Decision Trees and Ensemble Algorithms

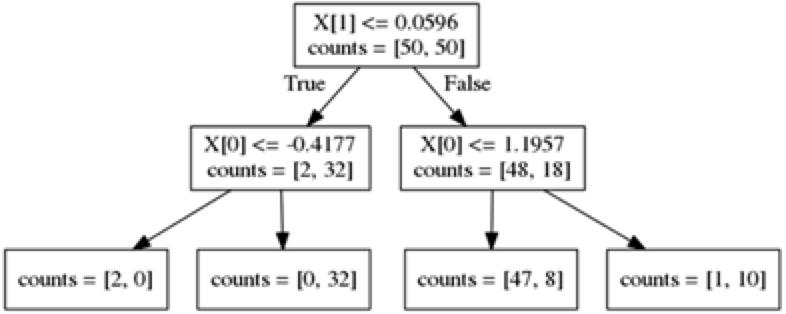
Injung Kim
Handong Global University

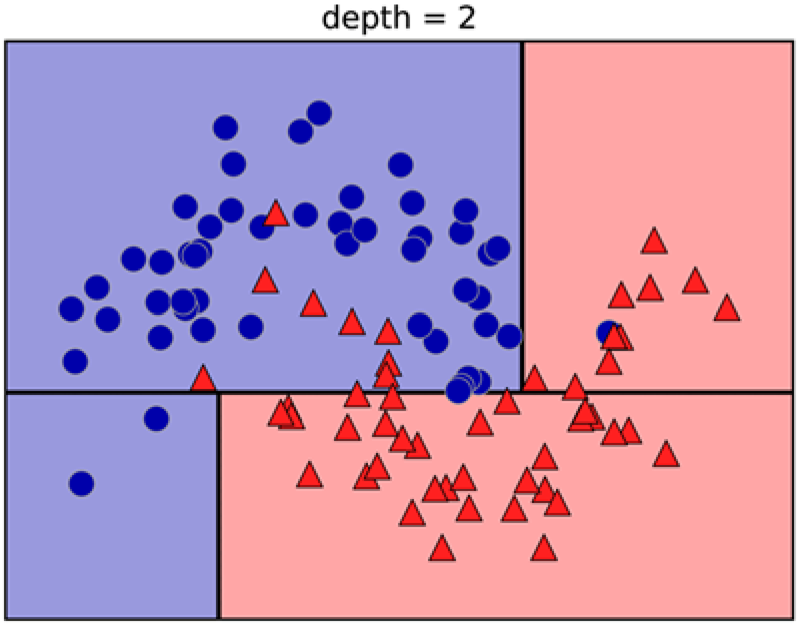
Agenda

- Decision Trees
- Random Forests
- AdaBoost
- Gradient Boost (incl. XGBoost)

Decision Trees

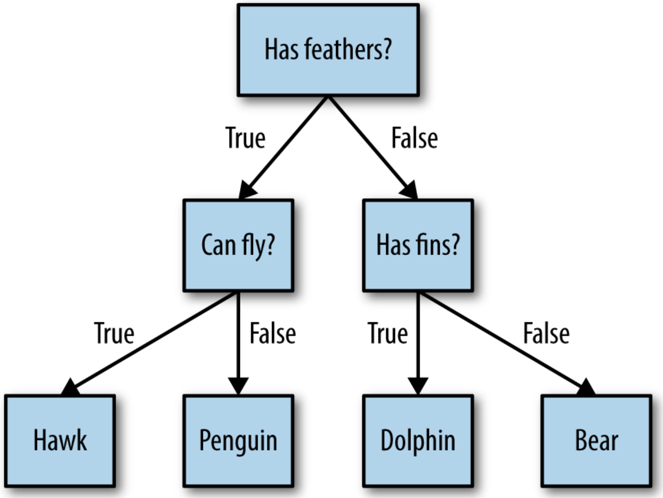
- Decision tree: a tree structure that represents a hierarchy of if~else questions leading to a decision.
  - Each if~else question (feature, threshold) splits feature space





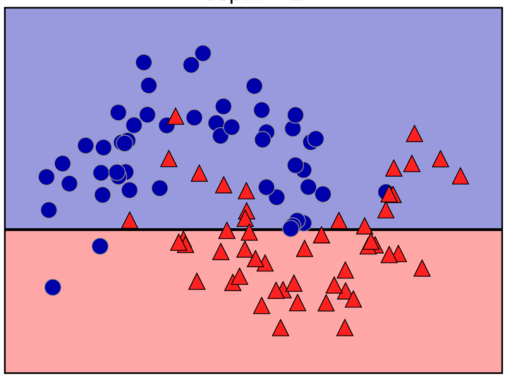
Decision Tree Learning

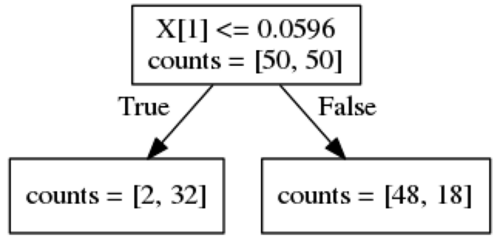
- Building decision tree
  - Learn a sequence of if~else questions that gets us to the true answer most quickly.
  - Iteratively find (feature, threshold) to split the region



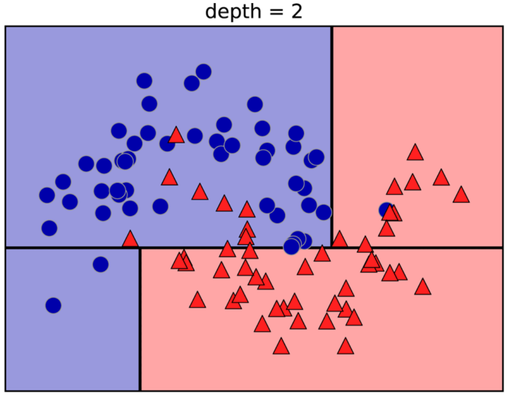
Building Decision Tree

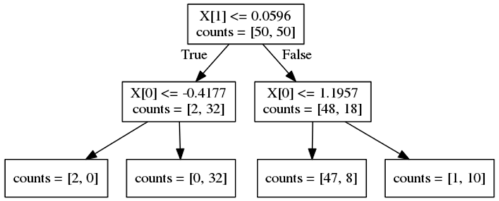
depth = 1



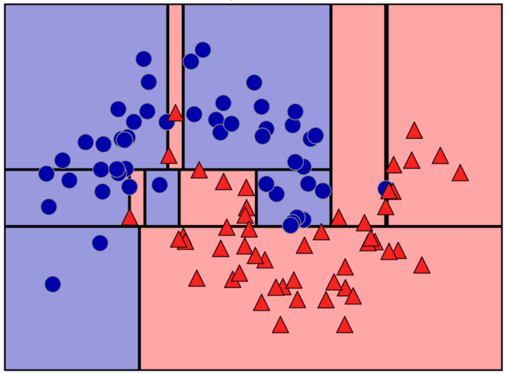


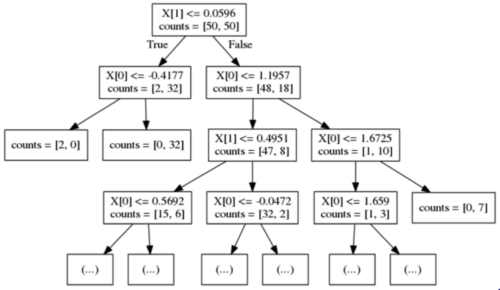
depth = 2





depth = 9





Decision Tree Learning

- Split algorithm
  - Iteratively at each node
    - Enumerate over all features
      - For each feature, sort 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

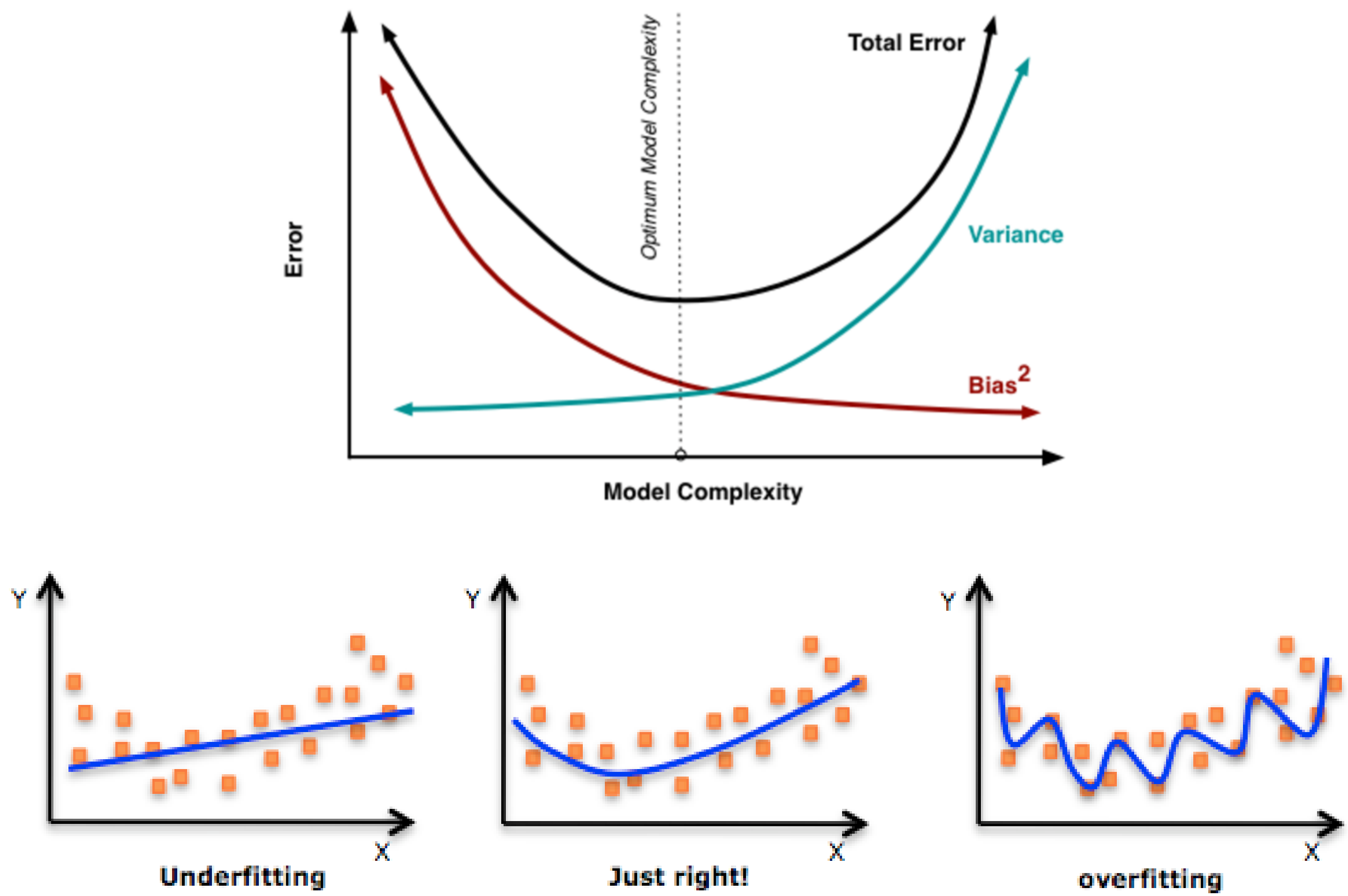
- Measurements for 'the best split'

$$\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t),$$

$$\text{Gini}(t) = 1-\sum_{i=0}^{c-1}[p(i|t)]^2 = \sum_{i=0}^{c-1} p(i|t)(1-p(i|t))$$

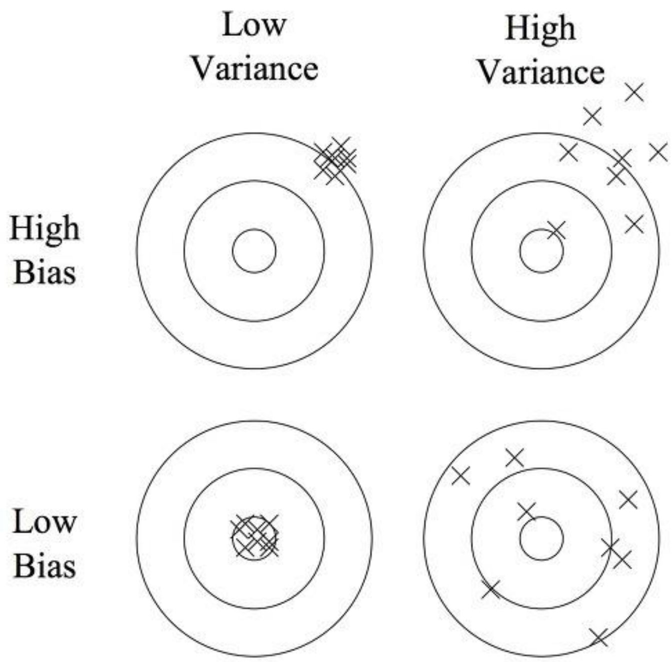
$$\text{Classification error}(t) = 1-\max_i[p(i|t)],$$

Bias–Variance Trade–off



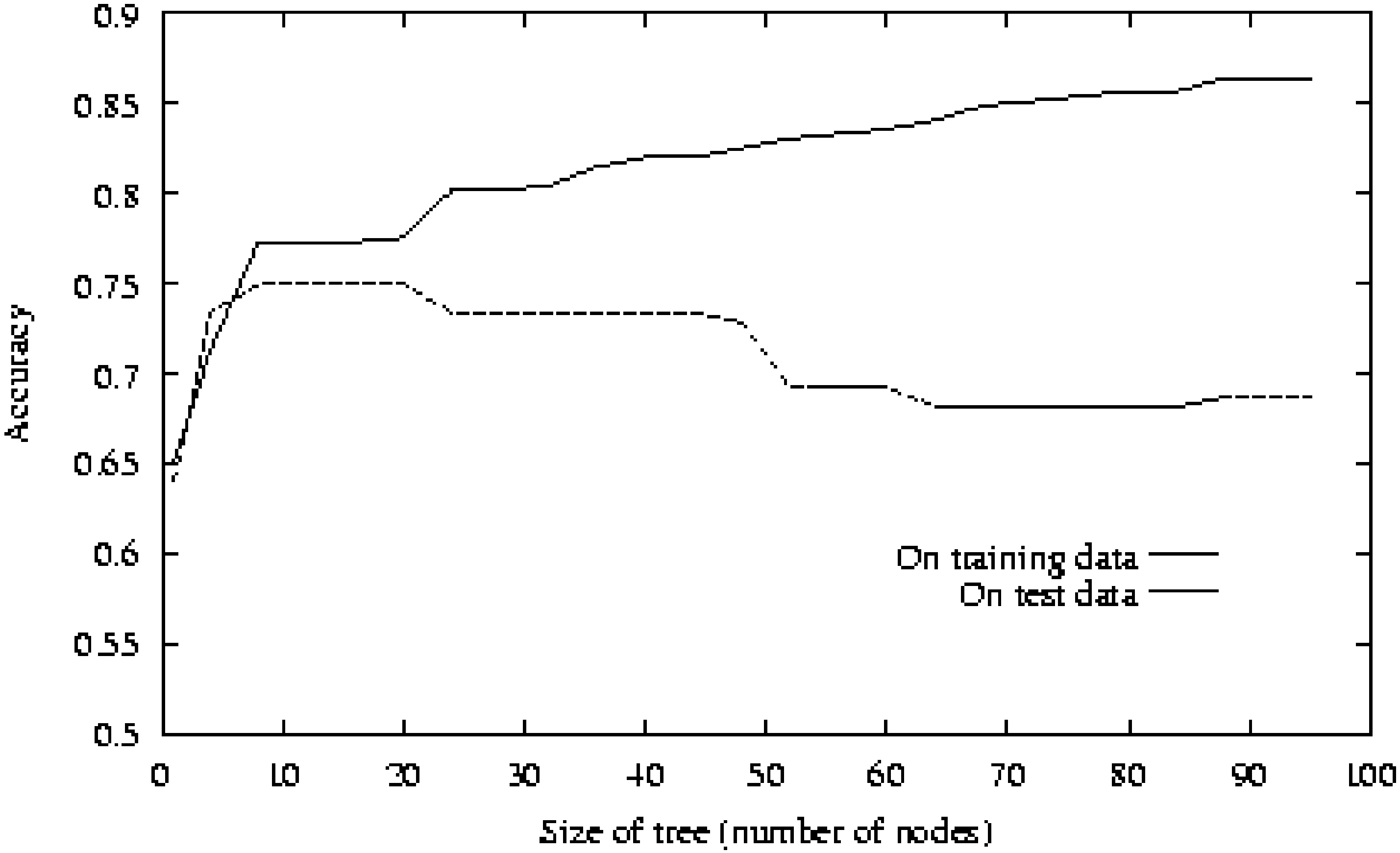
Bias-Variance Trade-off

- Approximating Y by $\hat{Y}$
  - $Y = f(X) + \epsilon,\ \hat{Y} = \hat{f}(X)$
  - $E\left(\hat{Y} - Y\right)^2 = E\left(\hat{f}(x) - Y\right)^2$
    $$= E\left[\hat{f}(X) - f(X)\right]^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + var(\epsilon)$$
- Bias $\hat{f}(X) - f(X)$
  - Error caused by inappropriate model (too simple model)
  - Related to under-fitting
- Variance $\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2$
  - Error caused by incorrect parameters (too complex model)
  - Related to over-fitting



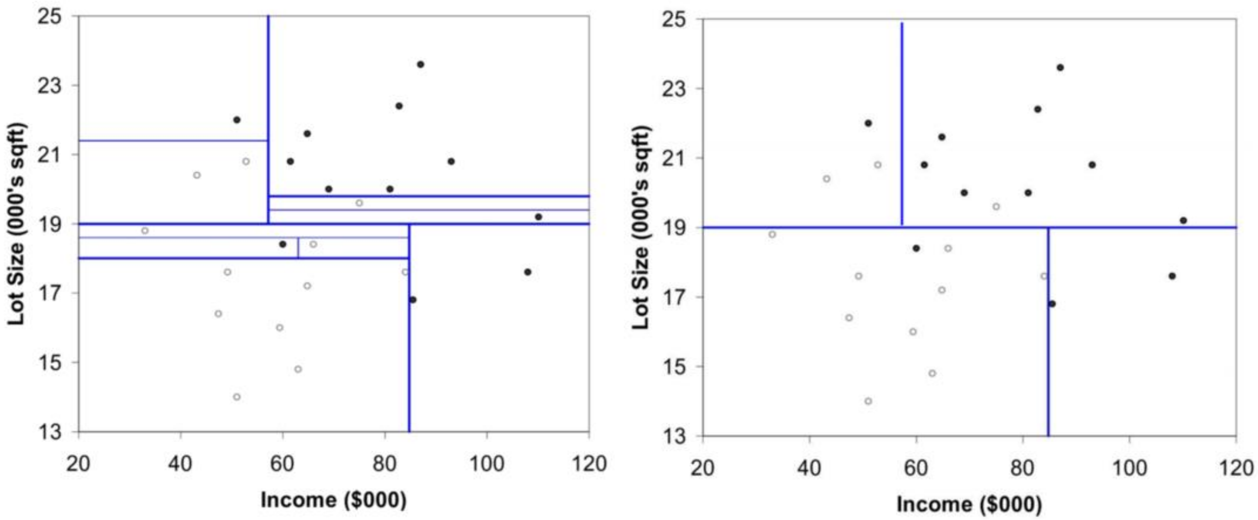
Overfitting of Decision Tree

- Decision tree needs pruning to control complexity



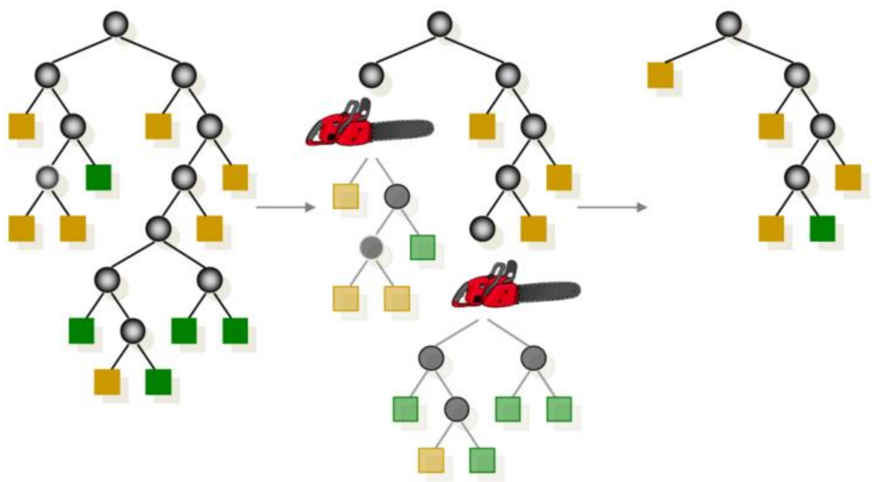
Pruning

- Too complex tree does not generalize well.
  - Vulnerable to overfitting



Pruning

- CART: first split to maximum size, than prune
- Cost function to find predictive and simple tree
  - $Cost(T) = Error(T) + \alpha \cdot L(T)$
    - $Error(T)$: error (e.g. $\sum(\hat{y} - y)^2$)
    - $L(T)$: complexity (# of nodes)
    - α : regularization factor (usually, 0.1 ~ 0.01)



Regression Tree

- Prediction
  - Average of samples in the leaf node
    - <# of possible values> = <# of leaf nodes>

c.f. Classification tree: voting at the leaf node

- Impurity of nodes
  - Residual sum of squares (deviance)

$$\sum_{x \in \chi_m} (y_i - \hat{y}_i)^2$$

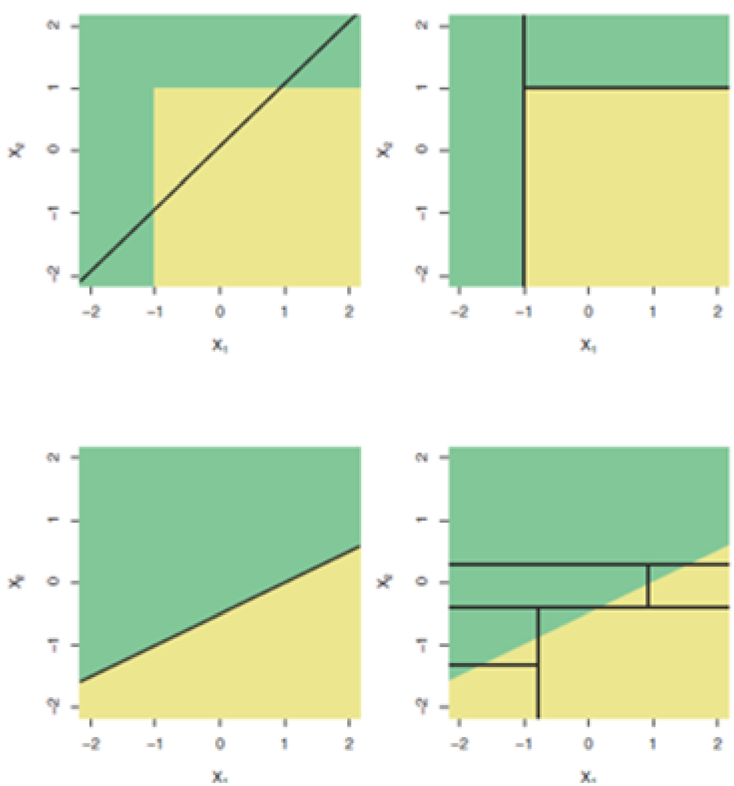
Pros and Cons

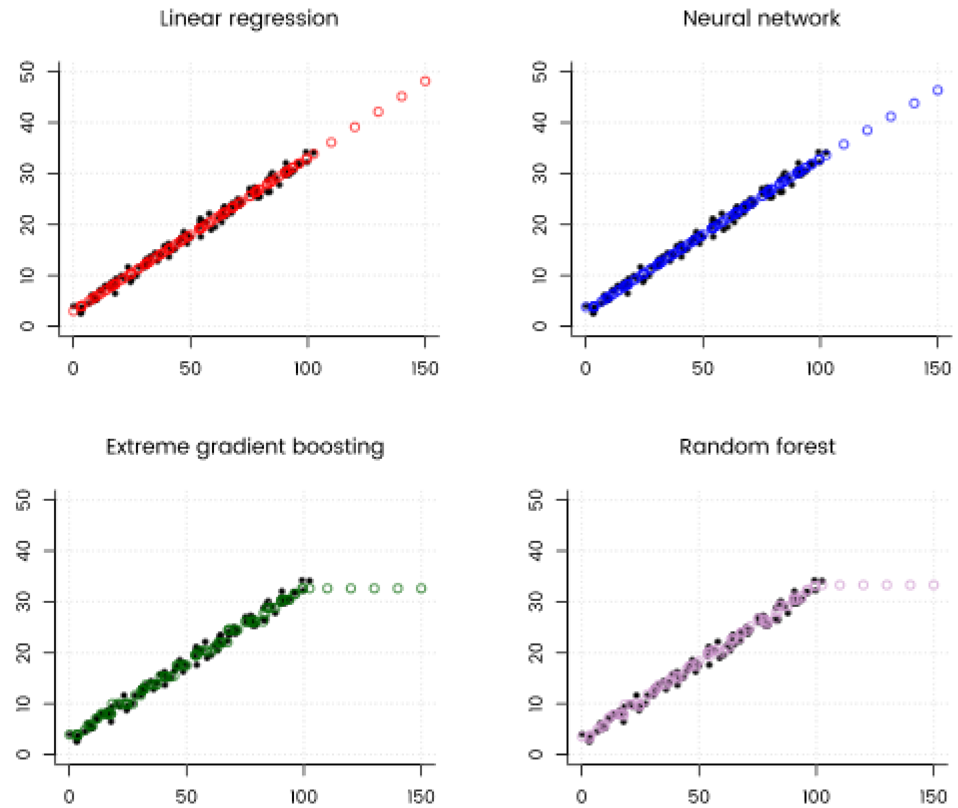
- Pros
  - Simple and efficient
  - Easy to analyze
    - Intuitive
    - Provides feature importance measure <u>specific to a particular tree</u>.
  - Not require data normalization
  - Scalable

- Cons
  - High variance (overfitting)
  - Learning algorithms does not guarantee optimum tree
  - Small difference can result in significantly different tree
  - Ineffective or inefficient for complex tasks (e.g. XOR)
    - Boundaries are perpendicular to feature axes
    - Cannot analyze correlation between variables
  - Cannot extrapolate

Limitations of Decision Trees

Linear model Decision trees





Pros and Cons

- Pros
  - Simple and efficient
  - Easy to analyze
    - Intuitive
    - Provides feature importance measure <u>specific to a particular tree</u>.
  - Not require data normalization
  - Scalable

Tree ensemble!

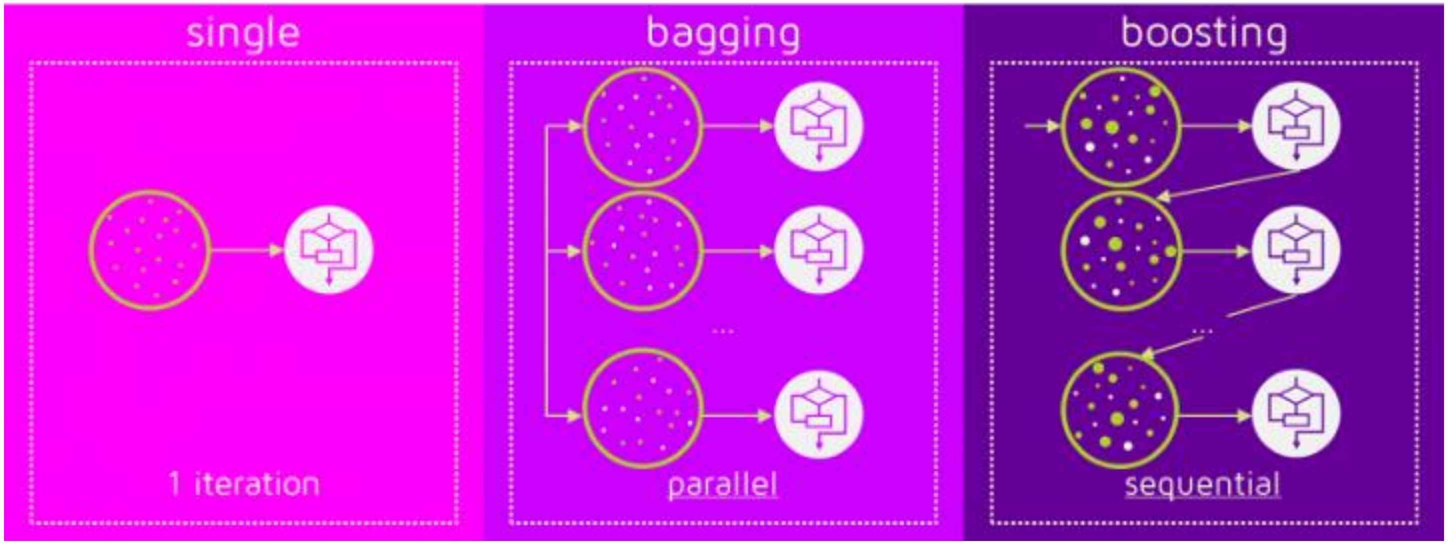
- Cons
  - High variance (overfitting)
  - Learning algorithms does not guarantee optimum tree
  - Small difference can result in significantly different tree
  - Ineffective or inefficient for complex tasks (e.g. XOR)
    - Boundaries are perpendicular to feature axes
    - Cannot analyze correlation between variables
  - Cannot extrapolate

Ensemble

- Bagging: parallel combination
  - Reduces variance

  Ex) Random forest
- Boosting: sequential combination
  - Primarily reduces bias, and also variance

  Ex) AdaBoost, gradient boosting (incl. XGBoost)

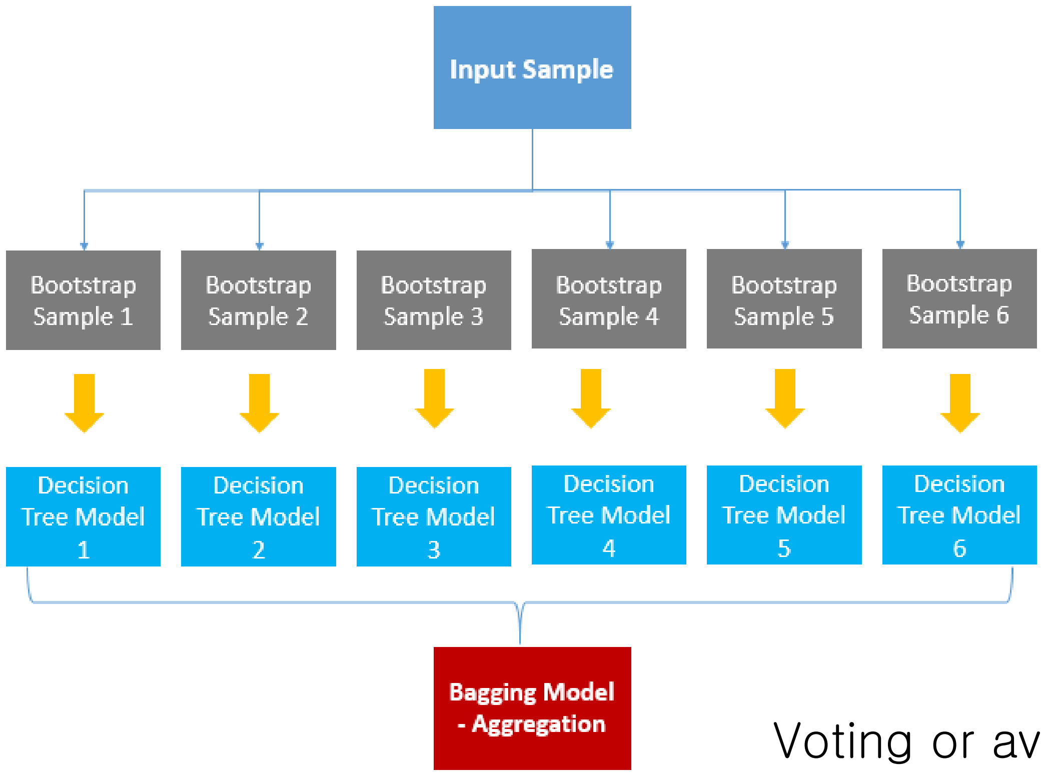


Agenda

- Decision Trees
- <u>Random Forests</u>
- AdaBoost
- Gradient Boost (incl. XGBoost)

Bagging

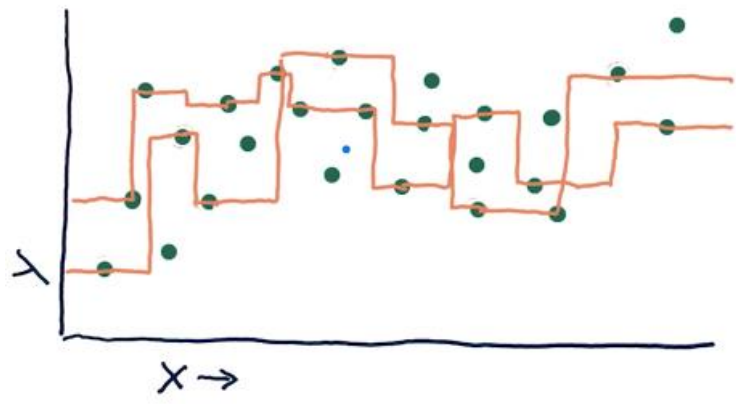
- Bagging (a.k.a. bootstrapping aggregation): An ensemble meta-algorithm designed to improve the stability and accuracy. Also reduces overfitting.



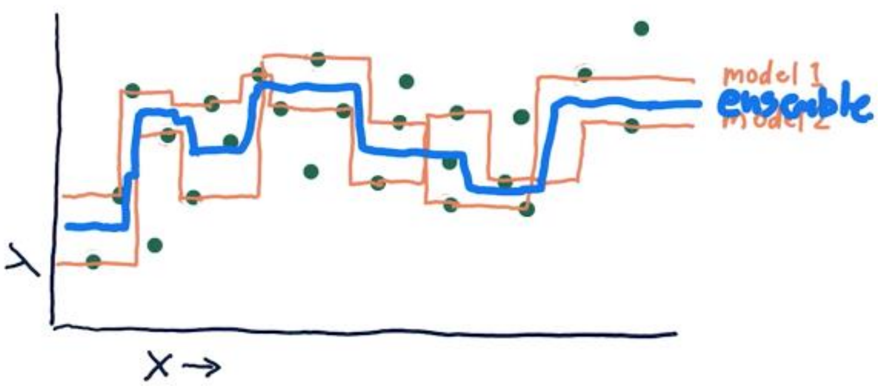
Voting or average

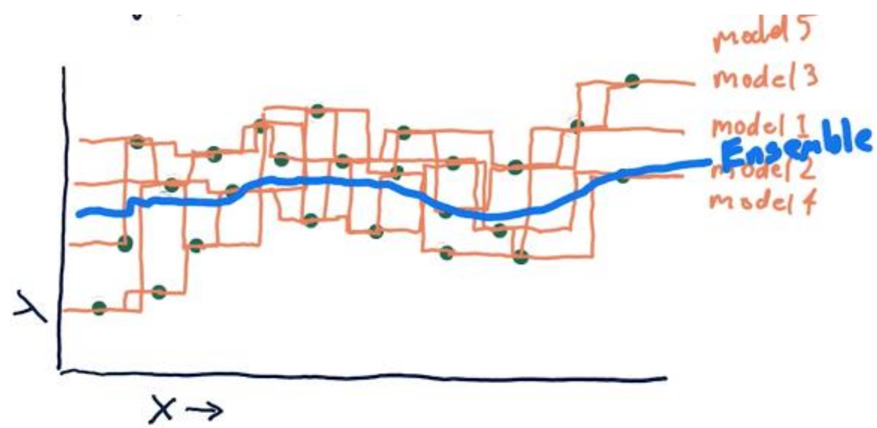
Bagging of Regression Model

- Learning by sampling

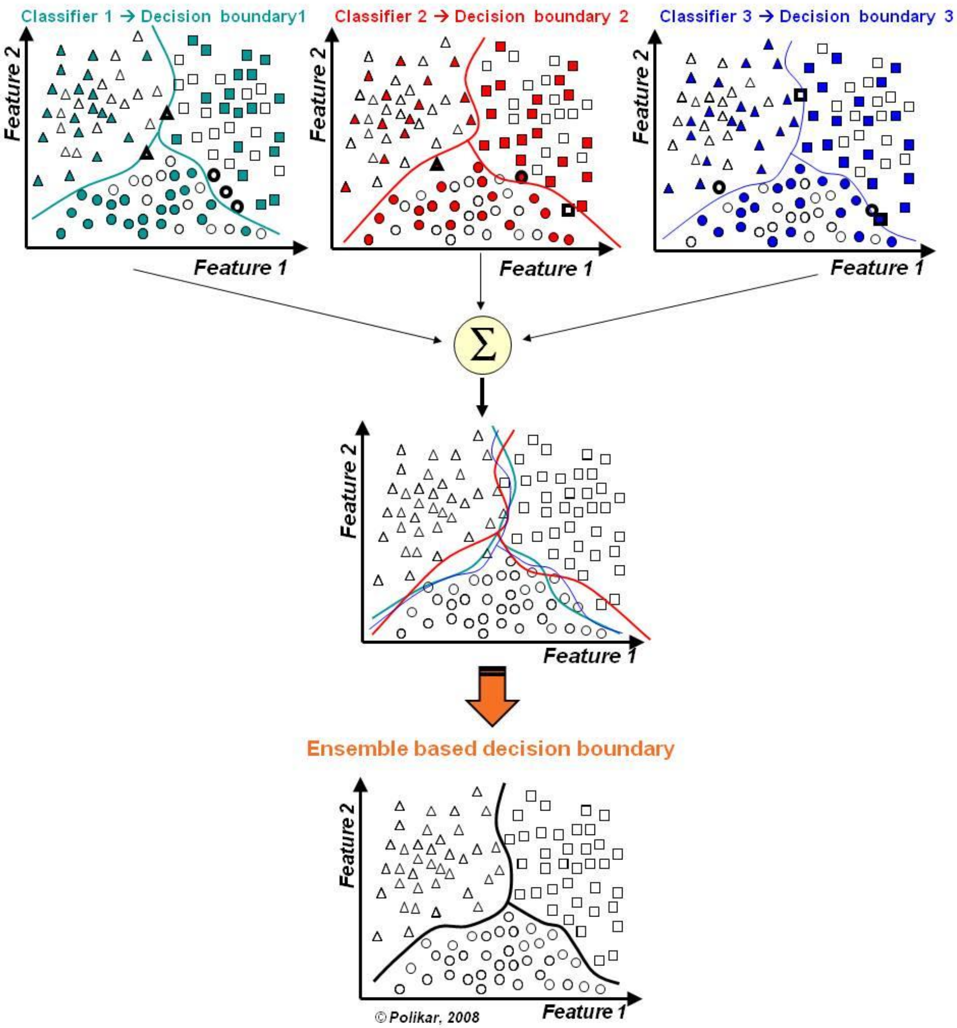


- Bagging





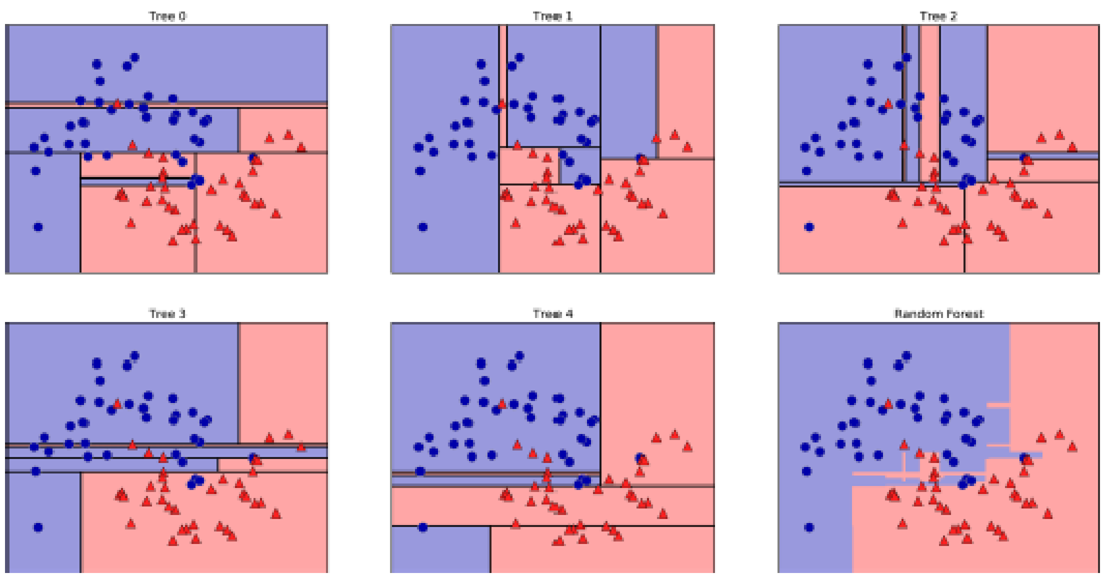
Bagging



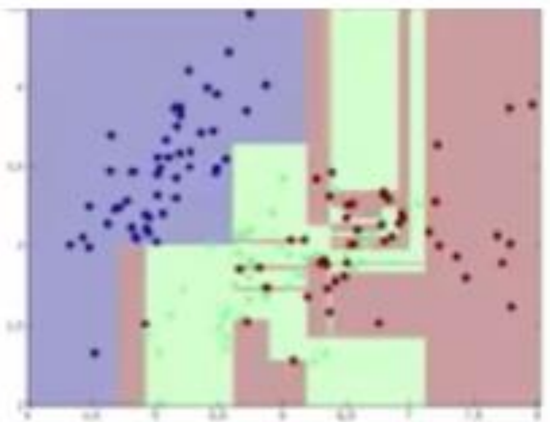
Classifier 1 → Decision boundary1
Classifier 2 → Decision boundary 2
Classifier 3 → Decision boundary 3
Feature 2
Feature 1
Σ
Ensemble based decision boundary
© Polikar, 2008

Random Forests

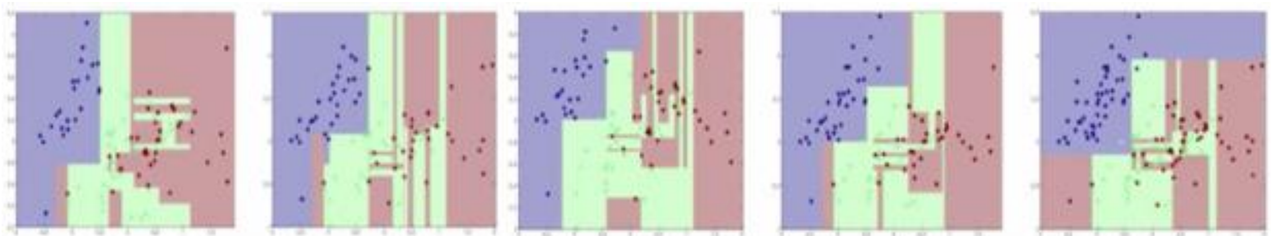
- Randomly generate multiple trees
  - From training dataset D , uniformly sample subsets D_i with replacement
  - Learn each tree T_i using D_i.
- Combine their decisions by bagging
  - $\hat{f}_{bag}(x) = \frac{1}{m}\sum_b \hat{f}_b(x)$



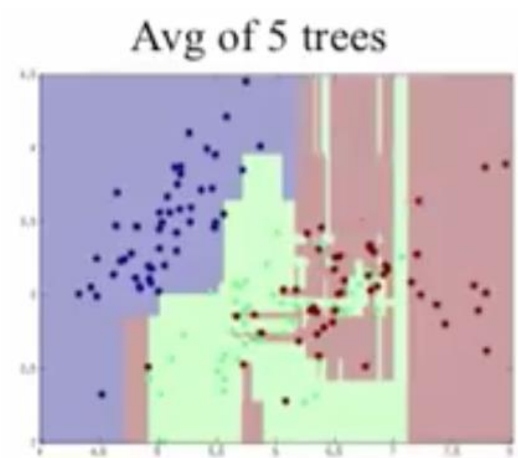
Random Forests

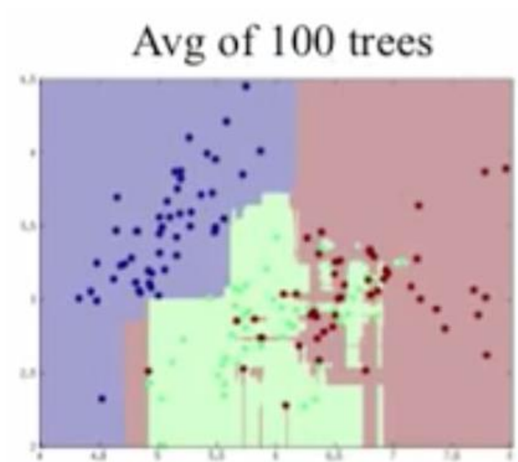


Trained on full dataset



Multiple trees learned from on sampled datasets





Random Forests

- How many trees in a Random Forest?
  - Authors: T. Oshiro, P. Perez, and J. Baranauskas
  - As the number of trees grows, it does not always mean the performance of the forest is significantly better than previous forests (fewer trees), and doubling the number of trees is worthless.
  - It is also possible to state there is a threshold beyond which there is no significant gain, unless a huge computational environment is available. In addition, it was found an experimental relationship for the AUC gain when doubling the number of trees in any forest.
  - Furthermore, as the number of trees grows, the full set of attributes tend to be used within a Random Forest, which may not be interesting in the biomedical domain.
  - Additionally, datasets' density-based metrics proposed here probably capture some aspects of the VC dimension on decision trees and low-density datasets may require large capacity machines whilst the opposite also seems to be true.

Agenda

- Decision Trees
- Random Forests
- AdaBoost
- Gradient Boost (incl. XGBoost)

Boosting

- Boosting
  - General ensemble method that creates a strong classifier from a number of weak classifiers (accuracy > 50%)

- Algorithm
  - Building a model from the training data
  - Then, creating a second model that attempts to correct the errors from the previous models.
  - Models are added until the training set is predicted perfectly or a maximum number of models are added.

Boosting Algorithms

알고리즘	특징	비고
AdaBoost	• 다수결을 통한 정답 분류 및 오답에 가중치 부여	
GBM	• Loss Function의 gradient를 통해 오답에 가중치 부여	gradient_boosting.pdf
Xgboost	• GBM 대비 성능향상 • 시스템 자원 효율적 활용 (CPU, Mem) • Kaggle을 통한 성능 검증 (많은 상위 랭커가 사용)	2014년 공개 boosting-algorithm-xgboost
Light GBM	• Xgboost 대비 성능향상 및 자원소모 최소화 • Xgboost가 처리하지 못하는 대용량 데이터 학습 가능 • Approximates the split (근사치의 분할)을 통한 성능 향상	2016년 공개 light-gbm-vs-xgboost

AdaBoost

- Adaptive Boosting
  - The first successful boosting algorithm that assigns weight to samples
- Algorithm

1. Initialize weights for training images
2. For T rounds

   2.1. Normalize the weights

   2.2. For available features from the set, <u>train a classifier using a single feature</u> and evaluate the training error

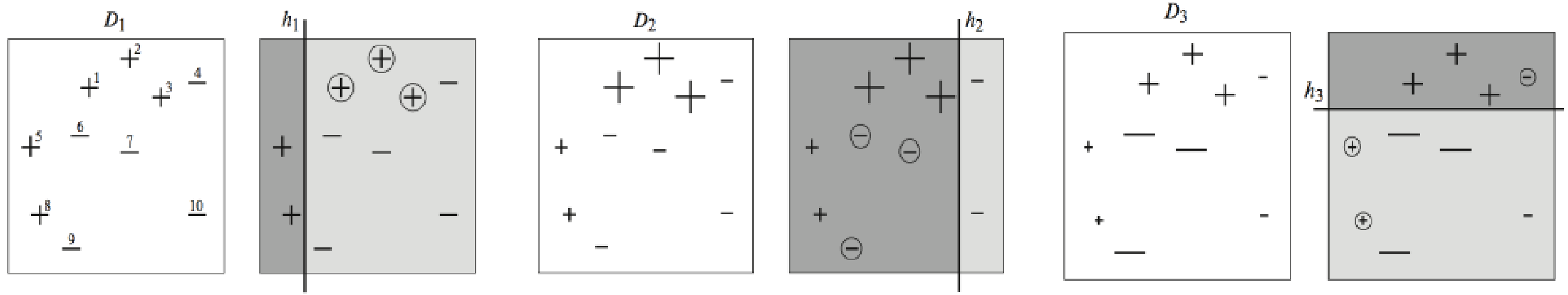
   2.3. Choose the classifier with the lowest error

   2.4. Update the weights of the training images:
   - Increase if classified wrongly by this classifier,
   - Decrease if correctly

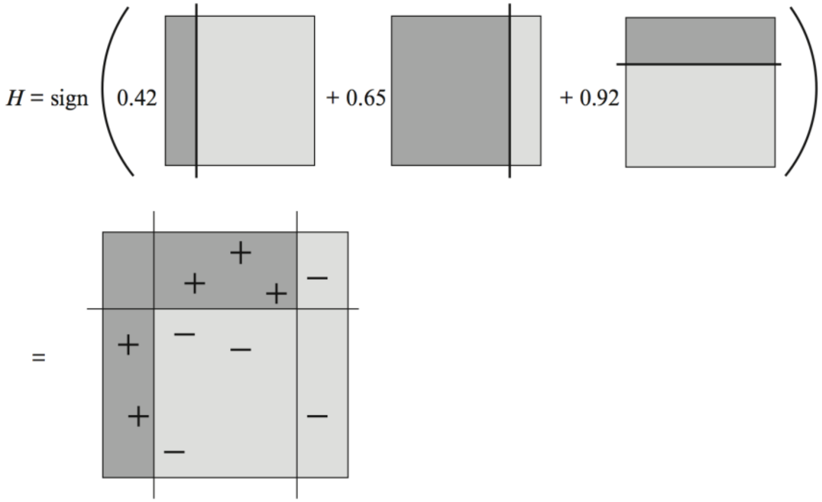
2. Form the final strong classifier as the linear combination of the T classifiers (coefficient larger if training error is small)

AdaBoost

- Learn tree sequentially
  - Each tries to correct previous error.



- Aggregation

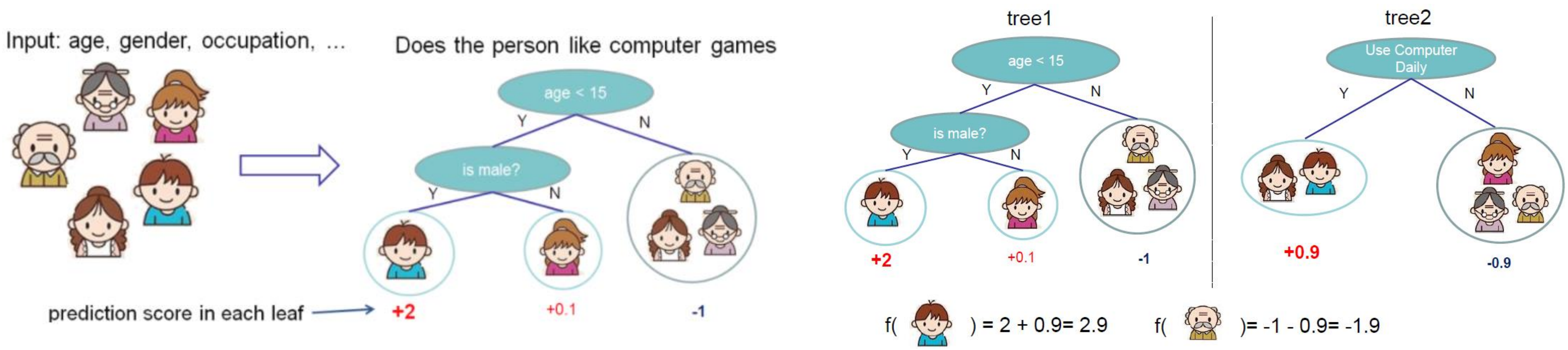


Agenda

- Decision Trees
- Random Forests
- AdaBoost
- <u>Gradient Boost (incl. XGBoost)</u>

Gradient Boosting (GBM)

- Prediction: $\hat{y}_i = \sum_k f_k(x_i)$,
  - $f_k(x_i)$: prediction of k^{th} tree for x_i
  - $f_k \in F = \{f(x) = w_{q(x)}\}$
  - q: structure of tree (mapping from x to leaf nodes)
  - w_i: score on i^{th} leaf node



Regularized Objective

- Objective function

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \quad \text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

- Objective for f_t

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

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

- 2nd order Taylor approximation

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \tfrac{1}{2} f''(x)\Delta x^2$$

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

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

Regularized Objective

- Instance set of a leaf j $\quad I_j = \{i | q(x_i) = j\}$

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

- Optimum score and loss
  - Can be obtained by setting derivative to zero

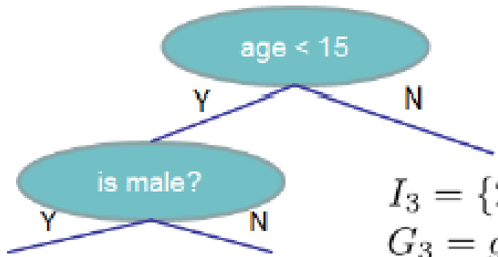
$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \qquad \tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

Regularized Objective

- The objective as a quality measurement of tree

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \qquad \tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

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

Greedy Split Algorithm

- Find a tree that minimizing the loss

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \qquad \tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T.$$

- Starting from a single leaf, iteratively adds branches to the tree
  - Splitting I into two nodes I_L and I_R, $(I = I_L \cup I_L)$
  - Then, loss reduction after split

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

Loss of left leaf

Loss of right leaf

Loss of original leaf

regularization

Preventing Overfitting

- Shrinkage: scaling newly added weights by a factor η after each step of tree boosting.
  - Similar to learning rate decay
  - Reduces the influence of each individual tree
    - Leaves space for future trees to improve the model

- Column (feature) subsampling
  - Additional speed up of computation

Configuring Gradient Boosting

- How to Configure the Gradient Boosting Algorithm
  - https://machinelearningmastery.com/configure-gradient-boosting-algorithm/

XGBoost

- XGBoost = eXtreme Gradient Boost
  - T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” Jun. 2016
- Extension of gradient boosting for better scalability
  - x10 times faster than existing popular solutions on a single machine
  - Scales to billions of examples in distributed or memory-limited settings

Split Finding

- Approximate algorithm for split finding
  - To summarize, the algorithm first proposes candidate splitting points according to percentiles of feature distribution
  - More efficient than the exact greedy algorithm

- Proposed methods
  - Weighted quantile

Weighted Quantile Sketch

- The loss function can be viewed as MSE weighted by h_i's

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

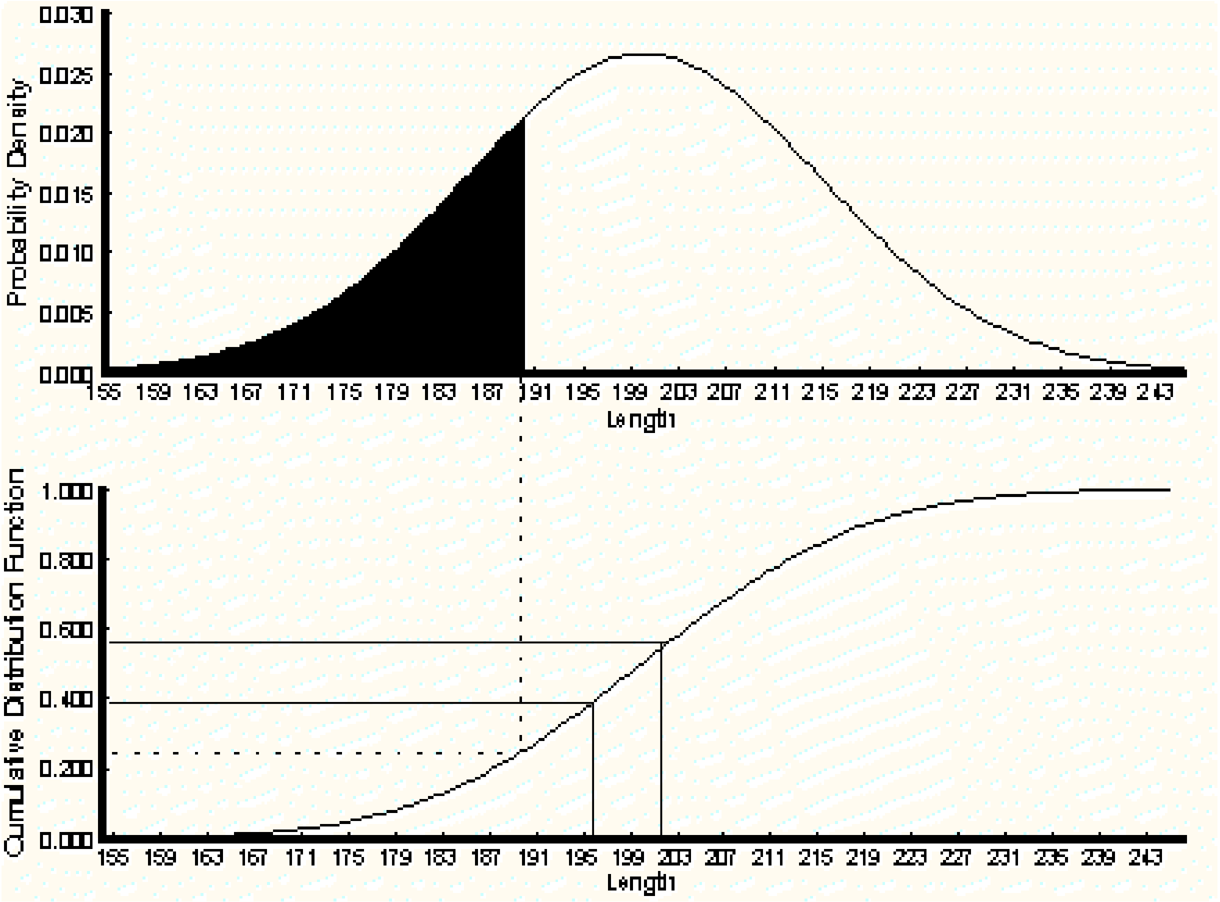
$$\sum_{i=1}^{n} \frac{1}{2} h_i (f_t(\mathbf{x}_i) - g_i/h_i)^2 + \Omega(f_t) + constant,$$

- Find candidate points $\{s_{k1}, s_{k2}, \dots, s_{kl}\}$
  - 2nd order gradients h_i at x_i as weights

$$r_k(z) = \frac{1}{\sum_{(x,h)\in\mathcal{D}_k} h} \sum_{(x,h)\in\mathcal{D}_k, x<z} h,$$

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \quad s_{k1} = \min_i \mathbf{x}_{ik}, s_{kl} = \max_i \mathbf{x}_{ik}.$$

Weighted Quantile Sketch



$r_k(s_{k,j+1})$

$r_k(s_{k,j})$

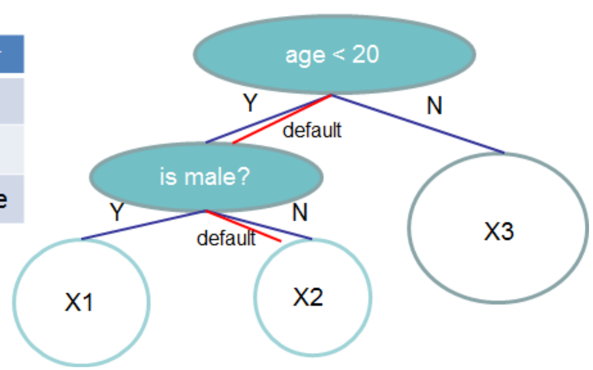
$s_{k,j}$ $s_{k,j+1}$

Sparsity-aware Split Finding

- In reality, data are often sparse
  - Missing values
  - Frequent zero entries in the statistics
  - Artifacts of feature engineering (e.g. one-hot encoding)
- Proposed method: adding default direction
  - Optimal default directions are learned from data
  - More than 50 times faster on Allstate-10K data

Data

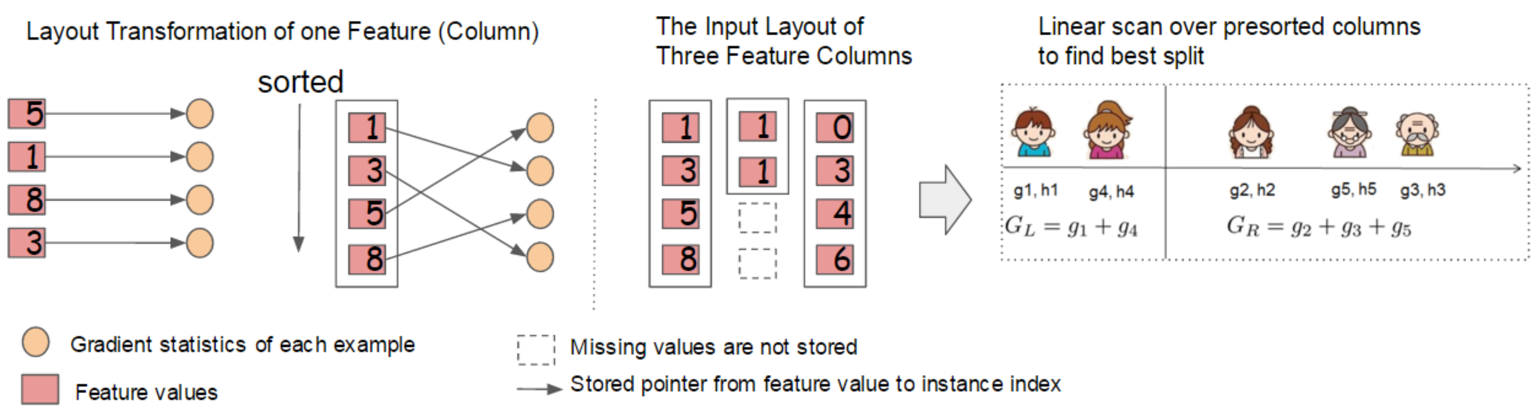
Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female



Implementation Techniques

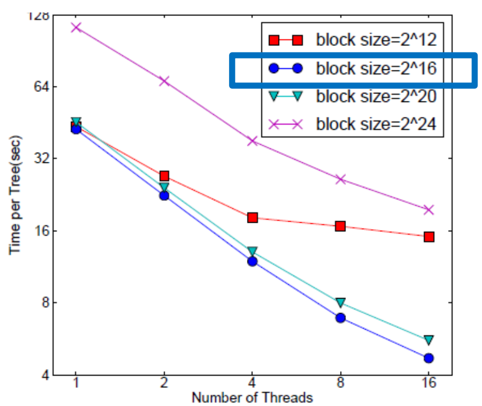
- Column block for parallel learning
  - Most time-consuming step is sorting
  - Store the data in in-memory units called blocks
  - Data in each block is sorted in CSC format
  - Exact greedy algorithm: entire dataset in a single block

    ➔ One scan of blocks can collect the statistics of split candidates in all leaf branches
  - Approximate algorithm: multiple blocks containing different subset of rows

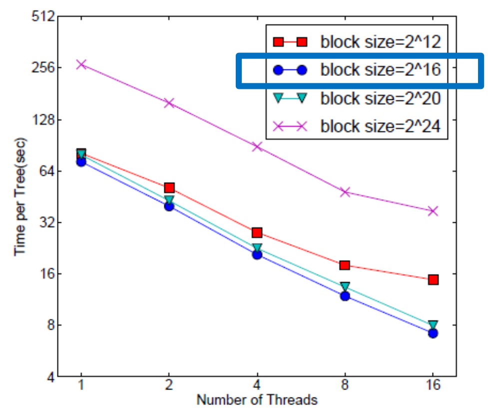


Implementation Techniques

- Cache-aware access
  - The block structures requires indirect fetches of gradient statistics (non-continuous memory access)
  - Exact greedy algorithm: cache-aware prefetching algorithm using internal bufffer (x2 faster)
  - Approximate algorithm: choosing block size to maximum # of examples in a cach block



(a) Allstate 10M



(b) Higgs 10M

Implementation Techniques

- Blocks for out-of-core computation
  - Divide the data into multiple blocks and store each block on disk
  - During computation, independent thread pre-fetch the block into memory
- Block compression
  - The block is compressed by columns
    - 26~29% compression ratio
  - Decompressed on the fly by an independent thread
- Block sharding
  - Shard data onto multiple disks in an alternative manner
  - Assign a pre-fetcher thread to each disk

Q&A

Thank you
for your attention!