Notes of Tree Models (I)

张春光

September 24, 2018

1 DecisionTree

1. 基本流程

   决策树的生成是递归产生叶结点的过程，而产生叶结点主要分为三种情况:

   (1) 当前节点包含的样本全部属于同一类别;

   (2) 当前属性值为空，没有其他属性进行划分，归类为结点中数目最多所属类别;

   (3) 当前节点包含样本为空，则按照父节点的样本分布作为该结点样本分布;

Decision Tree Pseudo Code

Input: Train dataset $D = \{(\mathbf{x}_i,\, y_i)\,|\,i = 1, 2, \ldots, n;\, y_i = 1, 2, ..., k\}$;
Attributes set $A = \{A_1, A_2, \ldots, A_m\}$;
Threshold ϵ.

Procedure: TreeGenerate(D, A)

- Generate a note N;
- **if** all instance in D belong to a class j **then**
  Label node N as Leaf Node of class j; **return**
  end if
- **if** $A = \emptyset$ or values of $\mathbf{x}_i's$ in D are same **then**
  Label node N as Leaf Node of the class which has the most instance in D; **return**
  end if
- Select a optimal partitioning attributes A_*.(**ID3, C4.5, Cart**);
  if $Gain(D,\, A_*) < \epsilon$; $ORGain\_RatioOrGini\_Index$; **return**
  end if
  for value a_*^v in A_*:
  Generate branch for N; denote D_v as subsample of D which have value a_*^v;
  if D_v is empty **then**
  Label node N as Leaf Node of class j; **return**
  else
  Denote TreeGenerate(D_v, $A \backslash a_*^v$) as child node;
  end if
  end for

Output: An Decision Tree

从上述伪代码中可以看出，决策树的主要学习内容为属性划分选择，但在实际应用中若决策树结点过多或多少往往会产生过拟合或拟合不足情况，因此较好的决策树需要进行剪枝处理;

2. 属性划分

   决策树的节点优劣常用纯度划分，因此节点属性划分通常以纯度提高来进行衡量，目前纯度常用的衡量指标包括信息增益、信息增益率以及基尼指数；对这三个指标来说，信息增益、增益率指标数值越高，节点属性划分的纯度提升越大，基尼指数则相反，但这三个指标从原理上来看对于指标选取都是有偏的，其他指标如 QUEST 算法采用 ANOVA 选取 p 值最小的属性进行划分;

(1) 信息熵–倾向多值属性

假设一节点 N 包含 K 类样本, 记 p_k 为第 k 类样本的比例，则节点 N 的信息熵为:

$$Ent(N) = -\sum_{k=1}^{K} p_k log_2 p_k$$

假定离散属性 a 有 V 个不同取值, 记 N^v 为 N 中属性 a 上取值为 a^v 的样本，记 $|S|$ 为样本 S 的容量，则采取属性 a 进行划分的信息增益为:

$$Gain(N, a) = Ent(N) - \sum_{v=1}^{V} \frac{|N^v|}{|N|} Ent(N^v)$$

从信息增益准则公式可以看出**当样本量不充分大**时，若一个划分属性的取值数目越多，则信息增益会越小，因此信息增益偏好属性取值较多的属性进行划分。

(2) 信息增益率–倾向不平衡属性

为消除信息增益准则的偏好性，衍生的 $C4.5$ 算法使用信息增益率为基础计算纯度的提升.

$$Gain\_ratio(N, a) = \frac{Gain(N, a)}{SplitInformation(N, a)}$$

$$SplitInformation(N, a) = -\sum_{v=1}^{V} \frac{|N^v|}{|N|} log_2 \frac{|N^v|}{|N|}$$

其中 SplitInformation(N, a) 称为分离信息，一般随着属性 a 取值个数增多而变大。**C4.5 算法并不直接使用信息增益率来选取属性划分，而是先取信息增益高于平均信息增益的属性，再从其中选择增益率高的属性**.

(3) 基尼指数–偏向多值属性

CART(Classification and Regression Tree) 可用于分类和回归树, 使用基尼指数来选择属性，节点属性衡量方法:

$$Gini\_index(N, a) = \sum_{v=1}^{V} \frac{|N^v|}{|N|} Gini(N^v)$$

$$Gini(N^v) = -\sum_{v=1}^{V} \frac{|N^v|}{|N|} log_2 \frac{|N^v|}{|N|}$$

(4) 其他如 $TwoingValue = \frac{|T_L|*|T_R|}{n^2} * (\sum_{v=1}^{V} |\frac{L_i}{|T_L|} - \frac{L_i}{|T_R|}|)^2$ 等;

(5) 连续属性

信息熵与基尼指数都是基于离散变量进行计算，而当碰到连续属性时，模型先进行连续数据离散化，$ID3$ 决策树无法对连续性数据构造决策书，而 $C4.5$ 则基于二分类方法对连续属性进行处理:

- 对属性特征进行排序;
- 对于每个特征，有 V 个特征值，则有 V-1 个中点值可将数据分为两部分，计算每个中值进行分裂的信息增益;
- 选择修正信息增益最大的分裂点作为该属性的划分点对数据划分;

$$Gain(N, a) = max_{t \in T_a} Gain(N, a, t)$$

$$= max_{t \in T_a} (Ent(N) - \sum_{\lambda \in \{-,+\}} \frac{|N_t^\lambda|}{|N|} Ent(N_t^\lambda))$$

$$T_a = \{\frac{a^i + a^{i+1}}{2} | 1 \le i \le V - 1\}$$

$$GainRevise(N, a) = Gain(N, a) - \frac{log_2(V-1)}{|N|}$$

3. 缺失值处理

   决策树在学习过程中对缺失值的处理主要从以下两方面:

   - 属性选择基于每个属性的无缺失样本 $\bar{N}$ 计算信息增益或基尼系数，而后乘以无缺失比例:

$$Gain(N, a) = \rho \times Gain(\bar{N}, a), \quad \rho = \frac{|\bar{N}|}{|N|}$$

   - 样本划分在基于无缺失的样本属性选择之后，将每个缺失样本按照无缺失样本的后验概率的权重划分入各个子节点中。

4. 多变量组合属性划分

   多变量决策树对于非叶结点划分时是对属性的线性组合进行测试，试图在每次属性划分时建立一个线性分类器。多变量决策树中较为常用的是 OC1 算法 (Oblique Classifier 1); 大致思想如下:

   (1) Coefficients Learning

   - CART Linear Combination:
     (1) 先基于单变量选择最优属性 $\mathbf{x}_{j.}$，建立多变量组合分割平面 H:$\sum_{i=1}^{m} a_i x_i + a_{m+1} = 0$
     (2) 每次基于当前结点数据优化一个 $\sum_{i=1}^{m} a_i x_i - \delta(x_i + \gamma) + a_{m+1} <= 0(>= 0)$, 寻求最优 a_i, a_{m+1} 使得当前结点内的样本分类正确并更新;
     (3) Randomization; 因为在步骤 (2) 中每次更新 a_i 都是局部更新，因此在得到变量组合分割的超平面之后会进行扰动优化，生成随机向量 $(r_1, r_2, \ldots, r_{m+1})$ 加入超平面使得 $H_1 = \sum_{i=1}^{m}(a_i + \alpha r_i)x_i + (a_{m+1} + r_{m+1})$ 再次优化;
   - Recursive Least Square procedure

$$\begin{aligned} W_k &= W_{k-1} - K_k(X_k^T W_{k-1} - y_k) \\ K_k &= P_k K_k \\ P_k &= P_{k-1} - P_{k-1} X_k [1 + X_k^T P_{k-1} X_k]^{-1} X_k^T P_{k-1} \end{aligned}$$

   (2) Feature Selection

   特征选取和回归中的 stepwise 类似采取 SBE(Sequential Backward Elimination) 和 SFE(Sequential Forward Selection) 等, 以 SBE 为例，对具有 n 个属性的组合寻求最优参数, 接下来依次删除其中一个属性，选择精度提高最多的一个属性进行剔除，以此类推，再用独立训练集验证;

5. 剪枝处理机器学习分类方法目的主要是为了提取某一类型样本的一般性特征，提高模型泛化识别能力，而非单一样例的个别特征，因此当决策书模型过深时，所产生的树模型往往会学习到个别特征，因此树模型一般需要进行剪枝处理来防止过拟合现象。

   (1) 预剪枝 (阈值)

   预剪枝即在模型划分前对结点进行估计计算衡量指标，若不能提升则停止进行划分并标记为叶结点；预剪枝的优点在于模型训练的成本降低，但决策树训练时很多分支未展开导致存在欠拟合现象。预剪枝一般有以下几种形式:

   - 设定决策树最大深度;
   - 节点中的样本个数小于阈值;
   - 结点分裂的信息增益小于阈值;

   (2) 后剪枝 (损失函数)

   后剪枝即在模型训练生成完整一决策树时从下而上，对非叶结点进行替换为叶结点计算模型的泛化能力，若有提升则将将该结点替换为叶结点进行。从原理上来看，预剪枝寻求局部优化，而后剪枝以成本开销为代价寻求全局优化。预剪枝一般在模型构建中使用阈值来处理，而后剪枝一般使用树损失函数来进行剪枝;

6. 三种决策树算法决策树算法主要分为二叉树与多叉树模型，前者代表有 CART 和 QUEST 算法，后者有 FACT、C4.5、CHAID、FRIM 算法等，这里主要介绍 ID3、C4.5 及 CART 算法;

   (1) ID3 决策树

   最早的决策树类型，采用信息增益指标进行评估和特征选择，也较为粗糙。

   - 只能处理离散属性，无法处理连续属性;

- 只生成树，未进行后剪枝处理;
- 信息增益准则偏好取值数目较多的属性;
- 适用于小规模数据集。

(2) C4.5 决策树

C4.5 决策树继承 ID3 决策树并进行改进:

- 结合信息增益和信息增益率进行属性划分选择，避免偏好取值多的属性;
- 使用二分法对连续属性离散化处理，对离散属性则采用多分法;
- 有对缺失值进行处理;
- 可以在树构造过程中进行剪枝 (信息增益率阈值等)，采用悲观剪枝法 (Pessimistic Error Pruning);
  * 对任意结点 t, 假设结点的样本个数为 $N(t)$, 该结点根据最优属性划分为 N_t;
  * 计算将结点 t 作为叶结点的错误率 $error_t = n(t) + \frac{1}{2}$;
  * 计算将结点作为内部结点的错误率 $error_{T_t} = n(T_t) + \frac{N_t}{2}$ 以及标准差 $SE = \sqrt{\frac{n(T_t)(N(t)-n(T_t))}{N(t)}}$
  * 若 $error_{T_t} + SE - error_t > 0$，则将结点 t 作为叶结点，反之不剪枝;
- **由于在构造树时需要对数据的属性进行多次扫描和排序，占内存、低效**

目前基于 C4.5 的基础上产生了 C5.0 算法，可应用于大数据集，采用 Boosting 方式提高模型准确率，又成为 BoostingTrees, 占用内存少，速度较快；但其本质上不是单决策树而是基于决策树的集成方法；FIRM 算法则是在 CHAID 算法上每个连续性属性划分为 10 个属性在进行树的生成, 但不进行剪枝;

(3) CART 分类树

CART 决策树为分类与回归树，假定决策树是二叉树，内部结点特征为是和否，递归二分每个特征，在进行回归时使用叶结点的样本的均值输出预测值;

- 使用 GINI 系数进行属性划分;
- 对于连续属性采用多次二分法，离散属性划分时每次选择其中一种分割数据，对属性特征重复性使用;
- 对于多分类目标，会将目标类别合并成两个超类别;
- 对于树的构建，CART 算法会生成尽量大的数目，而后用验证数据集根据损失函数对已生成的树进行剪枝，选择最优子树;
- 采用独立的验证集剪枝处理，否则剪枝结果为深度最大的子树;
  * 对整棵树分别计算以内部结点 t 为单结点的树（t）和根节点的树（T_t）的损失函数

$$C_\alpha(t) = C(t) + \alpha, \quad C_\alpha(T_t) = C(T_t) + \alpha|T_t|$$

  * 当且仅当 $C_\alpha(t) < C_\alpha(T_t)$ 时，决策树会进行剪枝剪除 T_t
  * 令 $g(t) = \frac{C(t)-C(T_t)}{|T_t|-1}$, 计算所有内部结点 t 的 g(t) 值，依次增加 α, 自下而上地在 T_i 中剪除 g(t) 最小的 $T_(t)$ 得到 T_{i+1} 子树
  * 利用独立测试集计算独立子树的损失函数，取最优子树.

$$\begin{aligned} C_\alpha(T) &= C(T) + \alpha\|T\| \\ &= \sum_{t=1}^{|T|} N_t H_t + \alpha\|T\| \\ H_t &= -\sum_k \frac{N_{tk}}{N_t} log \frac{N_{tk}}{N_t} \end{aligned}$$

α 为树模型复杂度的惩罚系数, $\|T\|$ 为树的叶结点数目, N_t 为叶结点 t 上的样本数，N_{tk} 为叶结点 t 上为类 k 的样本数；H_t 为叶结点 t 上的经验熵;

CART 回归树主要先将输入空间划分为 M 个空间 $R_1, R_2, \ldots, R_M$, 每个空间对应输出值 $c_1, c_2, \ldots, c_M$，而在对属性 A 选择和切分点 v 时不是选择信息增益或基尼指数，而是最小化:

$$min_{A,s}[min_{c_1} \sum_{x_i \in \{x|x^A \le s\}} (y_i - c_1)^2 + min_{c_2} \sum_{x_i \in \{x|x^A \ge s\}} (y_i - c_2)^2]$$

. 其中 c_i 在平方误差损失函数下一般取值为 R_i 区间内样本的均值;

(4) Scikitlearn 模块中使用的决策树算法为 CART 算法；

$DecisionTreeClassifier(criterion =' gini', splitter =' best', max\_depth = None,$
$min\_samples\_split = 2, min\_samples\_leaf = 1, min\_weight\_fraction\_leaf,$
$max\_features = None, random\_state = None, max\_leaf\_nodes = None,$
$min\_impurity\_decrease = 0.0, min\_impurity\_split = None,$
$class\_weight = None, presort = False)$

2 Bagging

2.1 RandomTreesEmbedding

$$\begin{aligned} DatasetD &\Rightarrow Bootstrapmsubdataset(D_1, D_2, \ldots, D_m) \\ &\Rightarrow TrainmlearnerI_i(x_i) \\ &\Rightarrow H(x) = argmax \sum_{m=1}^{M} I(I_t(x) = y) \end{aligned}$$

Bagging 结合 Bootstrap，使得模型有部分验证集对组合学习器进行包外估计，可以起到降低方差的效果，不易受干扰.

2.2 RandomForest

随机森林在 RandomTreesEmbedding 上基础上进行改进，在构建每克决策树过程中引入**随机属性选择**. 传统决策树在选择属性划分时候对当前结点的属性集合 (d) 中选择最优属性，而在随机森林中，先对当前结点的属性集合中随机选择 k(一般 $k = log_2 d$) 个属性子集，然后再从属性子集中选择最优属性。

2.3 StackingStrategy

Average, WeightedAverage, Majority Vote, MetaLearnerStacking,...

3 Boosting

(1) Boosting 基本流程:

- 从初始训练集用初始权重训练出一个基学习器 h_1；
- 根据基学习器的学习误差率 ϵ 表现更新训练样本权值，使得误差率高的训练样本权值 W_i 增加;
- 基于新的样本权值训练基学习器 h_2;
- 迭代进行，直到基学习器数目到达预设数目 T，然后根据基学习器的权重 α_i 组合基学习器的训练结果输出；

所以不同的 Boosting 方法的区别在于以下几点:

- 学习误差率的计算方式 Loss function；
- 如何根据误差率更新样本权重系数 W;
- 如何确定基学习器的权重系数（如何组合基学习器）;

针对大部分 Boosting 算法，基本都是采用加法模型来组合弱分类器，用前向分布算法来改变训练数据的权重和概率分布;

(2) 加法模型 Additive model

- 考虑由 K 棵树组成加法模型：

$$\hat{y} = \sum_{k=1}^{K} \beta_k f(x)$$

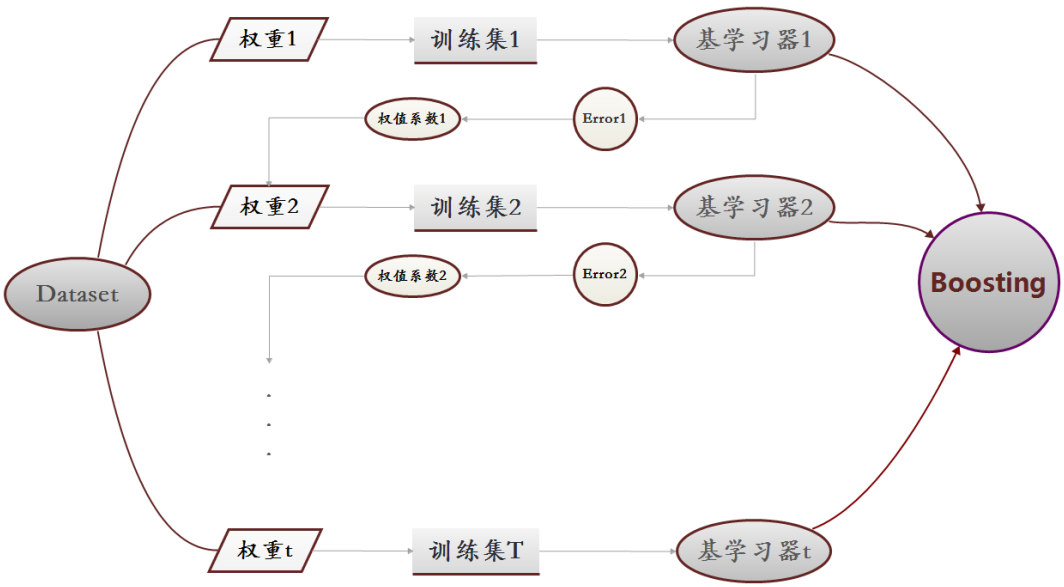

Figure 1: Boost

- 解决这一优化问题可采用前向分布算法从前往后每次只学习一个基函数及其系数，逐步逼近优化目标函数来简化复杂度，即

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

- 因此，最优化目标函数就转化为基于前一学习器的预测来最优化当前损失函数，即最小化 $\sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t)$；
- 以平方损失函数为例，则模型转为拟合上一轮的残差

$$\sum_{i=1}^{n} (y_i - \hat{y}_i^{t-1} + f_t(x_i))^2 + \Omega(f_t) = \sum_{i=1}^{n} (\epsilon_i - f_i(x))^2 + \Omega(f_t)$$

- 而损失函数的形式及其泰勒展开项数的不同产生就衍生出不同的 Boosting 算法；

3.1 AdaBoost

3.1.1 AdaBoost

AdaBoost 采用损失函数为最小化指数损失函数 $l_{exp}(H|D)$ 的线性 [权重] 加法**前向分布算法**模型输出 H(x)，假定 f 为真实分布，即

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$$

$$l_{exp}(H|W) = E_{\mathbf{x}|W}[exp(-f(x)H(x))]$$

Algorithm: AdaBoost for Binary Classification

Input: Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\ y_i = -1, 1\}$;
WeakLearn Algorithm I;
Number of Weak Learn Algorithm T.

Procedure: Initializing sample weight vector $W_1(\mathbf{x}) = \frac{1}{n}$;
for $t = 1, 2, \ldots, T$ **do**
$h_t = I(D, W_t)$;

$\epsilon_t = P_{\mathbf{x}|W_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$;
if $\epsilon_t > 0.5$ **then break**
$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$;
$W_{t+1} = \frac{W_t(\mathbf{x})}{Z_t} exp(-\alpha f(\mathbf{x}) h_t(\mathbf{x}))$
end for
Output: $H(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

(i) 样本权值更新方式: W_t

$$\begin{aligned} W_{t+1} &= \frac{W_t(\mathbf{x})}{Z_t} \times \begin{cases} exp(-\alpha_t), & if\ h_t(x) = f(x); \\ exp(\alpha_t), & if\ h_t(x) \neq f(x). \end{cases} \\ &= \frac{W_t(\mathbf{x})}{Z_t} exp(-\alpha f(\mathbf{x}) h_t(\mathbf{x})) \end{aligned}$$

(ii) 损失函数最小化：H(x)

$$\begin{aligned} \frac{\partial l_{exp}(H|W)}{\partial H(\mathbf{x})} &= -exp(H(\mathbf{x}))P(f(x)=1|x) + exp(H(x))P(f(x)=-1|x) = 0 \\ H(\mathbf{x}) &= \frac{1}{2} ln \frac{P(f(x)=1|x)}{P(f(x)=-1|x)} \\ sign(H(\mathbf{x})) &= sign(\frac{1}{2} ln \frac{P(f(x)=1|x)}{P(f(x)=-1|x)}) \\ &= \begin{cases} 1, & \text{P(f(x)=1|x) > P(f(x)=-1|x);} \\ -1, & \text{P(f(x)=1|x) < P(f(x)=-1|x).} \end{cases} \\ &= argmax_{y \in \{-1,1\}} P(f(x) = y) \end{aligned}$$

(iii) 损失函数最小化：α_t

$$\begin{aligned} l_{exp}(\alpha_t h_t|W_t) &= E_{x|W_t}[exp(-f(x)\alpha_t h_t(x)] \\ &= E_{x|W_t}[e^{-\alpha_t} I(f(x) = h_t(x)) + e^{\alpha_t} I(f(x) \neq h_t(x))] \\ &= e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t \\ \frac{\partial l_{exp}(\alpha_t h_t)}{\partial \alpha_t} &= -e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t = 0 \\ \alpha_t &= \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t} \end{aligned}$$

(iv) 损失函数最小化 [原理类似加法模型]：h_t,W(x)

$$\begin{aligned} h_t(x) &= argmin_h l_{exp}(H_{t-1} + \alpha_t h(x)|W) \\ &= argmin_h E_{x|W}[exp(-f(x)H_{t-1})exp(-f(x)\alpha_t h(x))] \\ &= argmin_h E_{x|W}[exp(-f(x)H_{t-1})exp(-f(x)h(x))] \\ &\equiv argmin_h E_{x|W}[exp(-f(x)H_{t-1})(1 - f(x)h(x) + \frac{f^2(x)h^2(x)}{2})] \\ &= argmin_h E_{x|W}[exp(-f(x)H_{t-1})(1 - f(x)h(x) + \frac{1}{2})] \\ &= argmax_h E_{x|W}[\frac{exp(-f(x)H_{t-1})}{E_{x|W}[exp(-f(x)H_{t-1}(x))]} f(x)h(x)] \\ &= argmax_h E_{x|W_t}[f(x)h(x)],\ where,\ W_t(x) = \frac{exp(-f(x)H_{t-1})}{E_{x|W}[exp(-f(x)H_{t-1}(x))]} \\ &= argmax_h E_{x|W_t}[1 - 2I(f(x) \neq h(x))] = argmin_h E_{x|W_t}[I(f(x) \neq h(x))] \\ W_{t+1} &= \frac{W_t exp(-f(x)H_t(x))}{E_{x|W}[exp(-f(x)H_t(x))]} \\ &= W_t exp(-f(x)\alpha_t h_t) \frac{E_{x|W}[exp(-f(x)H_{t-1}(x))]}{E_{x|W}[exp(-f(x)H_t(x))]} \end{aligned}$$

AdaBoost 作为弱分类器的组合器，基分类器越弱，效果提升越明显，结果也越好，且不容易过拟合，但由于 AdaBoost 的权值更新方式的原理，AdaBoost 对异常数据比较敏感，因异常数据在权值更新中往往会获得较高的权值；

3.1.2 AdaBoost.m1

AdaBoost 基本算法应用限制在二分类中，而衍生的 AdaBoost.m1 是基于 AdaBoost 的多分类方法，主要区别在于多分类问题在更新权值时，分类错误的样本权值不进行更新，而若分类正确的样本其权重乘以 $\frac{\epsilon_t}{1-\epsilon_t} < 1$ 来提高错误分类的样本，其基本算法如下:

AdaBoost.m1 for Multiclass Classification

Input: Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\ y_i = 1, 2, \ldots, k\}$;
WeakLearn Algorithm I;
Number of Weak Learn Algorithm T.
Procedure: Initializing sample weight vector $W_1(\mathbf{x}) = \frac{1}{n}$;
for $t = 1, 2, \ldots, T$ **do**
$h_t = I(D, W_t)$;
$\epsilon_t = P_{\mathbf{x}|W_t}(h_t(\mathbf{x}) \neq f(\mathbf{x}))$;
if $\epsilon_t > 0.5$ **then break**
$\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$;
$W_{t+1} = \frac{W_t}{Z_t}\beta_t^{1-[h_t(\mathbf{x})\neq y]}$
end for
Output: $H(\mathbf{x}) = argmax_{y\in C}\sum_{t=1}^{T} ln\frac{1}{\beta_t}[h_t(\mathbf{x}) = y]$

3.1.3 AdaBoost.m2

AdaBoost.m2 也是基于 AdaBoost 的多分类方法，不仅关注样本分类是否错误，而且关注样本分类错误是在错分在哪一类别上，因此在更新权值时会更新错误分类的权值（如下所示在更新 ϵ 时更新 β 进而提高错误分类样本的权重，同时在最后一步更新 ω 是降低修改其他类的权重）；

AdaBoost.m2 for Multiclass Classification

Input: Train dataset $D = \{(\mathbf{x}_i, y_i) \,|\, i = 1, 2, \ldots, n;\ y_i = 1, 2, \ldots, k\}$;
WeakLearn Algorithm I;
Number of Weak Learn Algorithm T.
Procedure: Initializing sample weight vector $W_1(i) = \frac{1}{n}$;
$\omega_{i,y}^1 = \frac{W_i}{k-1}$ for $i = 1, 2, \ldots, n;\quad y \in \{1, 2, \ldots, k\}/y_i$;
for $t = 1, 2, \ldots, T$ **do**
$\Omega_i^t = \sum_{y\neq y_i}\omega_{i,y}^t$;
$q_t(i, y) = \frac{\omega_{i,y}^t}{\Omega_i^t} for\, y \neq y_i$;
$W_t^i = \frac{\Omega_i^t}{\sum_{i=1}^{N}\Omega_i^t}$;
$h_t = I(D, W_t)$;
$\epsilon_t = \frac{1}{2}\sum_{i=1}^{N} W_t(i)(1 - h_i(x_i, y_i) + \sum_{i,y\neq y_i} q_t(i, y)h_t(x_i, y))$;
if $\epsilon_t > 0.5$ **then break**
$\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$;
$\omega_{i,y}^{t+1} = \omega_{i,y}^{t+1} * \beta_t^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_i,y)}$ for $y \in \{1, 2, \ldots, k\}/y_i$;
end for;
Output: $H(\mathbf{x}) = argmax_{y\in C}\sum_{t=1}^{T} ln\frac{1}{\beta_t}h_t(x, y)$;

3.1.4 AdaBoost.R

AdaBoost.R 算法是 AdaBoost 算法应用于回归问题的改进，但在使用标准 AdaBoost.R 算法会先将 response variable 映射在 $[0, 1]$ ；

AdaBoost.R for Regression Problem

Input: Train dataset $D = \{(\mathbf{x}_i,\, y_i)\,|\, i = 1, 2, \ldots, n;\, y_i \in [0,\, 1]\}$;
WeakLearn Algorithm I;
Number of Weak Learn Algorithm T.
Procedure: Initializing sample weight vector $W_1(i) = \frac{1}{n}$;
$\omega_{i,y}^1 = \frac{W_i * |y - y_i|}{Z}$, where $Z = \sum_{i=1}^{n} W(i) \int_0^1 |y - y_i| dy$;
for $t = 1, 2, \ldots, T$ **do**
$W^t = \frac{\omega^t}{\sum_{i=1}^{n} \int_0^1 \omega_{i,y}^t} dy$;
$h_t = I(D, W_t)$;
$\epsilon_t = \sum_1^n |\int_{y_i}^{h_t(x_i)} W_{i,y}^t| dy$;
if $\epsilon_t > 0.5$ **then break**
$\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$;
$\omega_{i,y}^{t+1} = \omega_{i,y}^t \beta_t^{1 - I(yin[y_i, h_t(x_i)]) - I(yin[h_t(x_i), y_i])}$;
end for;
Output: $H(\mathbf{x}) = inf\{y : \sum_{t:h_t(y) \le y} log(\frac{1}{\beta_t} \ge \frac{1}{2} log \frac{1}{\beta_t}\}$;

3.1.5 AdaBoost.R2

AdaBoost.R2 算法是 AdaBoost 算法应用于回归问题的改进，但在使用标准 AdaBoost.R 算法会先将 response variable 映射在 $[0, 1]$ 区间内;

AdaBoost.R2 for Regression Problem

Input: Train dataset $D = \{(\mathbf{x}_i,\, y_i)\,|\, i = 1, 2, \ldots, n;\, y_i \in [0,\, 1]\}$;
WeakLearn Algorithm I;
Number of Weak Learn Algorithm T.
Procedure: Initializing sample weight vector $W_1(i) = \frac{1}{n}$;
$\omega_{i,y}^1 = \frac{W_i * |y - y_i|}{Z}$, where $Z = \sum_{i=1}^{n} W(i) \int_0^1 |y - y_i| dy$;
for $t = 1, 2, \ldots, T$ **do**
$h_t = I(D, W_t)$;
$Z^t = max_{i \in \{1,2,dots,n\}} |y_i - h_t(\mathbf{x}_j)|$;
$e_i^t = \frac{|y_i - h_t(\mathbf{x}_j)|}{Z_t}$ for linear loss;
$\epsilon_t = \sum_{i=1}^{n} e_i^t \omega_i^t$;
if $\epsilon_t > 0.5$ **then break**
$\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$;
$\omega_{i,y}^{t+1} = \frac{\omega_{i,y}^t \beta_t^{1-e_i^t}}{\sum \omega_{i,y}^t \beta_t^{1-e_i^t}}$;
end for;
Output: $H(\mathbf{x}) = median(ln \frac{1}{\beta_t} h_t(x),\, i = 1, 2, ...)$;

3.1.6 AdaBoost.RT

AdaBoost.RT 是对 AdaBoost.R 的改进，在判断误差率时增加一项 threshold 来判断是否为误差;

AdaBoost.RT for Regression Problem

Input: Train dataset $D = \{(\mathbf{x}_i,\, y_i)\,|\, i = 1, 2, \ldots, n;\, y_i \in [0,\, 1]\}$;
WeakLearn Algorithm I;
Number of Weak Learn Algorithm T.
threshold ϕ.
Procedure: Initializing sample weight vector $W_1(i) = \frac{1}{n}$;
for $t = 1, 2, \ldots, T$ **do**
$h_t = I(D, W_t)$;

$\epsilon_t = \sum_{i=1}^{n} W_t(i) I[|\frac{y_i - h_t(\mathbf{x}_j)}{y_i}| > \phi]$
$\beta_t = \epsilon_t^2$;
$W_{t+1}(i) = \frac{W_t(i)}{Z_t} \beta_t^{I[|\frac{y_i - h_t(\mathbf{x}_j)}{y_i}| > \phi]}$;
end for;
Output: $H(\mathbf{x}) = \frac{\sum_{i=1}^{t} ln\frac{1}{\beta_t} h_t(x)}{\sum_{i=1}^{t} ln\frac{1}{\beta_t}}$;

3.1.7 AdaBoost.MR & MO

3.1.8 LPBoost

3.1.9 SAMME.R 与 SAMME

3.1.10 Scikitlearn

$AdaBoostClassifier(base\_estimator, n\_estimators, learning\_rate, algorithm)$

3.2 Boosting Decision Tree

Boosting Decision Tree 即提升树算法，采用前向分布算法，确定初始提升树之后对第 m 步的建模为:

$$f_m(x) = f_{m-1}(x) + T(x; \Theta_m),\ s.t.$$
$$\hat{\Theta}_m = argmin_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

对于一般的损失函数的提升树算法基本流程为

Boosting Decision Tree

- 训练集 $T = (x_i, y_i)$
- 初始化 $f_0(x) = 0$
- for m in 1, 2, ..., M
  - 计算残差 $r_{mi} = y_i - f_{m-1}(x_i)$
  - 训练一个回归树 $T(x; \Theta_m)$ 拟合残差 r_{mi}
  - 更新 $f_m(x) = f_{m-1}(x) + T(x; \Theta_m)$
- 得到回归树 $f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$

3.3 Stochastic Gradient Boosting

3.4 GBDT

3.4.1 GBDT

GBDT 在基于不同的损失函数上，利用梯度下降法展开目标函数进行优化，进行一阶泰勒展开 $f(x + \Delta x) = f(x) + f^{'}(x)\Delta x$，则有

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) \\ &= \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{t-1}) + \frac{\partial l(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} f_t(x_i)] \\ &= \sum_{i=1}^{n} [Obj_i^{t-1} + \hat{y}_i^{t-1}) + \frac{\partial l(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} f_t(x_i)] \end{aligned}$$
$$\frac{\partial Obj^{(t)}}{\partial f_i} = \frac{\partial l(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}}$$

目标函数 Obj 的梯度上升方向为 $\frac{\partial l(y_i,\hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}}$，若要使得 $Obj^{(t)} < Obj^{(t-1)}$, 只需要令 f_t 沿着负梯度方向进行赋值，即 $f_t = -\epsilon\frac{\partial l(y_i,\hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}}$, $\epsilon > 0$, ϵ 一般就是我们所说的步长，也即学习率;

Gradient Boosting Decision Tree

- 训练集 $T = (x_i, y_i)$
- 初始化 $f_0(x) = 0$
- for m in 1, 2, ..., M
  - 计算残差 $r_{mi} = -[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$
  - 训练一个回归树 $T(x;\Theta_m)$ 拟合梯度 r_{mi}, 得到 $T(x;\Theta_m)$ 的叶结点区域 R_{mj}
  - 对 $j = 1, 2, \ldots, J$, 计算出区域加权输出值使得 $c_{mj} = argmin_c \sum_{x_i \in R_{mj}} L(f_{m-1}(x_i), c)$;
  - $\alpha_m = argmin_\alpha \sum_i^n L(y_i, f_{m-1}(x_i) + \alpha h_t(x_i))$;
  - 更新 $f_m(x) = f_{m-1}(x) + \alpha_m h_m(x)$
- Return $f_M(x) = \sum_{m=1}^M \alpha_m h_m(x)$

GBDT 的基学习器分裂原则为选择方差增益最高的属性, 即:

$$V_{A_k|D}(d) = \frac{1}{n}(\frac{G_L(A_k)^2}{n_L} + \frac{G_R^2(A_k)}{n_R})$$

3.4.2 LSBoost

3.4.3 LogitBoost

3.4.4 LADBoost

3.4.5 MBoost

3.4.6 L_kBoost

3.5 XGBoost TGBoost

XGBoost 是 GBDT 的一个拓展，主要在四点上进行改进:

(1) 在损失函数中加入正则化项，当基学习器为 CART 树时，正则化项与树的叶子节点的数目和叶子节点的值相关;

$$\begin{aligned} Obj &= \sum_i^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{t=1}^T \Omega(f_t) \\ &= \sum_i^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{t=1}^T (\gamma|T_t| + \frac{1}{2}\lambda\|\omega\|^2) \\ &= \sum_i^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{t=1}^T (\gamma|T_t| + \frac{1}{2}\lambda \sum_{m=1}^M \omega_m^2) \end{aligned}$$

其中 $|T_t|$ 为树 T_t 的叶结点个数，$\omega_m(x) = f_t(x)$ 为 m 叶结点的取值;

(2) GBDT 是对损失函数上利用泰勒一阶展开的梯度优化算法，而 XGBoost 则对损失函数利用牛顿法

对目标函数进行二阶泰勒展开而进行的优化；

$$
\begin{aligned}
Obj^{(t)} &= \sum_i l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_k) + Constant \\
&= \sum_i [l(y_i, \hat{y}_i^{(t-1)}) + \frac{\partial l(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} f_t(x_i) + \frac{1}{2} \frac{\partial^2 l(y_i, \hat{y}_i^{t-1})}{\partial (\hat{y}_i^{t-1})^2} f_t^2(x_i)] + \Omega(f_k) \\
&\equiv \sum_i [l(y_i, \hat{y}_i^{(t-1)}) + g_i \omega_{q(x_i)} + \frac{1}{2} h_i \omega_{q(x_i)^2}] + \gamma |T| + \frac{1}{2} \sum_{t=1}^{T} \omega_t^2 \\
&= \sum_{t=1}^{T} [(\sum_{i \in R_t} g_i) \omega_t + \frac{1}{2} (\sum_{i \in R_t} + \lambda) \omega_t^2] + \gamma |T| \\
&\equiv \sum_{t=1}^{T} [G_j \omega_t + \frac{1}{2} (H_t + \lambda) \omega_t^2] + \gamma |T|
\end{aligned}
$$

在对梯度拟合求解最优得到 $\omega_t^* = -\frac{G_j}{H_j + \lambda} \quad \Rightarrow \quad Obj = -\frac{1}{2} \sum_T \frac{G_j^2}{H_j + \lambda} + \gamma |T|$ 等;

(3) GBDT 以 CART 回归树作为基学习器时对属性的最佳分割点的选择衡量标准为最小化均方差，而 XGBoost 在加入正则化项之后寻找分割点的标准是最大化分裂收益:

$$
\begin{aligned}
Obj\_node\_before\_split &= -\frac{1}{2} \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} + \gamma \times 1 \\
Obj\_node\_after\_split &= -\frac{1}{2} [\frac{(G_L)^2}{H_L + \lambda} + \frac{(G_R)^2}{H_R + \lambda}] + \gamma \times 2 \\
node\_split\_gain &= \frac{1}{2} [\frac{(G_L)^2}{H_L + \lambda} + \frac{(G_R)^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma
\end{aligned}
$$

Optimal Splits Search: Greedy Algorithm

Input: Current Node Weighted Dataset $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;

Current Node Attribute Set $A = \{A_i;\ i = 1, 2, \ldots, m\}$;

Initialize λ;

Procedure:

$Gain = 0, G = \sum_{i=1}^{n} g_i,\ H = \sum_{i=1}^{n} h_i$;

for $i = 1, 2, \ldots, m$ **do**

$G_L = 0,\ H_L = 0$;

for j in sorted(D) **do**

$G_L = G_L + g_j,\ G_R = G - G_L$;

$H_L = H_L + h_j,\ H_R = H - H_L$;

$score = max\{score, \frac{(G_L)^2}{H_L + \lambda} + \frac{(G_R)^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\}$;

end for

end for

Output: Split with max Score

Optimal Splits Search: Approximately Algorithm

Input: Current Node Weighted Dataset $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;

Current Node Attribute Set $A = \{A_i;\ i = 1, 2, \ldots, m\}$;

Initialize λ;

Procedure:

for $i = 1, 2, \ldots, m$ **do**

Propose $S_k = \{s_{k1}, s_{k2}, \ldots, s_{kl}\}$ by percentiles on feature A_k on per tree or per split;

end for

for $i = 1, 2, \ldots, m$ **do**

$G_{kv} = \sum_{i \in \{i \,|\, s_{k,v} \geq x_{ik} \geq s_{k,v-1}\}} g_i$

$H_{kv} = \sum_{i \in \{i \,|\, s_{k,v} \geq x_{ik} \geq s_{k,v-1}\}} h_i$

end for

Follow the same step in Greedy Algorithm.

Output: Split with max Score

---

(4) 支持 CART(gbtree) 树作为基学习器，也支持线性分类器 (gblinear);

(5) PreSorted+Exact Split/Approximate Percentile Split 对所有数据进行预排序后以 columb block 形式存于内存中，每个变量和标签作为一个 Block，之后每次计算分裂时只需要调用即可;

3.6 LightGBM

1. Basic

LightGBM 实际上也是 GBDT 的变种，在算法上并未进行任何改进但有其他特征;

(1) 在数据输入模型时，对连续属性进行离散化成 k 个整数，同时构造宽度为 k 的直方图存于内存中 [bin mapper]，后续遍历数据计算分裂时直接使用直方图的累计统计量即可，因此模型时间运行大幅降低;

(2) 与 XGBoost 采用 Level Wise 树生长后进行剪枝的策略不同，LightGBM 使用 Leaf Wise 生成策略;

2. GOSS as Bagging

Gradient Based One Side Sampling

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;

Number of Weak Learn Algorithms: T;

Sampling ratio of large gradient data: a;

Sampling ratio of small gradient data: b;

Loss function: L;

Weak Learn: I;

Procedure:

Initialize $models = \{\}$, $fact = \frac{1-a}{b}$, $W = \{1, 1, \ldots, 1\}$;

$topN = a \times len(D)$, $randN = b \times len(D)$

for $t = 1\, to\, T$ **do**

$h_t(\mathbf{x}) = models.predit(\mathbf{x})$;

$\epsilon_t = L(y, h_t(x), W_t)$;

sorted = GetSortedIndices(abs(ϵ_t));

topSet = sorted[1:topN];

randSet = RandomPick(sorted[topN:len(D)],randN);

usedSet = topSet + randSet;

$W_t[randSet]$ *= fact;

$newModel = L(D[usedSet], -\epsilon_t[usedSet], W_t[usedSet])$;

models.append(newModel)

---

3. EFB

Greedy Bundling

Input: F: features;

K: max conflict count;

G: Construct graph;

Procedure:

$SearchOrder$ = G.sortByDegree();

$bundles$ = , $bundlesConflict$ = ;

for i in $searchOrder$ **do**

$needNew$ = True;

for j in $len(bundles)$ **do**

cnt = ConflictCnt($bundles[j]$,$F[i]$);

if $cnt + bundlesConflict[i] \leq$ K **then**

$bundles[j]$.add($F[i]$);

$needNew$ = False;

if $needNew$ **then**

Add $F[i]$ as a new bundle to $bundles$;

Output: $bundles$

Merge Exclusive Features

Input: $numData$: number of data;

F: One bundle of exclusive features;

Procedure:

Initialize $binRanges = \{0\}, totalBin = 0$;

for f in F **do**

$totalBin += f.numBin$;

$binRanges$.append($totalBin$);

newBin = new Bin($numData$);

for $i = 1$ to F **do**

newBin[i]=0

for $j = 1$ to $len(F)$ **do**;

if $F[j].bin[i] \neq 0$ **then**

$newBin[i] = F[j].bin[i] + binRanges[j]$;

Output: $newBins, binRanges$

---

4. Histogram Based Algorithm

Histogrambased Algorithm in GBDT

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;

Tree Max Depth d;

$NodeSet$: tree nodes in current level, $RowSet$: Data Indices in Tree Nodes;

Procedure:

for $i = 1\,to\,d$ **do**
  for *node* in $NodeSet$ **do**
    usedRows = $RowSet[node]$;
    for $k = 1\,to\,m$ **do**
      H = new Histogram(); Build histogram;
      for j in usedRows **do**
        $bin = D.f[k][j].bin$;
        $H[bin].y = H[bin].y + D.y[j]$;
        $H[bin].n = H[bin].n + 1$;
      Find the best split on histogram H.
    Update rowSet and NodeSet according to the best split points.
    …

---

5. Gradient Based One Side Sample
6. Rules for Optimal Split Search

$$V_{A_k|D}(d) = \frac{1}{n}[\frac{(\sum_{x_i \in LargeG_L} g_i + \frac{1-a}{b}\sum_{x_i \in SmallG_L} g_i)^2}{n_L} + \frac{(\sum_{x_i \in LargeG_R} g_i + \frac{1-a}{b}\sum_{x_i \in SmallG_R} g_i)^2}{n_R}]$$

另外, 传统的分布式计算最佳分裂点是基于 Local Worker 平行计算之后汇总 Global 后找出 Global BSP 再去 Local Worker 进行分裂; 而 LightGBM 采用

FindBestSplit

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;
  Attribute Set $A = \{A_i;\ i = 1, 2, \ldots, m\}$;
Procedure:
  for all A_i **do**
    Construct Histogram();
    H = new Histogram();
    for all a in A_i **do**
      H.binAt(x.bin).Put(x.label);
    end for
    Find Best Split
    leftSum = new HistogramSum();
    for all bin in H **do**
      leftSum = leftSum + H.binAt(bin)
      rightSum = H.AllSum - leftSum
      split.gain = CalSplitGain(leftSum, rightSum)
      bestSplit.gain = ChoiceBetterOne(split, bestSplit)
    end for;
  end for;
Output: bestSplit

PV Tree FindBestSplit

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;

Procedure:

localHistograms = ConstructHistogramS(D);

Local Voting

splits = [] **for** all H in localHistograms **do**

splits.Push(H.FindBestSplit());

end for

localTop = splits.TopKByGain(K);

Gather all candidates;

allCandidates = AllGather(localTop)

Global Voting

globalTop = all.Candidates.TopKByMajority(2*K);

Merge Global Histograms

globalHistograms = Gather(globalTop, localHistograms);

bestSplit = globalHistograms.FindBestSplit()

Output: BestSplit

7. DART

The DART algorithm

Procedure:

Let N be the total number of trees to be added to the ensemble;

$S_1 = \{x, -L_x^{'}(0)\}$;

T_1 be a tree trained on the dataset S_1;

M = $\{T_1\}$;

for $t = 1\, to\, N$ **do**

D = the subset of M such that $T \in M$ is in D with probability p_{drop};

if D = $\emptyset$ **then**;

D = a random element from M;

$\hat{M} = M/D$;

$S_t = \{x, -L_x^{'}(\hat{M}(x))\}$;

T_t be a tree trained on the dataset S_t;

$M = M \cup \{\frac{T_t}{|D|+1}\}$;

for $T \in D$ **do**

Multiply T in M by a factor of $\frac{|D|}{|D|+1}$;

end for

end for

Output: M

3.7 CatBoost

1. Basic Algorithm

CatBoost Algorithm

Input: Training Data $D = \{(x_i, y_i);\ i = 1, 2, \ldots, n\}$;

Number of Iterations T;

Number of Random Permutation of Dataset S;

Train Mode $Mode$;

Loss funtion and α;

Procedure:

Initialize σ_i = random permutation of $\{1, 2, \ldots, n\}$ for $i = 1, 2, \ldots, s$;

$S_r(i) = 0$, $r = 1, 2, \ldots, s$ for $i = 1, 2, \ldots, n$;

$S^{'}_{r,j}(i) = 0$, $r = 1, 2, \ldots, s$ for $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, [log_2 n]$;

for i = 1 to T **do** $grad = ClacGradient(L, S, y)$;

$grad^{'} = ClacGradient(L, S^{'}, y)$;

$r = random(1, s)$;

$T_t = BuildTree(Mode, grad_r, grad^{'}_r, \sigma_r, \mathbf{x})$;

$leaf_{r,i} = GetLeaf(\mathbf{x}_i, T_t, \sigma_r)$ for $r = 0, 1, \ldots, s$; $i = 1, 2, \ldots, n$;

foreach $leafR^t_j$ in T_t **do**

$b^t_j = -avg(grad_0(i)\ for\ i : leaf_{r,i} = j)$;

$S, S^{'} = UpdateModel(Mode, leaf, T_t, \{b^t_j\}_j, S, S^{'}, grad, grad^{'}, \sigma_r{}^s_{r=1}$;

Output: $F(x) = \sum_{t=1}^{T} \sum_j \alpha b^t_j I(GetLeaf(x, T_t, ApplyMode) == j)$

Build Tree

Input: $Mode, grad_r, grad^{'}_r, \sigma_r, \mathbf{x}$

Procedure:

Initialize T = empty tree;

foreach step of topdown procedure **do**;

Form a set C of candidate splits;

for each $c \in C$ **do**

T_c = add split to T;

$leaf_i$ = GetLeaf($\mathbf{x}_i, T_c, \sigma$) for $i = 1, 2, \ldots, n$;

for each $leaf\ j\ in\ T_c$ **do**

for each $i : leaf_i = j$ **do**

if Mode == Plain **then**

$\Delta(i) = avg(grad(p)\ for\ p : leaf_p = j)$;

else

$\Delta(i) = avg(grad^{'}_{[log_2 \sigma_r(i)]}(p)\ for\ p : leaf_p = j, \sigma(p) < \sigma(i))$;

$loss(T_c) = \|\Delta - grad\|_2$

$T = argmin_{T_c}(loss(T_c))$

Output: T

UpdateModel

Input: $Mode, leaf, T_t, \{b_j^t\}_j, S, S^{'}, grad, grad^{'}, \sigma_r{}_{r=1}^{s}$

Procedure:

foreach $leaf\ j\ in\ T$ **do**

foreach $i\ s.t.\ leaf_{r,j} = j$ **do**

$S_0(i) = S_0(i) + \alpha b_j$;

for r = 1 to s **do**

foreach $i\ s.t.\ leaf_{r,j} = j$ **do**

if Mode == Plain **then**

$S_r(i) = S_r(i) - \alpha avg(grad_r(p)\ for\ p:\ leaf_{r,p} = j)$;

else

for l = 1 to $[log_2 n]$ **do**

$S_{r,l}^{'}(i) = S_{r,l}^{'}(i) - \alpha avg(grad_r(p)\ for\ p:\ leaf_{r,p} = j), \sigma_r(p) < 2^l$;

$S_r(i) = S_{r,[log_2 \sigma_r(i)]}^{'}(i)$; **Output:** $S, S^{'}$

2. Categorical Feature

(1) Greedy TBS

TBS 对离散属性数据连续化，处理方法为根据属性取值对应的 TARGET 取均值：

$$\hat{x}_i^k = \frac{\sum_{j=1}^{n} y_i I(x_j^k = x_i^k)}{\sum_{j=1}^{n} I(x_j^k = x_i^k)}$$

但以该方法进行处理可能会出现一个问题对于之前类别未出现的数据则在数据转化后的取值为零，所以需要进行拉普拉斯平滑

$$\hat{x}_i^k = \frac{\sum_{x_j \in D_k}^{n} y_i I(x_j^k = x_i^k) + aP}{\sum_{x_j \in D_k}^{n} I(x_j^k = x_i^k) + a}$$

$$\hat{x}_i^k = \frac{\sum_{x_j \in D_k}^{n} y_i I(x_j^k = x_i^k) + aP}{\sum_{j=1}^{n} I(x_j^k = x_i^k) + a}$$

(2) Holdout TBS

Where $D = D_1$

(3) Leave one out TBS

(4) Orderd TBS

3.8 PieceWise GBDT

PieceWise GBDT 算法与 GBDT 相同，在基学习器的每个结点的输出不是以损失函数为准，而是根据叶结点内的样本进行回归，即：

$$f(X) = a_1 + \sum_{i=1}^{m} b_j X_i$$

m 为叶结点中样本个数;

4 Loss Function

1. Square Loss

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

2. Absolute Loss

$$L(y, \hat{y}) = \frac{1}{2}|y - \hat{y}|$$

3. Logistic Loss for binary classification, that is $y \in \{+1, -1\}$

$$L(y, \hat{y}) = exp(-y\hat{y})$$

4. exponential Loss

$$L(y, \hat{y}) = \frac{1}{2}|y - \hat{y}|$$