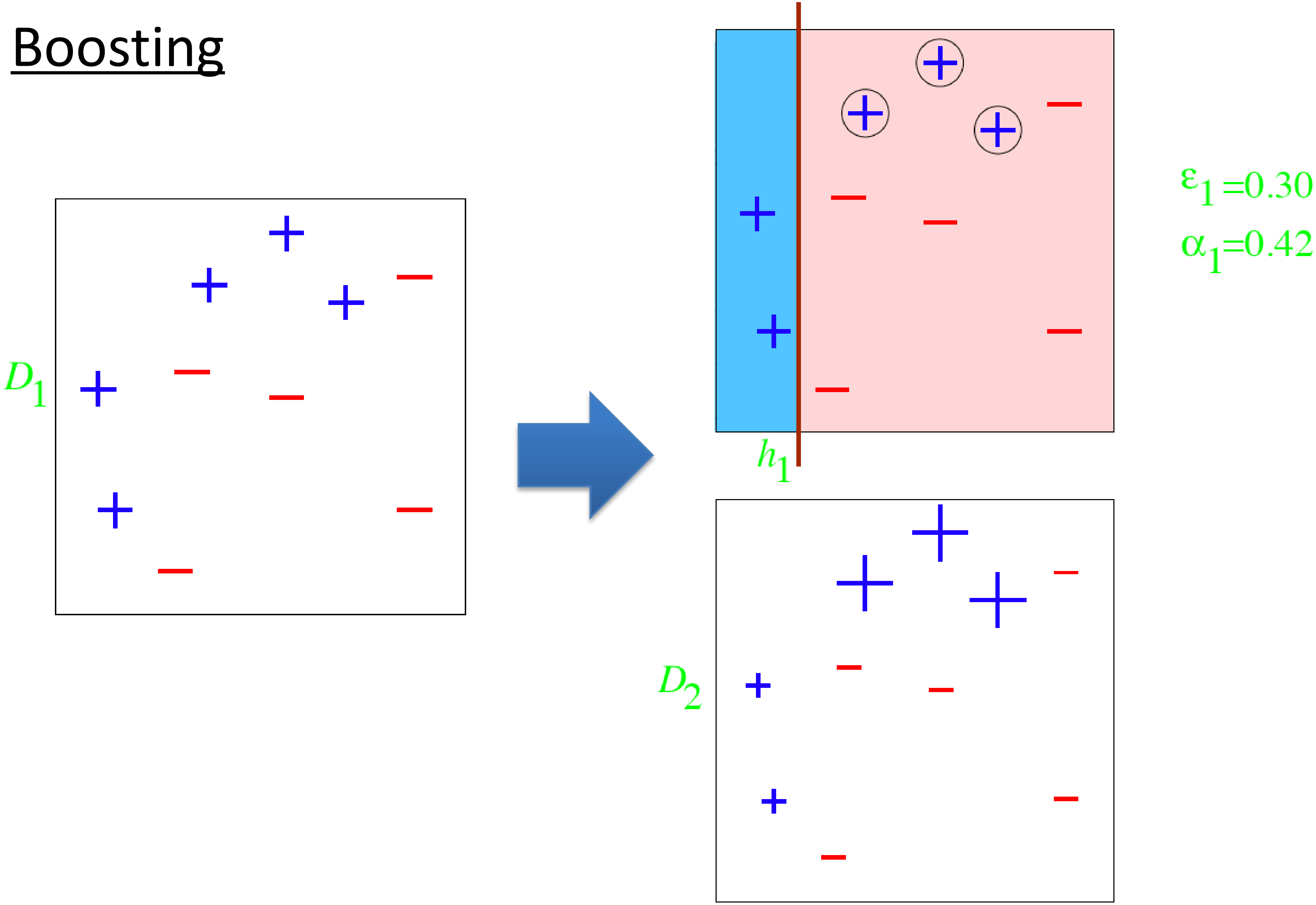
Weighted Sample $\dashrightarrow$ $G_M(x)$

Weighted Sample $\dashrightarrow$ $G_3(x)$

Weighted Sample $\dashrightarrow$ $G_2(x)$

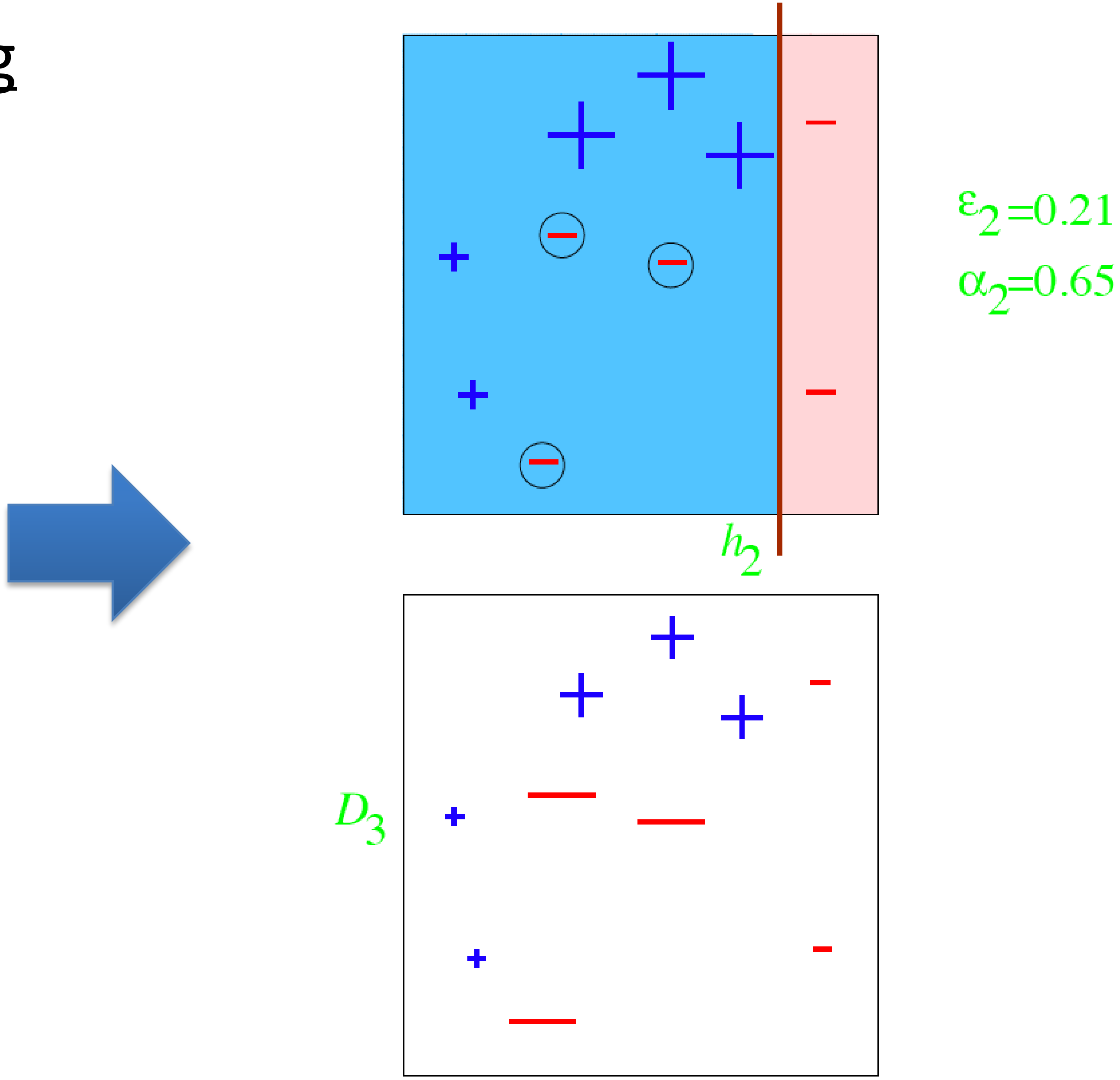
Training Sample $\dashrightarrow$ $G_1(x)$

Boosting



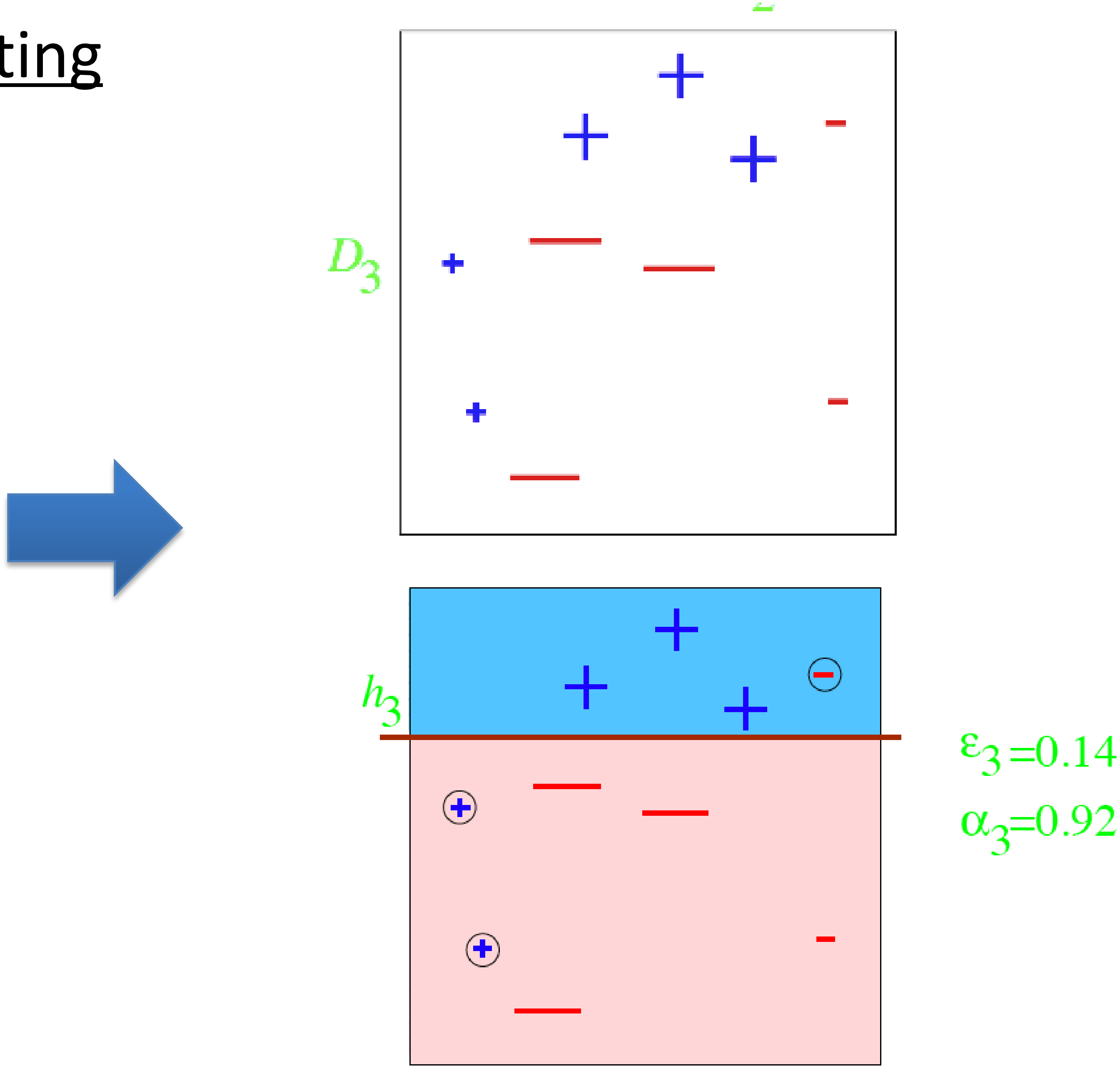
“A Tutorial on Boosting”,Yoav Freund and Rob Schapire

Boosting



“A Tutorial on Boosting”,Yoav Freund and Rob Schapire

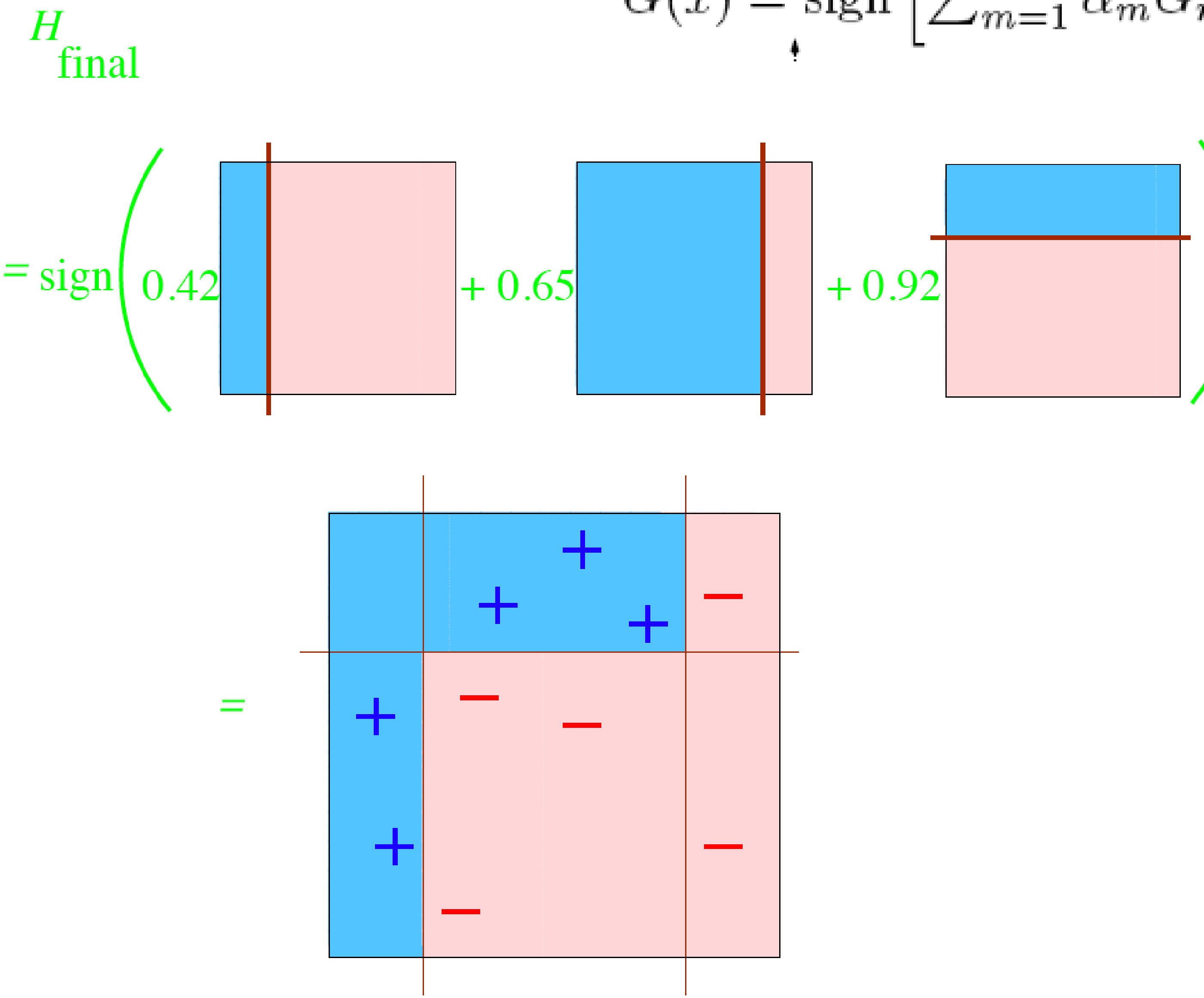
Boosting



“A Tutorial on Boosting”,Yoav Freund and Rob Schapire

Boosting

$$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$



“A Tutorial on Boosting”,Yoav Freund and Rob Schapire

AdaBoost (Algorithm)

- $W(x)$ is the distribution of weights over the N training points $\sum W(x_i)=1$
- Initially assign uniform weights $W_0(x) = 1/N$ for all x.
- At each iteration k :
  - Find best weak classifier $C_k(x)$ using weights $W_k(x)$
  - Compute ε_k the error rate as
    $\varepsilon_k = [\sum W(x_i) \cdot I(y_i \neq C_k(x_i))] / [\sum W(x_i)]$
  - weight α_k the classifier C_k's weight in the final hypothesis Set
    $\alpha_k = \log((1 - \varepsilon_k)/\varepsilon_k)$
  - For each x_i, $W_{k+1}(x_i) = W_k(x_i) \cdot \exp[\alpha_k \cdot I(y_i \neq C_k(x_i))]$
- $C_{FINAL}(x) = \text{sign}[\sum \alpha_i C_i(x)]$

AdaBoost (Freund & Schapire, 1996)

1. Initialize the observation weights
   $w_i = 1/N,\ i = 1, 2, \ldots, N$.

2. For $m = 1$ to M repeat steps (a)–(d):

   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.

   (b) Compute

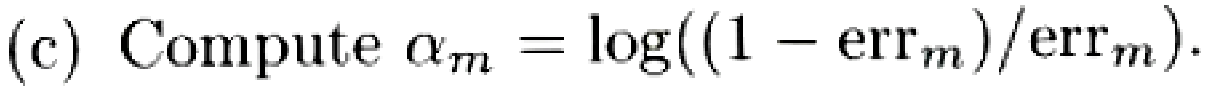
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.

   (d) Update weights for $i = 1, \ldots, N$:
   $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$
   and renormalize to w_i to sum to 1.

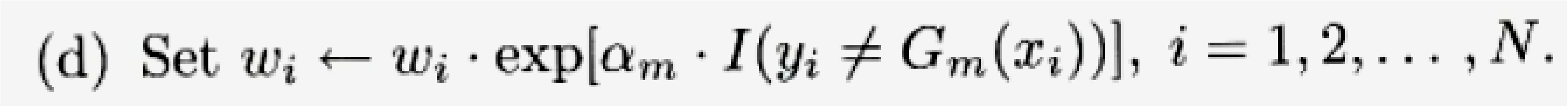
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Key Steps



(c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.

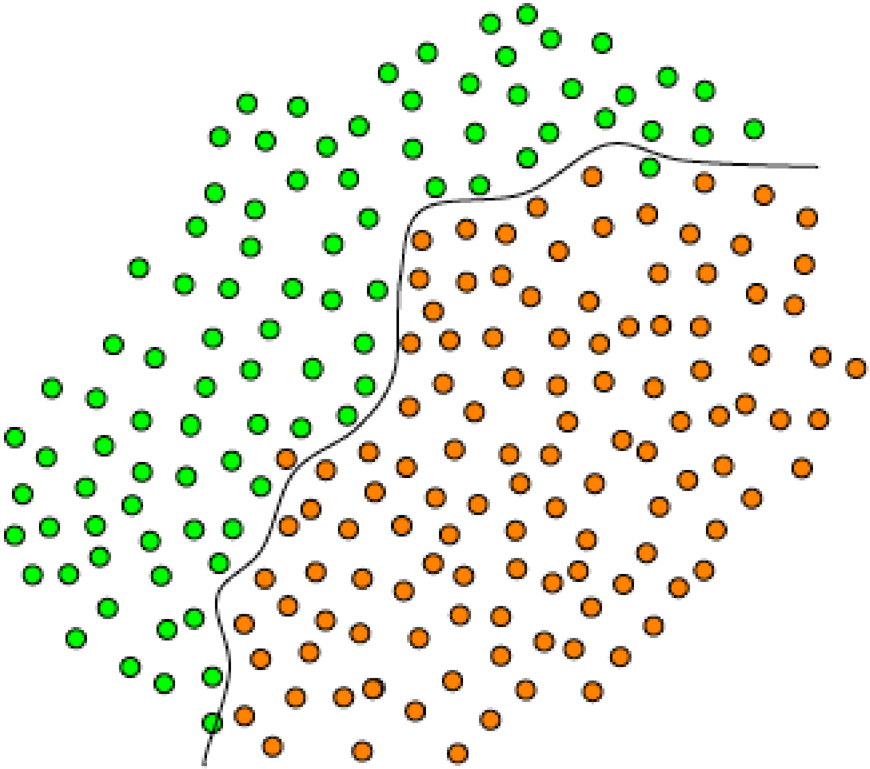
This is the weight of the current weak classifier in the final model.



(d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.

This weight is for individual observations.
Notice it is stacked from step 1.
If an observation is correctly classified at this step, its weight doesn’t change.
If incorrectly classified, its weight increases.

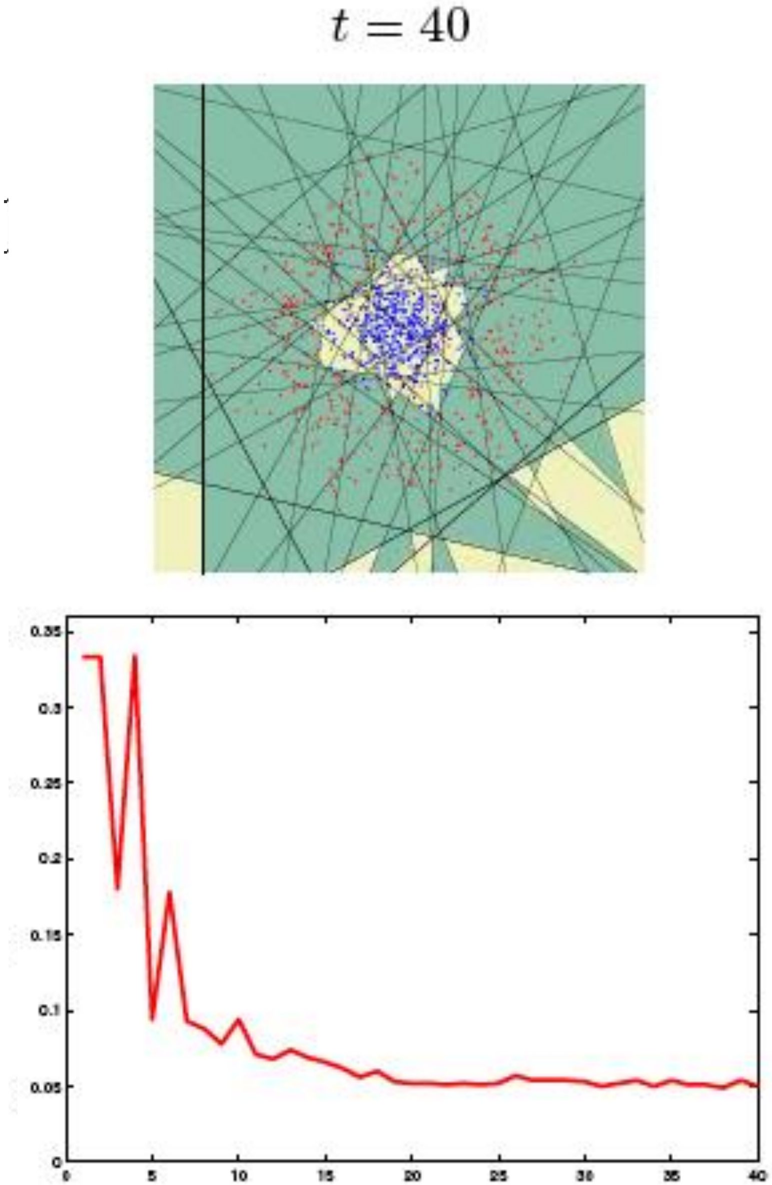
Boosting



Pengyu Hong

Statistical Machine Learning

Boosting



AdaBoost

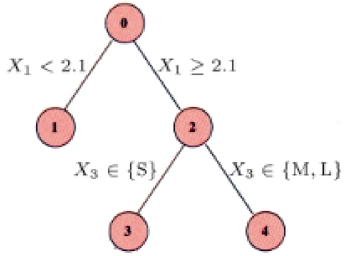
Jiri Matas and Jan Šochman

Centre for Machine Perception
Czech Technical University, Prague
http://cmp.felk.cvut.cz

Boosting

10 predictors

The weak classifier is a Stump: a two-level tree.



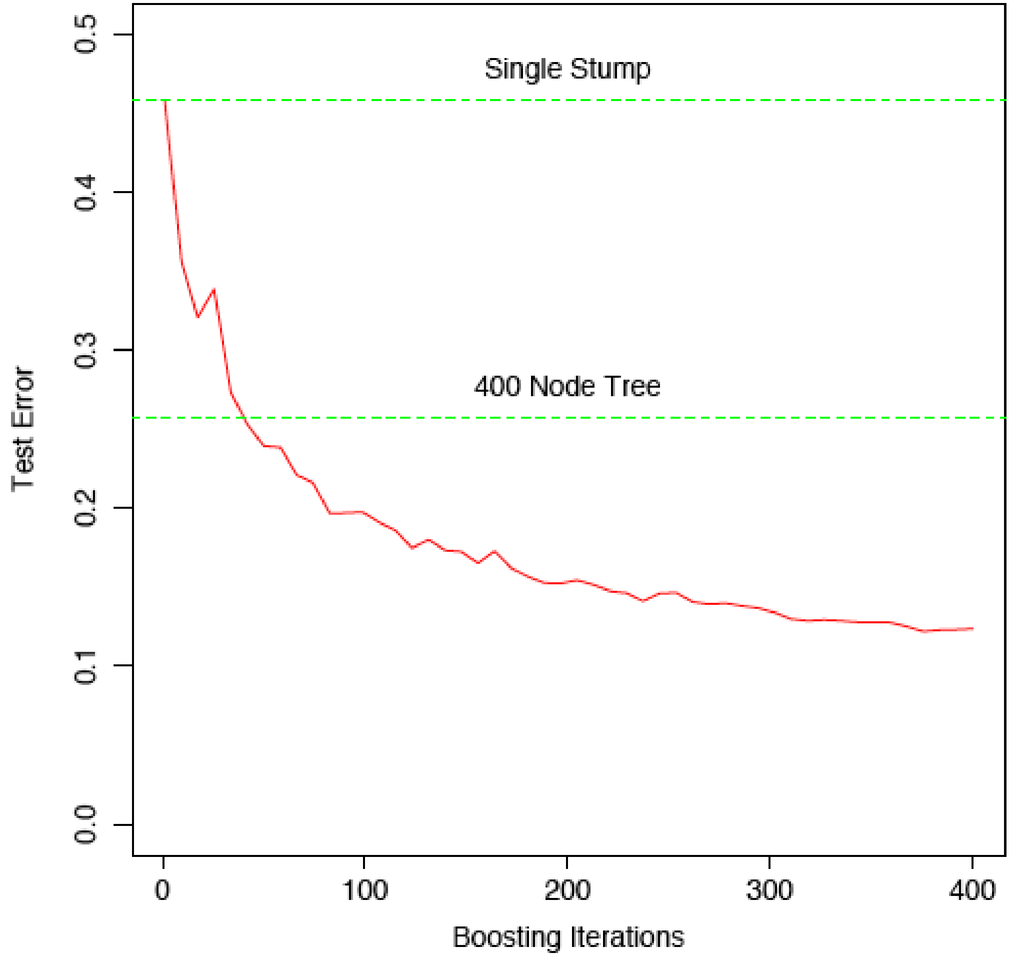
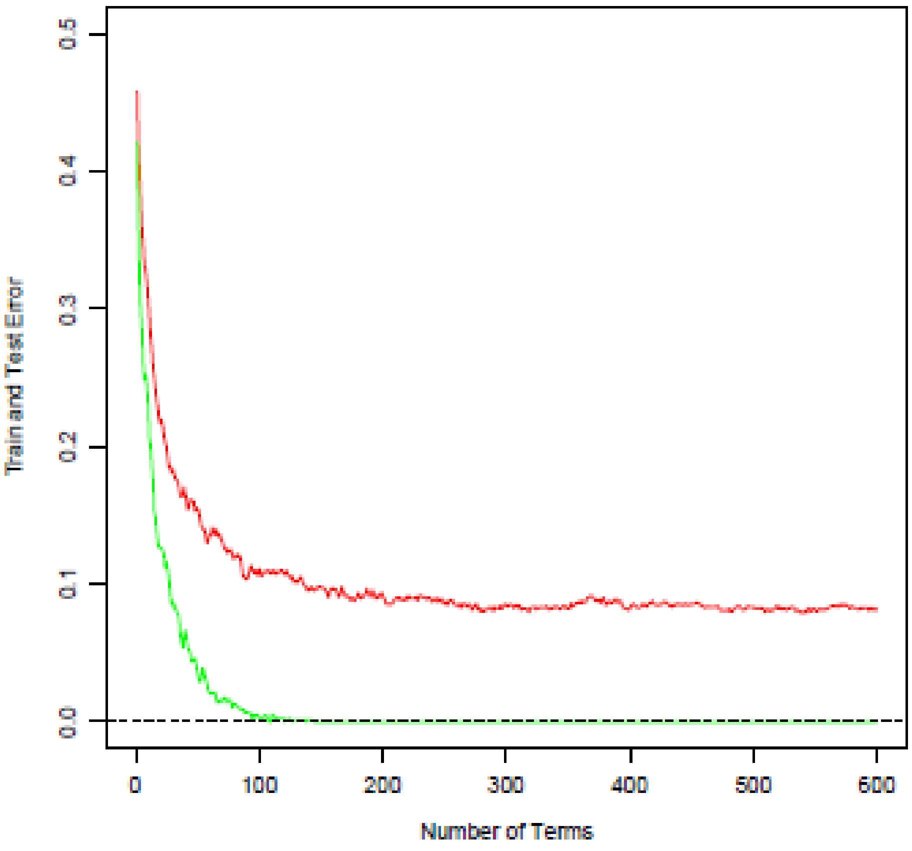


Figure 10.2: *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 400 node classification tree.*

Boosting & Training Error

Nested spheres in R^{10} — Bayes error is 0%.

Stumps

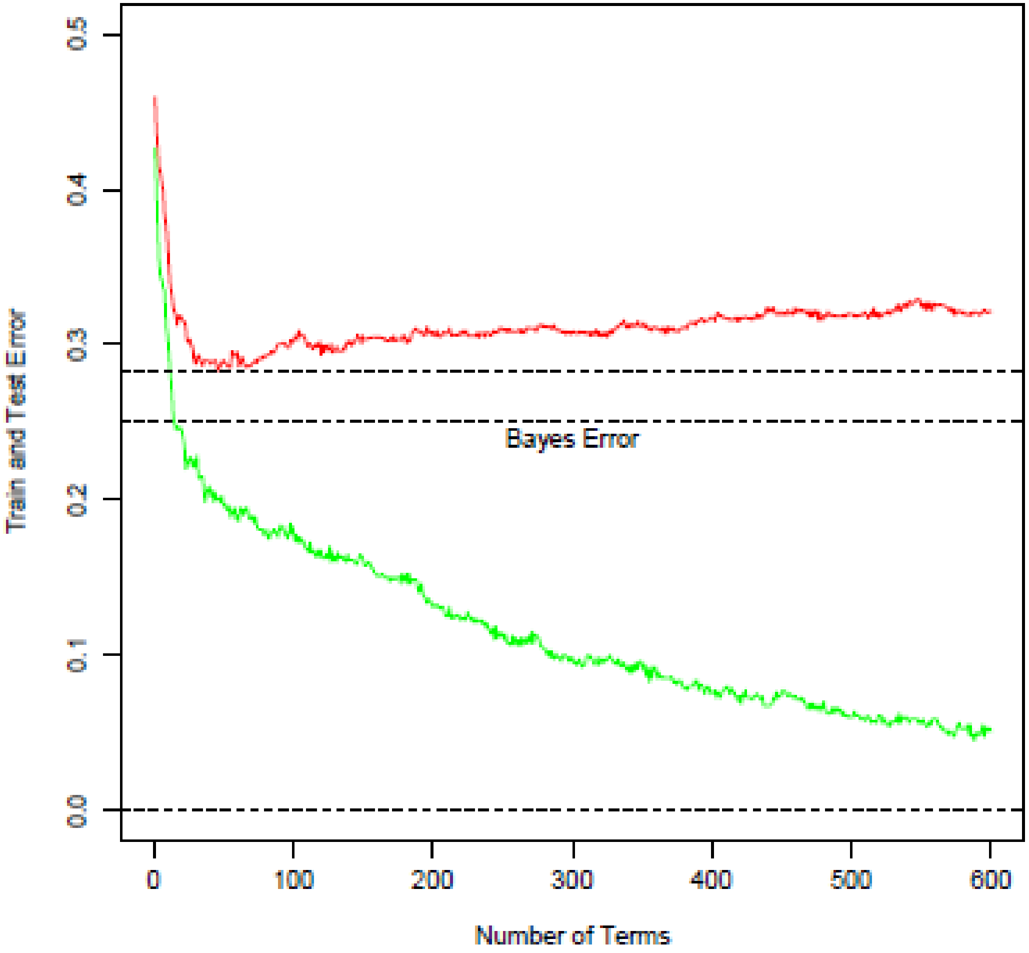


Boosting drives the training error to zero. Further iterations continue to improve test error in many examples.

Boosting Noisy Problems

Nested Gaussians in R^{10} — Bayes error is 25%.

Stumps



Here the test error does increase, but quite slowly.

General Scheme of Boosting

Boosting can be seen as fitting an additive model, with the general form:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m)$$

Expansion coefficients

Basis functions:
Simple functions of feature x, with parameters γ

Examples of γ:

Sigmoidal function in neural networks;

A split in a tree model;

General Scheme of Boosting

In general, such functions are fit by minimizing a loss function

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right)$$

This could be computationally intensive.

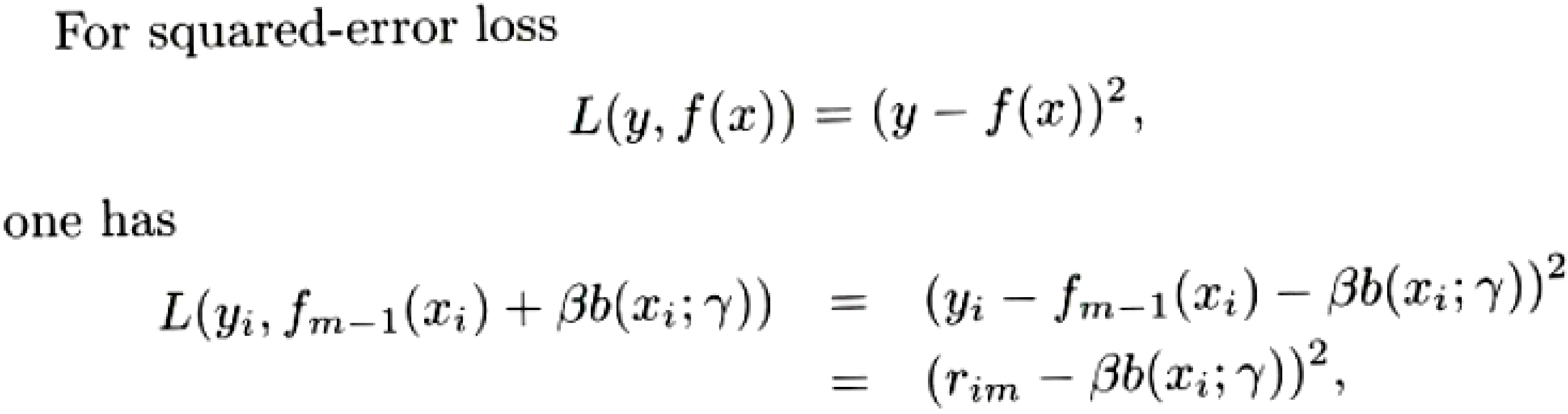
An alternative is to go stepwise, fitting a sub-problem of a single basis function

$$\min_{\beta,\gamma} \sum_{i=1}^{N} L\left(y_i, \beta b(x_i;\gamma)\right)$$

General Scheme of Boosting

Forward stagewise additive modeling --- add new basis functions without adjusting previously added ones.

Example:



For squared-error loss

$$L(y, f(x)) = (y - f(x))^2,$$

one has

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2 \\ &= (r_{im} - \beta b(x_i;\gamma))^2, \end{aligned}$$

* Squared loss function is not good for classification.

Stagewise Least Squares

Suppose we have available a basis family $b(x;\gamma)$ parametrized by γ. For example, a simple family is $b(x;\gamma_j) = x_j$.

- After $m-1$ steps, suppose we have the model $f_{m-1}(x) = \sum_{j=1}^{m-1} \beta_j b(x;\gamma_j)$.
- At the mth step we solve

$$\min_{\beta,\gamma} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2$$

- Denoting the residuals at the mth stage by $r_{im} = y_i - f_{m-1}(x_i)$, the previous step amounts to

$$\min_{\beta,\gamma} (r_{im} - \beta b(x_i;\gamma))^2,$$

- Thus the term $\beta_m b(x;\gamma_m)$ that best fits the current residuals is added to the expansion at each step.

Adaboost: Stagewise Modeling

- AdaBoost builds an additive logistic regression model

$$f(x) = \log \frac{\Pr(Y = 1|x)}{\Pr(Y = -1|x)} = \sum_{m=1}^{M} \alpha_m G_m(x)$$

  by stagewise fitting using the loss function

$$L(y, f(x)) = \exp(-y\, f(x)).$$

- Given the current $f_{M-1}(x)$, our solution for (β_m, G_m) is

$$\arg\min_{\beta,G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\, G(x))]$$

  where $G_m(x) \in \{-1, 1\}$ is a tree classifier and β_m is a coefficient.

- With $w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$, this can be re-expressed as

$$\arg\min_{\beta,G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i))$$

- We can show that this leads to the Adaboost algorithm; See pp 305.

Details of Adaboost

$E(e^{-yF(x)})$ achieves minimum at

$$F(x) = \frac{1}{2}\log\frac{P(y = 1|x)}{P(y = -1|x)}$$

Or equivalently,

$$P(y = 1|x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}, P(y = -1|x) = \frac{e^{-F(x)}}{e^{F(x)} + e^{-F(x)}}$$

Details of Adaboost

To be solved:

$$(\beta_m, G_m) = \arg\min_{\beta,G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\, G(x_i))]$$

$$= \arg\min_{\beta,G} \sum_{i=1} \boxed{\exp(-y_i\, f_{m-1}(x_i))}\, \exp(-\beta\, y_i\, G(x_i))$$

Independent from β and G

$$(\beta_m, G_m) = \arg\min_{\beta,G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i))$$

$$w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$$

Details of Adaboost

Observations are either correctly or incorrectly classified. Then the target function to be minimized is:

$$e^{-\beta} \cdot \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} \cdot \sum_{y_i \neq G(x_i)} w_i^{(m)}$$

$$= \quad (e^{\beta} - e^{-\beta}) \cdot \boxed{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))} + e^{-\beta} \cdot \sum_{i=1}^{N} w_i^{(m)}$$

For any β> 0, G_m has to satisfy:

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G(x_i))$$

G is the classifier that minimizes the weighted error rate.

Details of Adaboost

Solving for the Gm will give us a weighted error rate.

$$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}}$$

Plug it back to get ꞵ:

$$\beta_m = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m}$$

Update the overall classifier by plugging these in:

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$$

Details of Adaboost

The weight for next iteration becomes:

$$
\begin{aligned}
w_i^{(m+1)} &= \exp(-y_i\, f_m(x_i)) \\
&= \exp(-y_i\,(\, f_{m-1}(x) + \beta_m G_m(x)\,)) \\
&= w_i^{(m)} \cdot e^{-\beta_m y_i G_m(x_i)}
\end{aligned}
$$

Using $-y_i G_m(x_i) = 2 \cdot I(y_i \neq G_m(x_i)) - 1$

$$
w_i^{(m+1)} = w_i^{(m)} \cdot e^{\alpha_m I(y_i \neq G_m(x_i))} \boxed{\cdot e^{-\beta_m}}
$$

$$
\alpha_m = 2\beta_m
$$

Independent of i.
Ignored.

Gradient Boosted Decision Trees

Stage-wise Boosting

- 输入：训练数据集$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N\ , y_N)\}$

  损失函数$L\big(y, f(x)\big)$

  基函数集$\{b(x;\gamma)\}$

  输出：加分模型$f(x)$
- (1)初始化$f_0(x) = 0$
- (2)对$m = 1,2,3, \dots, M$
  - (a)极小化损失函数

$$(\beta_m\ , \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x;\gamma))$$

Stage-wise Boosting

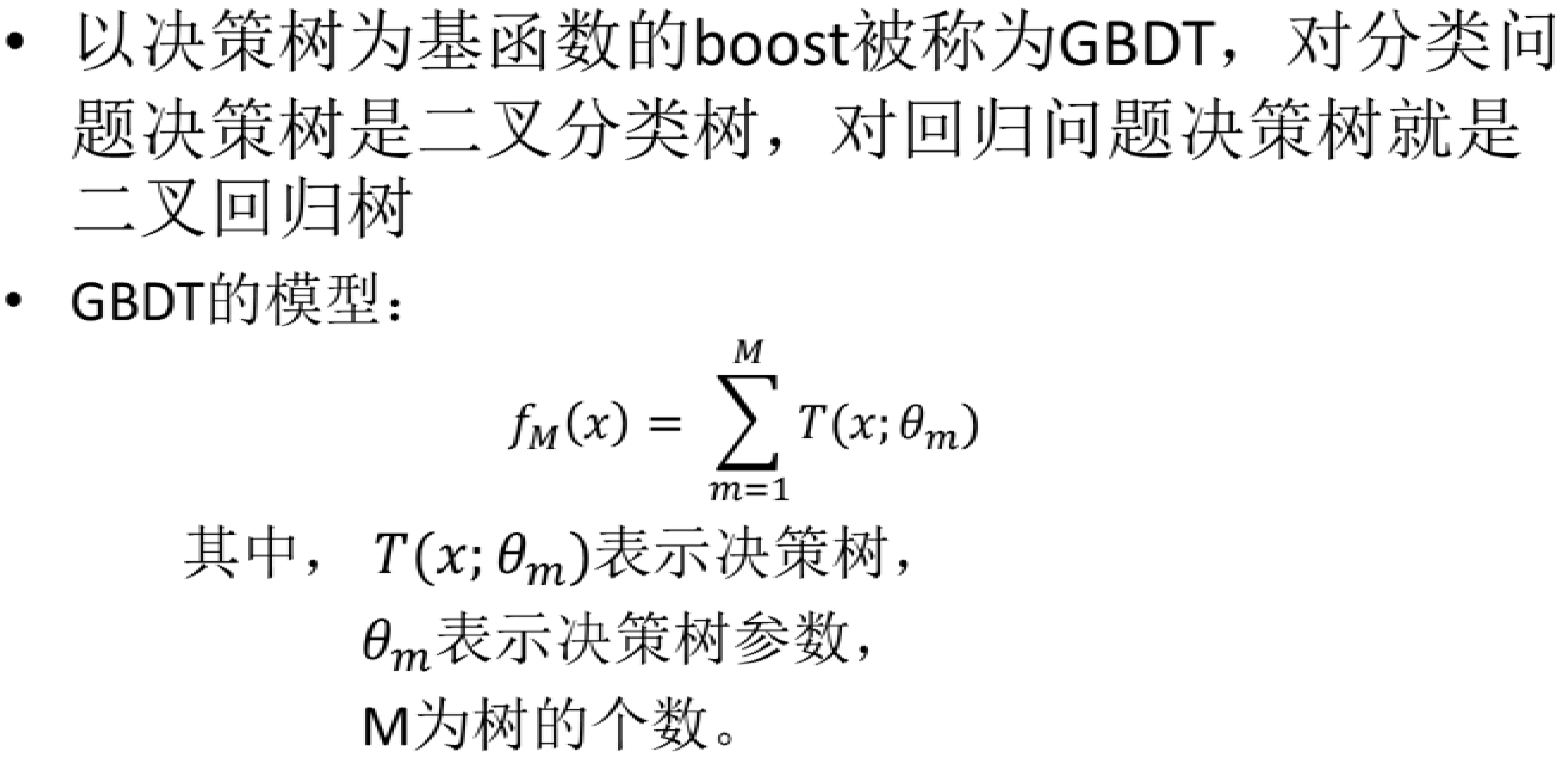
- b)更新

$$f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$$

- 3)得到加法模型

$$f(x) = f_M(x) = \sum_{m=1}^{M} \beta_m b(x; y_m)$$

Gradient Boosted Decision Tree



- 以决策树为基函数的boost被称为GBDT，对分类问题决策树是二叉分类树，对回归问题决策树就是二叉回归树
- GBDT的模型：

$$f_M(x) = \sum_{m=1}^{M} T(x;\theta_m)$$

其中，$T(x;\theta_m)$表示决策树，

θ_m表示决策树参数，

M为树的个数。

Gradient Boosted Decision Tree

- 已知一个训练数据集$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N\ , y_N)\}, x_i \in \ \chi \ \subseteq R^n$

χ为输入空间，$y_i \in \ \gamma \ \subseteq R$，$\gamma$ 为输出空间。如果将输入空间χ划分为J个互不相交的区域$R_1, R_2, \dots, R_J$,并且在每个区域上输出的常量c_j，那么树可以表示为

$$T(x; \Theta) = \sum_{j=1}^{J} c_j I(x \in R_j)$$

其中，参数$\Theta = \left\{(R_1, c_1), (R_2, c_2), \dots, \left(R_J, c_J\right)\right\}$表示树的区域划分和各区域上的常数。J是回归树的复杂度即叶节点个数。

GBDT for Regression

- 在回归问题中，使用以下的前向分步算法：

$$f_0(x) = 0$$

$$f_m(x) = f_{m-1}(x) + T(x;\Theta)，\ m = 1,2,\ldots,M$$

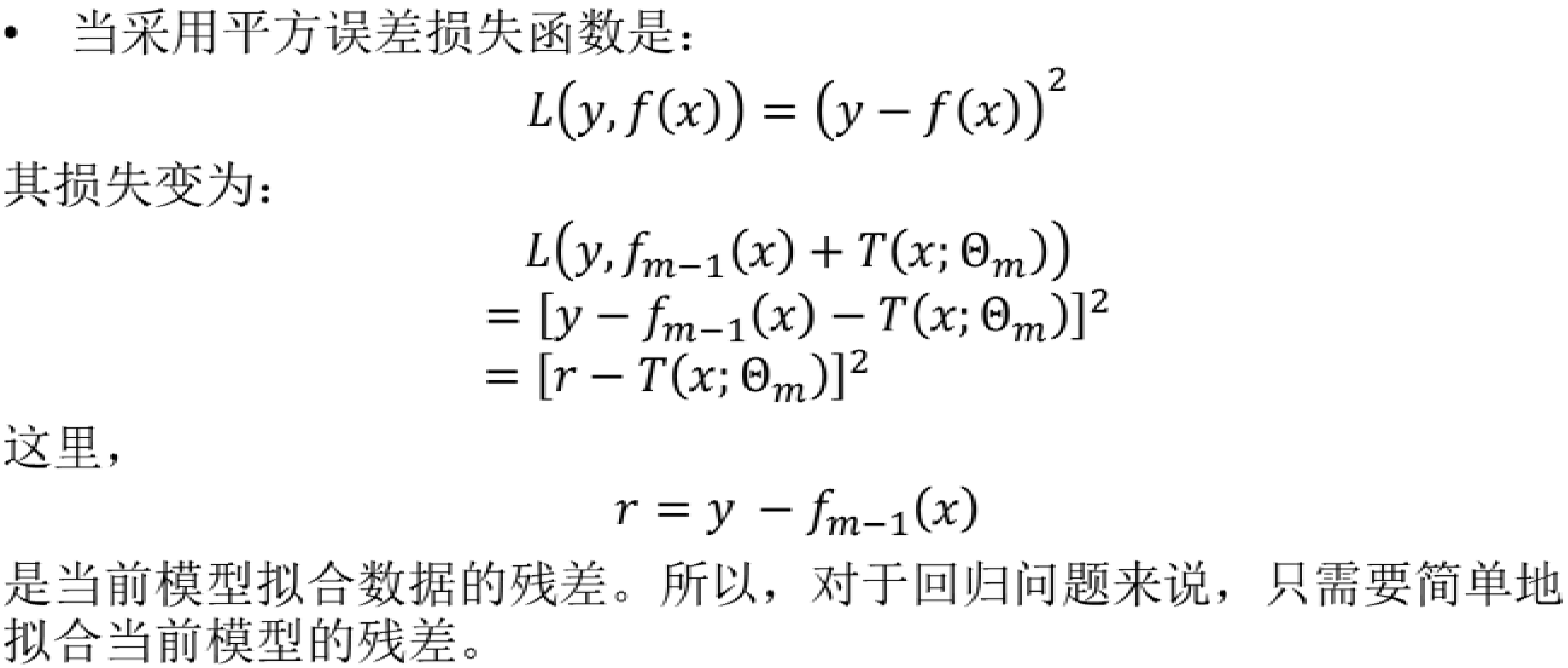
$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta)$$

在算法的第m步，给定当前的模型$f_{m-1}(x)$，需求解：

$$\Theta_{\mathrm{m}} = \arg\min_{\Theta_{\mathrm{m}}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x) + T(x;\Theta_m))$$

得到Θ_{m}，也就得到第m棵树的参数。

GBDT for Regression



- 当采用平方误差损失函数是：

$$L\big(y,f(x)\big)=\big(y-f(x)\big)^2$$

其损失变为：

$$\begin{aligned}&L\big(y,f_{m-1}(x)+T(x;\Theta_m)\big)\\&=[y-f_{m-1}(x)-T(x;\Theta_m)]^2\\&=[r-T(x;\Theta_m)]^2\end{aligned}$$

这里，

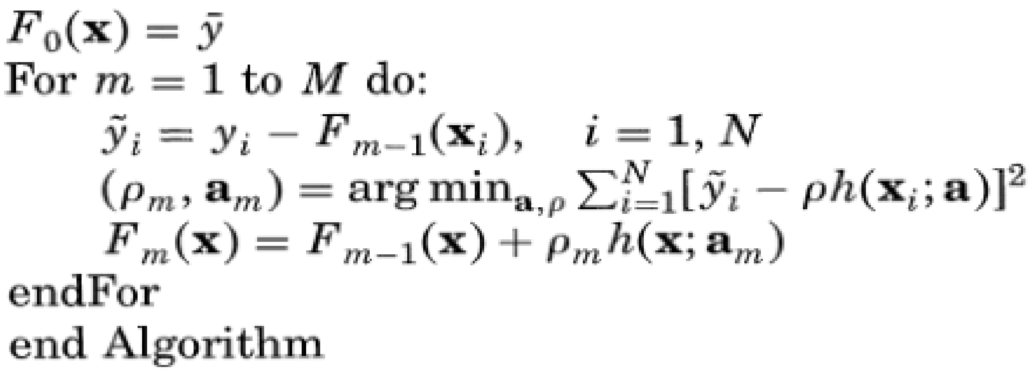
$$r=y\ -f_{m-1}(x)$$

是当前模型拟合数据的残差。所以，对于回归问题来说，只需要简单地拟合当前模型的残差。

GBDT for Regression (algorithm scheme)

- 1、Calculate the average:$\mathrm{y}^{'} = \bar{y} = \sum_{i=0}^{n} y_i / n$;
- 2、Build decision tree by the data sets$\left(\mathbf{X}, \mathrm{y}^{'} - \mathrm{y}\right)$;
- 3、Update the value:
  - $\mathrm{y}' = y' - learning\ rate * Tree$;
- 4、Repeat 2 && 3 until it ends.

GBDT for Regression (algorithm)



```
$F_0(\mathbf{x}) = \bar{y}$
For $m = 1$ to $M$ do:
    $\tilde{y}_i = y_i - F_{m-1}(\mathbf{x}_i), \quad i = 1, N$
    $(\rho_m, \mathbf{a}_m) = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^{N} [\tilde{y}_i - \rho h(\mathbf{x}_i; \mathbf{a})]^2$
    $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
endFor
end Algorithm
```

General GBDT

- 前面提到的GBDT中回归情况，当损失函数是平方损失的时候，优化比较简单。但是对于一般损失函数来说，往往每一步的优化并不容易。针对这个问题，Freidman提出了梯度提升（gradient boost）算法，其关键是利用损失函数的负梯度在当前模型的值：

$$-\left[\frac{\partial L\big(y, f(x_i)\big)}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

作为回归问题提升树算法中的残差的近似值，拟合一个回归树。

General GBDT

- 输入：训练数据集$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N\ , y_N)\}, x_i \in\ \chi\ \subseteq R^n$

  χ为输入空间，$y_i \in \gamma\ \subseteq R$，$\gamma$ 为输出空间；损失函数$L(y, f(x))$;

  输出：回归树$f(x)$

(1)初始化

$$f_0(x) = \arg\min_c \sum_{I=1}^{N} L(y_i, c)$$

(2)对m=1,2,…,M

(a)对$i = 1,2, \dots., N$,计算

$$r_{mi} = -\left[\frac{\partial L\big(y, f(x_i)\big)}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

(b)对r_{mi}拟合一个回归树，得到第m棵树的叶节点区域R_{mj}，j = 1,2,….,J

General GBDT

- (c) 对$j = 1,2, \ldots, J$,计算：

$$\mathrm{c_{mj}} = \arg\min_{c} \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + c)$$

(d)更新$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J} c_{mj}\, I(x \in R_{mj})$

(3)得到回归树

$$f(x) = f_M(x) = \sum_{m=1}^{M} \sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

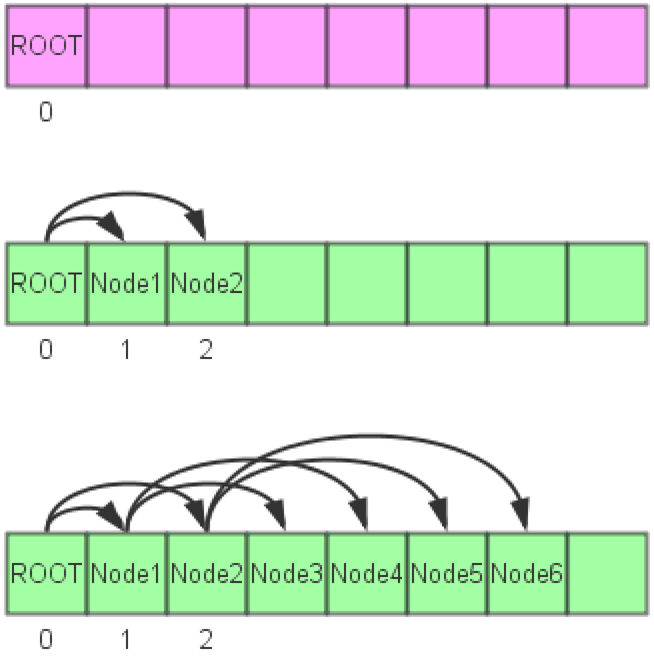
如何实现高效的并行化 GBDT/RF

GBDT/随机森林的并行化

- 随机森林
  - 并行地构造成千上万棵树
  - 策略相对简单
- GBDT
  - 每次构造一棵树
  - 需要更细致的并行策略

代码实现

- 非递归建树
  - 节点的存放



ROOT

0

ROOT Node1 Node2

0 1 2

ROOT Node1 Node2 Node3 Node4 Node5 Node6

0 1 2

代码实现

- 非递归建树
    - 终止条件
        - 树的节点数
        - 树的深度
        - 没有适合分割的节点

代码实现

- 特征值排序

　　在对每个节点进行分割的时候，首先需要遍历所有的特征，然后对每个样本的特征的值进行枚举计算。（CART）

　　在对单个特征量进行枚举取值之前，我们可以先将该特征量的所有取值进行排序，然后再进行计算。

代码实现

- 特征值排序

  1、避免计算重复的value值；

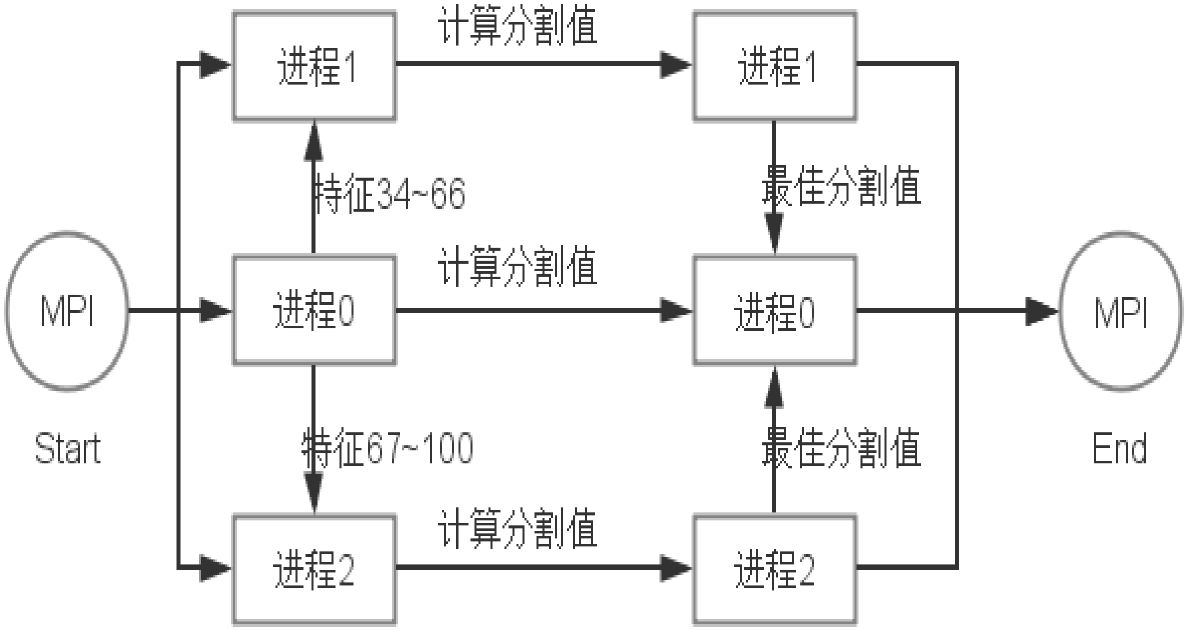
  2、方便最佳分割值的确定；

  3、减少信息的重复计算。

多线程/MPI并行化的实现

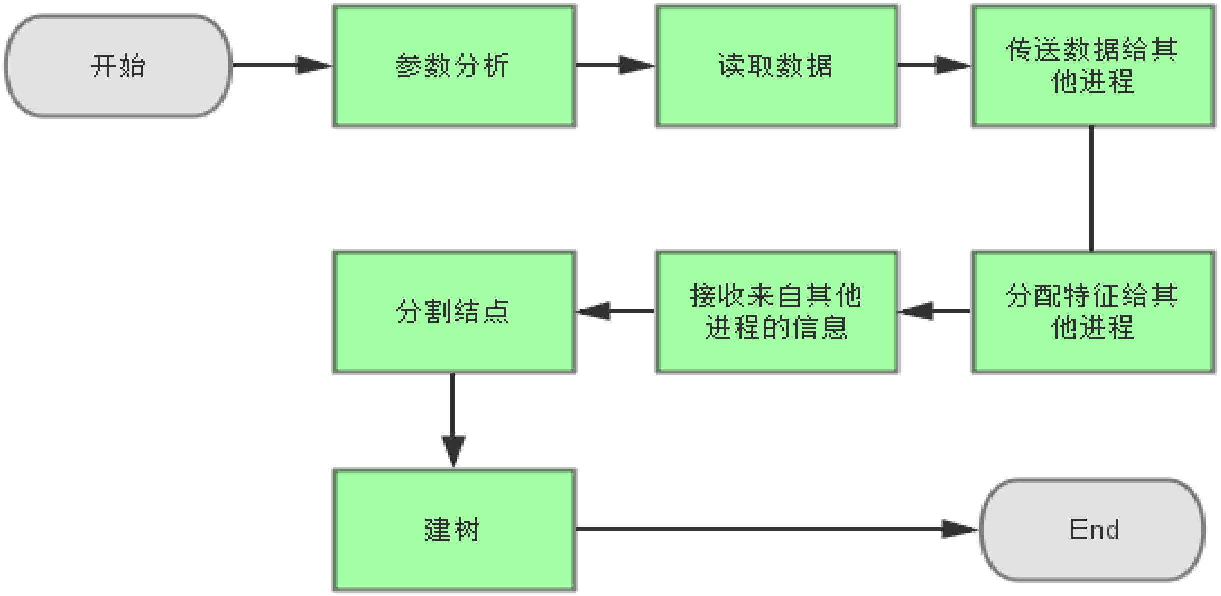
- 与RF相比，GBDT在构建第m棵树的时候，需要依赖前m-1棵树的信息，故每次只能建一棵树；
- 通过MPI实现对GBDT的并行化，最主要的步骤是在建树的过程中，由于每个特征值计算最佳分割值是相互独立的，故可以对特征进行平分，再同时进行计算。

MPI并行化的实现


MPI
Start
进程1
进程0
进程2
特征34~66
特征67~100
计算分割值
计算分割值
计算分割值
进程1
进程0
进程2
最佳分割值
最佳分割值
MPI
End

MPI并行化的实现

- 主进程



开始

参数分析

读取数据

传送数据给其他进程

分配特征给其他进程

接收来自其他进程的信息

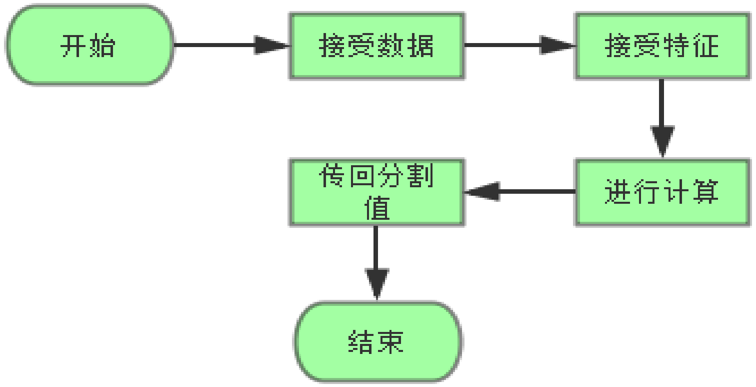
分割结点

建树

End

MPI并行化的实现

- 其他进程


开始
接受数据
接受特征
进行计算
传回分割值
结束