

树算法 原理与实现

04/04/2016 更新

1 Classification And Regression Trees

1.1 算法基础

CART(Classification And Regression Trees) 可以用作分类和回归，是一种贪心算法：它将特征空间进行递归地划分，切割成若干块区域，当对预测集的样本点进行预测时，先根据样本特征判断该样本点位于哪块区域，然后用该区域内的训练样本点集的众数 (分类) 或均值 (回归) 作为该样本点的预测值。如图 (1) 所示。

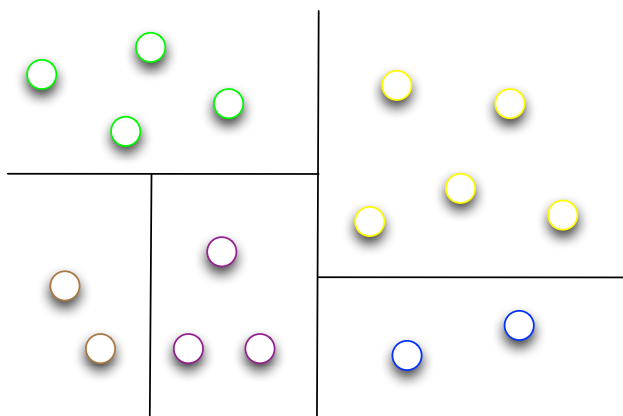


图 1: CART 算法

1.1.1 构造过程

CART 模型的构造采用的是二分递归分割的方法，从根节点开始“生长”。在每个节点处 (初始只有一个根节点)，选取某维特征的某个特征值作为分裂阈值，将当前节点所包含的样本集分成两个子样本集 (根节点包含所有的训练样本)，形成两个子节点，再对子节点采用同样的方法继续分割，直到不满足分割条件为止。如图 (2) 所示。

因此，CART 模型是一棵结构简单的二叉树。

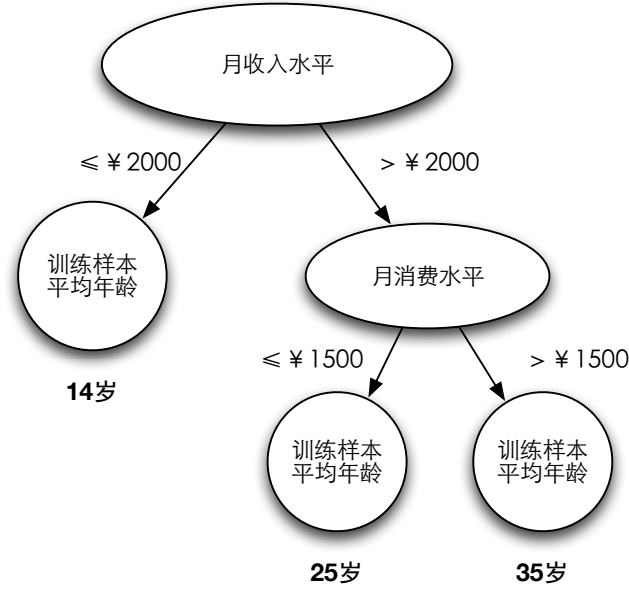


图 2: CART 回归模型构造过程

1.1.2 节点纯度

在 CART 模型的构造过程中，每次节点的分类都会生成两个新的节点。对于分类和回归，我们使用不同的标准来衡量每个节点的优劣程度（不纯度，Impurity）。不纯度越大，则对应节点越混乱，质量越差。

当 CART 用作分类的时候，我们使用每个节点所对应的训练样本集合的基尼不纯度 (Gini Impurity) 来衡量该节点的好坏。节点 N 的方差不纯度的计算方法如公式 (1) 所示。

$$I_g(N) = \sum_{i=1}^K f_i(1 - f_i) \quad (1)$$

其中， K 表示训练样本中的标签总数， f_i 表示该节点所对应的训练样本中第 i 个标签所占的比例。

当 CART 用做回归的时候，我们使用每个节点所对应的训练样本集合的方差 (Varicance) 作为该节点的不纯度。节点 N 的方差不纯度的计算方法如公式 (2) 所示。

$$I_v(N) = \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2} (y_i - y_j)^2 \quad (2)$$

其中， S 表示节点 N 所包含的训练样本集， y_i 表示节点 N 所包含的训练样本集中的第 i 个样本点的 *label* 值。

在实现的过程中，为了提高算法的效率，采用了公式 (2) 的另一种表示形式，如公式 (3) 所示。

$$I_v(N) = \frac{1}{|S|} \left(\sum_{i \in S} y_i^2 - \frac{1}{|S|} \sum_{i \in S} y_i^2 \right) \quad (3)$$

1.1.3 信息增益

CART 模型的每个节点在分裂时，需要确定分裂使用的特征及对应的特征值。我们使用信息增益 (Information Gain) 来评价分裂时选用不同特征及特征值的优劣程度，从而做出最佳选择。信息增益的计算方法如公式 (4) 所示。

$$IG(N) = I(N) - \left(\frac{|S_1|}{|S|} I(N_1) + \frac{|S_2|}{|S|} I(N_2) \right) \quad (4)$$

其中， N_1 、 N_2 表示分裂过程中由 N 新生成的两个子节点， S 表示节点 N 所包含的训练样本集， S_1 表示子节点 N_1 所包含的训练样本集， S_2 表示子节点 N_2 所包含的训练样本集。

1.1.4 终止条件

CART 算法中，节点在遇到如下情况的时候终止分裂，形成叶子节点：

- 节点分裂带来的信息增益小于用户指定的最小信息增益 (min_info_gain)。
- 节点树深大于等于用户指定的最大树深 (max_depth)。
- 节点权重小于等于用户指定的节点最小权重 (min_node_size)。

1.1.5 叶节点预测值

CART 模型的每个叶子节点对应一个预测值，分类和回归采用不同的方法得到该预测值。

当 CART 算法用作分类时，叶节点预测值的计算方法如公式 (5) 所示。

$$P_i = \arg \max_k \{c_{ik}\}, k = 1, K \quad (5)$$

其中， P_i 表示第 i 个叶节点的分类预测值， c_{ik} 表示第 i 个叶节点中 label 为 k 的训练样本个数。

当 CART 算法用作回归时，叶节点预测值的计算方法如公式 (6) 所示。

$$P_i = \frac{1}{|S_i|} \sum_{j \in S_i} y_{ij} \quad (6)$$

其中, P_i 表示第 i 个叶子节点的回归预测值, S_i 表示第 i 个叶子节点上的训练样本集, y_{ij} 表示第 i 个叶子节点所包含的训练样本集中的第 j 个样本点的 *label* 值。

1.2 分布式实现

1.2.1 特征值分箱

为了使模型能够适应大规模数据的需求, 在为节点分裂选择合适的特征及阈值的过程中, 算法没有遍历每个特征的所有特征值, 而是先对训练样本采样, 将采样的训练样本中每个特征的特征值尽可能均匀的分箱 (等频分箱, 箱中的训练样本个数尽可能相等), 使用箱与箱之间的边界值作为候选该特征的候选分裂值。如图 (3) 所示。

| | Bin #1 | Bin #2 | Bin #3 |
|-------|------------|---------------------|---------------|
| 特征 #1 | $f1 < a$ | $a \leq f1 < b$ | $b \leq f1$ |
| 特征 #2 | $f2 < a'$ | $a' \leq f2 < b'$ | $b' \leq f2$ |
| 特征 #3 | $f3 < a''$ | $a'' \leq f3 < b''$ | $b'' \leq f3$ |

图 3: 特征值分箱

1.2.2 节点分裂

在算法的分布式实现中, 每次会生成新的一层节点作为待分裂节点 (Splitting Nodes), 这些待分裂节点的最佳分裂特征及分裂阈值的选择并行完成。如图 (4) 所示。

在 Spark 中, 训练数据集以 RDD 的形式存储, 并被切分为不同的分区 (Partition)。为了确定每个待分裂节点的最佳分裂特征及分裂阈值, 为每个待分裂节点在各个训练数据集分区分配一个状态累加器 (Impurity Aggregator), 用以统计位于该分区的训练样本集对不同分裂节点的影响。统计完成后, 同一待分裂节点位于不同分区的状态累加器进行合并 (Merge), 得到一个用于描述该分裂节点总体状态的累加器。根据每个待分裂节点的最终状态累加器, 确定最佳分裂特征及对应的阈值, 从而实现并行化。如图 (5) 所示。

当 CART 用作分类的时候, 状态累加器中记录各维特征各个分箱中所包含的训练样本个数 (count) 以及各 label 数目 (count for label#i)。

当 CART 用作回归的时候, 状态累加器中记录各维特征各个分箱中所包含的训练样本个数 (count)、label 之和 (sum) 以及 label 的平方和 (squared sum)。

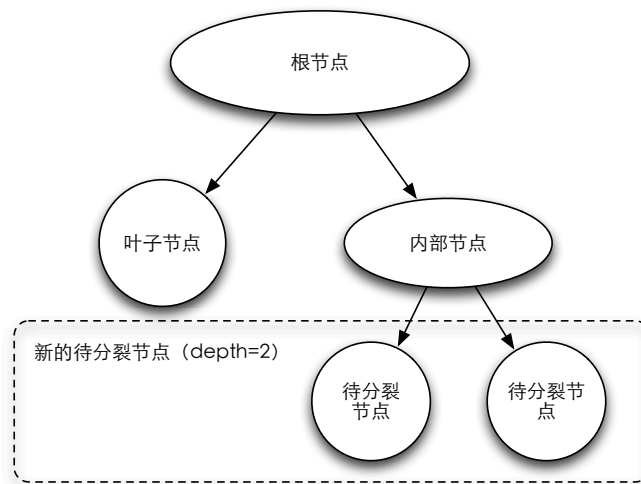


图 4: CART 模型构建过程

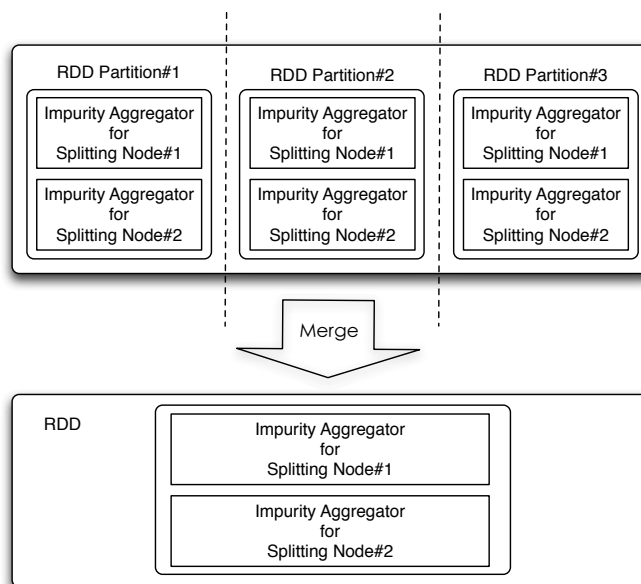


图 5: 待分裂节点状态统计

1.3 使用示例

1.3.1 分类

```
1 // read training data and testing data from disk
2 val train = Points.readLibSVMFile(sc, data_dir + "a1a")
3 val test = Points.readLibSVMFile(sc, data_dir + "a1a.t")
4
5 train.cache()
6 test.cache()
7
8 // train a model of CART for classification
9 val cart_model = CART.train(
10   train,
11   impurity = "Gini",
12   max_depth = 10,
13   max_bins = 32,
14   bin_samples = 10000,
15   min_node_size = 15,
16   min_info_gain = 1e-6)
17
18 // show struct of CART model
19 cart_model.printStructure()
20
21 // preict for testing data use the model
22 val preds = cart_model.predict(test)
23 // calculate testing error
24 val err = preds.filter(r => r._1 != r._2).count().toDouble / test.count()
25 println("Test Error = " + err)
```

RunCARTForClassificationDemo.scala

1.3.2 回归

```
1 // read training data and testing data from disk
2 val train = Points.readLibSVMFile(sc, data_dir + "cadata.train")
3 val test = Points.readLibSVMFile(sc, data_dir + "cadata.test")
4
5 train.cache()
6 test.cache()
7
8 // train a model of CART for regression
9 val cart_model = CART.train(
10   train,
11   impurity,
12   max_depth,
13   max_bins,
14   bin_samples,
15   min_node_size,
16   min_info_gain)
17
```

```

18 // show struct of CART model
19 cart_model.printStructure()
20
21 // preict for testing data use the model
22 val preds = test.map {
23     p =>
24         (p.label, cart_model.predict(p.fs))
25 }
26 // calculate testing error
27 println(s"Test RMSE: ${RMSE(preds)}")

```

RunCARTForRegressionDemo.scala

2 GBRT

2.1 算法基础

GBRT(Gradient Boosting Regression Trees) 是迭代的回归树算法。该算法模型由多棵回归树构成，所有回归树的预测结果的累加值即为最终的预测值。GBRT 是一种泛化能力 (Generalization) 很强的算法。

2.1.1 回归树

回归树是 GBRT 的基本组成单位，用来做回归分析。回归树在上一节进行了详细的说明，此处不再赘述。

2.1.2 梯度迭代

GBRT 的每一轮迭代都会生成一棵新的回归树，将所有训练得到的回归树的预测值按一定方式累加作为预测值，与真实值一起代入损失函数并求预测值偏导，得到的数据作为下一轮的输入，继续迭代。

伪代码 (1) 详细说明了 GBRT 算法中梯度迭代的执行过程，即如何构建回归树模型并将其进行组合。

2.1.3 缩减思想

缩减 (Shrinkage) 的思想认为，在模型的迭代过程中，每次走一小步逐渐逼近结果的效果，要比每次都一大步很快逼近结果的方式更容易避免过拟合，同时也更容易收敛。

也就是说，我们不能完全信任每一棵决策树，每一棵决策树只能学到真理的一小部分，累加的时候只累加一小部分，通过多棵决策树来弥补不足。

Algorithm 1 Gradient tree boosting for multiple additive regression trees

```
1:  $f_0 \leftarrow \operatorname{argmin}_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1 \rightarrow M$  do
3:   for  $i = 1 \rightarrow N$  do
4:      $r_{im} \leftarrow - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$ 
5:   end for
6:   Fit a regression tree to the targets  $r_{im}$  giving terminal regions
      $R_{jm}, j = 1, 2, \dots, J_m$ .
7:   for  $j = 1 \rightarrow J_m$  do
8:      $\gamma_{jm} \leftarrow \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$ 
9:   end for
10:   $f_m(x) \leftarrow f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ 
11: end for
12: Output  $\hat{f}(x) \leftarrow f_M(x)$ 
```

因此，对伪代码 (1) 第 (10) 行的公式进行修改，如公式 (7) 所示。其中， ρ_m 通常被称作学习率 (Learning Rate)。

$$f_m(x) \leftarrow f_{m-1}(x) + \rho_m \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}) \quad (7)$$

2.2 使用示例

```
1 // read training data and testing data from disk
2 val train = Points.readLibSVMFile(sc, data_dir + "cadata.train")
3 val test = Points.readLibSVMFile(sc, data_dir + "cadata.test")
4
5 train.cache()
6 test.cache()
7
8 // train a model of decision tree
9 val model: GradientBoostModel = GradientBoost.train(
10   train_data = train,
11   valid_data = null,
12   impurity = "Variance",
13   loss = "SquaredLoss",
14   max_depth = 10,
15   max_bins = 32,
16   min_samples = 10000,
17   min_node_size = 15,
18   min_info_gain = 1e-6,
19   row_rate = 0.6,
20   col_rate = 0.6,
21   num_iter = 20,
22   learn_rate = 0.02,
23   min_step = 1e-6,
```



```

24   silent = false)
25
26   // predict for testing data use the model
27   val predictions = model.predict(test).zip(test).map {
28     case (y, pn) => s"$y\t$pn"
29   }
30   // save predictions on disk
31   predictions.saveAsTextFile(prediction_pt)
32
33   // save the model on disk
34   model.save(sc, model_pt)

```

RunGradientBoostDemo.scala

3 随机森林

3.1 基本思想

随机森林算法生成很多棵回归树，当对新的样本进行预测的时候，随机森林中的每一棵树都会给出自己的预测值，并由此进行“投票”，在回归问题中，随机森林的输出是所有回归树输出的平均值。

3.1.1 随机采样

随机森林模型的构建中，存在两个随机过程：对样本的有放回采样和对特征的无放回采样。

- 对样本的有放回采样，用来训练每一棵回归树。
- 对特征的无放回采样，用做回归树节点的每一次分裂。

3.1.2 伪代码

随机森林的执行过程如伪代码 (2) 所示。

3.2 使用示例

```

1  // read training data and testing data from disk
2  val train = Points.readLibSVMFile(sc, data_dir + "cadata.train")
3  val test = Points.readLibSVMFile(sc, data_dir + "cadata.test")
4
5  train.cache()
6  test.cache()
7
8  // train a model of decision tree
9  val model: RandomForestModel = RandomForest.train(
10    train_data = train,
11    valid_data = null,

```

Algorithm 2 Random forest

```
1: for  $m = 1 \rightarrow M$  do
2:   Sample, with replacement from  $X, Y$ ; call these  $X_m, Y_m$ .
3:   Fit a regression tree to the targets  $y_{im} (y_{im} \in Y_m)$  giving terminal re-
     gions  $R_{jm}, j = 1, 2, \dots, J_m$  (Sample features without replacement during
     nodes splitting of the regression tree).
4:   for  $j = 1 \rightarrow J_m$  do
5:      $\gamma_{jm} \leftarrow \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$ 
6:   end for
7:    $f_m(x) \leftarrow f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ 
8: end for
9: Output  $\hat{f}(x) \leftarrow \frac{1}{M} f_M(x)$ 
```

```
12 | impurity = "Variance",
13 | loss = "SquaredLoss",
14 | max_depth = 10,
15 | max_bins = 32,
16 | min_samples = 10000,
17 | min_node_size = 15,
18 | min_info_gain = 1e-6,
19 | row_rate = 0.6,
20 | col_rate = 0.2,
21 | num_trees = 20,
22 | silent = false)
23 |
24 | // predict for testing data use the model
25 | val predictions = model.predict(test).zip(test).map {
26 |   case (y, pn) => s"$y\t$pn"
27 | }
28 | // save predictions on disk
29 | predictions.saveAsTextFile(prediction_pt)
30 |
31 | // save the model on disk
32 | model.save(sc, model_pt)
```

RunRandomForestDemo.scala