GBDT、TreeBoost 和 XGBoost

树模型的进化之路

颜发才

facai.yan@gmail.com

facaiy.github.io

新浪微博算法平台

2017 年 3 月 11 日

目录

决策树
- 直观印象
- 进化分支

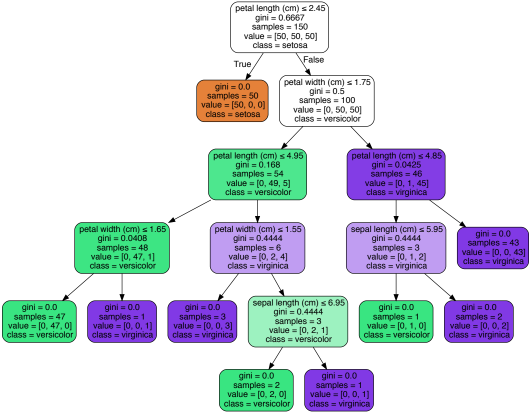


图: 决策树示意[1]

[1] Decision trees of iris data, scikit-learn

集成方法（Ensemble Method）

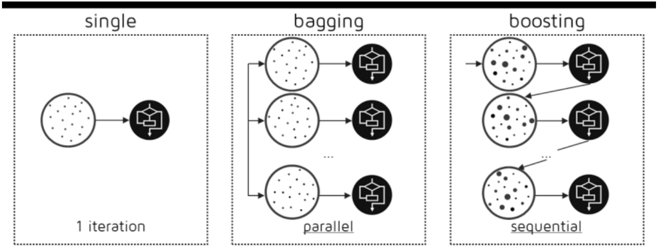


图: Bagging 和 Boosting 示意[2]

[2]What is the difference between Bagging and Boosting, xristica

Adaptive Boosting (AdaBoost)

图: AdaBoost，啊打，Boost![3]

[3]甄子丹对打李小龙全集视频，视觉网

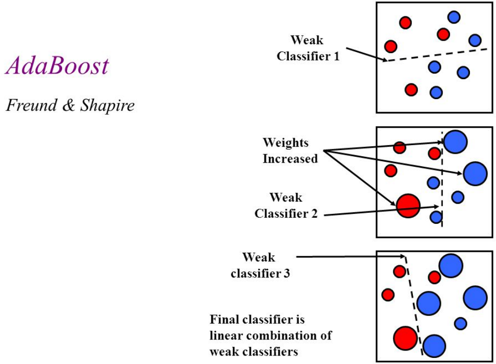


图: AdaBoost 训练示意[4]

[4]The Viola/Jones Face Detector (2001) (Most slides from Paul Viola)

Gradient Boost Decision Tree (GBDT)

- 直观印象
- 算法流程
- 从最优化角度的理解
- 从泛函角度的理解
- 从降维角度的理解
- spark 实现代码

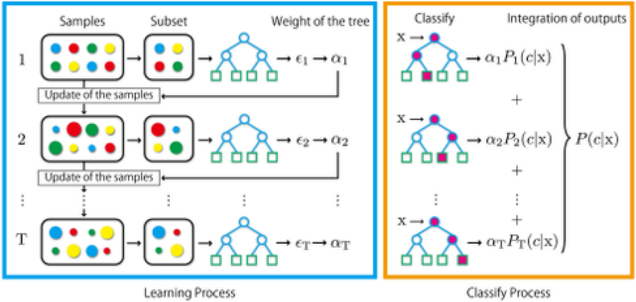


图: GBDT 示意[5]

[5] Boosted Random Forest and Transfer Learning

Algorithm 1: Gradient_Boost

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

for $m = 1$ **to** M **do**

$\quad \tilde{y} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, \quad i = 1, 2, \ldots, N$

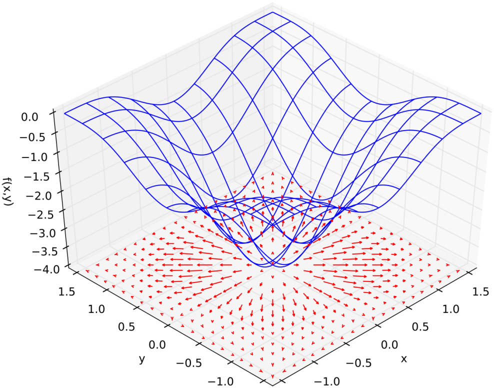
$\quad \mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} \left[\tilde{y}_i - \beta h(x_i; \mathbf{a})\right]^2$

$\quad \rho_m = \arg\min_\rho \sum_{i=1}^{N} L\left(y_i, F_{m-1}(x_i) + \rho h(x_i; \mathbf{a}_m)\right)$

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

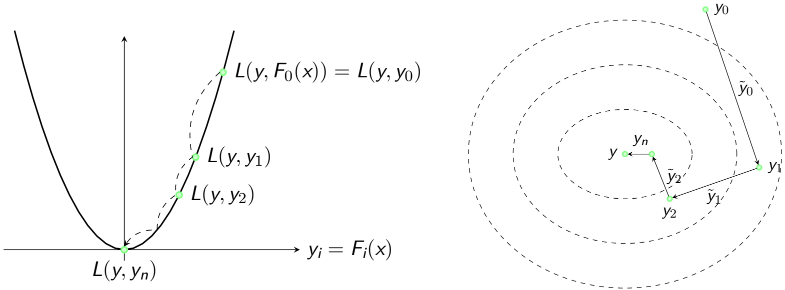
end

Greedy function approximation: A gradient boosting machine, Jerome H. Friedman

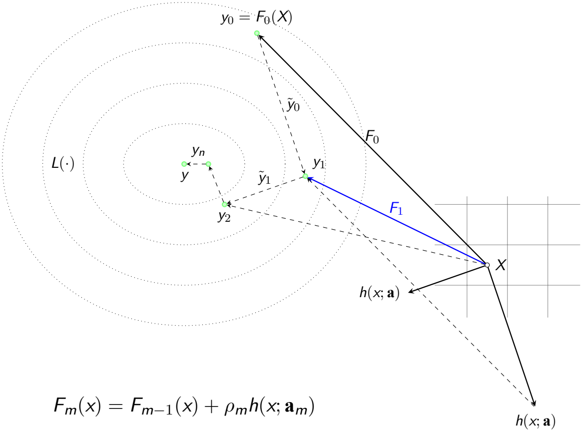


图：损失函数示意[6]

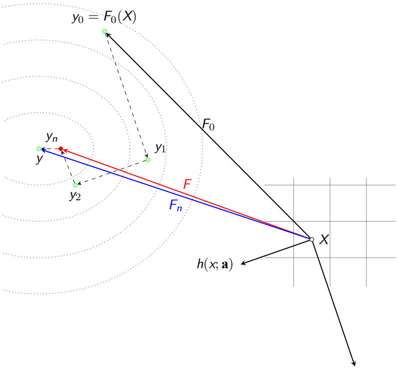
[6] Gradient Visual, Wikipedia


$L(y, F_0(x)) = L(y, y_0)$
$L(y, y_1)$
$L(y, y_2)$
$L(y, y_n)$
$y_i = F_i(x)$
y_0
$\tilde{y}_0$
y_1
$\tilde{y}_1$
y_2
$\tilde{y}_2$
y_n
y

(a) (b)



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



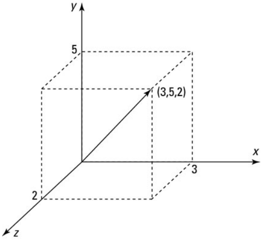


图: 三维向量空间[7]

[7] $\mathbf{e}_x = (1, 0, 0), \quad \mathbf{e}_y = (0, 1, 0), \quad \mathbf{e}_z = (0, 0, 1)$

泰勒展开：$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x-a)^n$

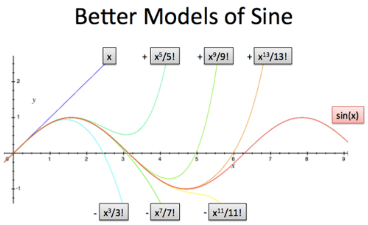
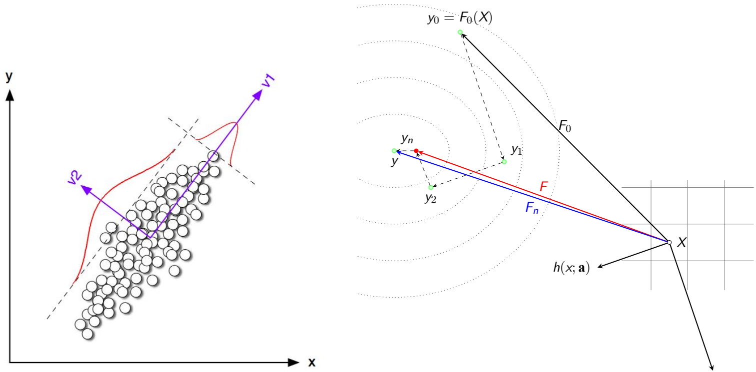


图: sin 函数泰勒展开示意[8]

GBDT: $F_n(x) = F_0(x) + \sum_n \rho_m h(x; \mathbf{a}_m)$

[8]Intuitive Understanding of Sine Waves



(a) (b)

图: PCA[9]比较示意

[9]Tutorial: Principal Components Analysis (PCA)

```scala
def boost(
  // +-- 46 lines: input: RDD[LabeledPoint],-------------------------------
  val firstTree = new DecisionTreeRegressor().setSeed(seed)
  val firstTreeModel = firstTree.train(input, treeStrategy)
  val firstTreeWeight = 1.0
  baseLearners(0) = firstTreeModel
  baseLearnerWeights(0) = firstTreeWeight
  // +-- 17 lines: var predError: RDD[(Double, Double)] =------------------
  while (m < numIterations && !doneLearning) {
    // Update data with pseudo-residuals
    val data = predError.zip(input).map { case ((pred, _), point) =>
    LabeledPoint(-loss.gradient(pred, point.label), point.features)
    }
    // +-- 5 lines: timer.start(s"building tree $m")----------------------
    val dt = new DecisionTreeRegressor().setSeed(seed + m)
    val model = dt.train(data, treeStrategy)
    baseLearners(m) = model
    baseLearnerWeights(m) = learningRate

    predError = updatePredictionError(
    input, predError, baseLearnerWeights(m), baseLearners(m), loss)
    // +-- 21 lines: predErrorCheckpointer.update(predError)---------------
    m += 1
  }
}
```

source: spark/ml/tree/impl/GradientBoostedTrees.scala
commit: 2eedc00b04ef8ca771ff64c4f834c25f835f5f44

TreeBoost
- 直观印象
- 算法推导
- 常见的损失函数
- sklearn 实现代码

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} \left[\tilde{y}_i - \beta h(x_i;\mathbf{a})\right]^2$$
$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L\left(y_i, F_{m-1}(x_i) + \rho h(x_i;\mathbf{a}_m)\right)$$

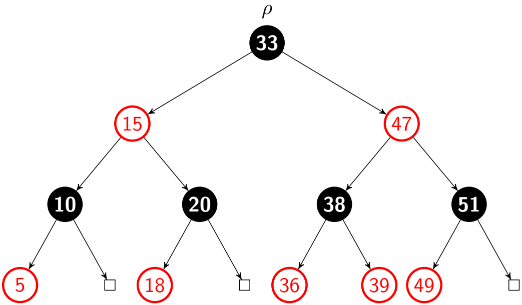
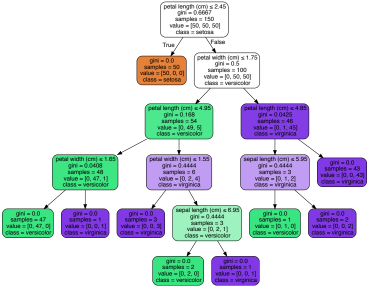


图: Tree boost 示意[10]

[10] Red-black tree, Madit

J-叶子树模型

$$h(x; \{b_j, R_j\}_1^J) = \sum_{j=1}^J b_j \mathbf{1}(x \in R_j)$$



$$\begin{aligned}
\rho_m &= \arg\min_\rho \textstyle\sum_{i=1}^N L\left(y_i, F_{m-1}(x_i) + \rho h(x_i; \mathbf{a}_m)\right) \\
&= \arg\min_\rho \textstyle\sum_{i=1}^N L\left(y_i, F_{m-1}(x_i) + \rho \sum_{j=1}^J b_j \mathbf{1}(x \in R_j)\right) \\
&= \arg\min_\rho \textstyle\sum_{i=1}^N L\left(y_i, F_{m-1}(x_i) + \sum_{j=1}^J \rho b_j \mathbf{1}(x \in R_j)\right)
\end{aligned}$$

$$\{\gamma_{jm}\}_1^J = \arg\min_{\{\gamma_j\}_1^J} \textstyle\sum_{i=1}^N L\left(y_i, F_{m-1}(x_i) + \sum_{j=1}^J \gamma_j \mathbf{1}(x \in R_{jm})\right)$$

$$\gamma_{jm} = \arg\min_\gamma \textstyle\sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$$

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$$

L2

$$\begin{aligned}\gamma_{jm} &= \arg\min_{\gamma} \sum_{x_i \in R_{jm}} (y_i, F_{m-1}(x_i) + \gamma)^2 \\ &= \mathrm{Ave}(y - F_{m-1}(x))\end{aligned}$$

L1

$$\gamma_{jm} = \mathrm{median}_W \left\{ \frac{y_i - F_{m-1}(x_i)}{h(x_i; \mathbf{a}_m)} \right\}_1^N$$

详细推导可见：TreeBoost 原理和实现（sklearn）简介，颜发才

```
    y_pred = self._decision_function(X)

    def _fit_stage(self, i, X, y, y_pred, sample_weight, sample_mask,
                   random_state, X_idx_sorted, X_csc=None, X_csr=None):
        for k in range(loss.K):
            if loss.is_multi_class:
                y = np.array(original_y == k, dtype=np.float64)

            residual = loss.negative_gradient(y, y_pred, k=k,
                                    sample_weight=sample_weight)
            tree = DecisionTreeRegressor(
                criterion='friedman_mse',
                splitter='best',
                presort=self.presort)

                tree.fit(X_csc, residual, sample_weight=sample_weight,
                         check_input=False, X_idx_sorted=X_idx_sorted)
            # update tree leaves
                loss.update_terminal_regions(tree.tree_, X, y, residual, y_pred,
                                             sample_weight, sample_mask,
                                             self.learning_rate, k=k)
            self.estimators_[i, k] = tree
        return y_pred
```

source: scikit-learn/sklearn/ensemble/gradient_boosting.py
commit: d161bfaa1a42da75f4940464f7f1c524ef53484f

XGBoost

- 思路来源
- 具体推导
- 重要参数

GBDT，每次迭代可描述成最优问题：

$$
\begin{aligned}
f_m &= \arg\min_f \sum_{i=1}^{n} L\left(y_i, \hat{y}_i + f(x_i)\right) \\
&= \arg\min \mathcal{L}(f)
\end{aligned}
$$

泰勒展开

$$\mathcal{L}(f) \approx \sum_{i=1}^{n} \left[L(y_i, \hat{y}_i) + g_i f(x_i) + \frac{1}{2} h_i f^2(x_i) \right] + \Omega(f)$$

$$g_i = \frac{\partial\, L(y_i, \hat{y}_i)}{\partial \hat{y}_i}$$

$$h_i = \frac{\partial^2\, L(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$$

$$\mathcal{L}(f) = \sum_{j=1}^{J} \left((\sum_{i \in I_j} g_i) b_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) b_j^2 \right) + \gamma \| R_j \|$$

详细推导可见：xgboost 的基本原理与实现简介，颜发才

叶子值

$$
\begin{aligned}
b_j &= \arg\min_{b_j} \mathcal{L} \\
&= \arg\min_{b_j} \sum_{j=1}^{J} \left((\sum_{i \in I_j} g_i) b_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) b_j^2 \right) + \gamma \| R_j \| \\
&= \sum_{j=1}^{J} \arg\min_{b_j} \left((\sum_{i \in I_j} g_i) {\color{red} b_j} + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) {\color{red} b_j^2} \right) + \gamma \| R_j \|
\end{aligned}
$$

$$
b_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}
$$

不纯性度量 (impurity)

$$
\begin{aligned}
\mathcal{L} &= -\frac{1}{2}\sum_{j=1}^{J}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i+\lambda}+\gamma\|R_j\| \\
&= -\frac{1}{2}H+\gamma T
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{L}_{\text{split}} &= \mathcal{L}-\mathcal{L}_L-\mathcal{L}_R \\
&= \frac{1}{2}(H_L+H_R-H)+\gamma(T-T_L-T_R) \\
&= \frac{1}{2}(H_L+H_R-H)-\gamma
\end{aligned}
$$

最终，树生成公式：

$$\mathcal{L} = -\frac{1}{2}H + \gamma T$$

$$b_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

```
void UpdateOneIter(int iter, DMatrix* train) override {
   CHECK(ModelInitialized())
      << "Always call InitModel or LoadModel before update";
   if (tparam.seed_per_iteration || rabit::IsDistributed()) {
      common::GlobalRandom().seed(tparam.seed * kRandSeedMagic + iter);
   }
   this->LazyInitDMatrix(train);
   this->PredictRaw(train, &preds_);
   obj_->GetGradient(preds_, train->info(), iter, &gpair_);
   gbm_->DoBoost(train, this->FindBufferOffset(train), &gpair_);
}
```

source: xgboost/src/learner.cc
commit: 49bdb5c97fccd81b1fdf032eab4599a065c6c4f6

- XGBoost Parameters
  - objective
    reg:linear, binary:logistic, multi:softmax
  - num_round, max_depth
  - eta
  - lambda (L2 reg), alpha (L1 reg)
- Notes on Parameter Tuning
- 稀疏数据：0 和 missing

总结

总结

- CART: 统一回归和分类问题
- AdaBoost：加权
- GBDT：残差
- TreeBoost：叶子权重
- XGBoost：损失函数指导决策树

谢谢！

GBDT、TreeBoost 和 XGBoost

树模型的进化之路

颜发才

facai.yan@gmail.com
facaiy.github.io

新浪微博算法平台

2017 年 3 月 11 日