大数据分析

Scalable Machine Learning
decision tree

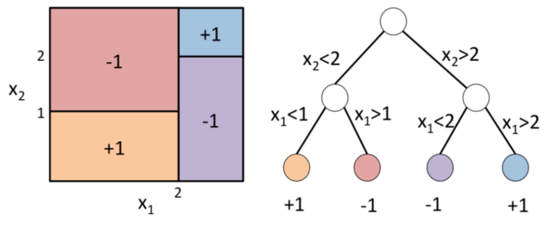
刘盛华

Outline

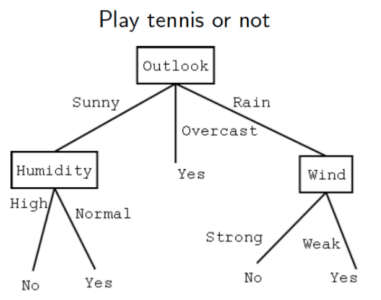
- **Decision Tree**
- **Random Forest**
- **Gradient Boosted Decision Tree (GBDT)**

Decision Tree

- **Each node checks one feature x_i :**
  - **Go left if x_i < threshold**
  - **Go right if $x_i \geq$ threshold**



A real example



Decision Tree

- Strength:
  - it's a nonlinear classifier
  - Better interpretability
  - Can naturally handle categorical features

非线性
更好的解释性
自然处理分类特征

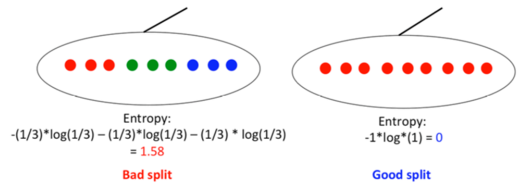
- Computation:
  - Training: slow
  - Prediction: fast
  - *h* operations (h: depth of the tree, usually ≤ 15)

Splitting the node

- Classification tree: Split the node to maximize entropy
- Let S be set of data points in a node, *c = 1,⋯ ,C* are labels:

$$\text{Entroy}: H(S) = -\sum_{c=1}^{C} p(c)\log p(c),$$

- where p(c) is the proportion of the data belong to class c.
  - Entropy=0 if all samples are in the same class
  - Entropy is large if p(1) = ⋯ = p(C)



Information Gain

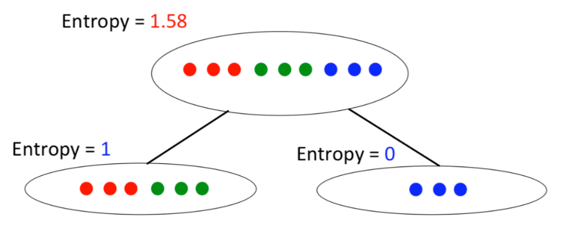
- The averaged entropy of a split S → S1, S2

$$\frac{|S_1|}{|S|}H(S_1) + \frac{|S_2|}{|S|}H(S_2)$$

- Information gain: measure how good is the split

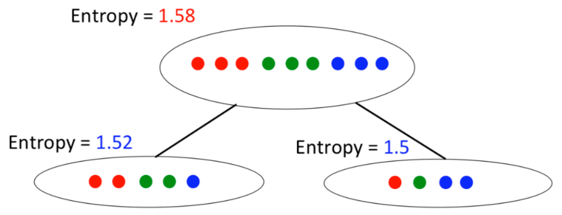
$$H(S) - \left((|S_1|/|S|)H(S_1) + (|S_2|/|S|)H(S_2)\right)$$

Information Gain



Averaged entropy: 2/3*1 + 1/3*0 = 0.67
Information gain: 1.58 – 0.67 = 0.91

Information Gain



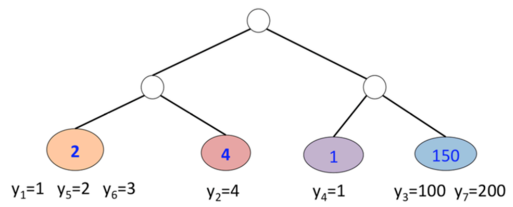
Averaged entropy: 1.51
Information gain: 1.58 – 1.51 = 0.07

Splitting the node

- **Given the current note, how to find the best split?**
- **For all the features and all the threshold**
  - **Compute the information gain after the split**
  - **Choose the best one (maximal information gain)**
- **For *n* samples and *d* features: need O(*nd*) time**

Regression Tree

- **Assign a real number for each leaf**
- **Usually averaged *y* values for each leaf (minimize square error)**



Regression Tree

Objective function:

$$\min_F \frac{1}{n}\sum_{i=1}^{n}(y_i - F(\boldsymbol{x}_i))^2 + (\text{Regularization})$$

The quality of partition $S = S_1 \cup S_2$ can be computed by the objective function:

$$\sum_{i\in S_1}(y_i - y^{(1)})^2 + \sum_{i\in S_2}(y_i - y^{(2)})^2,$$

where $y^{(1)} = \frac{1}{|S_1|}\sum_{i\in S_1} y_i$, $y^{(2)} = \frac{1}{|S_2|}\sum_{i\in S_2} y_i$

Find the best split:

Try all the features & thresholds and find the one with minimal objective function

Parameters

- Maximum depth: (usually ~ 10)
- Minimum number of nodes in each node: (10, 50, 100)
- Single decision tree is not very powerful· · ·
- Can we build multiple decision trees and ensemble them together?

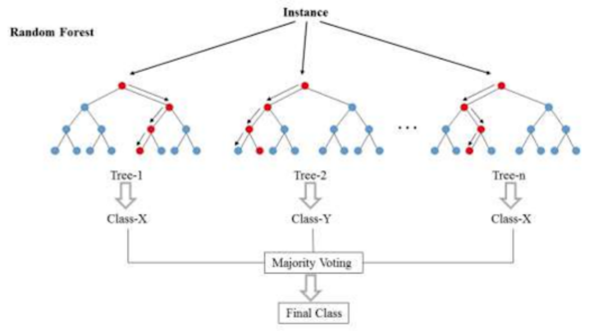
Outline

- Decision Tree
- Random Forest
- Gradient Boosted Decision Tree (GBDT)

Random Forest

- Random Forest (Bootstrap ensemble for decision trees):
  - Create *T* trees
  - Learn each tree using a subsampled dataset S_i and subsampled feature set D_i
  - Prediction: Average the results from all the *T* trees
- Benefit:
  - Avoid over-fitting
  - Improve stability and accuracy
- Good software available:
  - R: “randomForest” package Python: sklearn

An example



Building Decision Trees using MapReduce

- **Parallel Learner for Assembling Numerous Ensemble Trees [Panda et al., VLDB '09]**
  - **A sequence of MapReduce jobs that builds a decision tree**
  - **Spark MLlib Decision Trees are based on PLANET**

Outline

- **Decision Tree**
- **Random Forest**
- **Gradient Boosted Decision Tree (GBDT)**

Boosted Decision Tree

- Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees

$$F^* = \operatorname*{argmin}_F \sum_{i=1}^{n} \ell(\mathbf{y}_i, F(\boldsymbol{x}_i)) \quad \textbf{with} \quad F(\boldsymbol{x}) = \sum_{m=1}^{T} f_m(\boldsymbol{x})$$

(each f_m is a decision tree)

- Direct loss minimization: at each stage m, find the best function to minimize loss
  - solve $f_m = \operatorname{argmin}_{f_m} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(\boldsymbol{x}_i) + f_m(\boldsymbol{x}_i))$
  - update $F_m \leftarrow F_{m-1} + f_m$

  $F_m(\boldsymbol{x}) = \sum_{j=1}^{m} f_j(\boldsymbol{x})$ is the prediction of $\boldsymbol{x}$ after m iterations.
- **Two problems:**
  - **Hard to implement for general loss**
  - **Tend to overfit training data**

Gradient Boosted Decision Tree (GBDT)

- **Approximate the current loss function by a quadratic approximation:**
  用二次逼近法逼近电流损失函数:

$$\sum_{i=1}^{n} \ell_i(\hat{y}_i\,, \quad f_m(\boldsymbol{x}_i)) \approx \sum_{i=1}^{n} \left(\ell_i(\hat{y}_i) - g_i f_m(\boldsymbol{x}_i) + \frac{1}{2} h_i f_m(\boldsymbol{x}_i)^2\right)$$

residual = actual - prediction

$$= \sum_{i=1}^{n} \frac{h_i}{2} \| f_m(\boldsymbol{x}_i) - g_i / h_i \|^2 + \text{constant}$$

where $g_i = \partial_{\hat{y}_i} \ell_i(\hat{y}_i)$ is gradient,
$h_i = \partial^2_{\hat{y}_i} \ell_i(\hat{y}_i)$ is second order derivative

Gradient boosting (Freidman 1999)

Details

Gradient boosting (Friedman 1999) approximately solves (3) for arbitrary (differentiable) loss functions $\Psi(y, F(\mathbf{x}))$ with a two step procedure. First, the function $h(\mathbf{x}; \mathbf{a})$ is fit by least–squares

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\rho} \sum_{i=1}^{N} [\tilde{y}_{im} - \rho h(\mathbf{x}_i; \mathbf{a})]^2 \tag{5}$$

to the current "pseudo"–residuals

$$\tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}. \tag{6}$$

Then, given $h(\mathbf{x}; \mathbf{a}_m)$, the optimal value of the coefficient β_m is determined

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} \Psi\left(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a}_m)\right). \tag{7}$$

This strategy replaces a potentially difficult function optimization problem (3) by one based on least–squares (5), followed by a single parameter optimization (7) based on the general loss criterion Ψ.

Gradient Boosted Decision Tree (GBDT)

- Finding $f_m(\boldsymbol{x}, \theta_m)$ by minimizing the loss function:

$$\boxed{\underset{f_m}{\text{argmin}} \sum_{i=1}^{N} [f_m(\boldsymbol{x}_i, \theta) - g_i/h_i]^2 + R(f_m)}$$

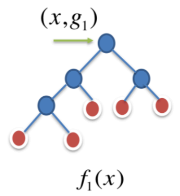
  - reduce the training of any loss function to regression tree (just need to compute g_i for different functions)
  - $h_i = \alpha$ (fixed step size) for original GBDT.
  - XGboost shows computing second order derivative yields better performance
- **Algorithm:**

  Computing the current gradient for each $\hat{y}_i$.
  Building a base learner (decision tree) to fit the gradient.
  Updating current prediction $\hat{y}_i = F_m(\boldsymbol{x}_i)$ for all i.

Gradient Boosted Decision Tree (GBDT)

Key idea:

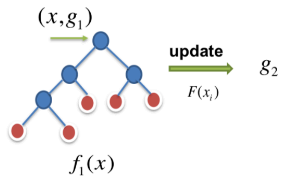
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$



Gradient Boosted Decision Tree (GBDT)

Key idea:

- Each base learner is a decision tree
- Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$

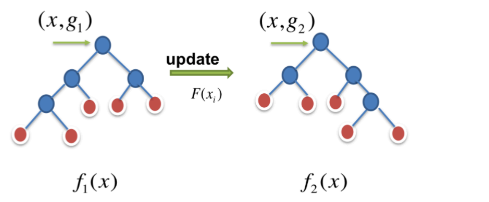


$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \qquad g_m(x_i) = \left.\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)}\right|_{F(x_i)=F_{m-1}(x_i)}$$

Gradient Boosted Decision Tree (GBDT)

Key idea:

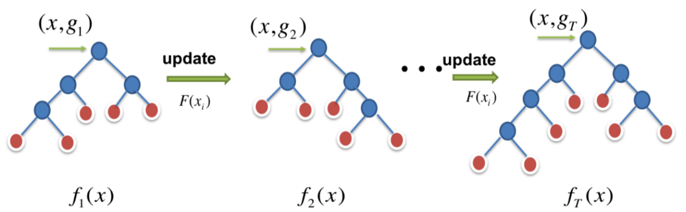
- Each base learner is a decision tree
- Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$



$F_{m-1}(x_i)=\sum_{j=1}^{m-1} f_j(x_i)$ $\quad g_m(x_i)=\frac{\partial \ell(y_i,F(x_i))}{\partial F(x_i)}\Big|_{F(x_i)=F_{m-1}(x_i)}$

Gradient Boosted Decision Tree (GBDT)

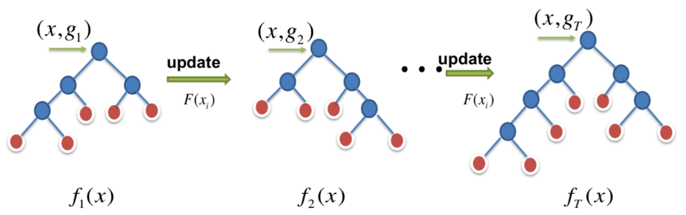
- Key idea:
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$



$F_{m-1}(x_i)=\sum_{j=1}^{m-1} f_j(x_i)$ $\quad g_m(x_i)=\frac{\partial \ell(y_i,F(x_i))}{\partial F(x_i)}\Big|_{F(x_i)=F_{m-1}(x_i)}$

Gradient Boosted Decision Tree (GBDT)

- Key idea:
  - Each base learner is a decision tree
  - Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial f}$



Final prediction $F(x_i)=\sum_{j=1}^{T} f_j(x_i)$

Questions?