# Melt/LightGBM中GBDT的实现

chenghuige

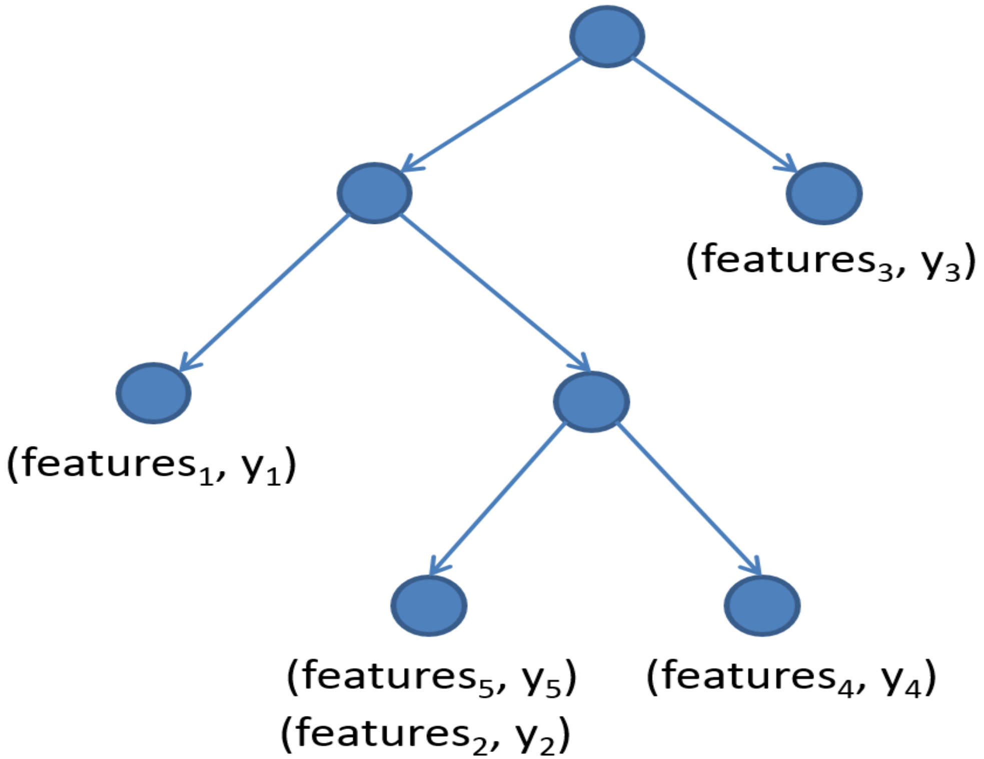
# outline

- MELT中的GBDT特点
- GBDT简介
- GBDT算法流程
- MELT中的实现

# Melt GBDT的特点

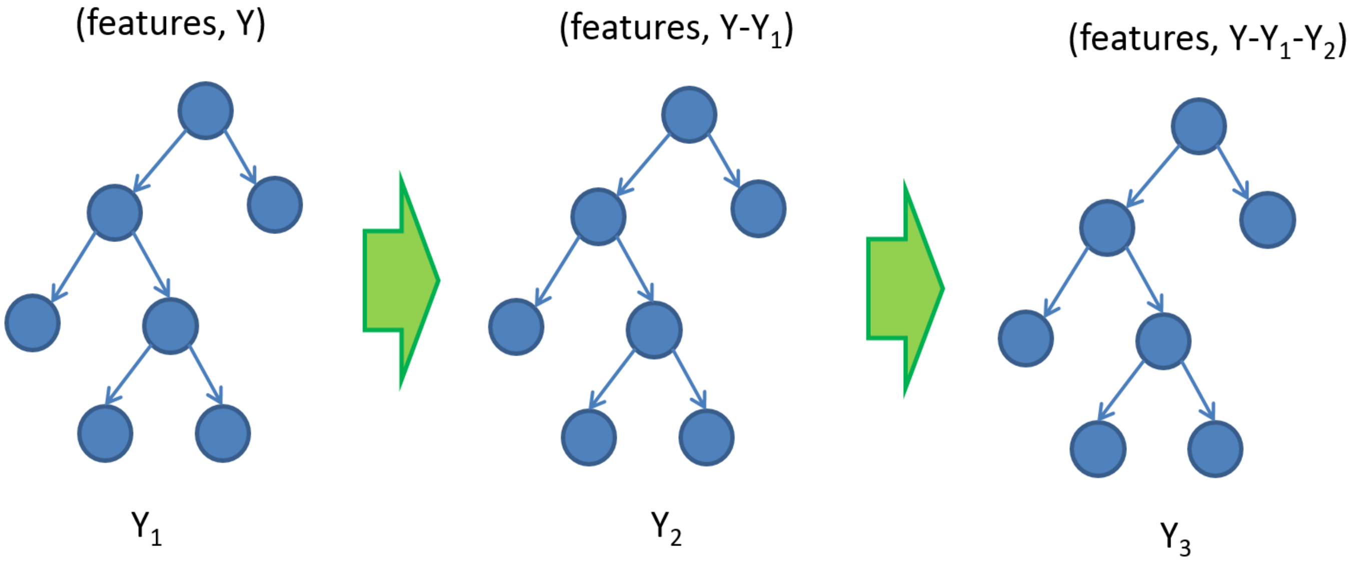
- 相比xgboost速度更快
- 采用针对二分类的负二项式分布对数似然损失函数
- 可调节内存占用(更快的速度or更少的内存占用)
- 支持打印模型特征权重
- 支持打印决策树路径
- 支持打印单次预测中的特征权重
- 支持样本weight
- 支持early stop，支持bagging+gbdt (TODO）

# 回归树



(features$_3$, y$_3$)
(features$_1$, y$_1$)
(features$_5$, y$_5$)
(features$_2$, y$_2$)
(features$_4$, y$_4$)

# Boosted Regression Tree



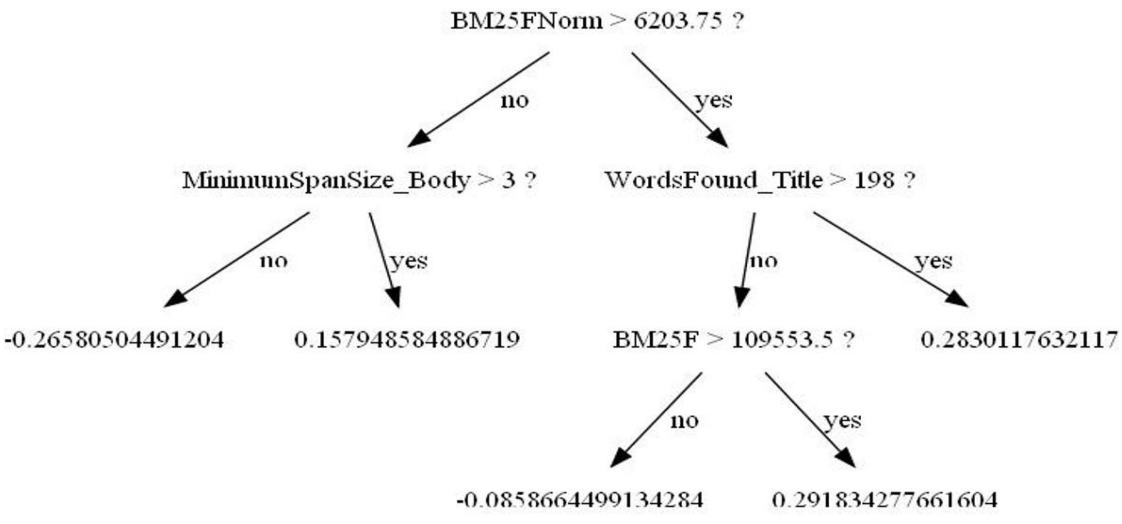

**Combine many weak models to make a strong committee**

$$Y = Y_1 + Y_2 + Y_3$$

# Boosted Regression Trees

- Output of GBDT is an ensemble of boosted decision trees

Example tree:




Score ($d_i$) = $\Sigma_T$ $tree_t(d_i)$

# 回归树


(features$_1$, y$_1$)

(features$_2$, y$_2$)

(features$_3$, y$_3$)

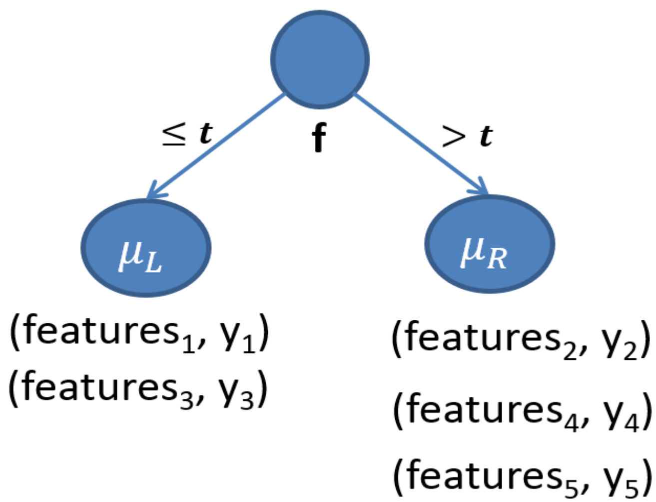
(features$_4$, y$_4$)

(features$_5$, y$_5$)

$$Error = \sum_i (y_i - \mu)^2$$

最初所有样本都在根节点
根节点输出值是所有目标值的均值

# 回归树




$$S = \sum_i (y_i - \mu)^2 \quad \Longrightarrow \quad S_j = \sum_{i \in L} (y_i - \mu_L)^2 + \sum_{i \in R} (y_i - \mu_R)^2$$

$$Split\ gain = S - S_j$$

# 回归树

$$S_j = \sum_{i \in L}({y_i}^2 + {\mu_L}^2 - 2y_i\,\mu_L) + \sum_{i \in R}({y_i}^2 + {\mu_R}^2 - 2y_i\,\mu_R)$$

$$= \left(\sum_i {y_i}^2\right) + \|L\|{\mu_L}^2 + \|R\|{\mu_R}^2 - 2\mu_L \sum_{i \in L} y_i - 2\mu_R \sum_{i \in R} y_i = \left(\sum_i {y_i}^2\right) - \|L\|{\mu_L}^2 - \|R\|{\mu_R}^2$$

$$= \left(\sum_i {y_i}^2\right) - \left(\frac{{sum_L}^2}{\|L\|} + \frac{{sum_R}^2}{\|R\|}\right)$$

$$S = \sum_i (y_i - \mu)^2 = \left(\sum_i {y_i}^2\right) + \|total\|\mu^2 - 2\mu \sum_i y_i = \left(\sum_i {y_i}^2\right) - \|total\|\mu^2$$

$$= \left(\sum_i {y_i}^2\right) - \frac{sum^2}{\|total\|}$$

$$\boxed{Split\ gain = S - S_j = \left(\frac{{sum_L}^2}{\|L\|} + \frac{{sum_R}^2}{\|R\|}\right) - \frac{sum^2}{\|total\|}}$$

# GBDT算法

**Algorithm 1: Gradient_Boost**

$F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} \Psi(y_i, \rho)$

For $m = 1$ to $M$ do:

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

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

$\rho_m = \arg\min_\rho \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$

$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$

endFor

end Algorithm

**Algorithm 5: $L_2$_TreeBoost**

$F_0(\mathbf{x}) = \frac{1}{2}\log\frac{1+\bar{y}}{1-\bar{y}}$

For $m = 1$ to $M$ do:

$\tilde{y}_i = 2y_i / (1 + \exp(2y_i F_{m-1}(\mathbf{x}_i)), \ i = 1, N$

$\{R_{lm}\}_1^L = L\text{-terminal node } tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$

$\gamma_{lm} = \sum_{\mathbf{x}_i \in R_{lm}} \tilde{y}_i / \sum_{\mathbf{x}_i \in R_{lm}} |\tilde{y}_i| (2 - |\tilde{y}_i|), \ l = 1, L$

$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \gamma_{lm} 1(\mathbf{x} \in R_{lm})$

endFor

end Algorithm

Here the loss function is negative binomial log–likelihood (FHT98)

$$\Psi(y, F) = \log(1 + \exp(-2yF)), \quad y \in \{-1, 1\},$$

where

$$F(\mathbf{x}) = \frac{1}{2}\log\left[\frac{\Pr(y = 1 \mid \mathbf{x})}{\Pr(y = -1 \mid \mathbf{x})}\right].$$

**Algorithm 16.4:** Gradient boosting

1 Initialize $f_0(\mathbf{x}) = \operatorname{argmin}_{\gamma} \sum_{i=1}^{N} L(y_i, \phi(\mathbf{x}_i; \gamma))$;

2 **for** $m = 1 : M$ **do**

3 Compute the gradient residual using $r_{im} = -\left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x}_i)=f_{m-1}(\mathbf{x}_i)}$;

4 Use the weak learner to compute $\gamma_m$ which minimizes $\sum_{i=1}^{N} (r_{im} - \phi(\mathbf{x}_i; \gamma_m))^2$;

5 Update $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu\phi(\mathbf{x}; \gamma_m)$;

6 Return $f(\mathbf{x}) = f_M(\mathbf{x})$

$$r_{im} = \frac{2y_i * learning\_rate}{1 + \exp(2y_i\, f_{m-1}(x) * learning\_rate)}$$

# GBDT算法流程

1. 对所有特征进行分桶归一化(bin normalizing)

$$
\begin{array}{c|ccc}
 & f_1 & \cdots & f_m \\
\begin{matrix} d_1 \\ \vdots \\ d_n \end{matrix} & \multicolumn{3}{c}{\begin{bmatrix} v_{11} & \cdots & v_{1m} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nm} \end{bmatrix}}
\end{array}
\Rightarrow
\begin{array}{c|ccc}
 & f_1 & \cdots & f_m \\
\begin{matrix} d_1 \\ \vdots \\ d_n \end{matrix} & \multicolumn{3}{c}{\begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nm} \end{bmatrix}}
\end{array}
$$

2. 计算初始梯度值($f_{m-1}(x)$设置为0或随机值) $\dfrac{2y_i * learning\_rate}{1+\exp(2y_i f_{m-1}(x) * learning\_rate)}$

$$
\begin{array}{c|ccc}
 & f_1 & \cdots & f_m \\
\begin{matrix} d_1 \\ \vdots \\ d_n \end{matrix} & \multicolumn{3}{c}{\begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nm} \end{bmatrix}}
\end{array}
\Rightarrow
\begin{array}{c|cccc}
 & f_1 & \cdots & f_m & \lambda \\
\begin{matrix} d_1 \\ \vdots \\ d_n \end{matrix} & \multicolumn{3}{c}{\begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nm} \end{bmatrix}} & \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix}
\end{array}
$$

# GBDT算法流程

3. 建立树

a) 计算直方图

$$\begin{array}{c|c|c} & \begin{array}{ccc} f_1 & \cdots & f_m \end{array} & \lambda \\ \begin{array}{c} d_1 \\ \vdots \\ d_n \end{array} & \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nm} \end{bmatrix} & \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix} \end{array} \Rightarrow \begin{array}{cc} & \begin{array}{ccc} f_1 & \cdots & f_m \end{array} \\ \begin{array}{c} 0 \\ \vdots \\ k-1 \end{array} & \begin{bmatrix} h_{11} & \cdots & h_{1m} \\ \vdots & \ddots & \vdots \\ h_{k1} & \cdots & h_{km} \end{bmatrix} \end{array}$$

$$h_{ij} = (c_{ij}, l_{ij}) \qquad c_{ij} = \sum_{k=1}^{n} 1 | (b_{kj} = i - 1) \qquad l_{ij} = \sum_{k=1}^{n} \lambda_k | (b_{kj} = i - 1)$$

b) 从直方图获得分裂收益，选取最佳分裂特征，分裂阈值

$$\begin{array}{cc} & \begin{array}{ccc} f_1 & \cdots & f_m \end{array} \\ \begin{array}{c} 0 \\ \vdots \\ k-1 \end{array} & \begin{bmatrix} h_{11} & \cdots & h_{1m} \\ \vdots & \ddots & \vdots \\ h_{k1} & \cdots & h_{km} \end{bmatrix} \end{array} \Rightarrow \begin{array}{ccc} f_1 & \cdots & f_m \\ [G_1 & \cdots & G_m] \\ [I_1 & \cdots & I_m] \end{array}$$

$$G_j = \max_{1 \le x < k} \left( \frac{(\sum_{i=1}^{x} l_{ij})^2}{\sum_{i=1}^{x} c_{ij}} + \frac{(\sum_{i=x+1}^{k} l_{ij})^2}{\sum_{i=x+1}^{k} c_{ij}} \right)$$

$$I_j = \operatorname*{argmax}_{1 \le x \le k} \left( \frac{(\sum_{i=1}^{x} l_{ij})^2}{\sum_{i=1}^{x} c_{ij}} + \frac{(\sum_{i=x+1}^{k} l_{ij})^2}{\sum_{i=x+1}^{k} c_{ij}} \right)$$

# GBDT算法流程

c) 建立根节点

$$\begin{array}{ccc} f_1 & \cdots & f_m \\ [G_1 & \cdots & G_m] \\ [I_1 & \cdots & I_m] \end{array}$$

$$s = \underset{1 \le i \le m}{\operatorname{argmax}}(G_i)$$

$$\text{Node} = (s, G_s, I_s)$$

d) 根据最佳分裂特征，分裂阈值将样本切分

$$\begin{array}{cc} & \begin{array}{ccc} f_1 & \cdots & f_m \end{array} \\ \begin{array}{c} d_1 \\ \vdots \\ d_n \end{array} & \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nm} \end{bmatrix} \end{array}$$

L:
$$\begin{array}{ccc} & \begin{array}{ccc} f_1 & \cdots & f_m \end{array} & \lambda \\ \begin{array}{c} d_{p_1} \\ \vdots \\ d_{p_k} \end{array} & \begin{bmatrix} b_{p_1 1} & \cdots & b_{p_1 m} \\ \vdots & \ddots & \vdots \\ b_{p_k 1} & \cdots & b_{p_k m} \end{bmatrix} & \begin{bmatrix} \lambda_{p_1} \\ \vdots \\ \lambda_{p_k} \end{bmatrix} \end{array}$$

R:
$$\begin{array}{ccc} & \begin{array}{ccc} f_1 & \cdots & f_m \end{array} & \lambda \\ \begin{array}{c} d_{p_{k+1}} \\ \vdots \\ d_{p_n} \end{array} & \begin{bmatrix} b_{p_{k+1} 1} & \cdots & b_{p_{k+1} m} \\ \vdots & \ddots & \vdots \\ b_{p_n 1} & \cdots & b_{p_n m} \end{bmatrix} & \begin{bmatrix} \lambda_{p_{k+1}} \\ \vdots \\ \lambda_{p_n} \end{bmatrix} \end{array}$$

$$where\ b_{p_{1..k},s} \le I_s \ \ and\ b_{p_{k+1..n},s} > I_s$$

# GBDT算法流程

e) 重复3.a-3.d选取最佳分裂叶子，分裂特征，分裂阈值，切分样本，直到达到叶子数目限制或者所有叶子不能分割

f) 更新当前每个样本的输出值

$f_k(x) = f_{k-1}(x) + now\_output(x)$*learning_rate

4. 根据之前得到的树更新梯度值 $\frac{2y_i * learning\_rate}{1 + \exp(2y_i f_{m-1}(x) * learning\_rate)}$
5. 重复3,4直到所有的树都建立好

# Melt中核心数据结构Instance表示

- Instance采用稀疏和稠密混合存储方式
- 根据当前Instance非0的特征数目占比确定稀疏或者稠密(默认sparse_ratio < 0.1 表示稀疏 可配置)

```
vector<int>        indices;  //indices空,values非空表示稠密
vector<double>  values;  //indices,values非空,长度相同稀疏
```

# 针对稀疏和稠密表示的优化-计算直方图

```
inline void SumupRootDense(IntArray& bins, SumupInputData& input)
{
  bins.ForEachDense([&, this](int index, int featureBin)
  {
    double output = input.Outputs[index];
    SumTargetsByBin[featureBin] += output;
    CountByBin[featureBin]++;
  });
}

inline void SumupRootSparse(IntArray& bins, SumupInputData& input)
{
  double totalOutput = 0.0;
  bins.ForEachSparse([&, this](int index, int featureBin)
  {
    double output = input.Outputs[index];
    SumTargetsByBin[featureBin] += output;
    CountByBin[featureBin]++;
    totalOutput += output;
  });

  SumTargetsByBin[bins.ZeroValue()] += input.SumTargets - totalOutput;
  CountByBin[bins.ZeroValue()] += input.TotalCount - bins.Count();
}
```

# 针对稀疏和稠密表示的优化-计算直方图

```
inline void SumupLeafDense(IntArray& bins, SumupInputData& input)
{
  for
  {
    do
    in
    in
    Su
    Co
  }
}
```

```
inline void SumupLeafSparse(IntArray& bins, SumupInputData& input)
{
  int iDocIndices = 0;
  int totalCount = 0;
  double totalOutput = 0.0;

  int len = bins.indices.size();
  for (int i = 0; i < len; i++)
  {
    int index = bins.indices[i];
    while (index > input.DocIndices[iDocIndices])
    {
      iDocIndices++;
      if (iDocIndices >= input.TotalCount)
        goto end;
    }
    if (index == input.DocIndices[iDocIndices])
    {
      double output = input.Outputs[iDocIndices];
      int featureBin = bins.values[i];
      SumTargetsByBin[featureBin] += output;
      totalOutput += output;
      CountByBin[featureBin]++;
      totalCount++;
      iDocIndices++;
      if (iDocIndices >= input.TotalCount)
        break;
    }
  }
end:
  SumTargetsByBin[bins.ZeroValue()] += input.SumTargets - totalOutput;
  CountByBin[bins.ZeroValue()] += input.TotalCount - totalCount;
}
```

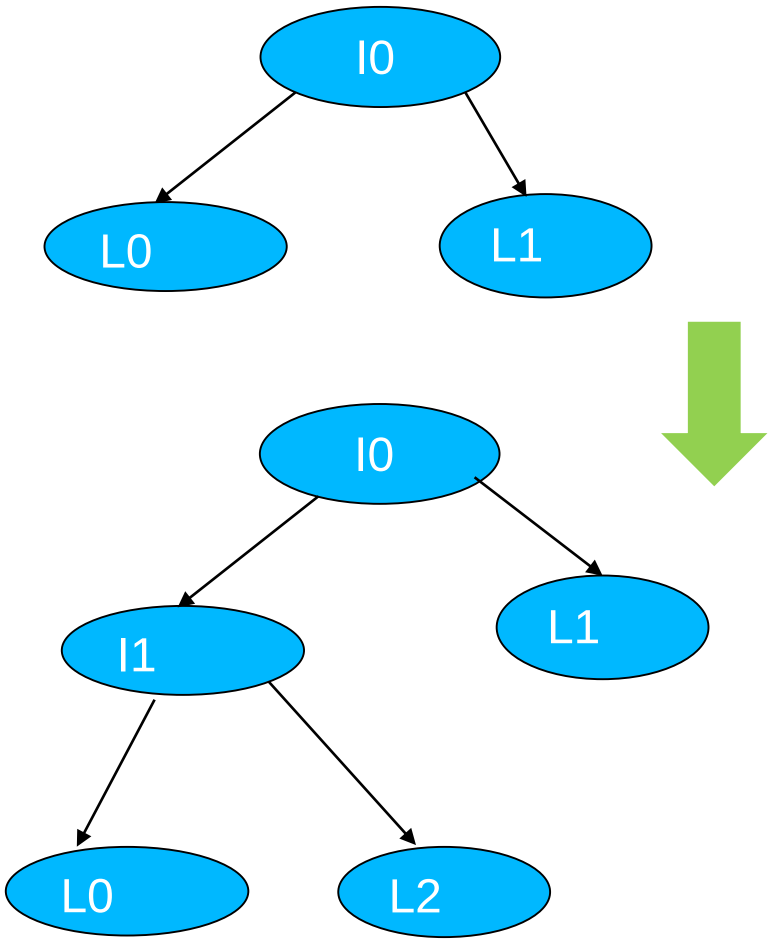
# Melt GBDT的二叉树表示

- 采用数组表示二叉树
  - 右子树gtChild,左子树lteChild,leafValue, parent,splitFeature,threThold,splitGain…

```
int GetLeaf(const FeatureBin& featureBin)
{
   if (NumLeaves == 1)
   {
      return 0;
   }
   int node = 0;

   while (node >= 0)
   {
      if (featureBin[_splitFeature[node]] <= _threshold[node])
      {
         node = _lteChild[node];
      }
      else
      {
         node = _gtChild[node];
      }
   }
   return ~node;
}
```

# Melt GBDT的二叉树 节点分裂




| Index | 0 | 1 | 2 |
|---|---|---|---|
| lteChild | -1 | | |
| gtChild | -2 | | |
| leafValue | v0 | v1 | |

| | | | |
|---|---|---|---|
| lteChild | 1 | -1 | |
| gtChild | -2 | -3 | |
| leafValue | v0 | v1 | v2 |

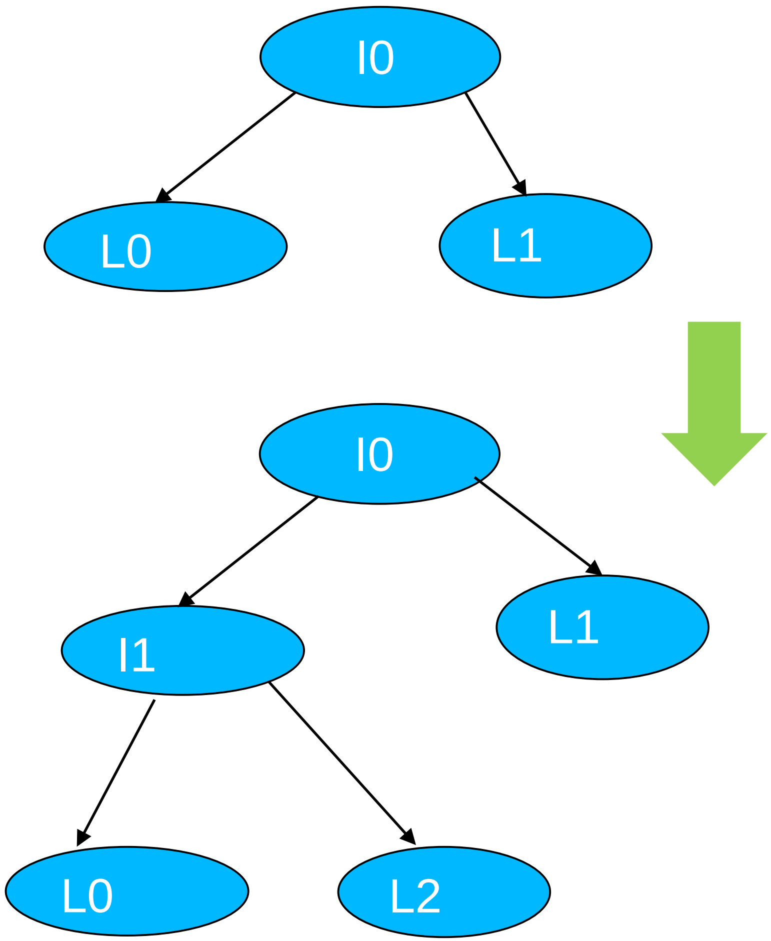
# 使用OPENMP进行并行加速

```
#pragma omp parallel for
        for (int query = 0; query < Dataset.NumDocs; query++)
        {
          if ((query % _gradSamplingRate) == sampleIndex)
          {
            GetGradientInOneQuery(query, scores);
          }
        }
```

```
#pragma omp parallel for
      for (int featureIndex = 0; featureIndex < TrainData.NumFeatures; featureIndex++)
      {
        if (IsFeatureOk(featureIndex))
          FindBestThresholdForFeature(featureIndex);
      }
```

```
#pragma omp parallel for
        for (int d = 0; d < dataset.NumDocs; d++)
        {
          int leaf = tree.GetLeaf(dataset.GetFeatureBinRow(d));
#pragma omp critical
          {
            perLeafDocumentLists[leaf].push_back(d); //注意可能并行
          }
        }
```

# 核心优化-避免重复计算直方图




1.假如上图L0中对应Feature f分裂无收益，那么后续也不对f计算

2.判断上图L0的分裂收益的时候计算了I1直方图，那么后面对L0，L2，判断哪一个样本数目少，对少的计算直方图，假设L0小，那么L2计算直方图只需要hist(I1)-hist(L0)