Ensemble Learning Models

Yanyan Lan

lanyanyan@ict.ac.cn

Bagging

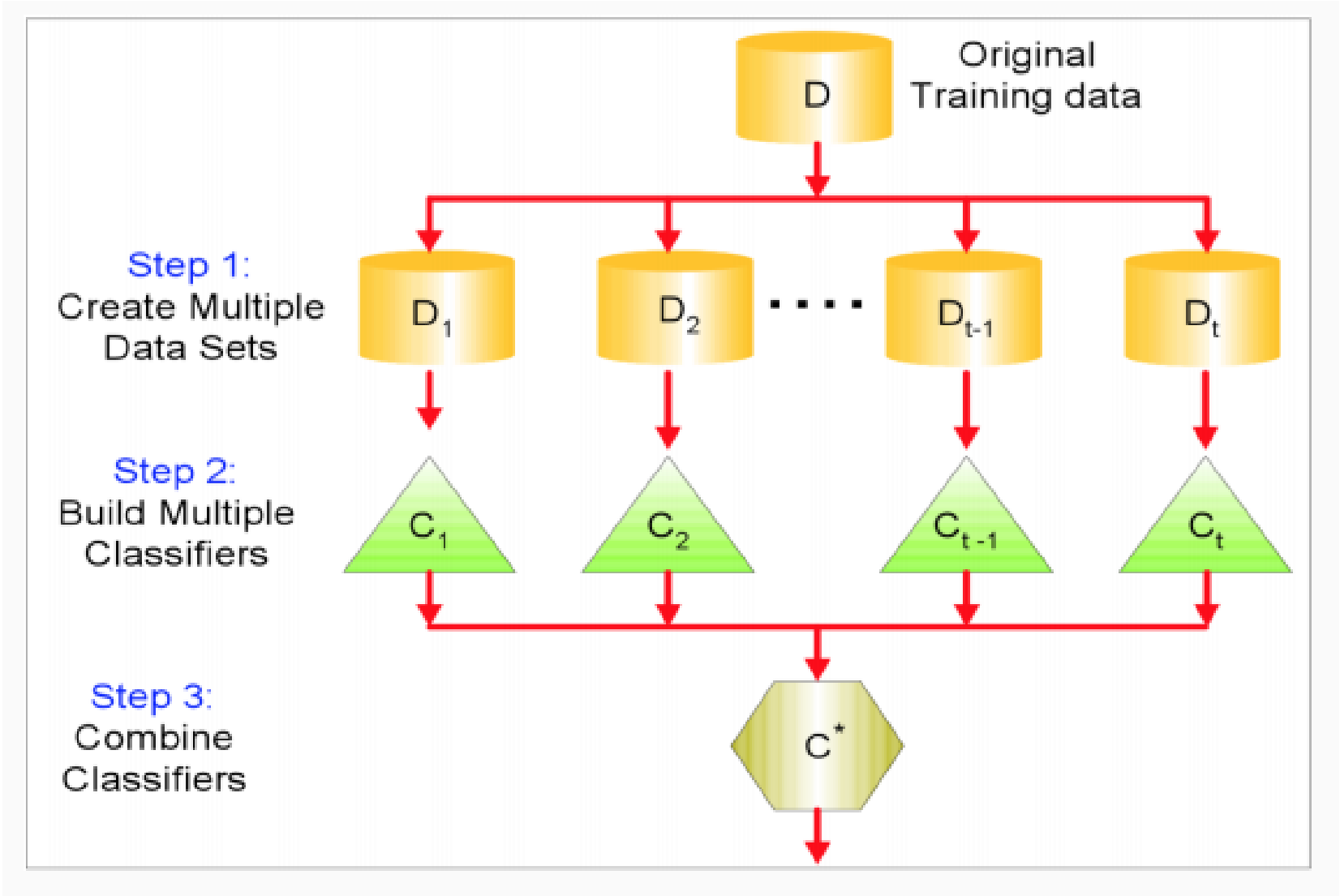
Comes from **B**ootstrap **AGG**regat**ING**

Bootstrap Sampling

Bagging

- Leo Breiman (1994)
- Idea: Take repeated bootstrap samples from training set D
- Bootstrap sampling: Given set D containing N training examples, create D' by drawing N examples at random with replacement from D
- Bagging:
  - Create k bootstrap samples $\{D_1, D_2, \ldots, D_t\}$
  - Train distinct classifier on each D_i
  - Classify new instance by majority vote / average

The Process of Bagging



Bagging

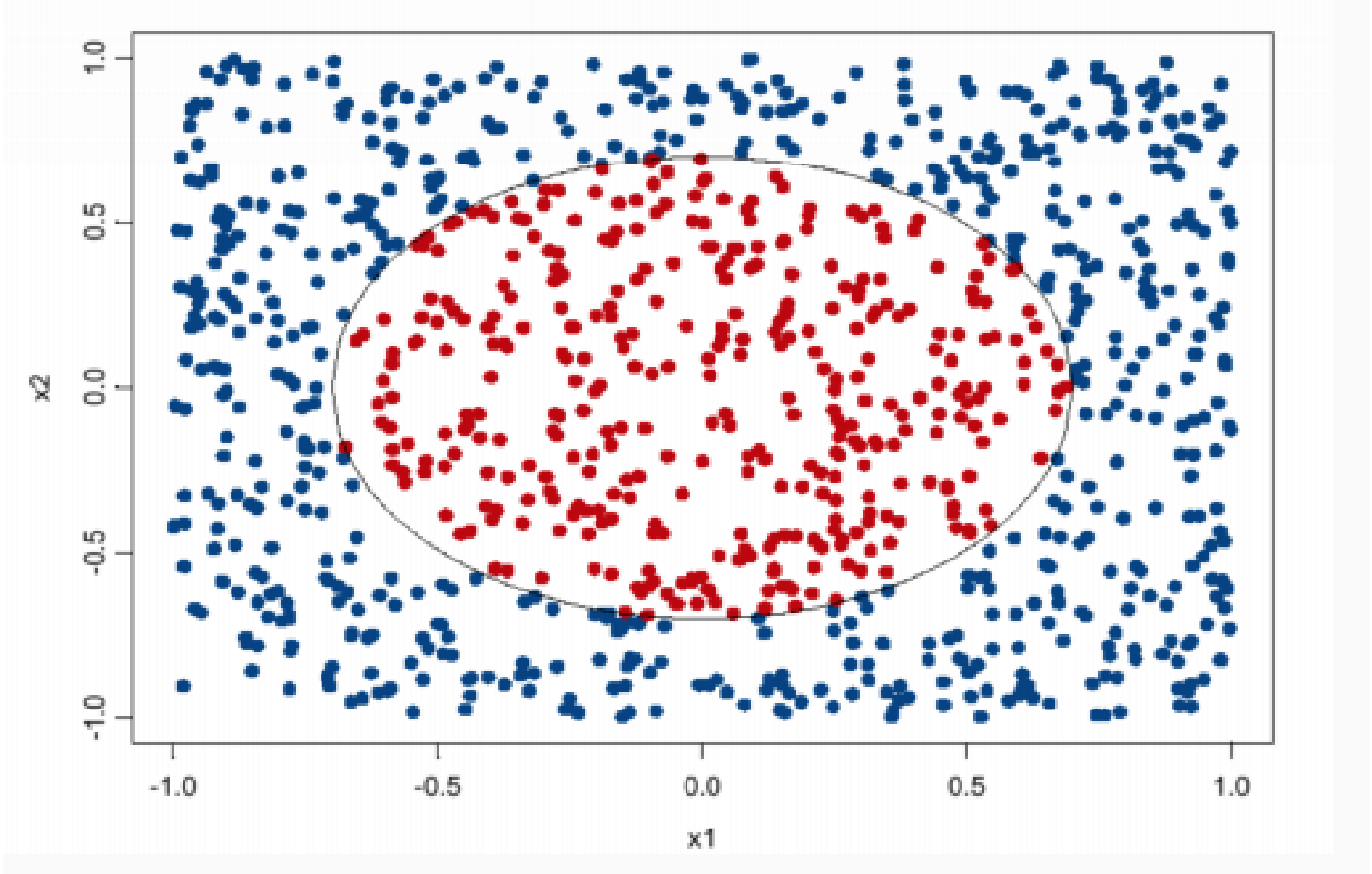
Data	1	2	3	4	5	6	7	8	9	10
BS 1	7	1	9	10	7	8	8	4	7	2
BS 2	8	1	3	1	1	9	7	4	10	1
BS 3	5	4	8	8	2	5	5	7	8	8

- Build a classifier from each bootstrap sample
- In each bootstrap sample, each data point has probability $\left(1 - \frac{1}{N}\right)^N$ of not being selected
- Expected number of data points in each samples is then

$$N\left(1 - \left(1 - \frac{1}{N}\right)^N\right) \approx 0.632N$$

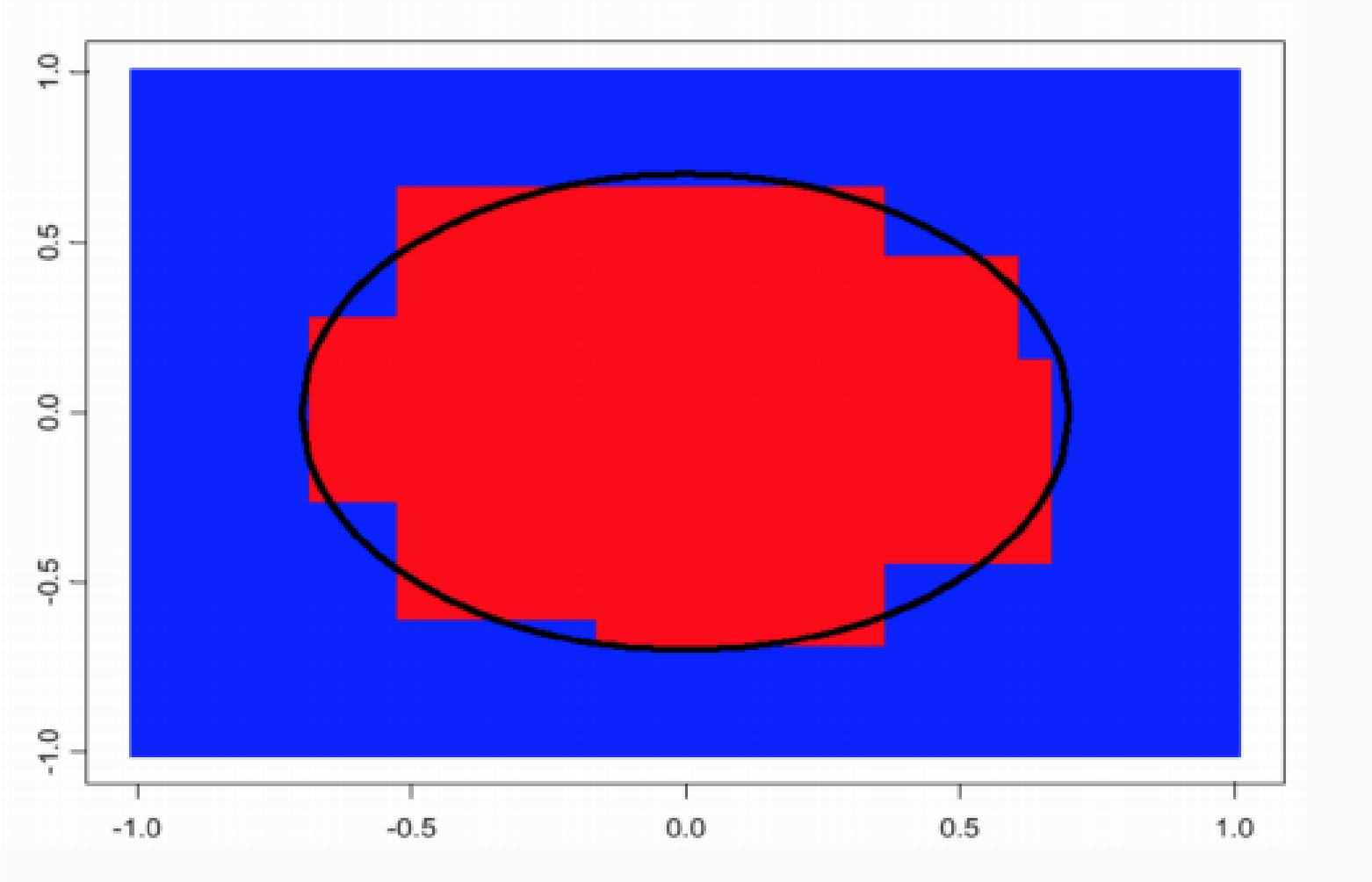
Example

Bagging Example



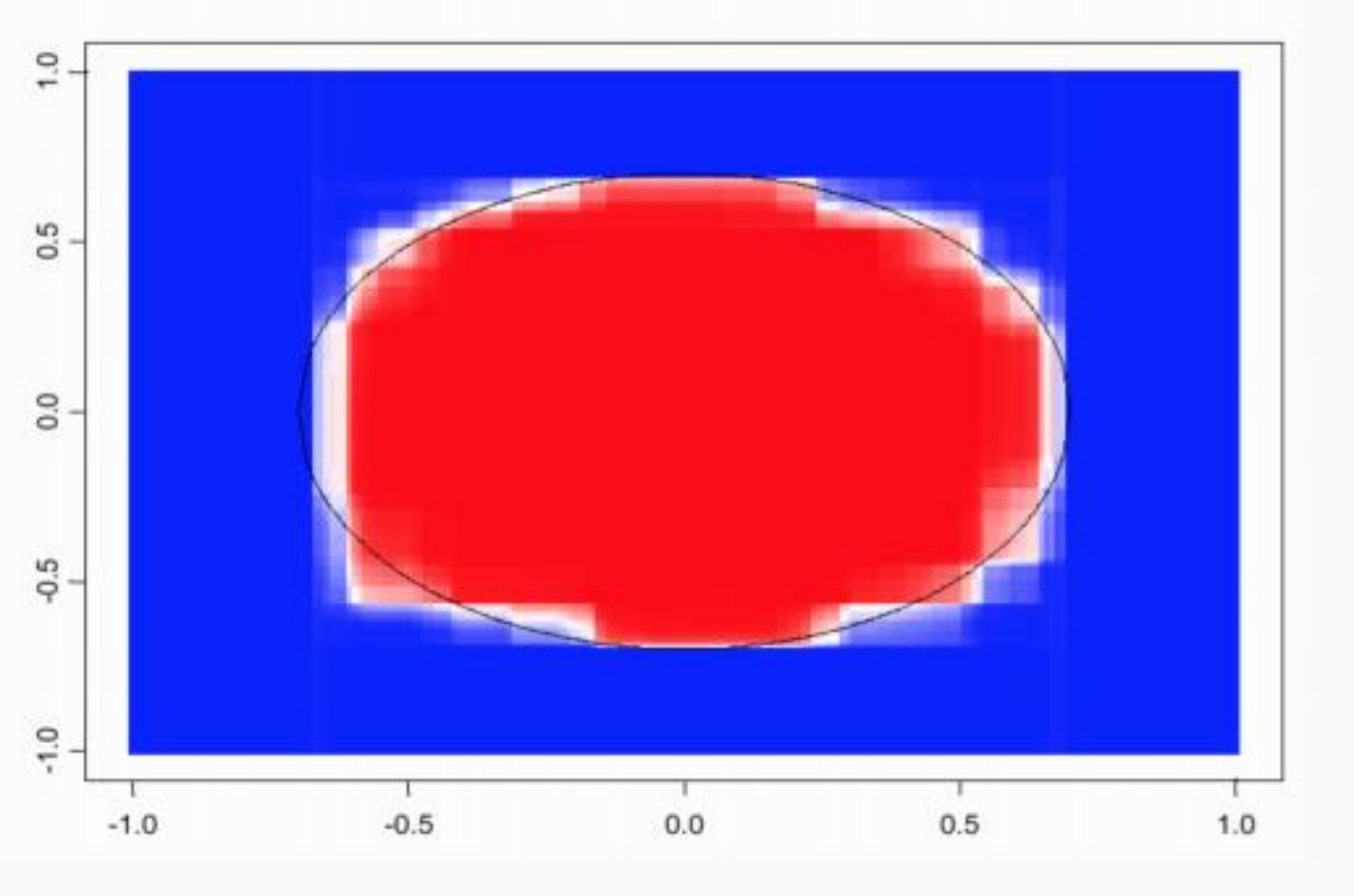
Single learner

CART decision boundary



Bagging advantage

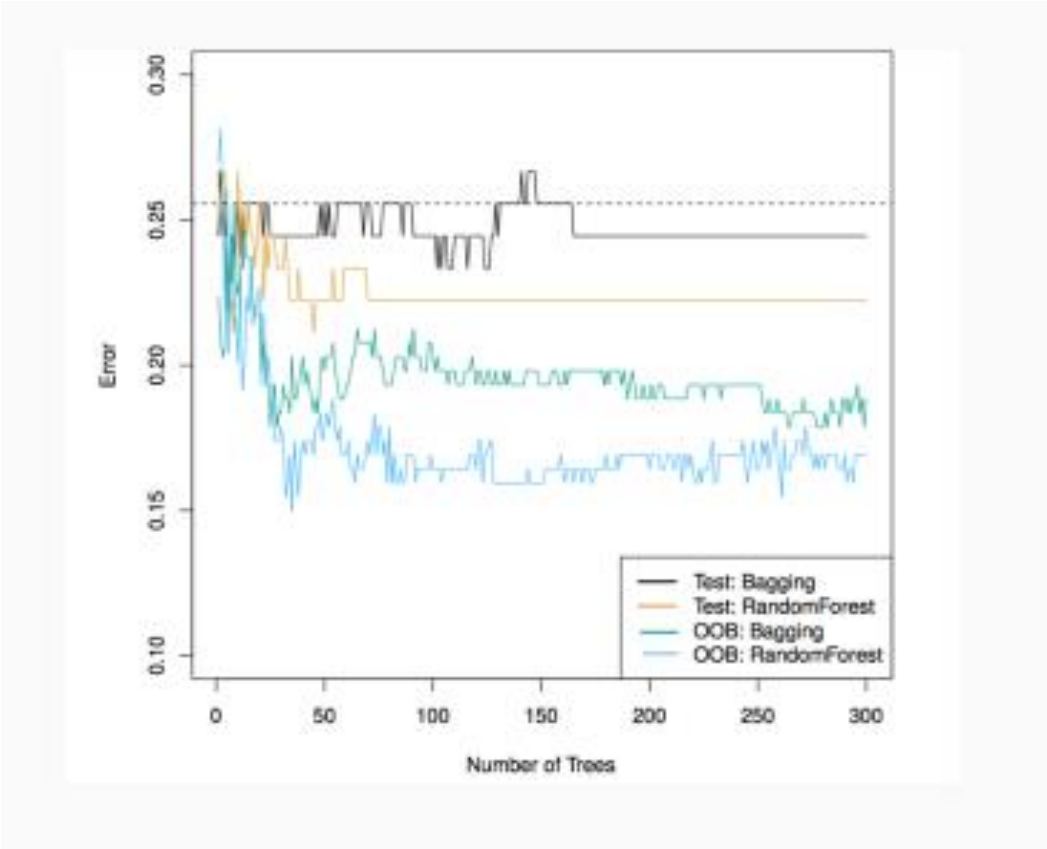
100 bagged trees



Test Error

- Out-of-bag (OOB) error
- Recall that each time we draw a Bootstrap sample, we only use 63% of the observations.
- Idea: use the rest of the observations as a test set!
- Out-of-bag (OOB) error:
  - For each sample x^i, find the prediction y_b^i for all bootstrap samples b which do not contain x^i. There should be around $0.37B$ of them.
  - Average these predictions to obtain $y^{i,oob}$.
  - Compute the error $\left(y^i - y^{i,oob}\right)^2$.
  - Average the errors over all observations $i = 1,2, \dots, n$.

Out-of-bag (OOB) error



- The test error converges as we increase number of trees B (dashed line is the error for a single classification tree).

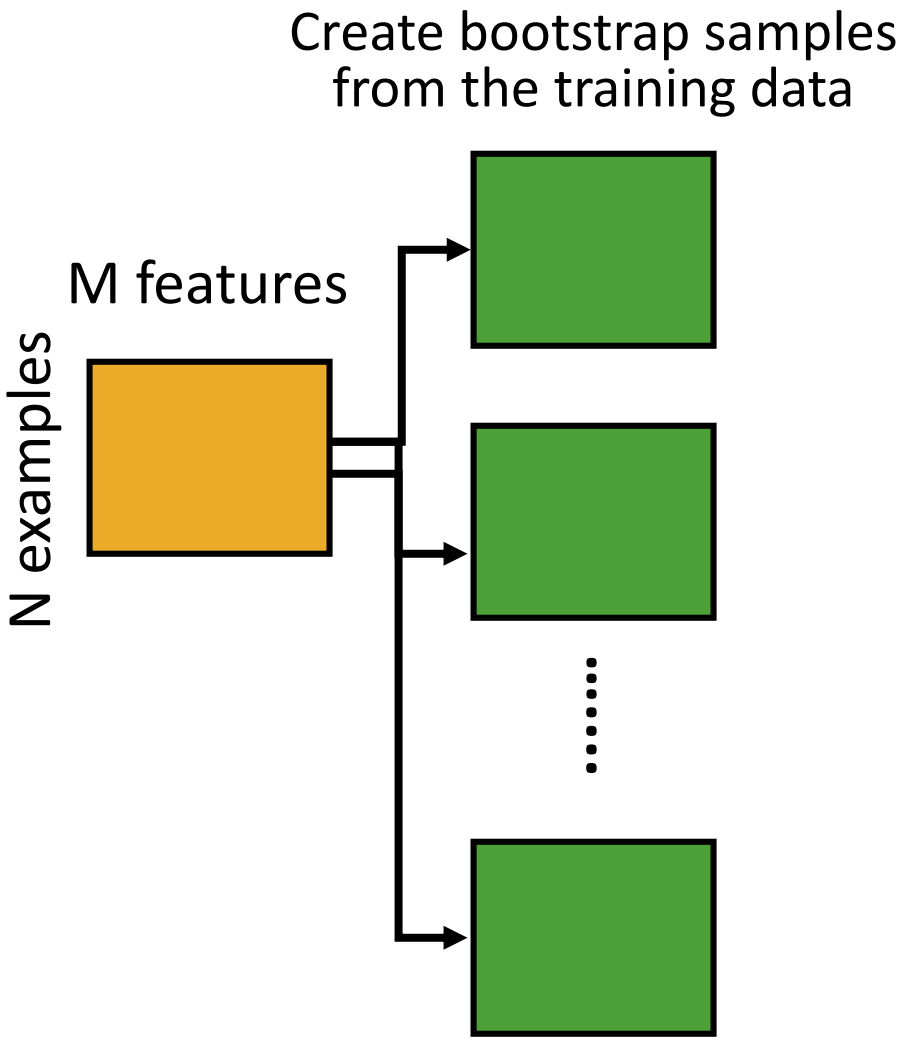
Random Forests

- Bagging has a weakness:

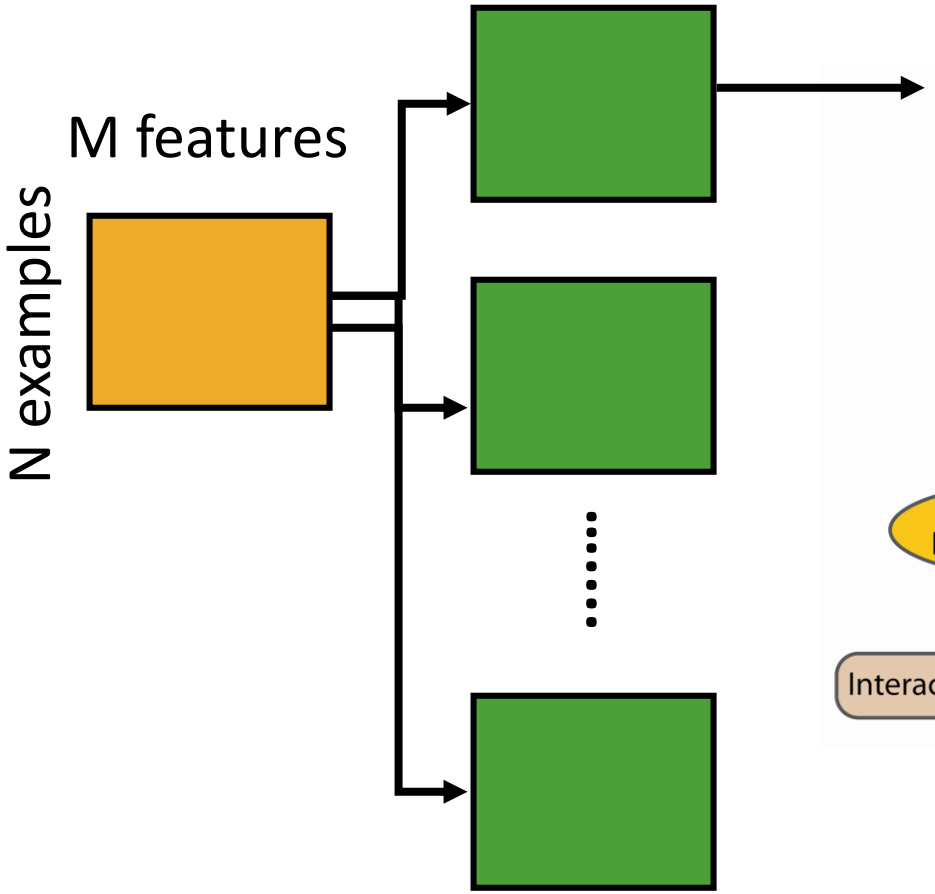
  The trees produced by different Bootstrap samples can be very similar.

- Random Forests:
  - We fit a decision tree to different Bootstrap samples.
    - When growing the tree, we select a random sample of $m < p$ features to consider at each branch of the tree.
    - This will lead to less similar trees with less correlated predictions.
  - Finally, average the prediction of each tree.

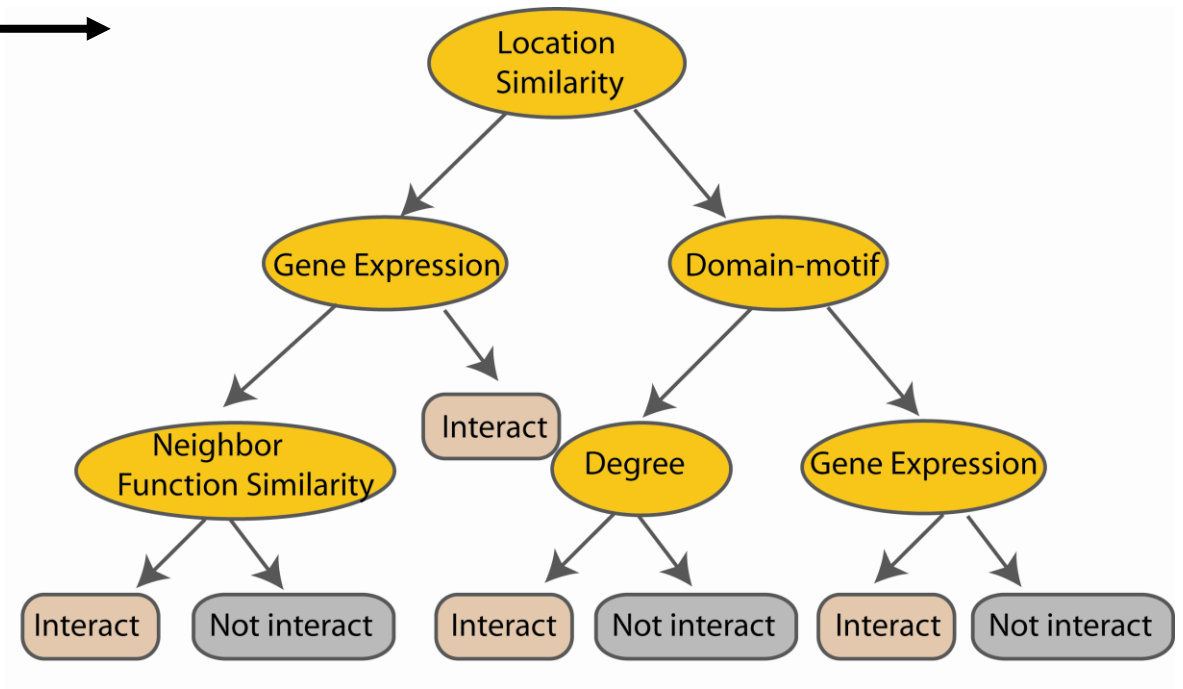
Random Forests



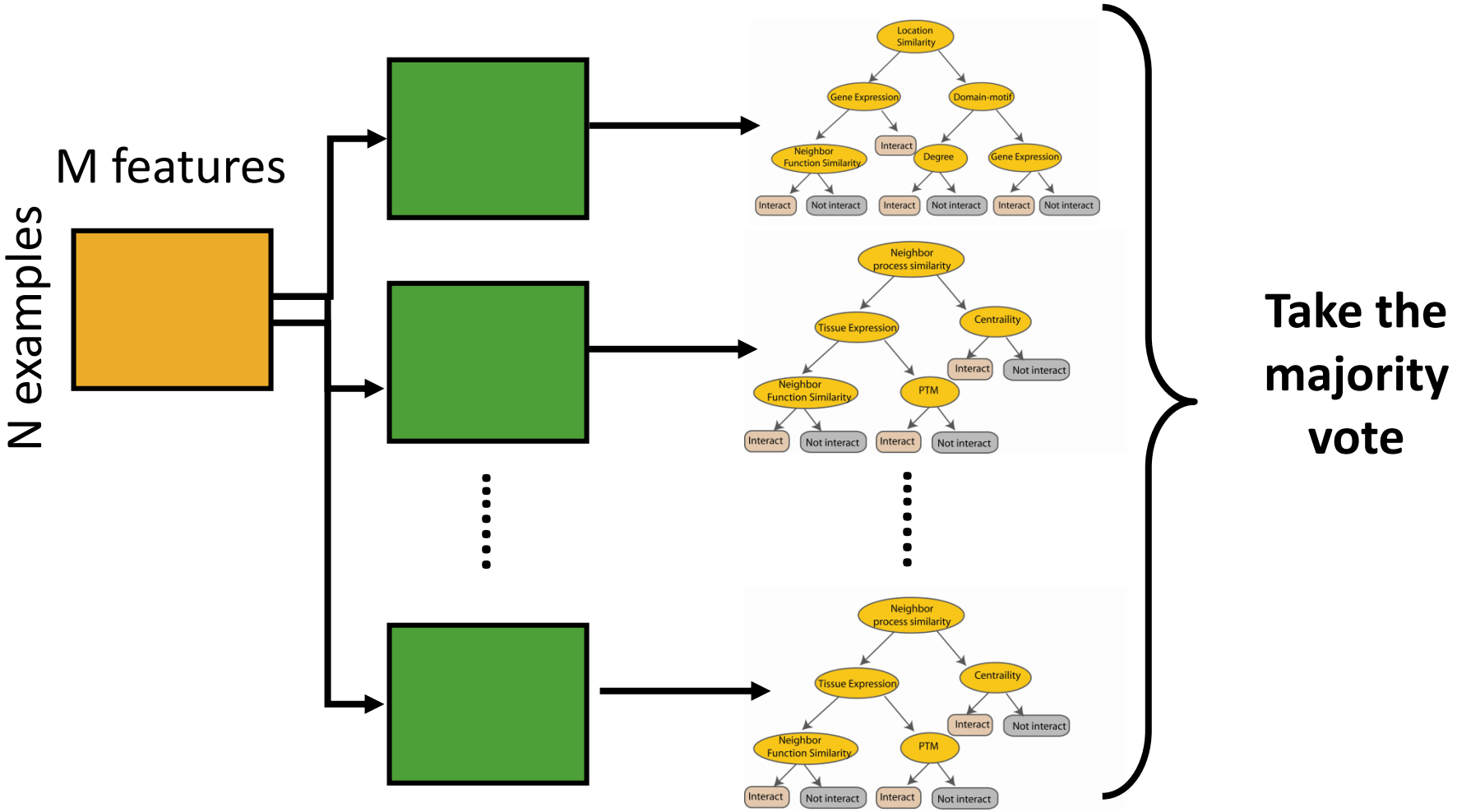
Random Forests



Construct a decision tree



Random Forests

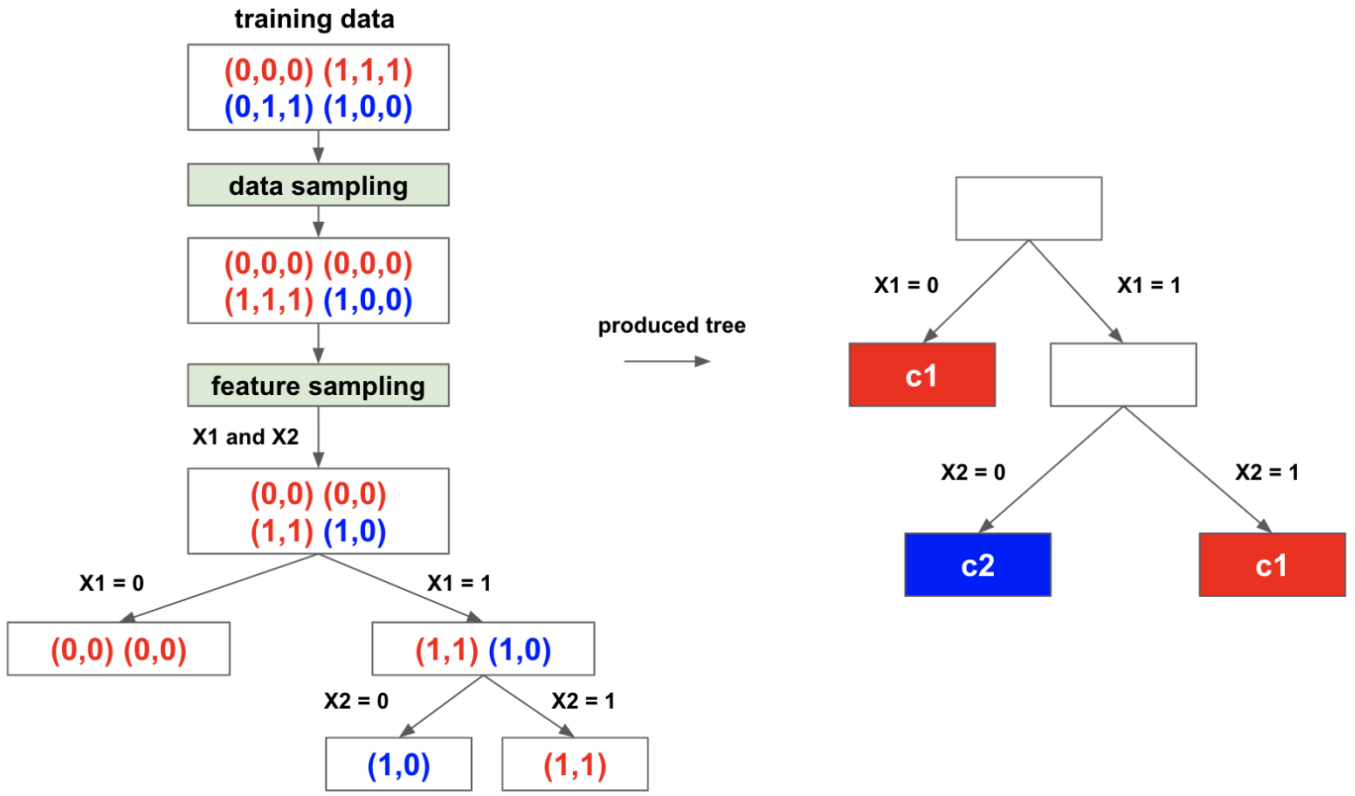


Random Forests - Example

X1	X2	X3	Class
0	0	0	c1
1	1	1	c1
0	1	1	c2
1	0	0	c2

An illustrative dataset with two classes colored in red and blue.

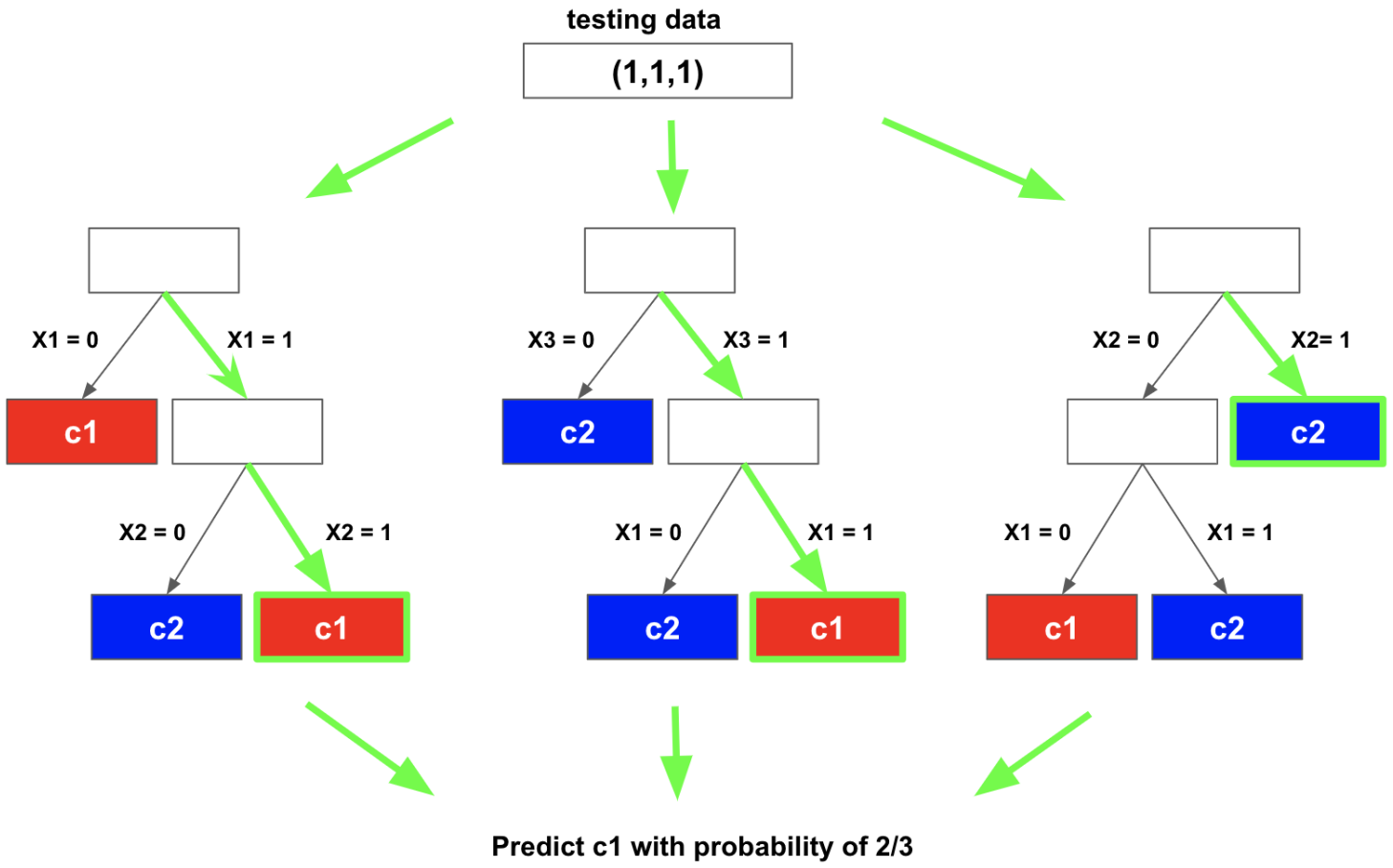
Random Forests - Build trees



X1	X2	X3	Class
0	0	0	c1
1	1	1	c1
0	1	1	c2
1	0	0	c2

The same process is applied to build multiple trees

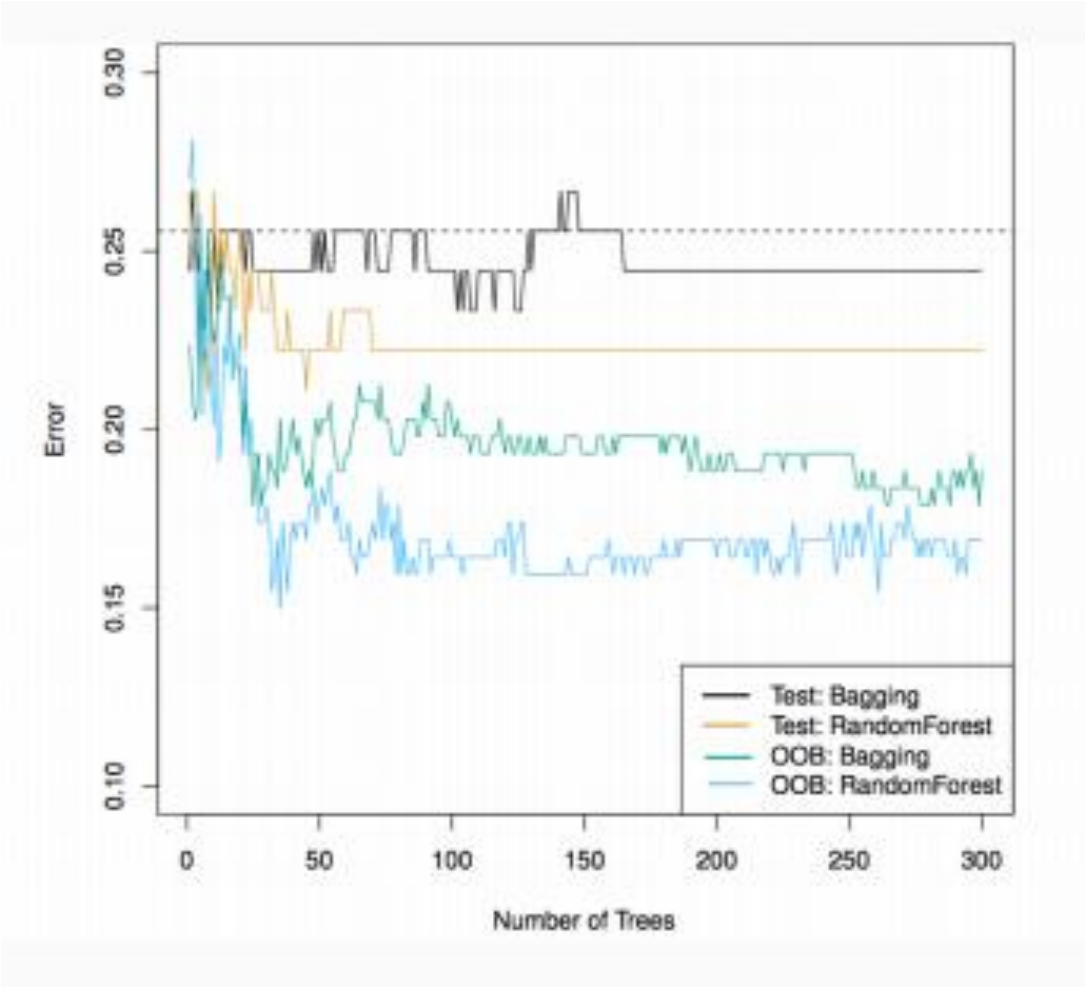
Random Forests - Predict



X1	X2	X3	Class
0	0	0	c1
1	1	1	c1
0	1	1	c2
1	0	0	c2

The flow of predicting a testing instance with a random forest with 3 trees.

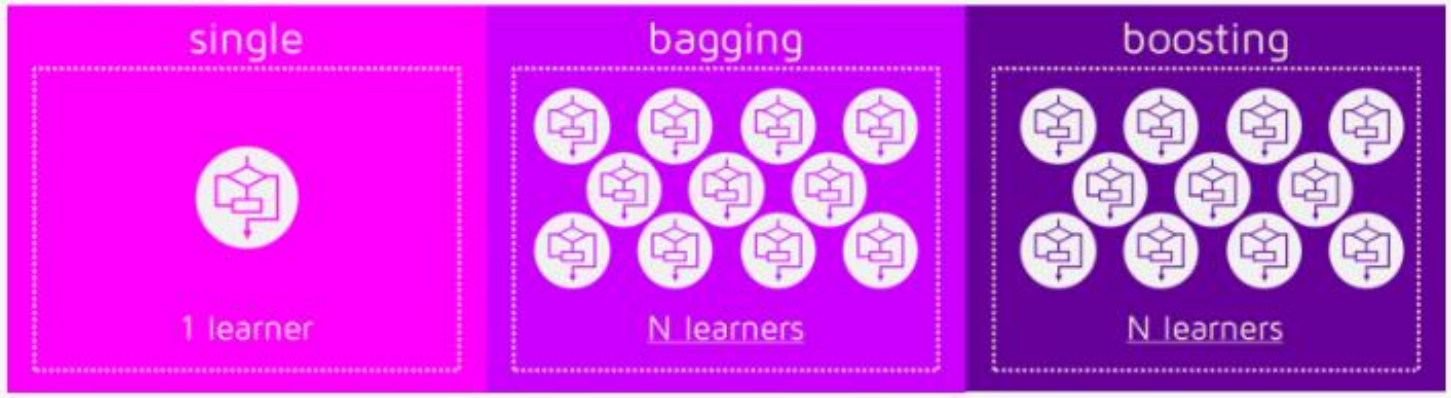
Bagging vs. RF



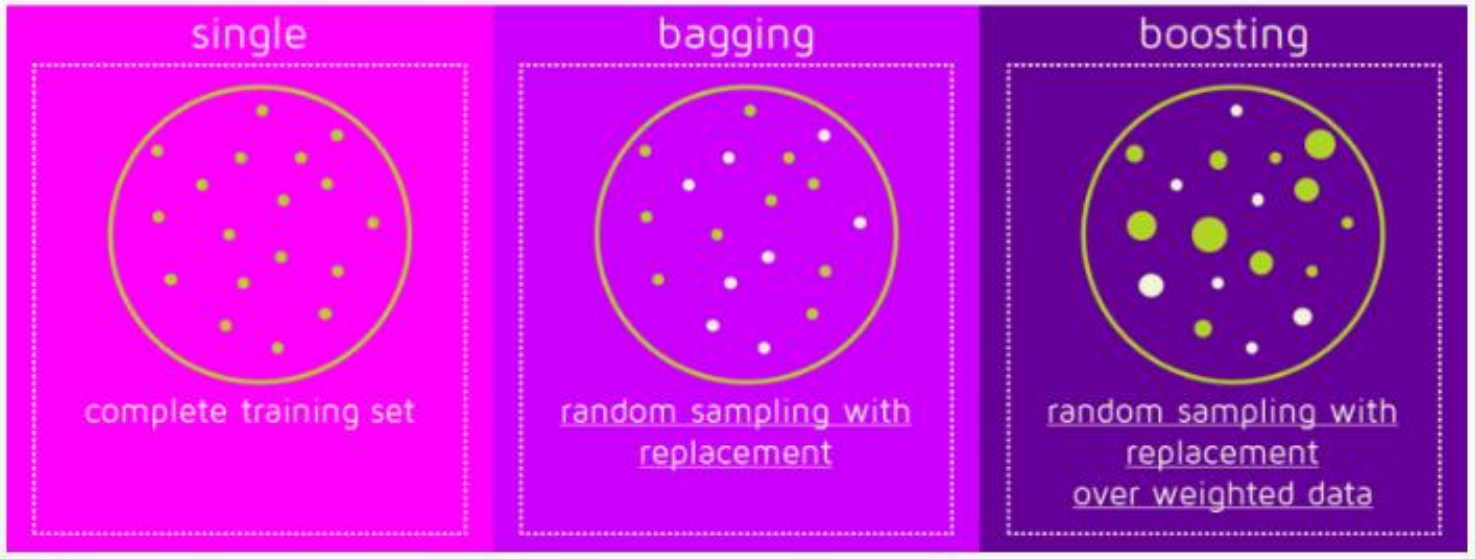
- Random forests trained with $m = \sqrt{p}$
- Bagging trees = Random forests with $m = p$

Bagging vs Boosting

- **They are both ensemble methods**

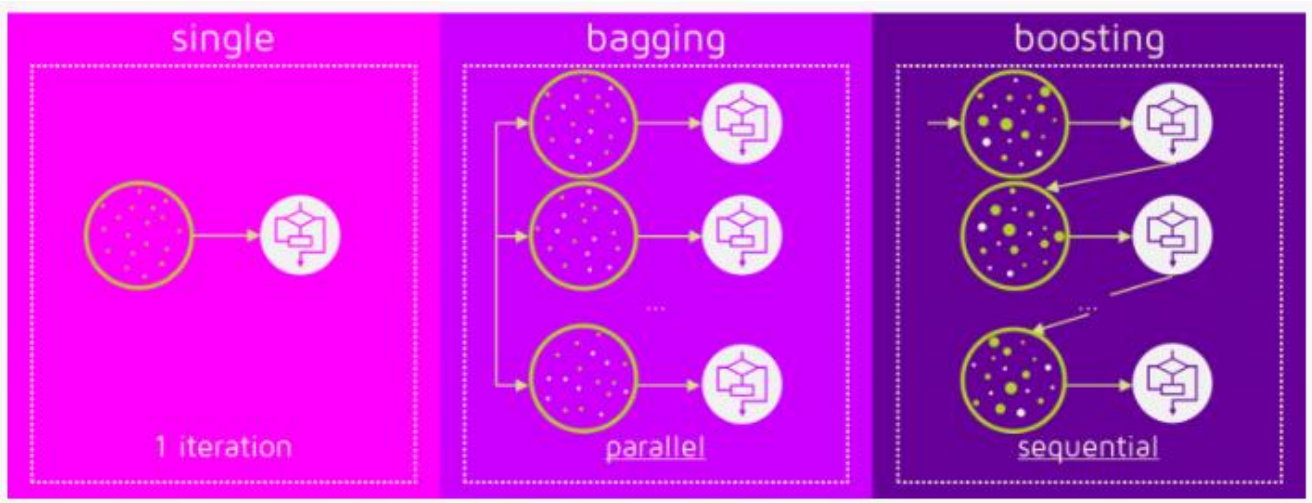


- **How do Bagging and Boosting get N learners?**

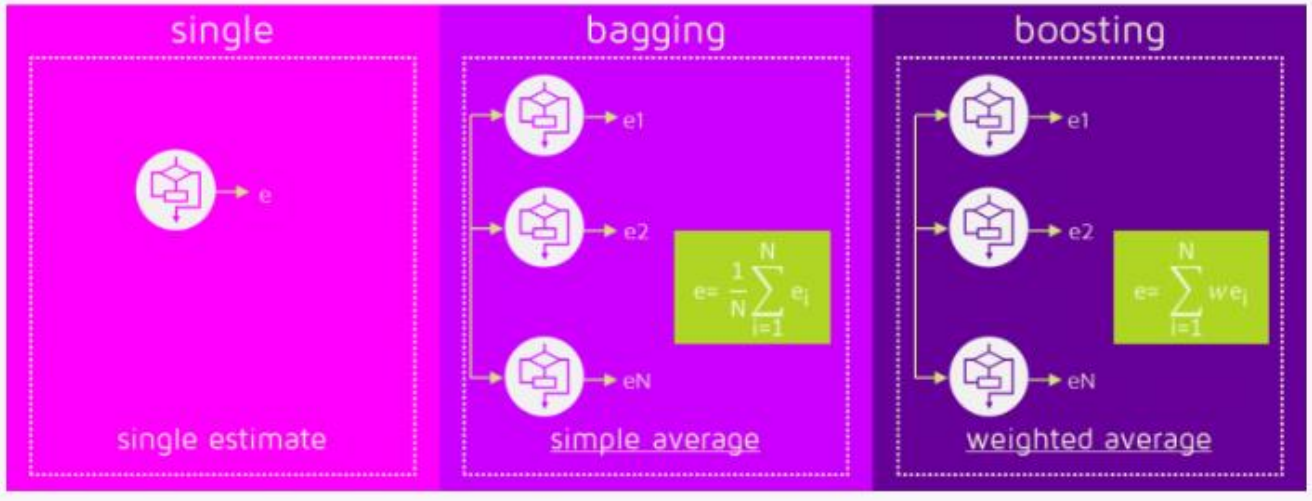


Bagging vs Boosting

- **Training stage**



- **Classification stage**



Ensemble Strategy

Ensemble ?

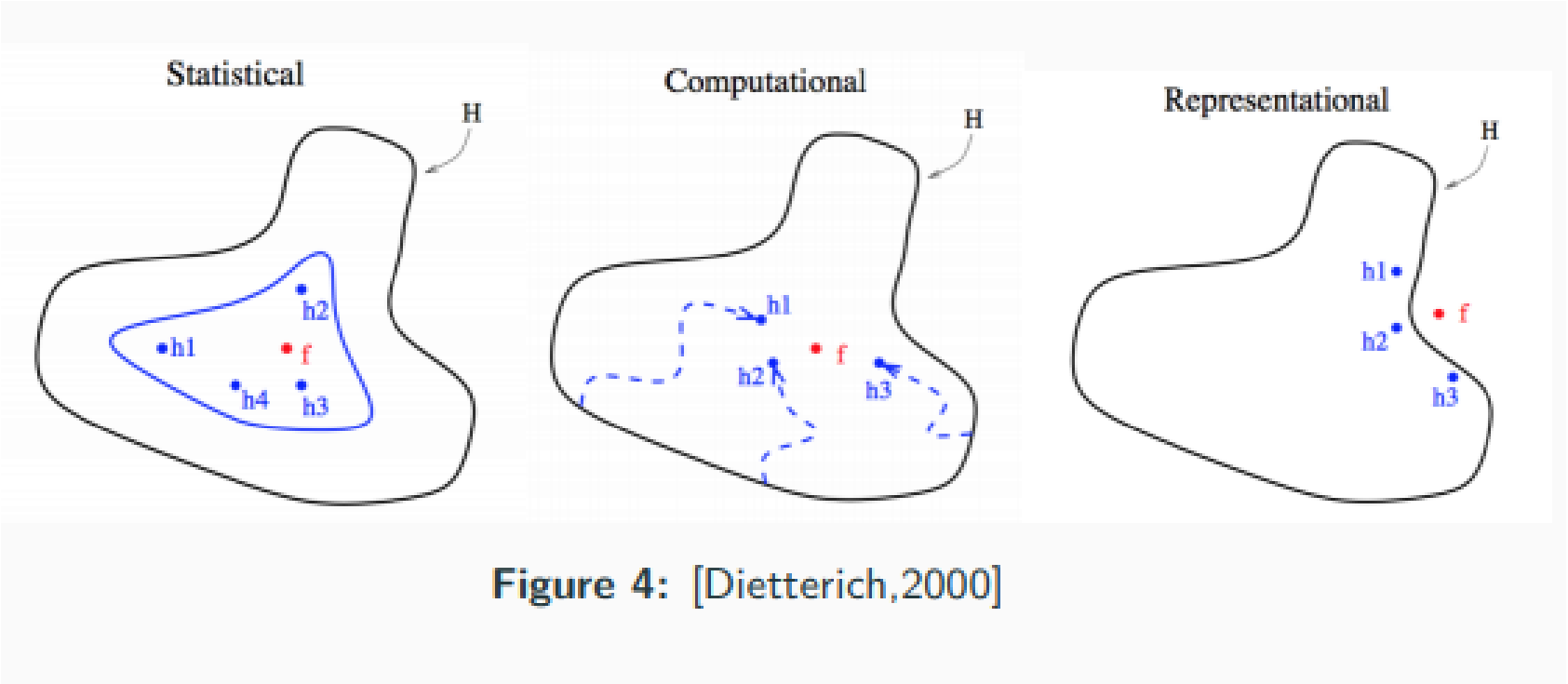


Figure 4: [Dietterich,2000]

- Statistical: equal performance on training set but different generality performance
- Computational: bad generality of local minimal
- Representational: true hypothesis not in hypothesis space

Ensemble Strategy

- Suppose we ensemble T base learners: $\{h_1, h_2, \ldots, h_T\}$
- Denote the output of h_i on instance $\boldsymbol{x}$ as $h_i(\boldsymbol{x})$
- There are three common ensemble strategies:
  - Averaging
  - Voting
  - Learning: Stacking [wolpert, 1992; Breiman, 1996b]

Averaging

- For numerical output $h_i(\boldsymbol{x}) \in \Re$,

  the most common ensemble strategy is averaging

- Averaging:
  - simple averaging: $H(x) = \frac{1}{T}\sum_{i=1}^{T} h_i(x)$
  - weighted averaging: $H(x) = \sum_{i=1}^{T} w_i h_i(x)$, $w_i \geq 0$, $\sum_{i=1}^{T} w_i = 1$.
  - simple averaging preferred, especially for large T or similar learners

Voting

- In classification task, base learner h_i predicts a label from label set $\{c_1, c_2, \ldots, c_N\}$, where the voting is the most common ensemble strategy.

- Voting:
  - majority voting: $H(\boldsymbol{x}) = \begin{cases} c_j, & if \ \sum_{i=1}^{T} h_i^j(\boldsymbol{x}) > 0.5 \sum_{k=1}^{N} \sum_{i=1}^{T} h_i^k(\boldsymbol{x}) \\ reject, & otherwise \end{cases}$
  - plurality voting: $H(\mathbf{x}) = c_{\underset{j}{argmax} \sum_{i=1}^{T} h_i^j(\mathbf{x})}$
  - weighted voting: $H(\mathbf{x}) = c_{\underset{j}{argmax} \sum_{i=1}^{T} w_i h_i^j(\mathbf{x})}, w_i \geq 0, \ \sum_{i=1}^{T} w_i = 1$

Voting

- Class label: $h_i^j(\boldsymbol{x})\epsilon\{0,1\}$, $h_i^j(\boldsymbol{x})$ = 1 if h_i predicts c_j to 1, otherwise 0.

The voting using it called hard voting

- Class probability: $h_i^j(\boldsymbol{x})\epsilon[0,1]$, it's an estimation about posterior $P(c_j|\boldsymbol{x})$

The voting using it called soft voting

- We can't mixture above two kind of voting and learn a ensemble classifier
- Empirically, the classifier learned from class probability is always better than the one learned from class tag.

Stacking

- use a learner to ensemble
  - elementary learners $\{h_1, h_2, \ldots, h_T\}$ generate a new data set
  - training a meta-learner h' on the new data set
- avoid overfitting: k-fold cross-validation
  - divide D into $D_1, D_2, \ldots, D_k$
  - any $x \in D_j$ train h_t^j on $\overline{D_j}$
- meta-learner and its input

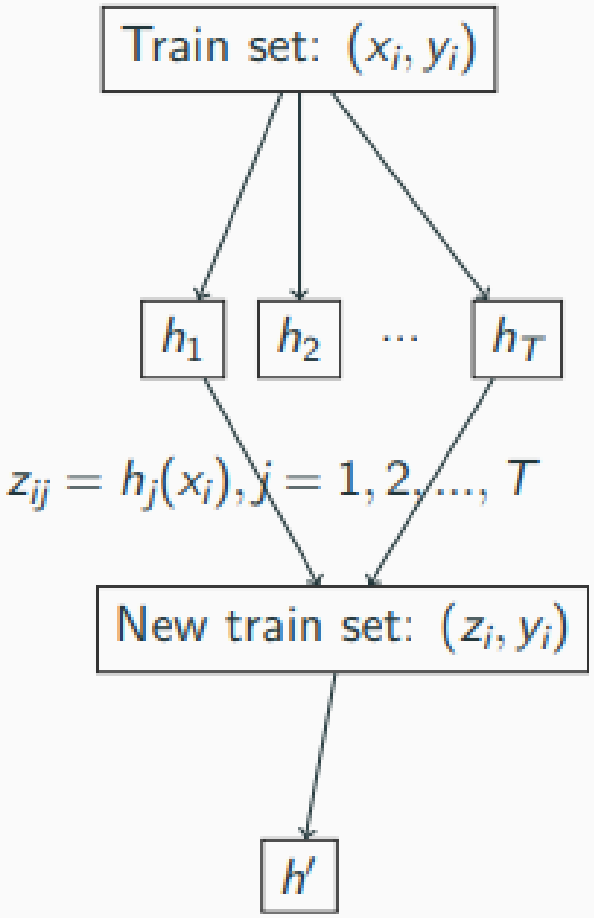


Figure 5: Stacking

Stacking

Input: train set $D = (x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$

primal-learner $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_T$

meta-learner $\mathcal{L}$

Process:

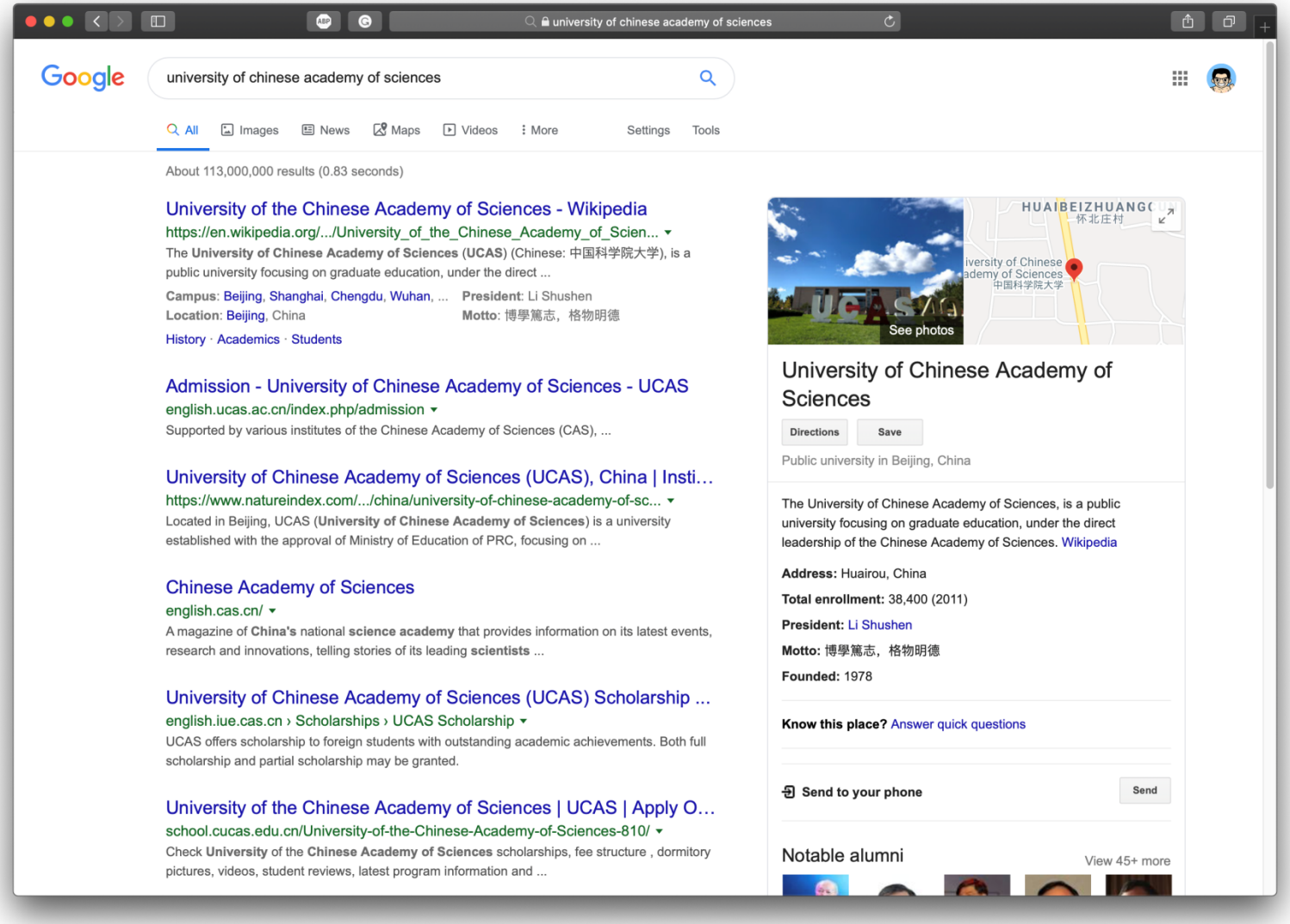
1. **for** $t = 1,2, \ldots, T$ **do**
2. $h_t = \mathcal{L}_t(D)$
3. **end for**
4. $D' \neq \phi$
5. **for** $i = 1,2, \ldots, m$ **do**
6. **for** $t = 1,2, \ldots, T$ **do**
7. $z_{it} = h_t(x_i)$
8. **end for**
9. $D' = D' \cup (z_{i1}.z_{i2}, \ldots, z_{iT}, y_i)$
10. **end for**
11. $h' = \mathcal{L}(D')$

Output: $H(x) = h'(h_1(x), h_2(x), \ldots, h_T(x))$

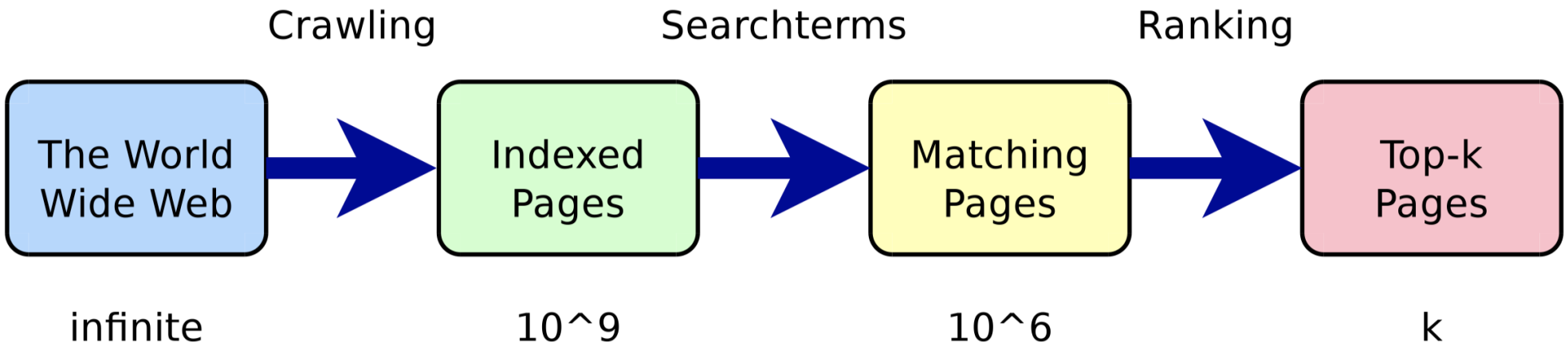
Applications: Web Search & Recommender System

How to check if a document is relevant or not?

How to rank these relevant documents?

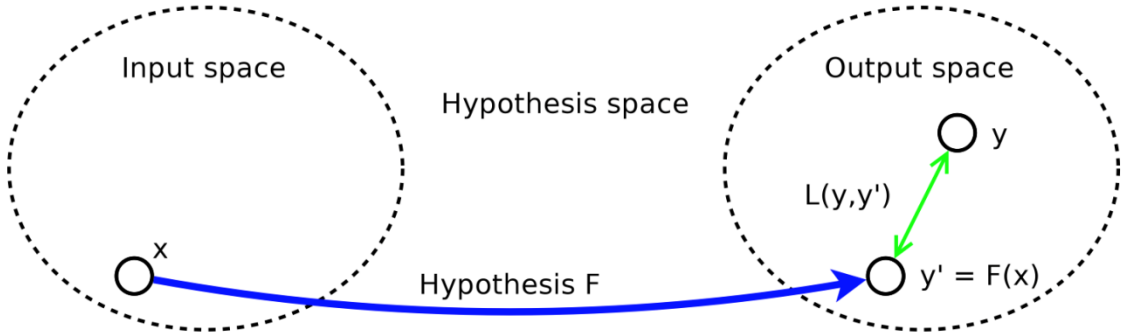


Web Scale Information Retrieval



The “retrieval pipeline” must reduce the number of pages significantly!

Learning to Rank



From: Liu (2010), *Learning to Rank for Information Retrieval*.

- Basic Idea of Machine Learning:
  - Hypothesis *F* transforms input object x to output object *y ′ = F(x).*
  - *L(y, y ′)* is the loss, i.e. the difference between the predicted *y′* and the target *y*.
  - "Learning" process: find the hypothesis minimizing *L* by tuning *F*.

- Learning a ranking function with machine learning techniques:

  Learning to Rank (LTR)

Features for Learning

- To learn a ranking function, each query-document pair is represented by a vector of features of three categories:
  - Features modelling web document, d (static features): inbound links, PAGE rank, document length, etc.
  - Features modelling query-document relationship (dynamic features): frequency of search terms in document, cosine similarity, etc.
  - Features modelling user query, q: number of words in query, query classification, etc.

- In supervised training, the ranking function is learned using vectors of known ranking levels

Evaluation Metric - Discounted Cumulative Gain

- Popular measure for evaluating web search and related tasks

- Two assumptions:
  - Highly relevant documents are more useful than marginally relevant documents
  - The lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined

Evaluation Metric - Discounted Cumulative Gain

- Uses graded relevance as a measure of usefulness, or gain, from examining a document
- Gain is accumulated starting at the top of the ranking and may be reduced, or discounted, at lower ranks

- Typical discount is 1/log (rank)
  - With base 2, the discount at rank 4 is 1/2, and at rank 8 it is 1/3

Evaluation Metric - Discounted Cumulative Gain

- What if relevance judgments are in a scale of [0,r]? r>2
- Cumulative Gain (CG) at rank n
  - Let the ratings of the n documents be $r_1, r_2, \ldots r_n$ (in ranked order)
  - CG = $r_1 + r_2 + \ldots r_n$

- Discounted Cumulative Gain (DCG) at rank n
  - DCG = $r_1 + r_2 / \log_2 2 + r_3 / \log_2 3 + \ldots r_n / \log_2 n$
  - We may use any base for the logarithm

Evaluation Metric - Discounted Cumulative Gain

- DCG is the total gain accumulated at a particular rank p:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

- Used by some web search companies
- Emphasis on retrieving highly relevant documents

Evaluation Metric - Discounted Cumulative Gain

- 10 ranked documents judged on 0-3 relevance scale:
  - 3, 2, 3, 0, 0, 1, 2, 2, 3, 0

- Discounted gain:

  3, 2/1, 3/1.59, 0, 0, 1/2.59, 2/2.81, 2/3, 3/3.17, 0

  = 3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0

- DCG:

  3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61

Evaluation Metric – Normalized Discounted Cumulative Gain

- Normalized Discounted Cumulative Gain (NDCG) at rank n
  - Normalize DCG at rank n by the DCG value at rank n of the ideal ranking
  - The ideal ranking would first return the documents with the highest relevance level, then the next highest relevance level, etc
- Normalization useful for contrasting queries with varying numbers of relevant results
- NDCG is now quite popular in evaluating Web search

Evaluation Metric – Normalized Discounted Cumulative Gain

4 documents: d_1, d_2, d_3, d_4

i	Ground Truth		Ranking Function$_1$		Ranking Function$_2$	
	Document Order	r_i	Document Order	r_i	Document Order	r_i
1	d4	2	d3	2	d3	2
2	d3	2	d4	2	d2	1
3	d2	1	d2	1	d4	2
4	d1	0	d1	0	d1	0
	$NDCG_{GT}$=1.00		$NDCG_{RF1}$=1.00		$NDCG_{RF2}$=0.9203	

$$DCG_{GT} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

$$DCG_{RF1} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

$$DCG_{RF2} = 2 + \left(\frac{1}{\log_2 2} + \frac{2}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.2619$$

$$MaxDCG = DCG_{GT} = 4.6309$$

LambdaMART

- It had repeatedly appeared in various machine learning contests before deep learning
- The algorithm used by the champion of 2008 Yahoo! Learning to Rank Challenge is LambdaMART
- It was used by Bing and Facebook.

6.2. Methods used

The similarity between the methods used by the winners is striking: all of them used decision trees and ensemble methods.

Burges et al. (2011) used a linear combination of 12 ranking models, 8 of which were LambdaMART (Burges, 2010) boosted tree models, 2 of which were LambdaRank neural nets, and 2 of which were logistic regression models. While LambdaRank was originally instantiated using neural nets, LambdaMART implements the same ideas using the boosted-tree style MART algorithm, which itself may be viewed as a gradient descent algorithm. Four of the LambdaMART rankers (and one of the nets) were trained using the ERR measure, and four (and the other net) were trained using NDCG. Extended training sets

(Yahoo! Learning to Rank Challenge Overview, 2008)

RankNet

- Differentiable function of the model parameters, typically neural nets
- RankNet maps a feature vector x to a value $f(x;\ w)$
- Learned probabilities URL $U_i \succ U_j$ modelled via a sigmoid function

$$P_{ij} = P\left(U_i \succ U_j\right) = \frac{1}{1+e^{-\sigma(s_i - s_j)}}$$

- with $s_i = f(x_i), s_j = f\left(x_j\right)$
- Cost function calculates cross entropy

$$C = -\bar{P}_{ij}\log P_{ij} - (1 - \bar{P}_{ij})\log\,(1 - P_{ij})$$

- P_{ij} is the model probability, $\bar{P}_{ij}$ is the known probability from training

RankNet Algorithm

Algorithm 2 RankNet Training.

Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for each query $q \in Q$ **do**
    for each pair of URLs U_i, U_j with different label **do**
        $s_i = f(\mathbf{x}_i)$, $s_j = f(\mathbf{x}_j)$
        Estimate cost C
        Update model scores $w_k \rightarrow w_k - \eta \frac{\partial C}{\partial w_k}$
    end for
end for
Return **w**

RankNet λ's

- The crucial part is the update

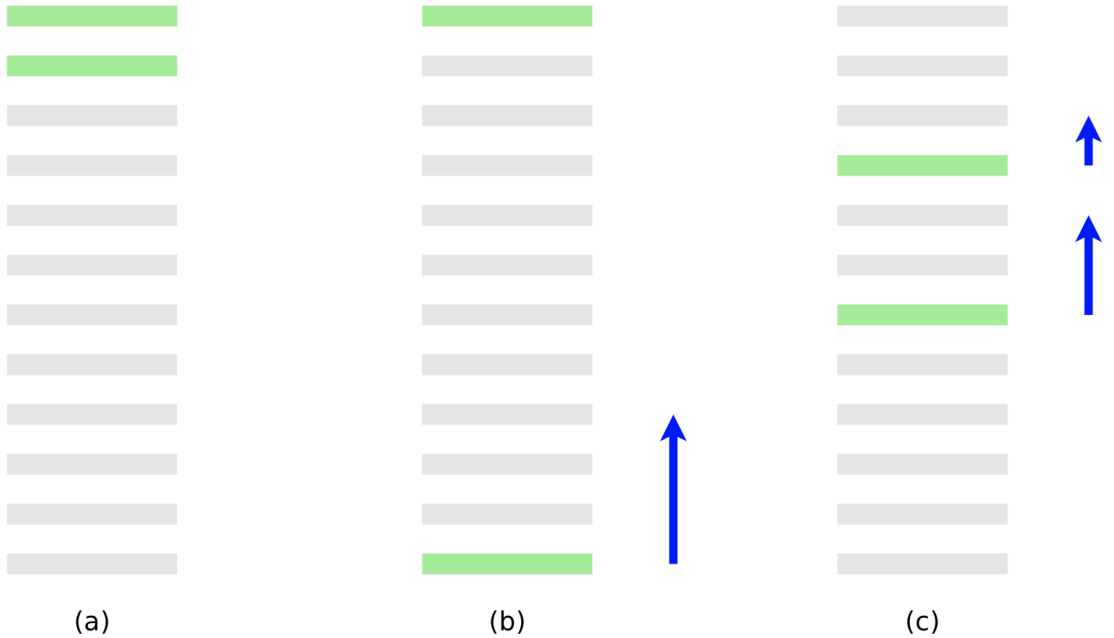
$$\frac{\partial C}{\partial w_k} = \frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial w_k} = \lambda_{ij}\left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}\right)$$

- λ_{ij} describes the desired change of scores for the pair U_i and U_j
- The sum over all λ_{ij}'s and λ_{ji}'s of a given query-document vector x_i w.r.t. all other differently labelled documents is

$$\lambda_i = \sum_{j:\{i,j\}\in I} \lambda_{ij} - \sum_{k:\{k,i\}\in I} \lambda_{ki}$$

- λ_i is (kind of) a gradient of the pairwise loss of vector x_i .

RankNet Example



(a) is the perfect ranking, (b) is a ranking with 10 pairwise errors, (c) is a ranking with 8 pairwise errors. Each blue arrow represents the λ_i for each query-document vector $\mathbf{x}_i$.

From: Burges (2010), *From RankNet to LambdaRank to LambdaMART: An Overview.*

LambdaRank

- Many times only the wrong pair number is not enough. Such as the NDCG or ERR only focus on the ranking of top k.
- It is necessary to improve the existing loss or the gradient of loss, but NDCG (or others) is not differentiable.
- So skip loss and directly multiply another term on the gradient, thereby defining a new Lambda gradient
- From RankNet to LambdaRank:
  - Multiply λ's with |ΔZ|, i.e. the difference of an IR measure when U_i and U_j are swapped
  - E.g. |ΔNDCG| is the change in NDCG when swapping U_i and U_i :

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} |\Delta \text{NDCG}|$$

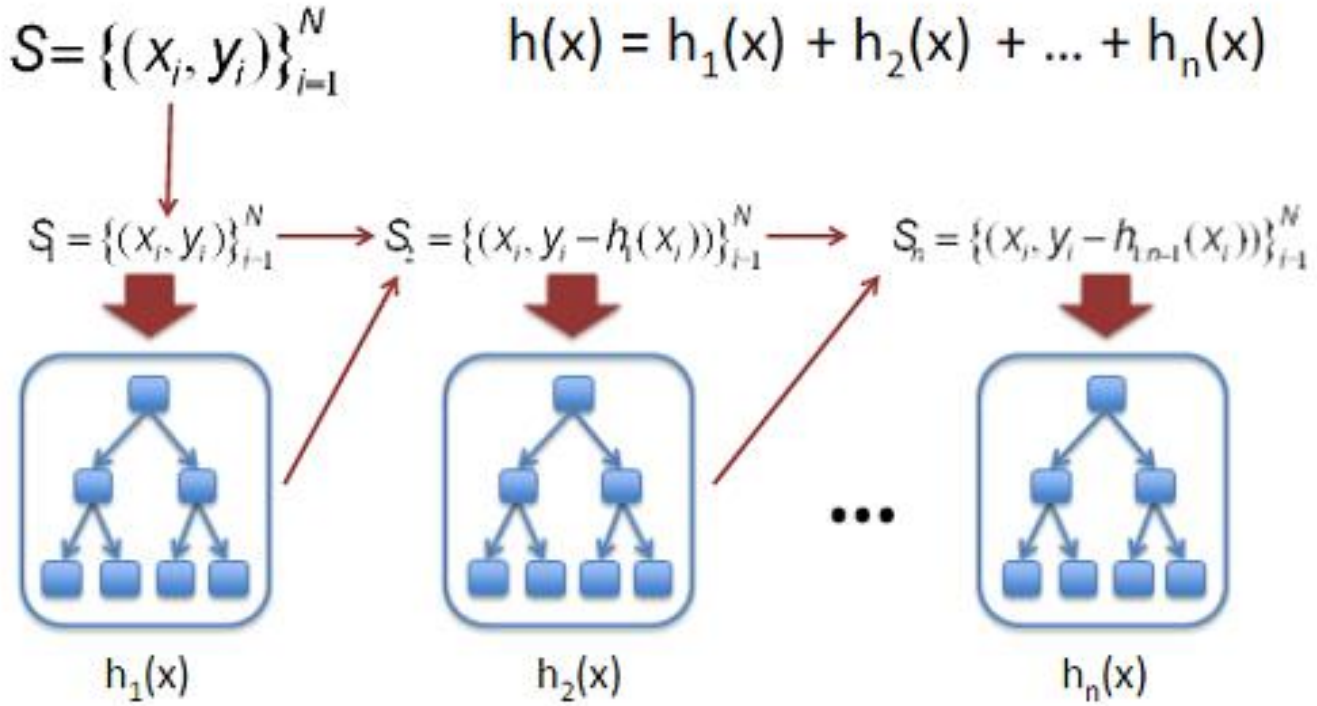
LambdaRank Example

Problem: RankNet is based on pairwise error, while modern IR measures emphasize higher ranking positions. Red arrows show better λ's for modern IR measures.

From: Burges (2010), *From RankNet to LambdaRank to LambdaMART: An Overview.*

Multiple Additive Regression Trees (MART, GBDT)

- Ensemble of Multiple Decision Tree



Multiple Additive Regression Trees (MART, GBDT)

- The goal of MART is to find a function that minimizes loss

$$F^* = \arg\min_F E_{y,\mathbf{x}} \left(L(y, F(\mathbf{x})) \right)$$

- This function has a certain form, that is, an additive combination of a group of weak learners

$$F(\mathbf{x}; \rho_m, \mathbf{a}_m) = \sum_{m=0}^{M} \rho_m h(\mathbf{x}; \mathbf{a}_m)$$

- MART adopts a greedy strategy, and the goal of each iteration is to minimize the loss, that is, to reduce

$$\tilde{y}_i = \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})}$$

Multiple Additive Regression Trees (MART)

Algorithm 1 Multiple Additive Regression Trees.

1: Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2: **for** $m = 1, ..., M$ **do** M个决策树
3: **for** $i = 1, ..., N$ **do** N个样本
4: $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$
5: **end for**
6: $\{R_{km}\}_{k=1}^{K}$ // Fit a regression tree to targets $\tilde{y}_{im}$
7: **for** $k = 1, ..., K_m$ **do** Km个叶子节点
8: $\gamma_{km} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 每个叶子节点的预测值
9: **end for**
10: $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$
11: **end for**
12: Return $F_M(\mathbf{x})$

LambdaMART

- Clearly, since MART models derivatives, and since LambdaRank works by specifying the derivatives at any point during training, the two algorithms are well matched: LambdaMART is the marriage of the two.
- MART is a class of algorithms, rather than a single algorithm, because it can be trained to minimize general costs.
- From LambdaRank to LambdaMART:
  - LambdaRank models gradients
  - MART works on gradients
  - Combine both to get LambdaMART:

    ⇒ MART with specified gradients and Newton step

LambdaMART Algorithm

Algorithm 3 LambdaMART.

for $i = 0, ..., N$ **do**
    $F_0(\mathbf{x}_i) = \text{BaseModel}(\mathbf{x}_i)$ // Set to 0 for empty BaseModel
end for
for $m = 1, ..., M$ **do**
    for $i = 0, ..., N$ **do**
        $y_i = \lambda_i$ // Calculate λ-gradient
        $w_i = \frac{\partial y_i}{\partial F_{k-1}(\mathbf{x}_i)}$ // Calculate derivative of gradient for $\mathbf{x}_i$
    end for
    $\{R_{km}\}_{k=1}^{K}$ // Create K-leaf tree on $\{\mathbf{x}_i, y_i\}$
    $\gamma_{km} = \frac{\sum_{x_i \in R_{km}} y_i}{\sum_{x_i \in R_{km}} w_i}$ // Assign leaf values
    $F_m(\mathbf{x}_i) = F_{m-1}(\mathbf{x}_i) + \eta \sum_k \gamma_{km} 1(\mathbf{x}_i \ in R_{km})$
end for

Thanks !