Data Mining

Chapter 5: Decision Tree and Ensemble Learning

Yunming Ye, Baoquan Zhang

School of Computer Science

Harbin Institute of Technology, Shenzhen

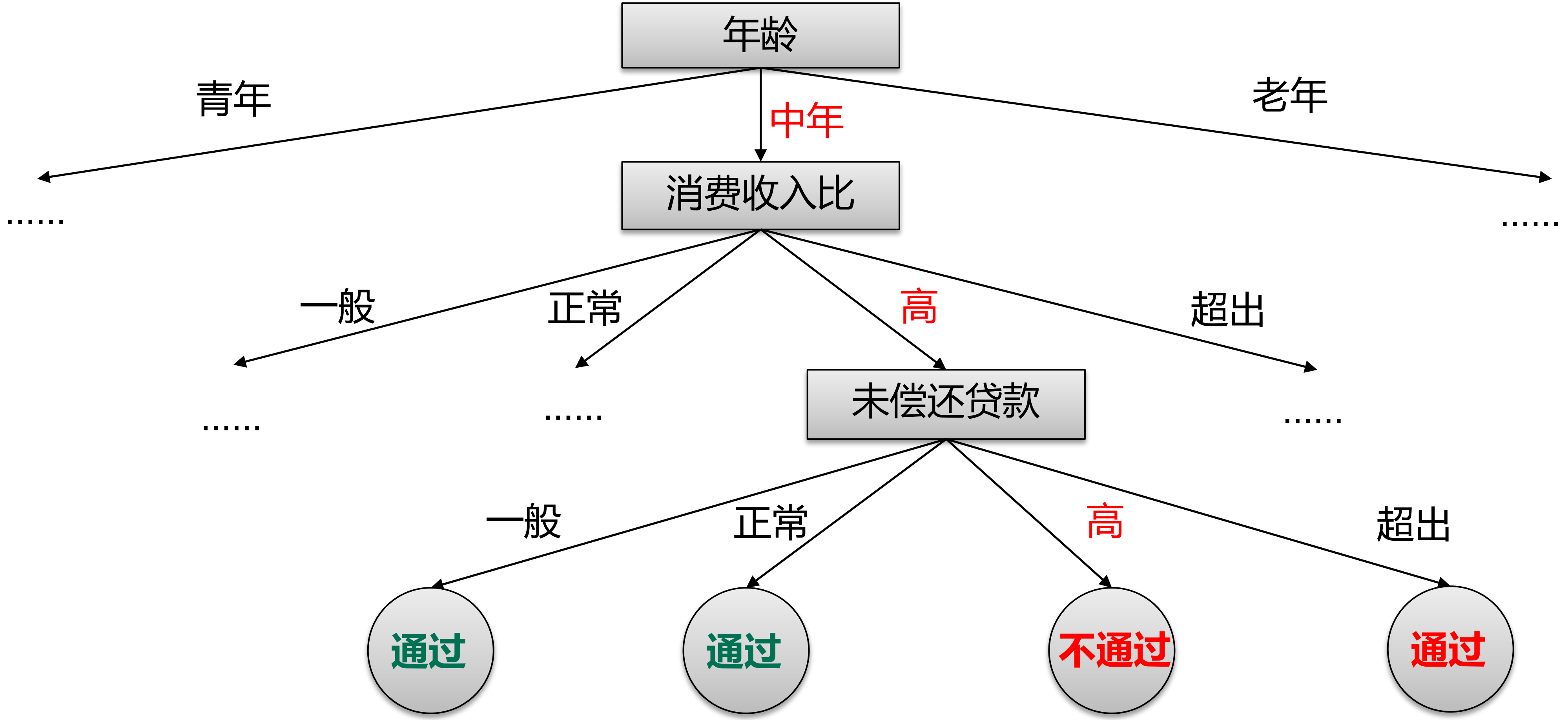
Agenda

- Decision Tree
  - Basic Idea of Decision Tree Induction
  - Decision Tree Classification Algorithms
  - Performance Evaluation and Tree Pruning
  - Regression Tree
- Ensemble Learning
  - Basic Idea of Ensemble Learning
  - Gradient Boosting Methods
  - Deep Forest

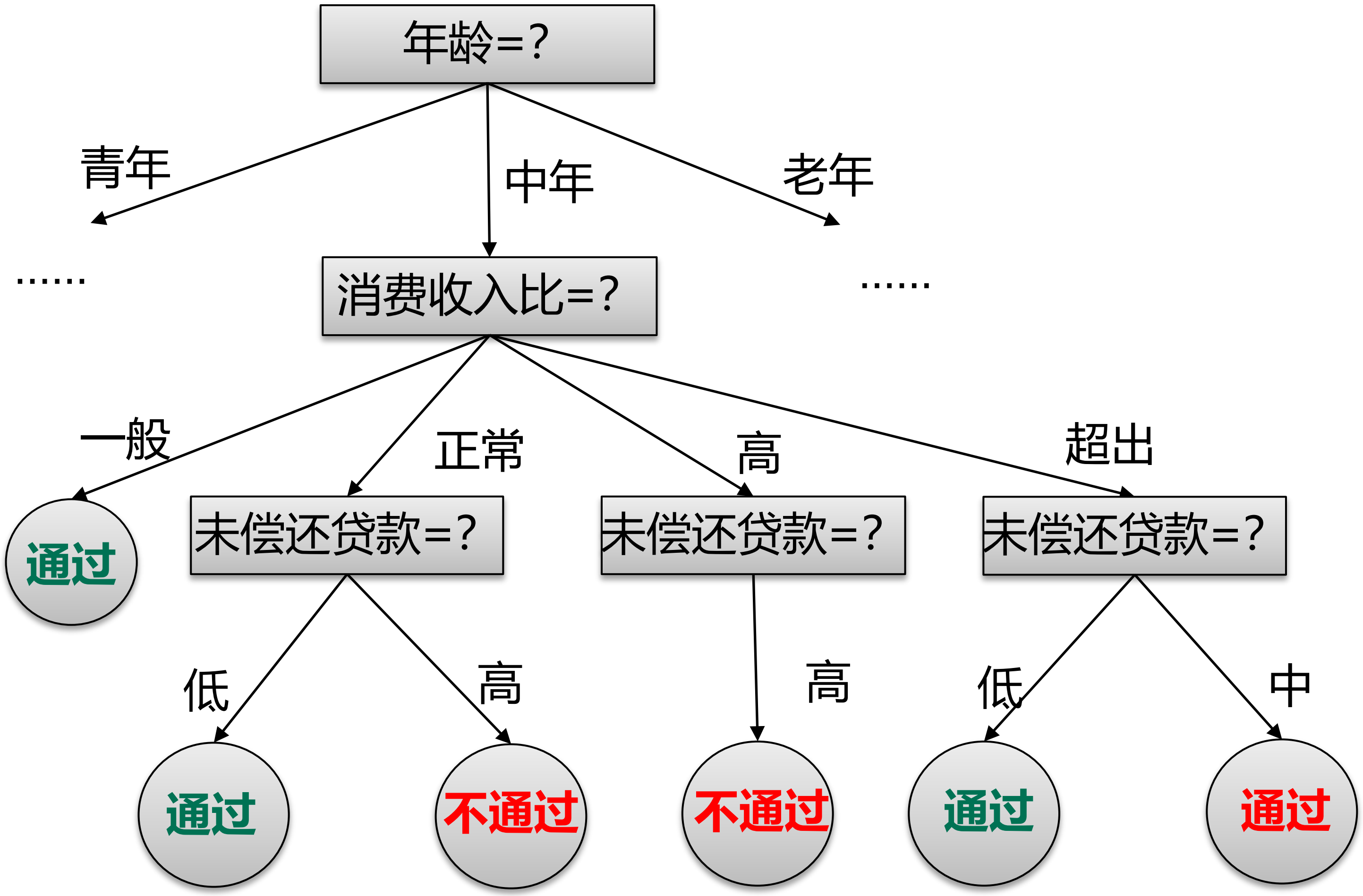
6.1 Basic Idea of Decision Tree Induction

Tree-like Decision Process: an example

年龄	消费收入比	未偿还贷款
中年	高	高



Basic idea of Decision Tree Induction for Classification



序号	年龄	消费收入比	未偿还贷款	审批
1	中年	高	高	不通过
2	中年	一般	高	通过
3	中年	一般	较高	通过
4	中年	一般	低	通过
5	中年	一般	高	通过
6	老年	正常	低	通过
7	中年	超出	中	通过
8	中年	一般	较高	通过
9	青年	超出	低	通过
10	中年	正常	低	通过
11	中年	一般	较高	通过
12	中年	正常	低	不通过
13	中年	超出	中	不通过
14	中年	正常	高	不通过
15	中年	正常	低	不通过

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

6.2 ID3、C4.5、CART

Attribute Selection Measure: Information Gain

$$H(D) = -\sum_{i=1}^{N} p_i \log p_i = -\sum_{i=1}^{N} \frac{|D_i|}{|D|} \log \frac{|D_i|}{|D|}$$

$$H(D|A) = \sum_{j=1}^{v} \frac{\left|D^{(j)}\right|}{|D|} H\left(D^{(j)}\right)$$

$$I(D; A) = H(D) - H(D|A)$$

Compute H(D)

$$H(D) = \boxed{-\frac{10}{15}\log\frac{10}{15}} \boxed{-\frac{5}{15}\log\frac{5}{15}} = 0.918$$

序号	年龄	消费收入比	未偿还贷款	审批
1	中年	高	高	不通过
2	中年	一般	高	通过
3	中年	一般	较高	通过
4	中年	一般	低	通过
5	中年	一般	高	通过
6	老年	正常	低	通过
7	中年	超出	中	通过
8	中年	一般	较高	通过
9	青年	超出	低	通过
10	中年	正常	低	通过
11	中年	一般	较高	通过
12	中年	正常	低	不通过
13	中年	超出	中	不通过
14	中年	正常	高	不通过
15	中年	正常	低	不通过

Compute H(T|A=年龄)

$$H(D|A_1) = \boxed{\frac{1}{15}H(T^{青年})} + \boxed{\frac{13}{15}H(T^{中年})} + \boxed{\frac{1}{15}H(T^{老年})}$$

$$= \boxed{\frac{1}{15} \times \left(-\frac{1}{1}\log\frac{1}{1}\right)} +$$

$$\frac{13}{15} \times \left(\boxed{-\frac{8}{13}\log\frac{8}{13}} - \boxed{\frac{5}{13}\log\frac{5}{13}}\right) +$$

$$\boxed{\frac{1}{15} \times \left(-\frac{1}{1}\log\frac{1}{1}\right)}$$

$$= 0.833$$

序号	年龄	消费收入比	未偿还贷款	审批
1	中年	高	高	不通过
2	中年	一般	高	通过
3	中年	一般	较高	通过
4	中年	一般	低	通过
5	中年	一般	高	通过
6	老年	正常	低	通过
7	中年	超出	中	通过
8	中年	一般	较高	通过
9	青年	超出	低	通过
10	中年	正常	低	通过
11	中年	一般	较高	通过
12	中年	正常	低	不通过
13	中年	超出	中	不通过
14	中年	正常	高	不通过
15	中年	正常	低	不通过

Compute H(D|A=消费收入比)、 H(D|A=未偿还贷款)

$$H(T|A_2) = \frac{6}{15}H(T^{\text{一般}}) + \frac{5}{15}H(T^{\text{正常}}) + \frac{1}{15}H(T^{\text{高}}) + \frac{3}{15}H(T^{\text{超出}})$$
$$= \frac{6}{15} \times \left(-\frac{6}{6}\log\frac{6}{6}\right) + \frac{5}{15} \times \left(-\frac{2}{5}\log\frac{2}{5} - \frac{3}{5}\log\frac{3}{5}\right) + \frac{1}{15} \times \left(-\frac{1}{1}\log\frac{1}{1}\right) +$$
$$\frac{3}{15} \times \left(-\frac{2}{3}\log\frac{2}{3} - \frac{1}{3}\log\frac{1}{3}\right) = 0.507$$

$$H(T|A_3) = \frac{6}{15}H(T^{\text{低}}) + \frac{2}{15}H(T^{\text{中}}) + \frac{3}{15}H(T^{\text{较高}}) + \frac{4}{15}H(T^{\text{高}})$$
$$= \frac{6}{15} \times \left(-\frac{4}{6}\log\frac{4}{6} - \frac{2}{6}\log\frac{2}{6}\right) + \frac{2}{15} \times \left(-\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2}\right) +$$
$$\frac{3}{15} \times \left(-\frac{3}{3}\log\frac{3}{3}\right) + \frac{4}{15} \times \left(-\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4}\right) = 0.767$$

序号	年龄	消费收入比	未偿还贷款	审批
1	中年	高	高	不通过
2	中年	一般	高	通过
3	中年	一般	较高	通过
4	中年	一般	低	通过
5	中年	一般	高	通过
6	老年	正常	低	通过
7	中年	超出	中	通过
8	中年	一般	较高	通过
9	青年	超出	低	通过
10	中年	正常	低	通过
11	中年	一般	较高	通过
12	中年	正常	低	不通过
13	中年	超出	中	不通过
14	中年	正常	高	不通过
15	中年	正常	低	不通过

Compute I(D;A)

$$I(D;A_1) = H(D) - H(D|A_1) = 0.918 - 0.833 = 0.085$$

$$I(D;A_2) = H(D) - H(D|A_2) = 0.918 - 0.507 = 0.411$$

$$I(D;A_3) = H(D) - H(D|A_3) = 0.918 - 0.767 = 0.151$$

$$I(D;A_{max}) = I(D;A_2) = 0.411$$

序号	年龄	消费收入比	未偿还贷款	审批
1	中年	高	高	不通过
2	中年	一般	高	通过
3	中年	一般	较高	通过
4	中年	一般	低	通过
5	中年	一般	高	通过
6	老年	正常	低	通过
7	中年	超出	中	通过
8	中年	一般	较高	通过
9	青年	超出	低	通过
10	中年	正常	低	通过
11	中年	一般	较高	通过
12	中年	正常	低	不通过
13	中年	超出	中	不通过
14	中年	正常	高	不通过
15	中年	正常	低	不通过

6.3 How to Compute Information Gain for Continuous Attributes?

C4.5: Gain Ratio

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

$$I_R(D;A) = \frac{I(D;A)}{SplitInfo_A(D)}$$

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

Gini index (CART, IBM IntelligentMiner)

- If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in D

- If a data set D is split on A into two subsets D_1 and D_2, the *gini* index $gini(D)$ is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

6.4 How to Compute Gini index for Categorical Attributes?

Other Attribute Selection Measures

- CHAID: a popular decision tree algorithm, measure based on χ^2 test for independence
- C-SEP: performs better than info. gain and gini index in certain cases
- G-statistics: has a close approximation to χ^2 distribution
- MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):
  - The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- Multivariate splits (partition based on multiple variable combinations)
  - CART: finds multivariate splits based on a linear comb. of attrs.
- Which attribute selection measure is the best?
  - Most give good results, none is significantly superior than others

6.5 Performance Evaluation and Tree Pruning

Evaluation of Misclassification Error

- Evaluation data
- Training set/validation set/testing set

Classifier Accuracy Measures

- **Accuracy** of a classifier M, Acc(M): percentage of test set tuples that are correctly classified by the model M.
  - Error rate(misclassification rate) of M=1-acc(M)

Confusion matrix

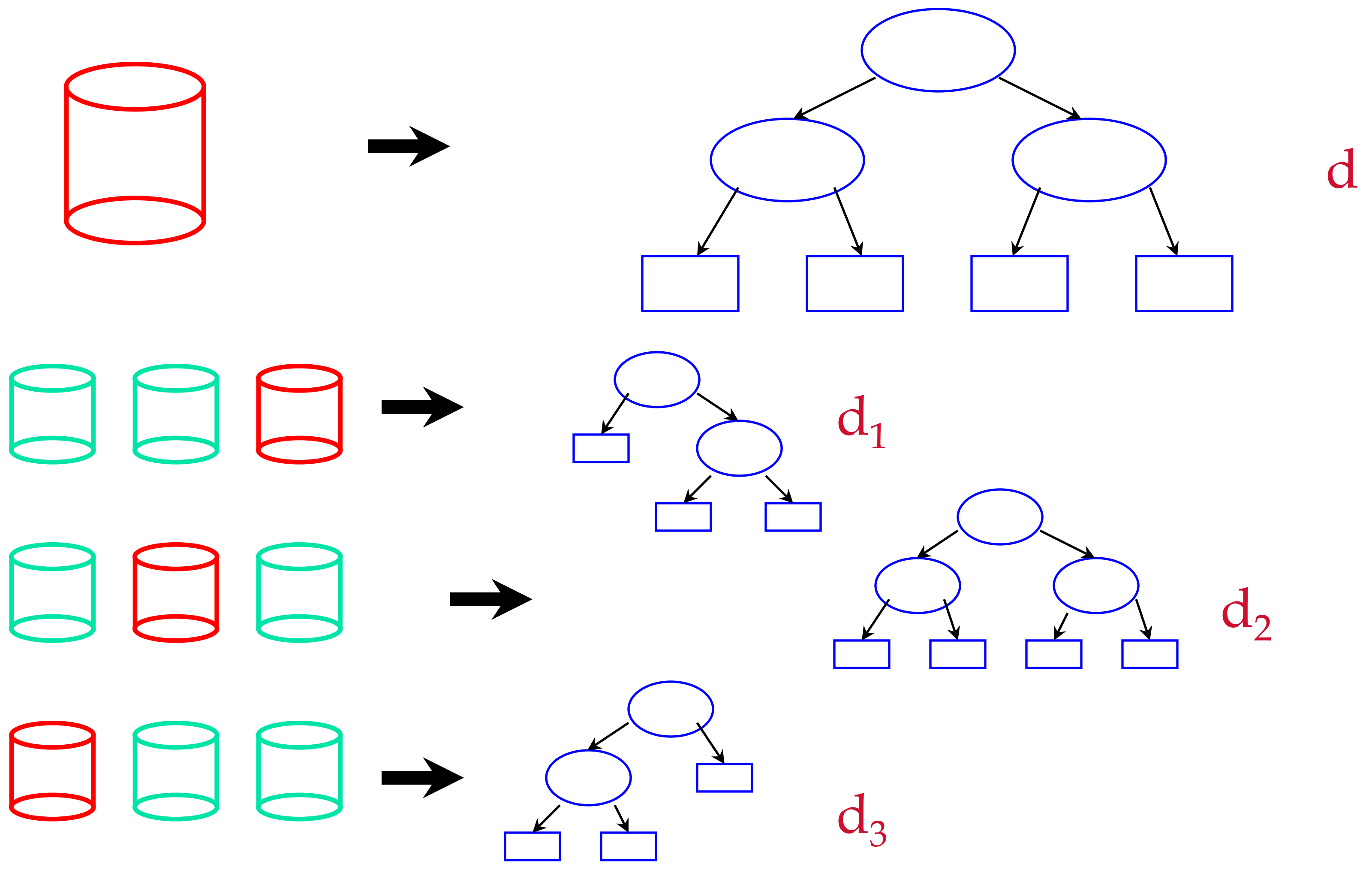
Predicted class

Actual class

	C1	C2
C1	true positives	false negatives
C2	false positives	true negatives

classes	buy_computer=yes	buy_computer=no	total	recognition(%)
buy_computer=yes	6954	46	7000	99.34
buy_computer=no	412	2588	3000	86.27
total	7366	2634	10000	95.52

Cross-Validation



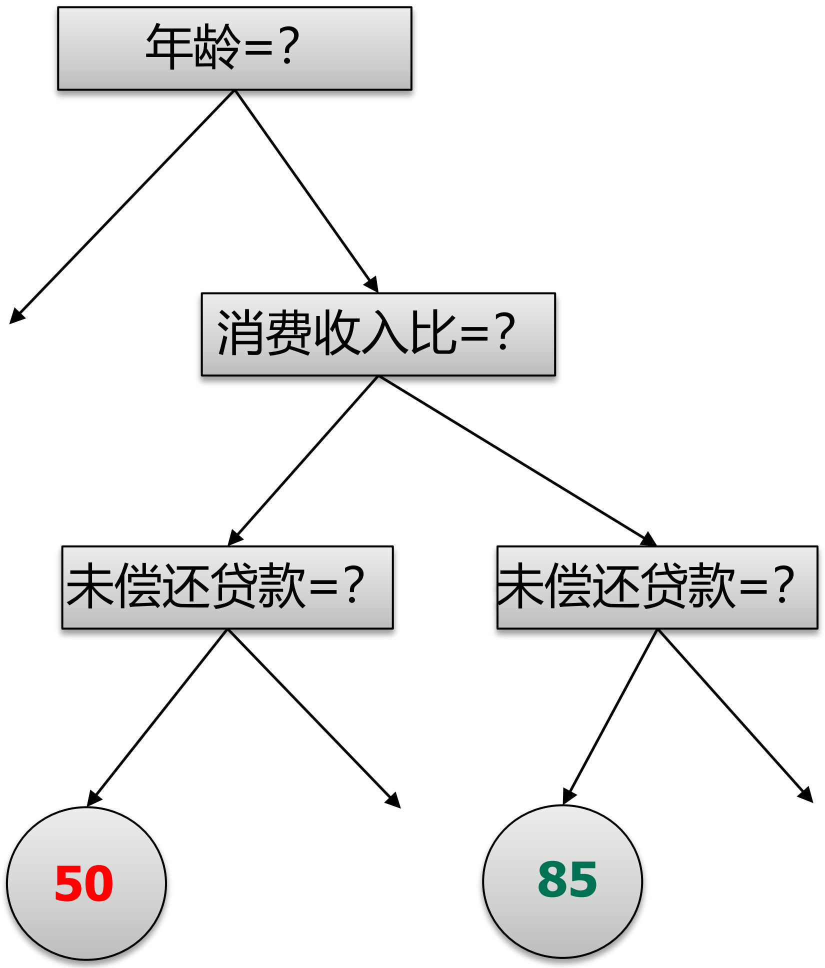
Overfitting Problem & Pruning Strategies in Tree Induction

- Overfitting problem of decision tree

- Tree pruning：
  - Pre-pruning
  - Post-pruning

6.6 Regression Tree

Regression Tree Induction



序号	未偿还贷款比	年龄	消费收入比	评估分
1	0.7	35	0.6	50
2	0.3	36	0.7	85
3	0.2	25	1.2	45
4	0.1	40	0.3	90
5	0.5	50	1.1	60
…	…	…	…	…

- **Loss function**(MSE):

Minimize: $$\frac{1}{n}\sum_{m=1}^{M}\sum_{\boldsymbol{x}_i\in R_m}(c_m-y_i)^2$$

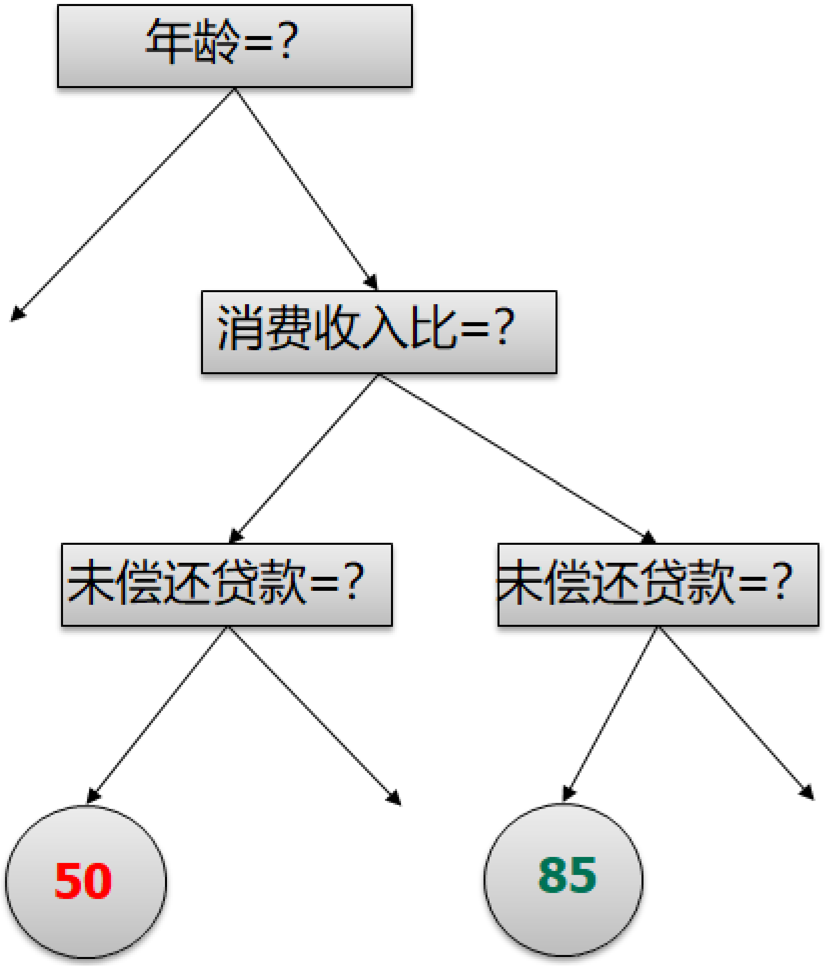
$$c_m = ave(y_i|\boldsymbol{x}_i \in leaf_m)$$

➤ If we select attribute *j* and its value *s* as a splitting point:

$$\sum_{x_i\in R_1}(y_i-c_1)^2+\sum_{x_i\in R_2}(y_i-c_2)^2$$

$$\min_{j,s}[\min_{c_1}\sum_{x_i\in R_1}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2}(y_i-c_2)^2]$$

$$c_1=\frac{1}{N_1}\sum_{x_i\in R_1}y_i \qquad c_2=\frac{1}{N_2}\sum_{x_i\in R_2}y_i$$



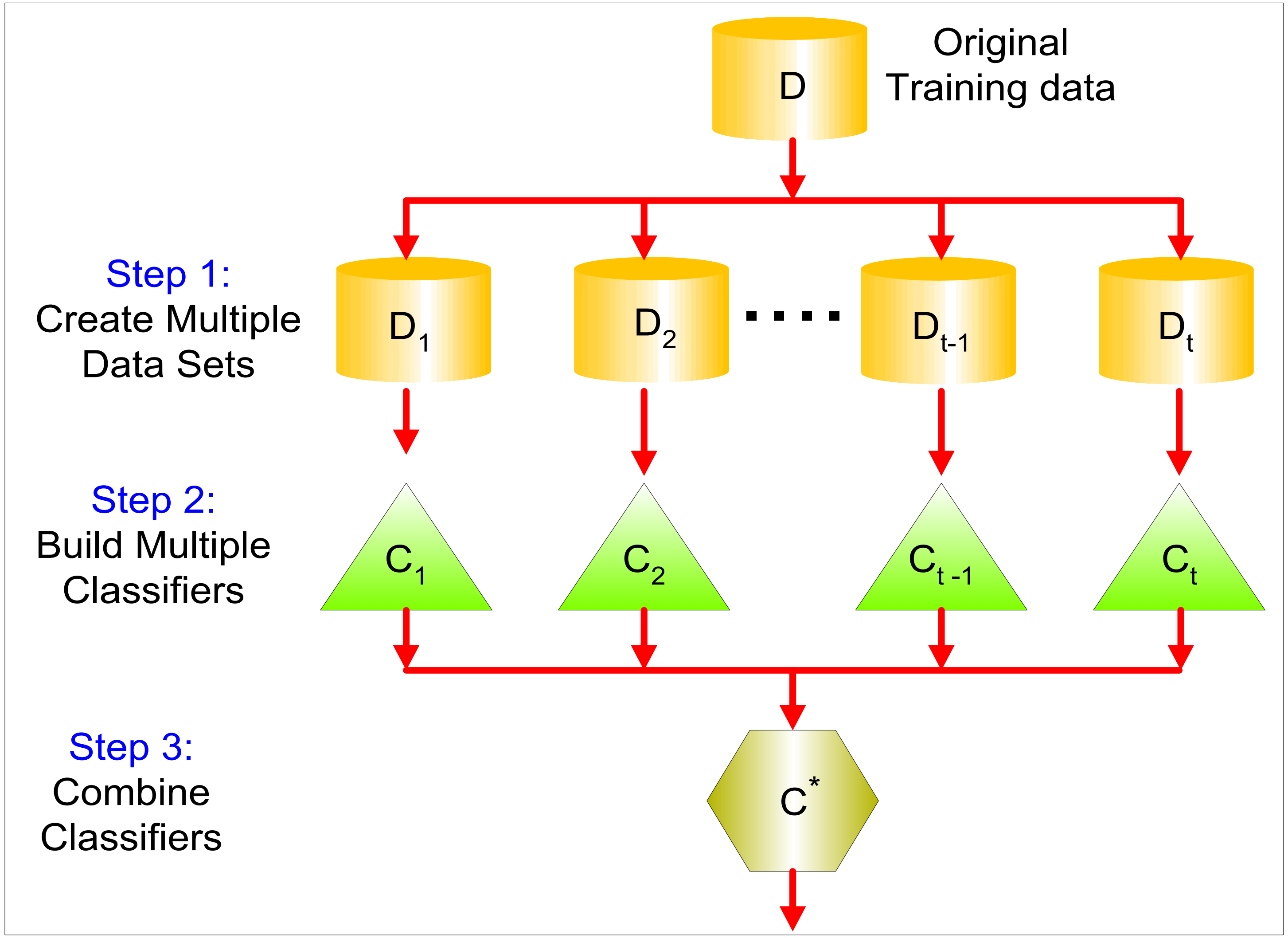
6.7 Introduction to Ensemble Learning

Ensemble Methods

- Construct a set of classifiers from the training data

- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers

Main Idea


Original
Training data
D
Step 1:
Create Multiple
Data Sets
D_1
D_2
....
D_{t-1}
D_t
Step 2:
Build Multiple
Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3:
Combine
Classifiers
C^*

Why does it work?

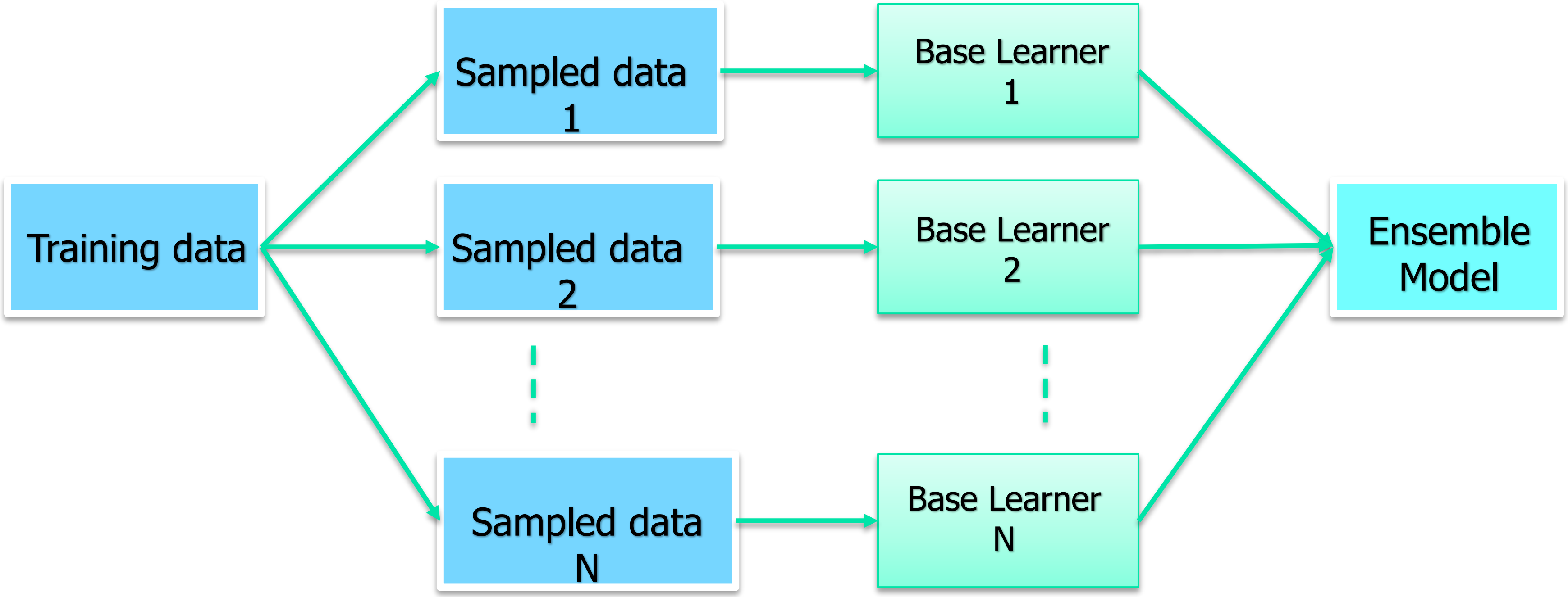
- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction:

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

Examples of Ensemble Methods

- How to generate an ensemble of classifiers?
  - Stacking
  - Bagging
  - Boosting

stacking


Training data
Sampled data 1
Sampled data 2
Sampled data N
Base Learner 1
Base Learner 2
Base Learner N
Ensemble Model

Bagging

- Sampling with replacement
- Build classifier on each bootstrap sample
- Each sample has equal probability of being selected

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

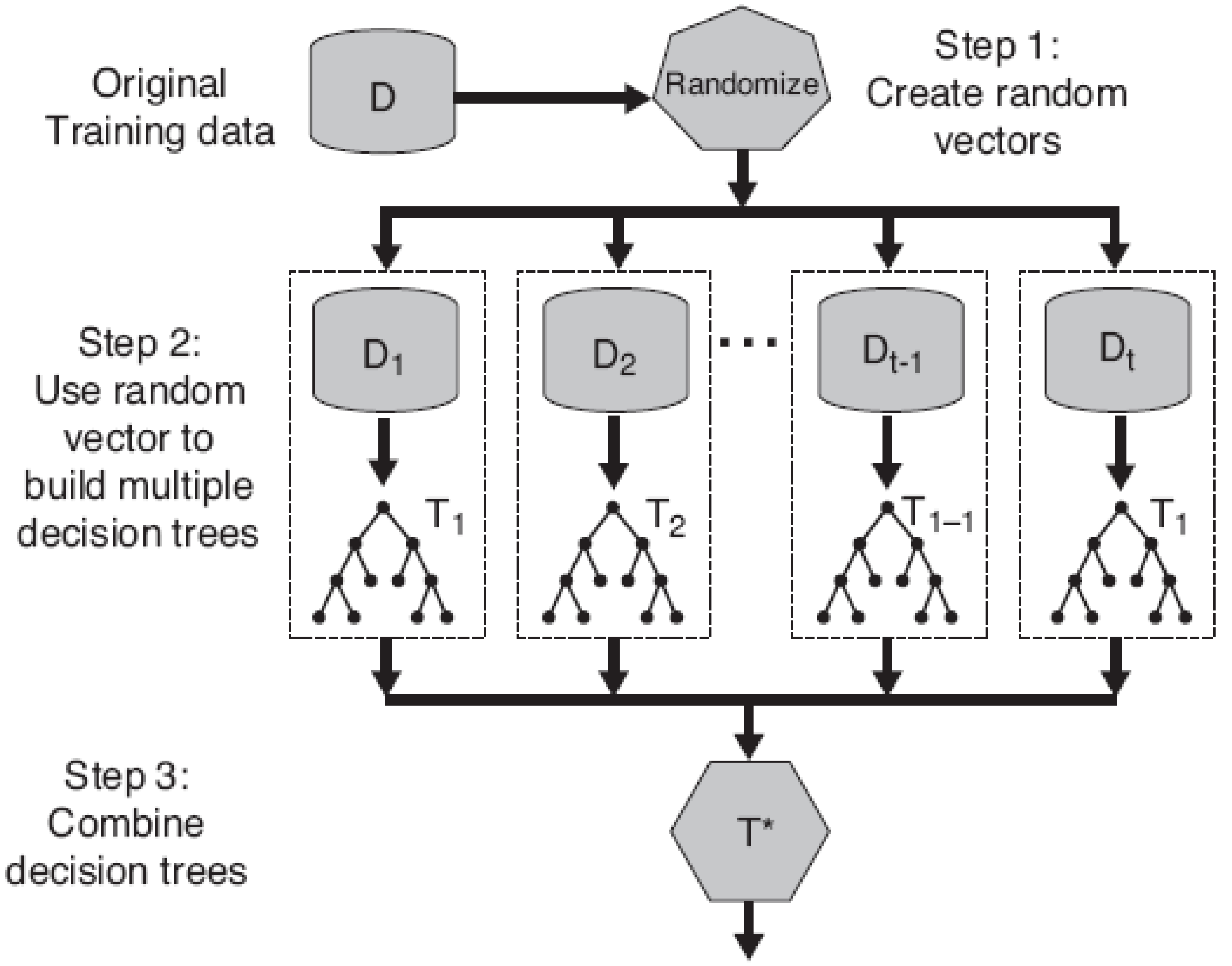
Random Forests

- Ensemble method specifically designed for decision tree classifiers
- Random Forests grows many classification trees (that is why the name!)
- Ensemble of unpruned decision trees
- Each base classifier classifies a “new” vector
- Forest chooses the classification having the most votes (over all the trees in the forest)

Random Forests

- Introduce two sources of randomness: “Bagging” and “Random input vectors”
  - Each tree is grown using a bootstrap sample of training data
  - At each node, best split is chosen from random sample of m_{try} variables instead of all variables

Random Forests



Adaboost - Adaptive Boosting

- Instead of sampling, re-weight
  - Previous weak learner has only 50% accuracy over new distribution
- Can be used to learn weak classifiers
- Final classification based on weighted vote of weak classifiers

  - Records that are wrongly (correctly) classified will have their weights increased (decreased)

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

Adaboost

- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:
- The weight of classifier M_i's vote is

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

$$\log \frac{1 - error(M_i)}{error(M_i)}$$

Some Comments on Ensemble Learning

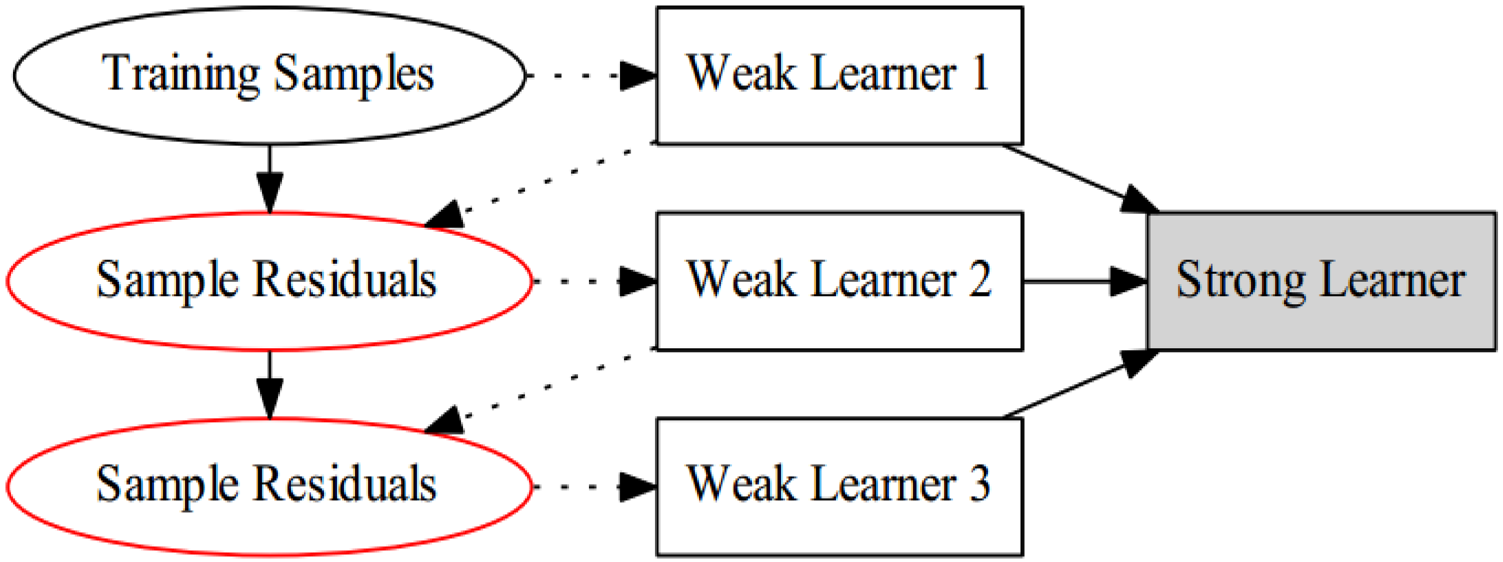
- Strength & Correlation
  - Out-of-bag (OOB) error
- High dimensional data
- Parallelized Algorithms

6.8 Gradient Boosting Methods

Reference

J. Friedman(1999). Greedy Function Approximation: A Gradient Boosting Machine.

Gradient Boosting



Guarantee: sum of residuals is monotonously decreasing. Residuals are highly related to loss.

Gradient Boosting

Notations

A training set $\mathcal{D} = \{(x_i, y_i)\}_1^N$. A loss function L. The model F.

F is an additive model

$$F(x; w) = \sum_{k=0}^{K} \alpha_k h_k(x; w_k) = \sum_{k=0}^{K} f_k(x; w_k) \tag{3}$$

Define:

$$F_k = \sum_{i=0}^{k} f_i \tag{4}$$

$\{h_k(x; w_k)\}_1^K$, $\{\alpha_k\}_1^K$, $\{w_k\}_1^K$: weak learners and their weights, parameters.

Gradient Boosting

Goal

Overall Loss Function

$$\mathcal{L} = \underbrace{\sum_{i=1}^{N} L(y_i, F(x_i; w))}_{Training\ \ loss} + \underbrace{\sum_{k=1}^{K} \Omega(f_k)}_{Regularization} \tag{5}$$

Goal

$$F^* = \arg\min_{F} \mathcal{L} \tag{6}$$

This is a NP hard problem

Gradient Boosting

Learn Greedily

Iteration 0

Choose a f0, usually a constant

Iteration k

$$f_k = \arg\min_{f_k} \mathcal{L}(f_k) \tag{7}$$

$$= \arg\min_{f_k} \sum_{i=1}^{N} L(y_i, F_{k-1}(x_i; w) + f_k(x_i)) + \Omega(f_k) \tag{8}$$

Finally

$$F^* = \sum_{k=1}^{K} f_k$$

Boosting Tree

输入：训练数据集 $T=\{(x_1,y_2),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i \in X \subseteq R^n, y_i \in Y \subseteq R$;

输出：提升树：$f_M(x)$。

（1）初始化$f_0(x)=0$。

（2）对$m=1,2,\cdots$，M。

(a) 计算残差：

$$r_{mi}=y_i-f_m-1(x_i),i-1,2,\cdots,N$$

(b)拟合残差：r_{mi}学习一个回归树，得到$T(x;\Theta_m)$。

(c)拟合残差：更新$f_m(x)=f_{m-1}(x)+T(x;\Theta_m)$。

（3）得到回归问题提升树

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

Gradient Boosting Machine

$\Omega(f_k) = 0$ in [Friedman(1999)].

1 choose an initial f_0, let $F_0 = f_0$

2 for $k = 1, 2, \ldots, K$

  2.1 $\tilde{y}_i = -\frac{\partial L(y_i, F_{k-1}(x_i))}{\partial F_{k-1}(x_i)}$, $i = 1, 2, \ldots, N$
  $\{\tilde{y}_i\}_1^N$: pseudo responses, a measurement of residuals, namely $\{y_i - F_{k-1}(x_i)\}_1^N$

  2.2 $w^* = \arg\min_w \sum_{i=1}^N [\tilde{y}_i - h_k(x_i; w)]^2$
  Train h_k to fit $\{(x_i, \tilde{y}_i)\}_1^N$ using square error loss

  2.3 $\rho^* = \arg\min_\rho \sum_{i=1}^N L(y_i, F_{k-1}(x_i) + \rho h_k(x_i; w^*))$
  Perform a line search, so that $F_{k-1} + \rho^* h_k$ reduces L most

  2.4 let $f_k = \rho^* h_k(x; w^*)$, $F_k = F_{k-1} + f_k$
  Update F_k

3 output F_K

GBDT

输入：训练数据集 $T=\{(x_1,y_2),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i\in X\subseteq R^n$，$y_i\in Y\subseteq R$；损失函数$L(y,f(x))$;

输出：回归树$\hat{f}(x)$。

（1）初始化

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

（2）对 $m=1,2,\cdots$，M

(a)对 $i=1,2,\cdots$，N计算

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

(b)对r_{mi}拟合一个回归树，得到第m棵树的叶结点区域$R_{mj},j=1,2,\cdots,J$。

(c)对$j=1,2,\cdots,J$，计算

$$c_{mj}=agr\min_c\sum_{x_i\in R_{mj}}L(y_i,f_{m-1}(x)+c)$$

(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

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

6.9 XGBoost

Tianqi Chen, Carlos Guestrin(2016). **XGBoost: A Scalable Tree Boosting System**. KDD'16.

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 Trainging(Boosting)
  - Start from constant prediction, add a new function each time

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

New function

Model at training round t

Keep functions added in previous round

Additive Training

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

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

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

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

Our New Goal

- Objective, with constants removed

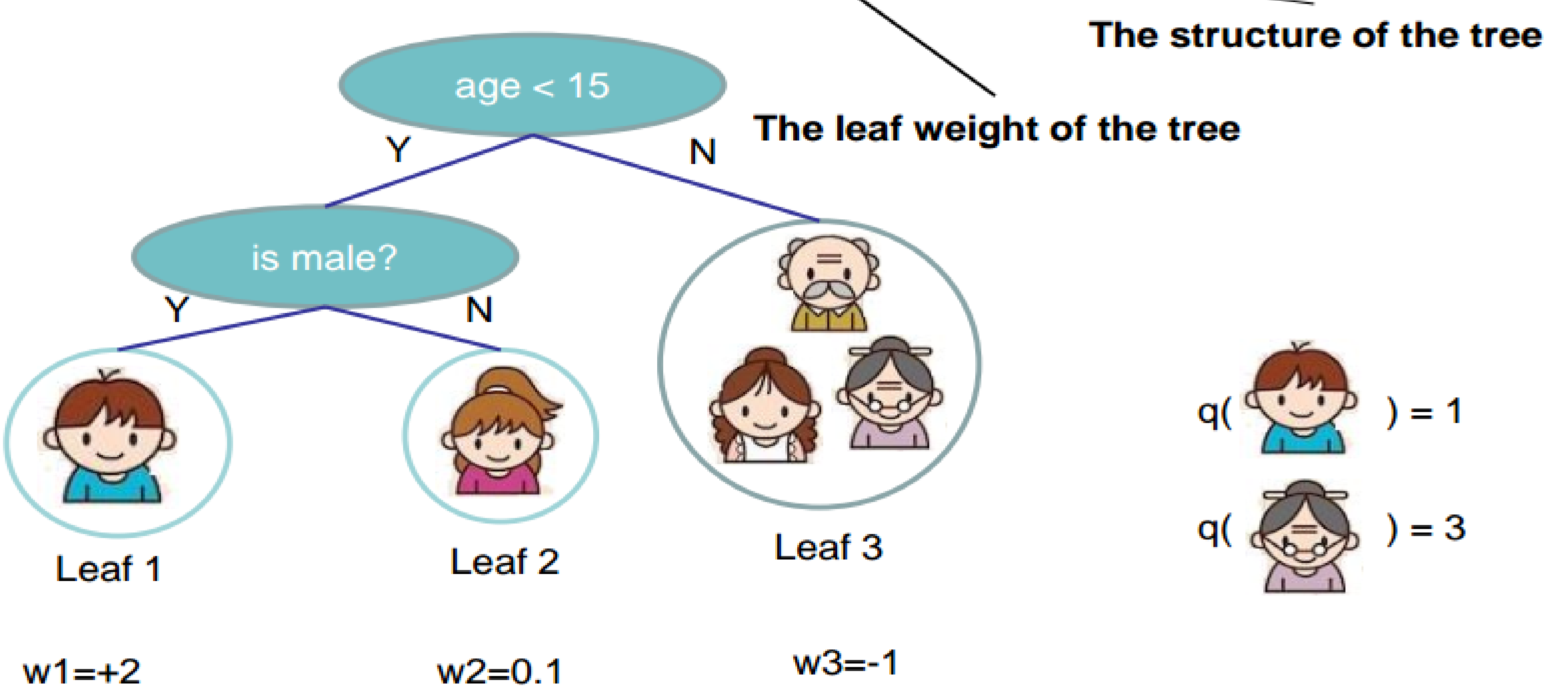
$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

  - 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 much efforts to derive the objective, why not just grow trees ...
  - **Theoretical benefit**:know what we are learning, convergence
  - **Engineering benefit**,recall elements of supervised learning
    - and g_i comes from h_i definition of loss function
    - the learning of function only depend on the objective via and
    - think of how you can separate g_i modules h_i of you code when you are asked to implement boosted tree for square loss and logistic loss

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

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



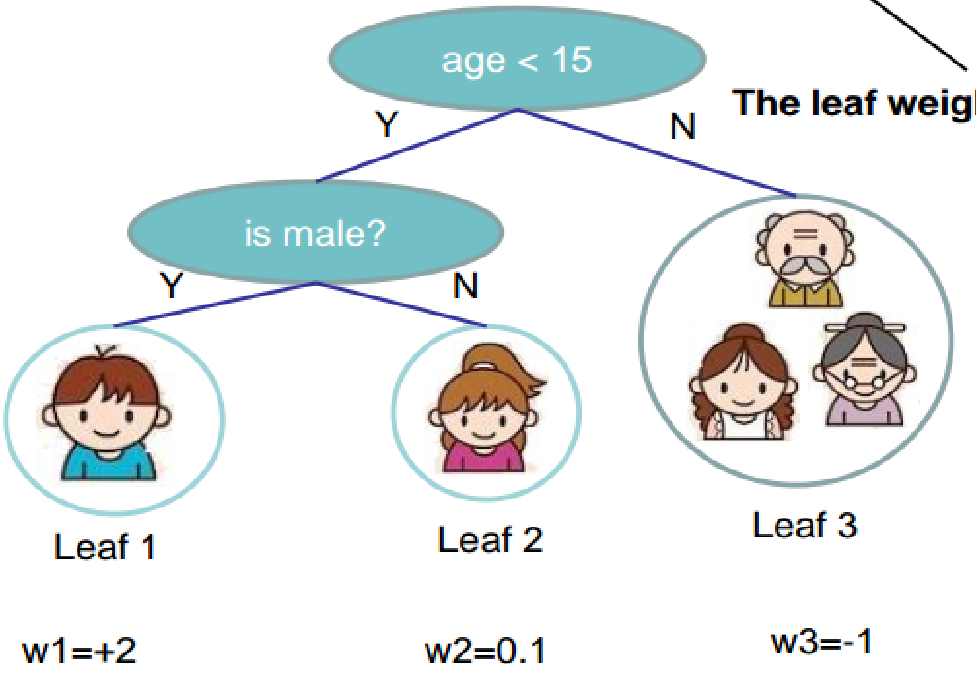
Define Complexity of a Tree

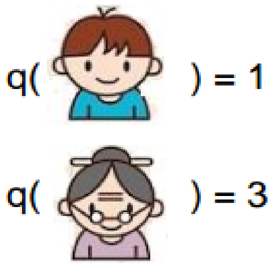
- Define complexity as sum of leaf weights (this is not the only possible definition)

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

The structure of the tree

The leaf weight of the tree





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 $\quad G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

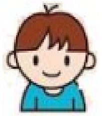
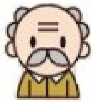
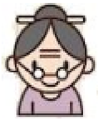
$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

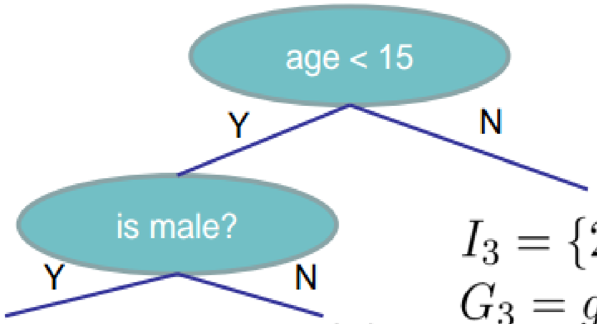
- Assume the structure of tree (q(x)) is fixed, the optimal weight in each leaf, and the resulting objective value are

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

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



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

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

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

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

The smaller the score is, the better the structure is

Searching Algorithm for Single Tree

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

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

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

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

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

Greedy Learning of the Tree

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

The complexity cost by introducing additional leaf

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

the score of left child

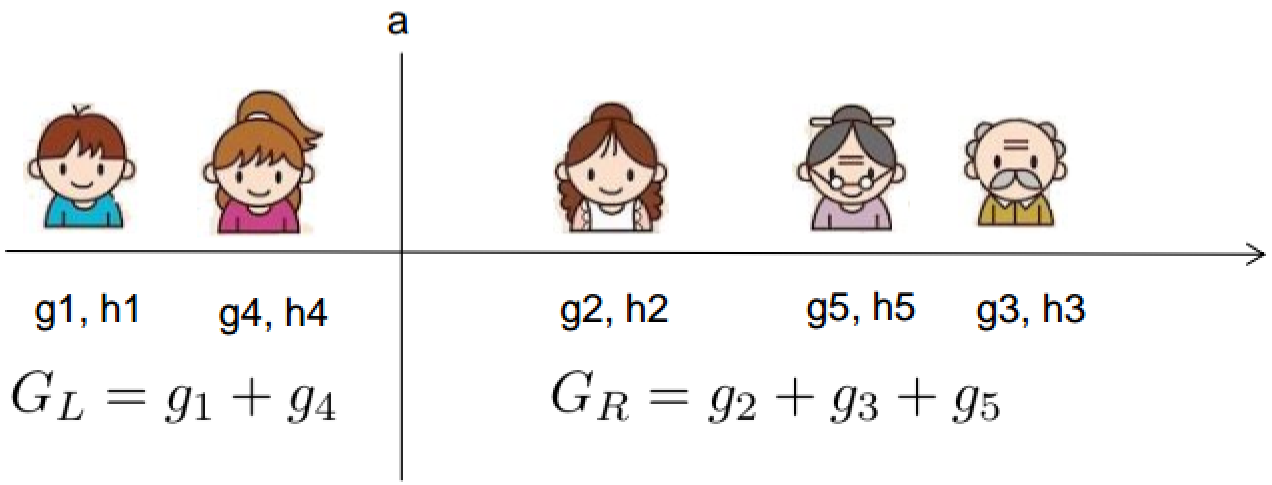
the score of if we do not split

the score of right child

  - 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 future.

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 L
  - It is O(n d k logn):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.

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 may a split can benefit futures 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 $f_t(x)$

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

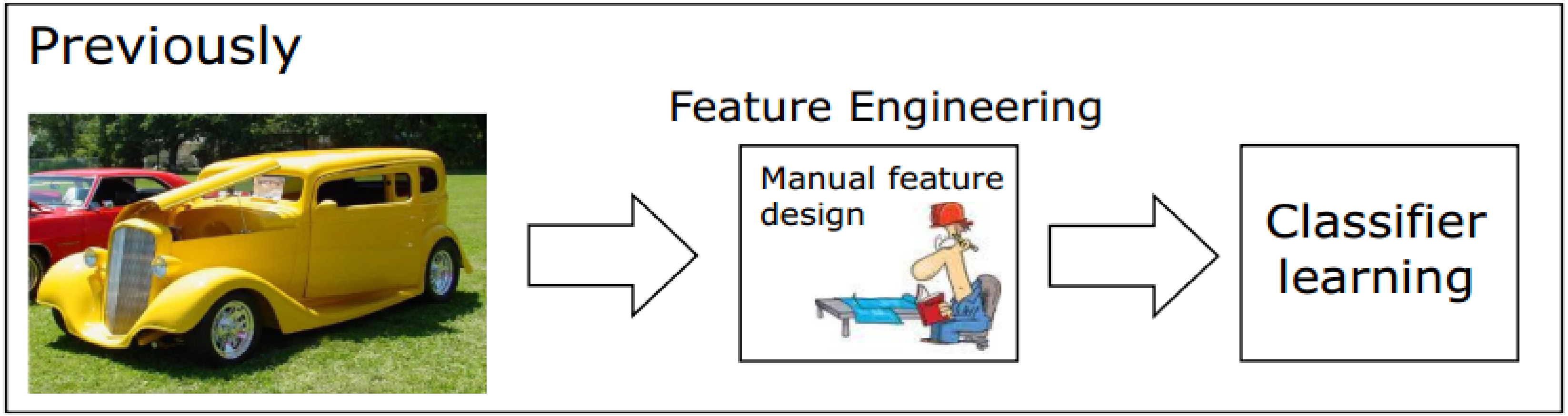
- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This is means we do not do full optimization in each step and reserve chance for future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

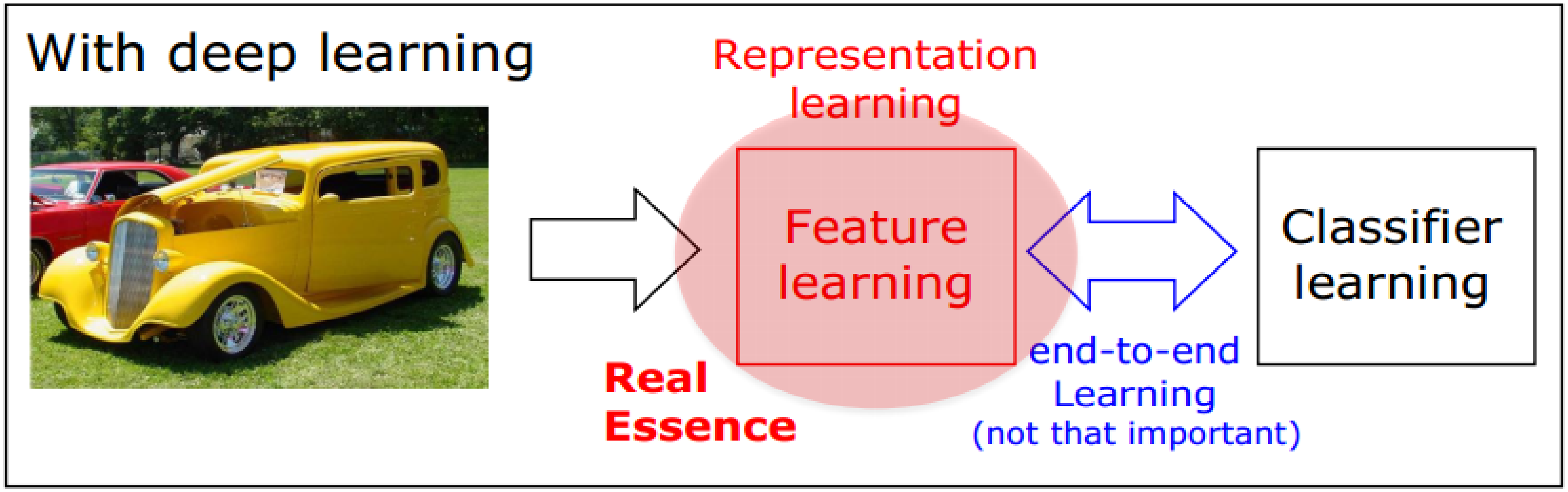
6.10 Deep Forest

Zhi-Hua Zhou, Ji Feng(2017). **Deep Forest: Towards An Alternative to Deep Neural Networks**. IJCAI 2017.

gcForest

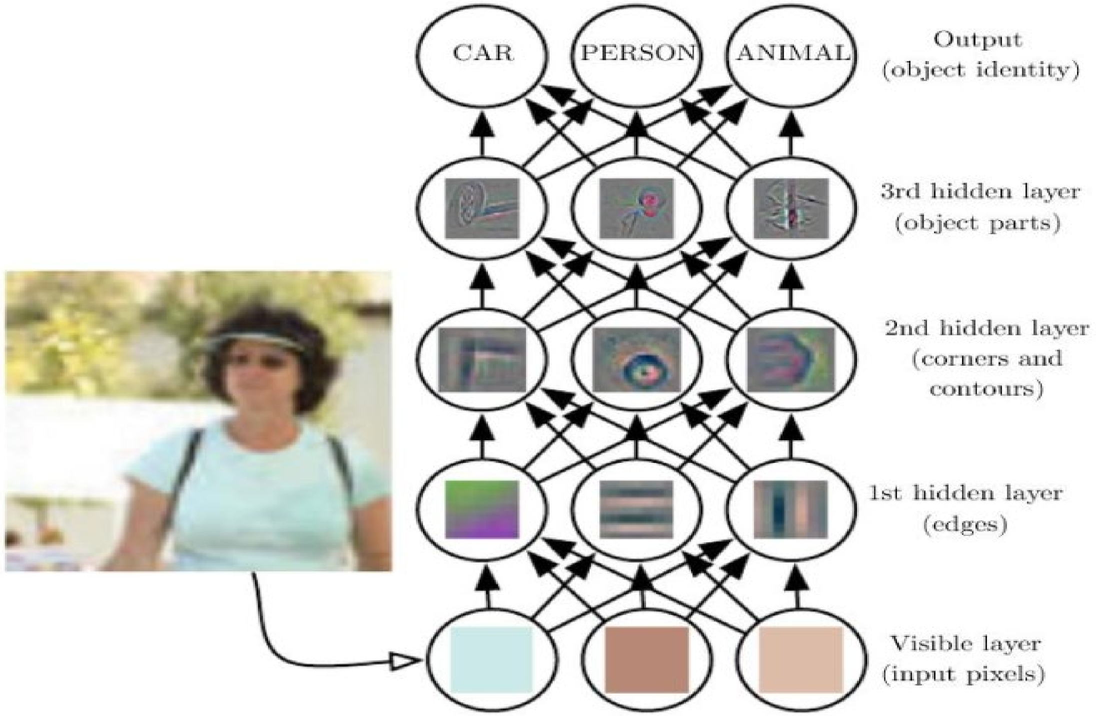
What's essential with DNNs? --Representation learning





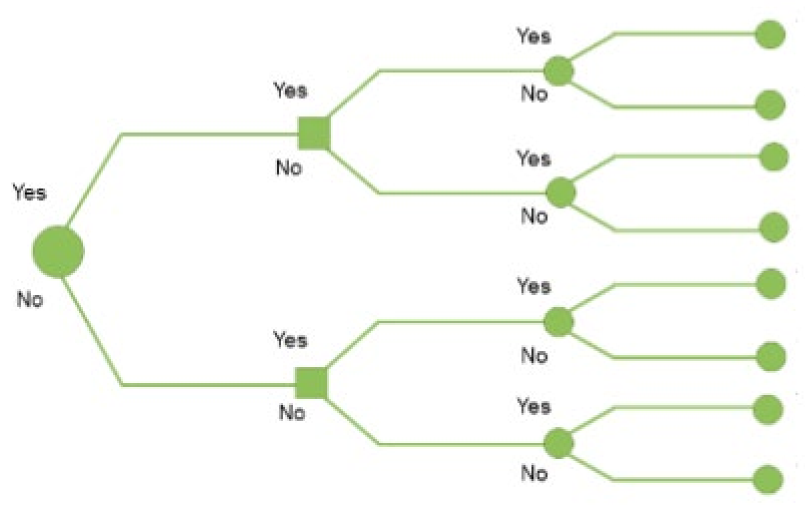
gcForest

Layer-by-Layer processing is crucial

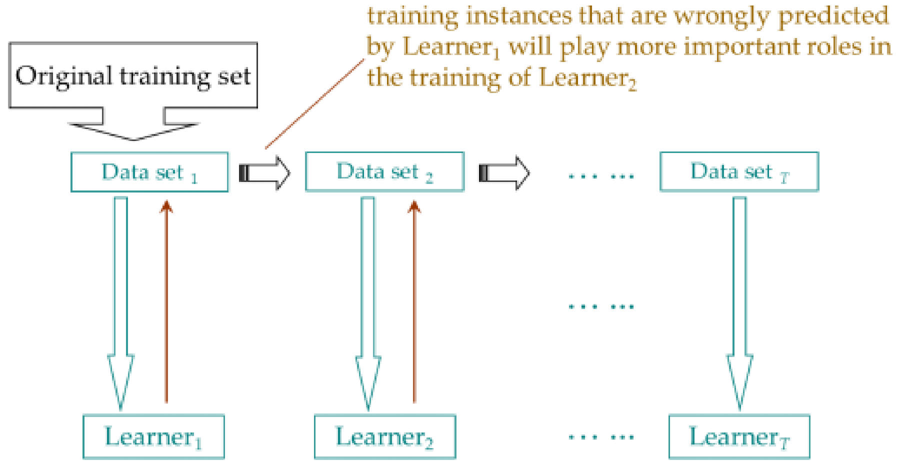


gcForest

Decision trees ?



Boosting?



Layer-by-layer processing, but ...

insufficient complexity

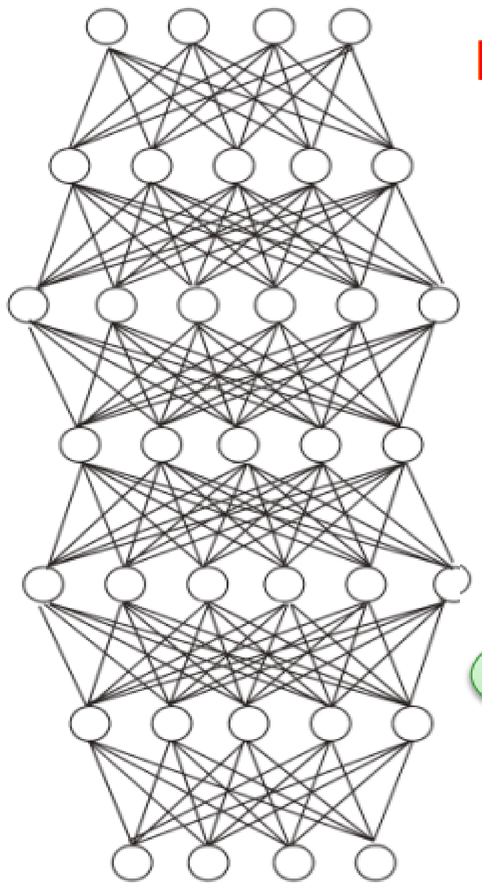
always on original features

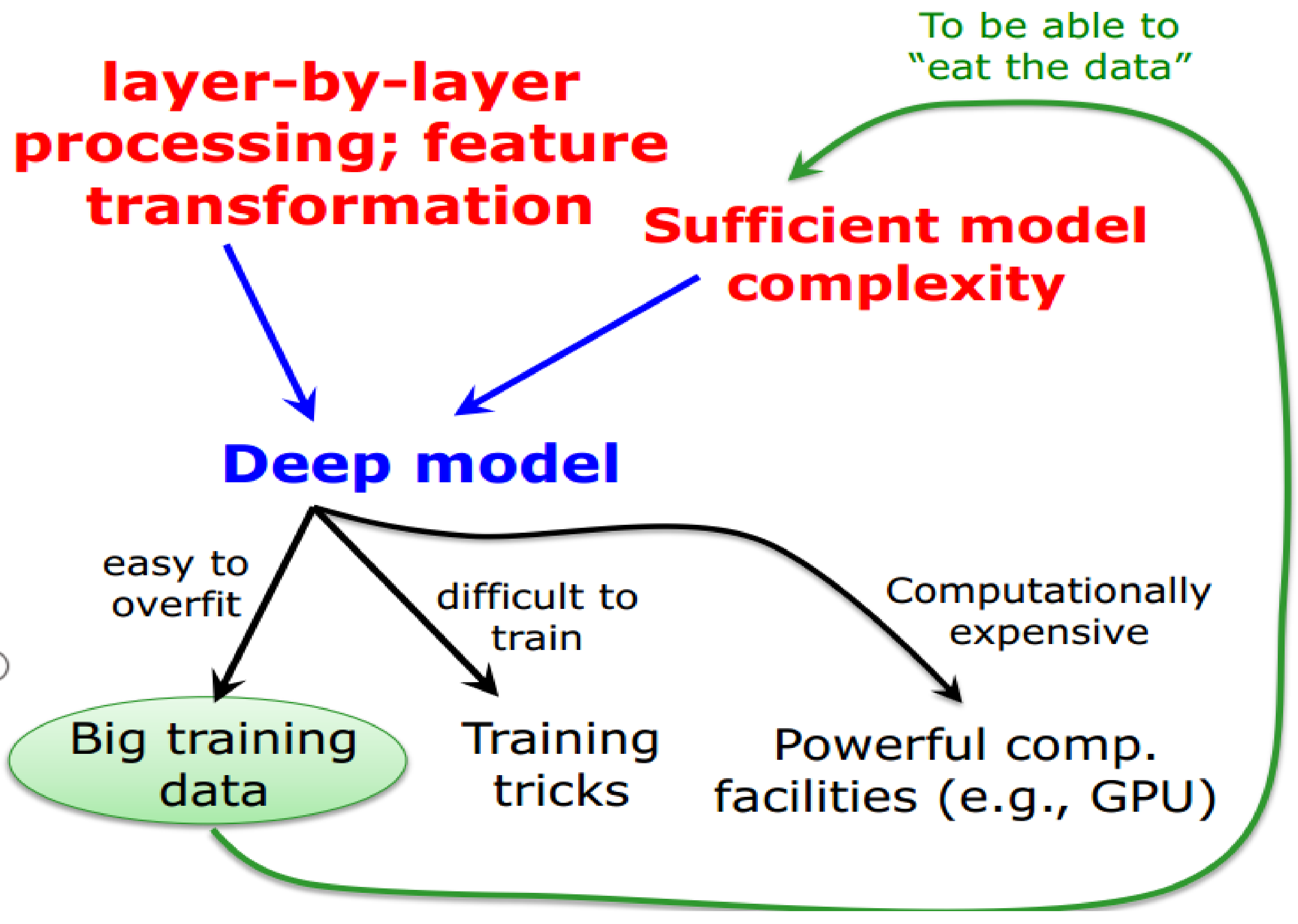
still, insufficient complexity

always on origin features

gcForest

Another view





gcForest

Most crucial for deep models:

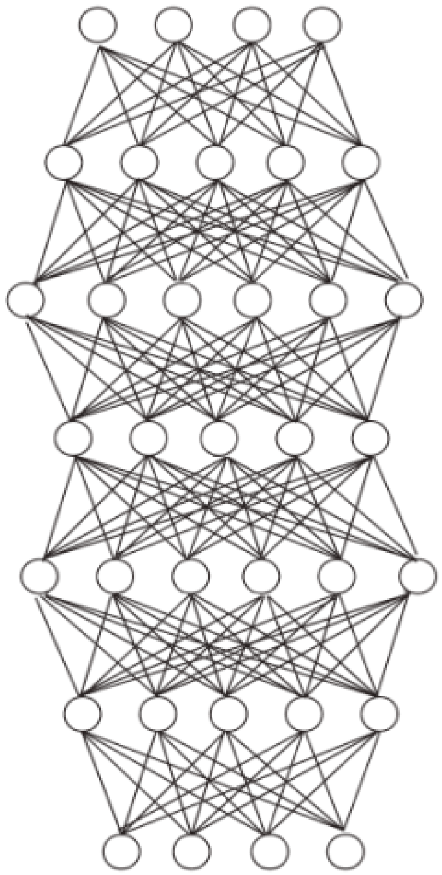
Layer-by-layer processing

Feature transformation

Sufficient model complexity

gcForest

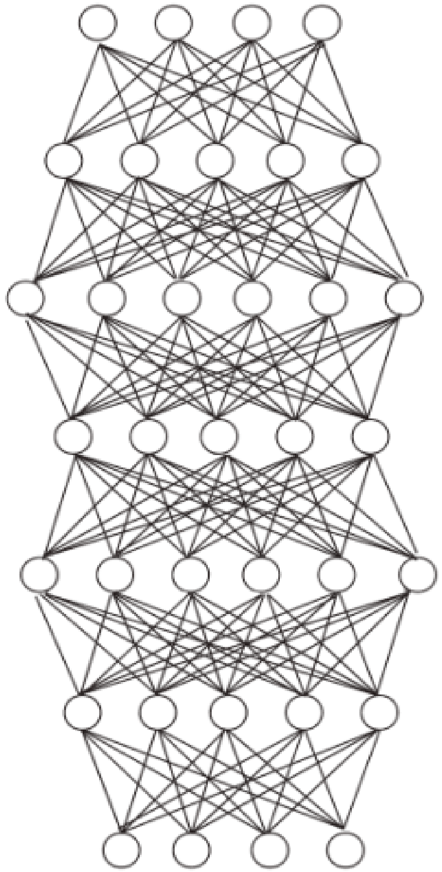
Using neural networks



- Too many hyper-parameters
  - tricky tuning, particularly when across tasks
  - Hard to repeat others' results; e.g., even when several authors all use CNNs, they are actually using different learning models due to the many different options such as convolutional layer structures
- Model complexity fixed once structure decided; usually, more than sufficient
- Big training data required
- Theoretical analysis difficult
- Blackbox

gcForest

Deep models revisited



- Currently, Deep Models are DNNs: multiple layers of parameterized differentiable nonlinear modules that can be trained by backpropagation

- Not all properties in the world are "differentiable", or best modelled as "differentiable"

- There are many non-differentiable learning modules (not able to be trained by backpropagation)

gcForest

A Grand Challenge

Can we realize deep learning with
non-differentiable modules?

This is fundamental for understanding:

- Deep models ?= DNNs
- Can do DEEP with non-differentiable modules? (without backpropagation)
- Can enable Deep model to win more tasks?
- ... ...

gcForest

To have

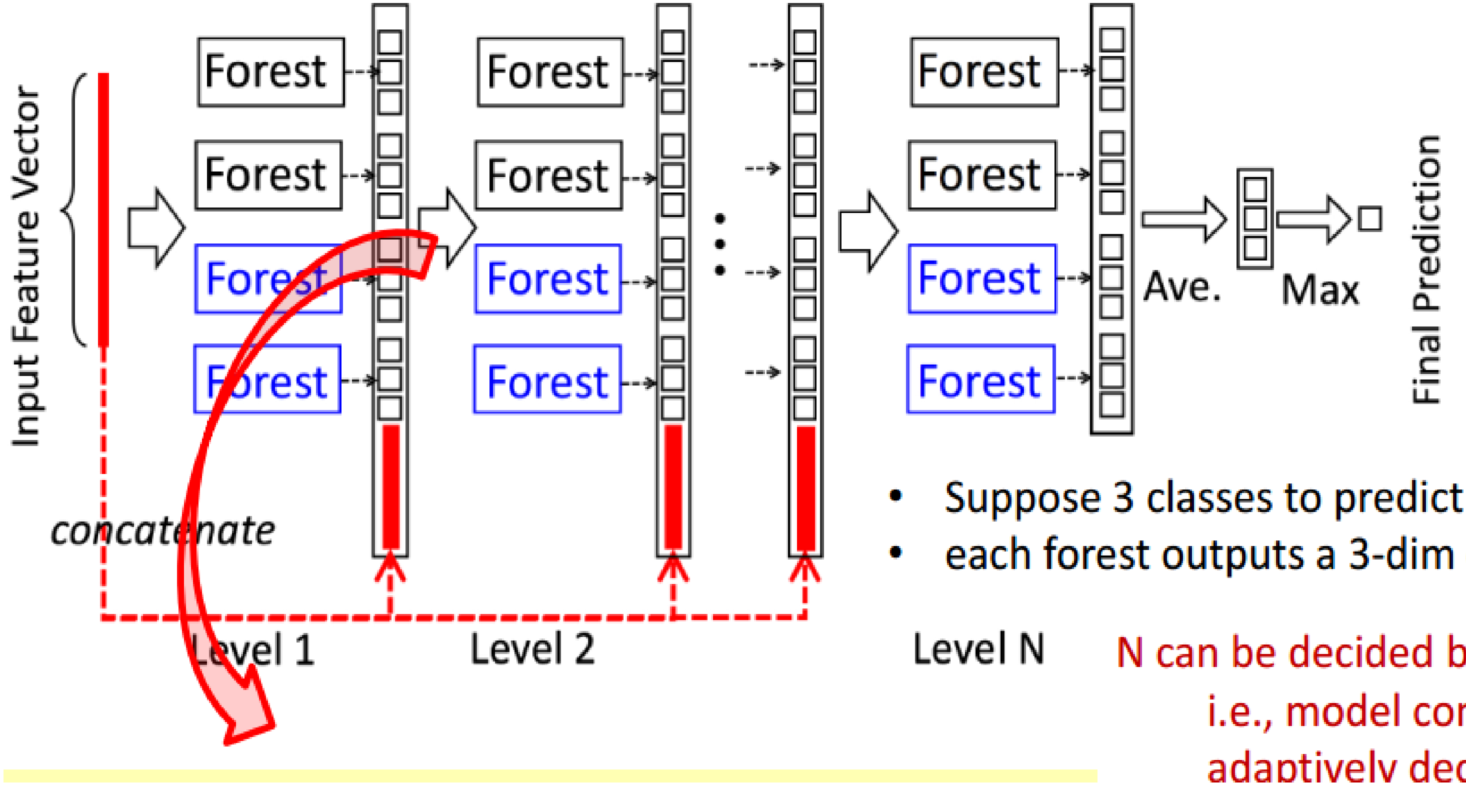
- Layer-by-layer processing, and
- Feature transformation, and
- Sufficient model complexity

The gcForest approach

gcForest(muti-Gained Cascade Forest)

- A decision tree forest(**ensemble**)approach
- **Performance highly competitive** to DNNS across a broad range of tasks
- **Much less hyper-parameters**
  - Easier to set
  - Default setting works well across a broad range of tasks
- **Adaptive model complexity**
  - Automatically decided upon data
  - Small data applicable

Cascade Forest structure



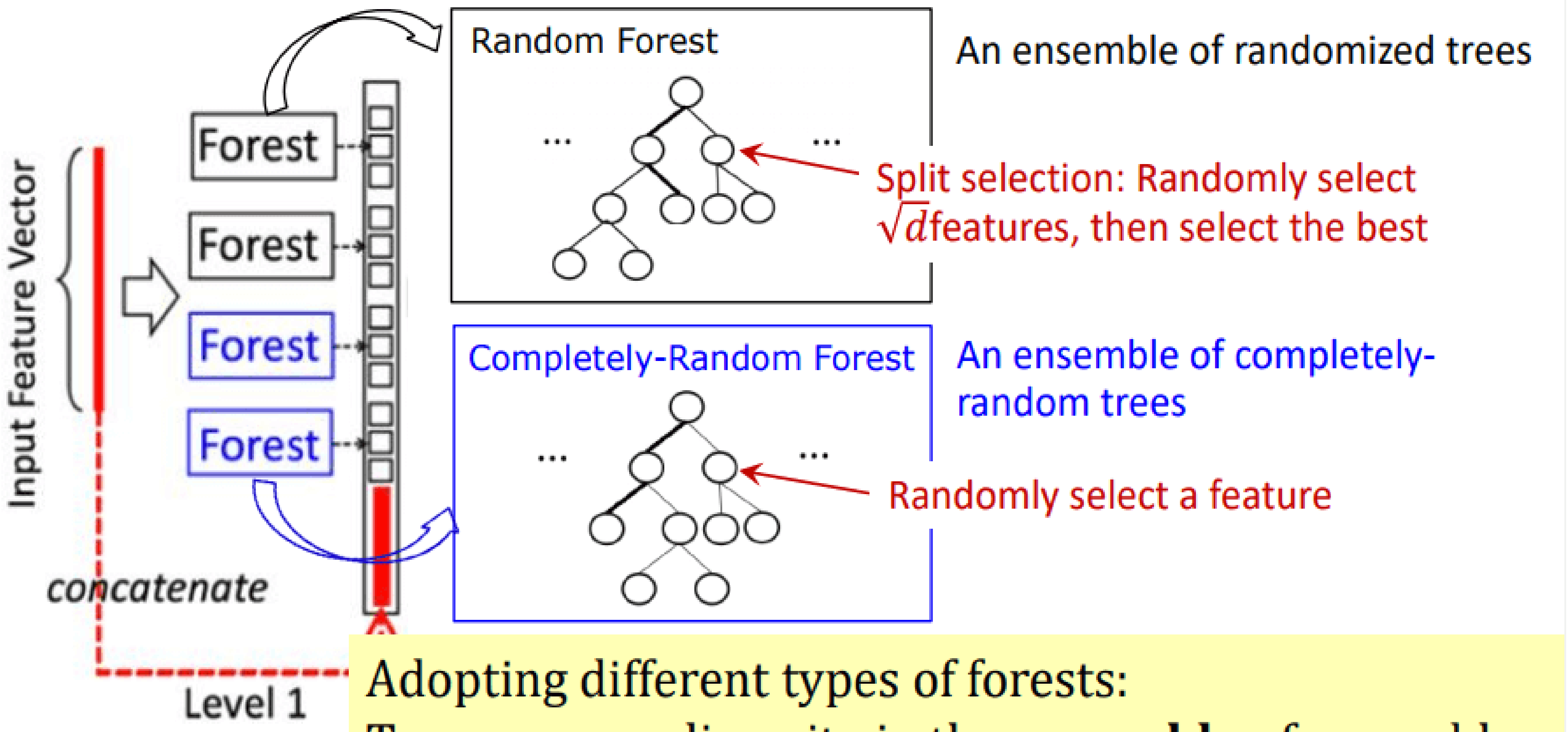
- Suppose 3 classes to predict
- each forest outputs a 3-dim **class vector**

N can be decided by CV
i.e., model complexity
adaptively decided

Passing the output of one level as input to another level:

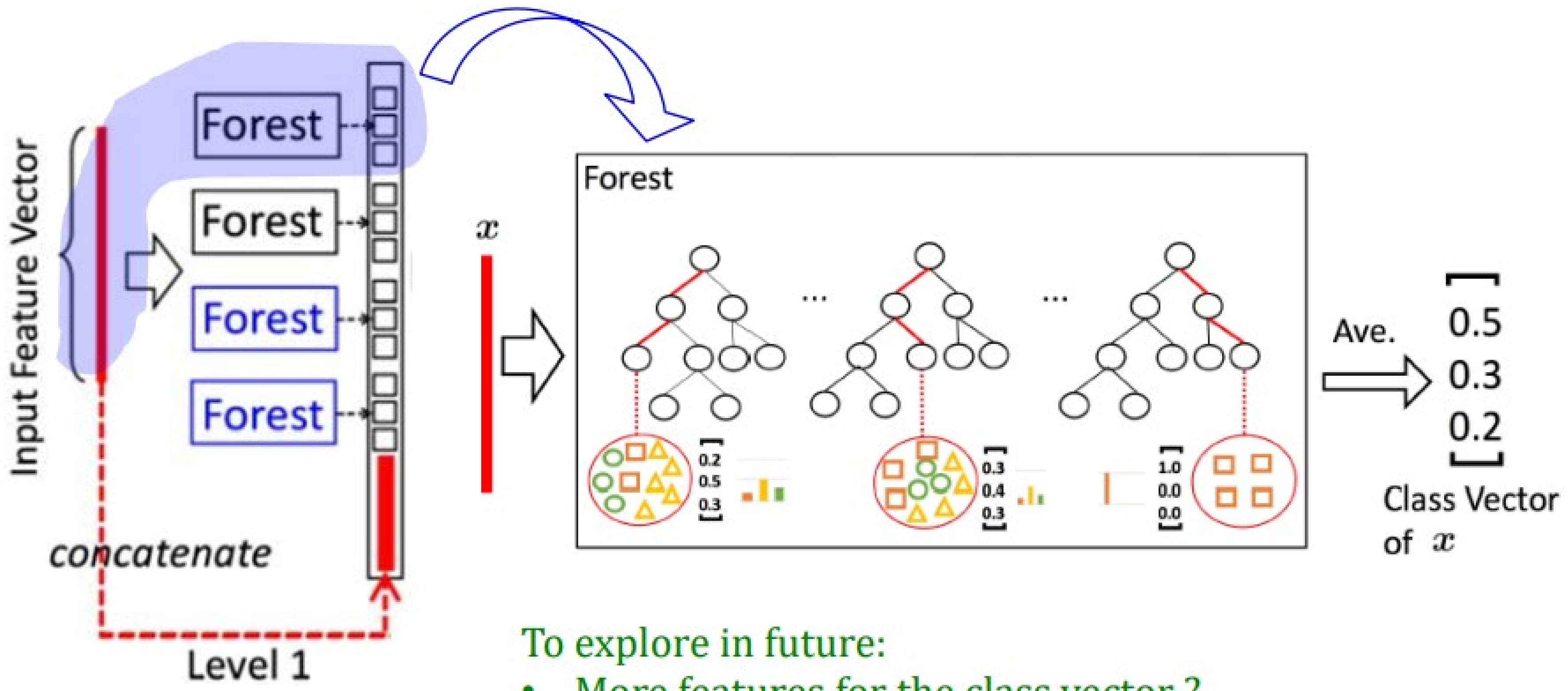
- Related to Stacking [Wolpert, NNJ 1992; Breiman, MLJ 1996], a famous ensemble method
- Stacking usually one or two levels, as it is easy to overfit with more than two levels; could not enable a deep model by itself

ensembles



Adopting different types of forests:
To encourage diversity in the **ensemble** of ensembles

Generation of class vectors



To explore in future:

- More features for the class vector ?

  ... such as parents nodes (prior distribution), sibling nodes (complement distribution), decision patch encoding, ...

Acknowledgements

- Some text, figures and formulations are from WWW. Thanks for their sharing. If you have copyright claim please contact with me at yym@hit.edu.cn.
- This lecture is distributed for nonprofit purpose.

Thank You for Your Attention

Contact me at: yym@hit.edu.cn

Tel: 26033008, 13760196623

Address: Rm.1402, H# Building