树算法 原理与实现

BDA 小组，侯建鹏
中国科学院计算技术研究所

04/21/2016 更新

目录

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。

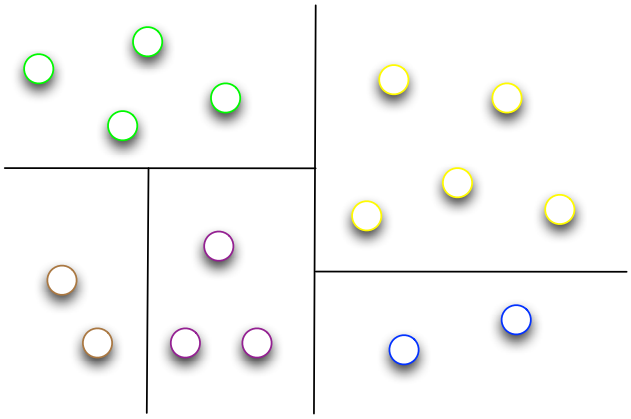

图 1: CART 算法

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。

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。

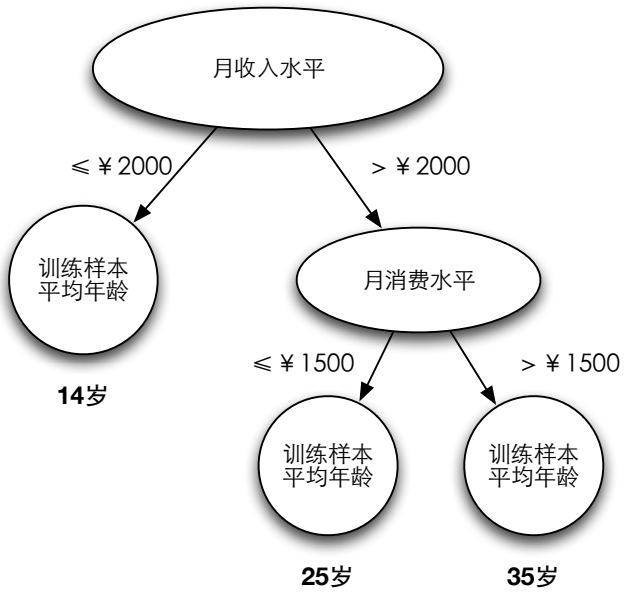

图 2: CART 回归模型构造过程

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

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。

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

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。

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

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。

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

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。

$$P_i = arg\ max_k\{c_{ik}\}, k = 1, K \tag{5}$$

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。

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

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。

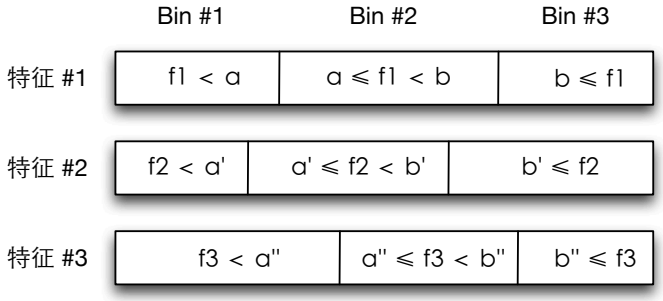

图 3: 特征值分箱

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。

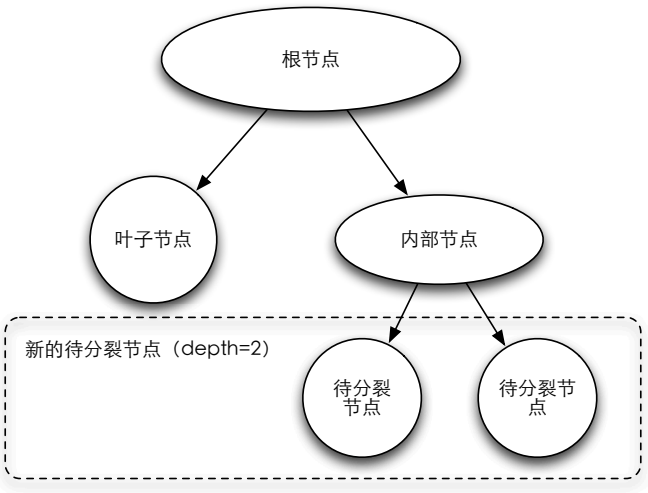

图 4: CART 模型构建过程

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 (ImpurityAggregator)，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。

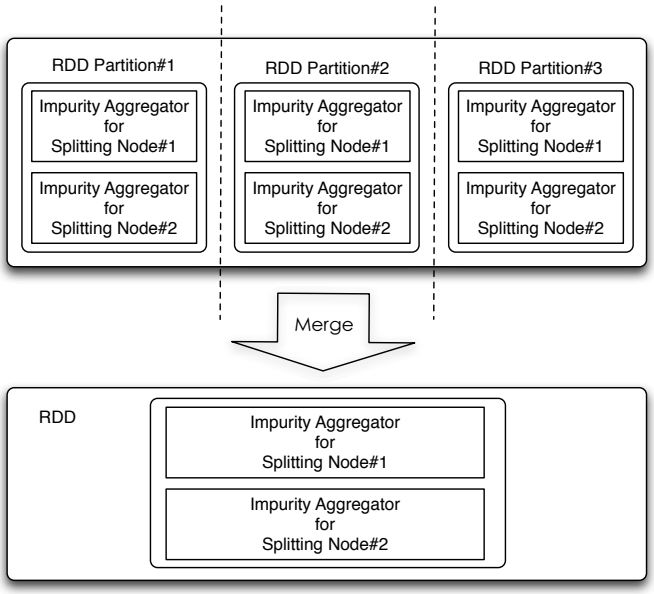

图 5: 待分裂节点状态统计

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)。

1.3 使用示例

1.3.1 分类

```
// read training data and testing data from disk
val train = Points.readLibSVMFile(sc, data_dir + "a1a").cache()
val test = Points.readLibSVMFile(sc, data_dir + "a1a.t").cache()

// train a model of CART for classification
val cart_model = CART.train(
  train,
  impurity = "Gini",
  max_depth = 10,
  max_bins = 32,
```

```
    bin_samples = 10000,
    min_node_size = 15,
    min_info_gain = 1e-6)

// show structure of CART model
cart_model.printStructure()

// predict for testing data using the model
val preds = cart_model.predict(test)
// calculate testing error
val err = preds.filter(r => r._2 != r._3).count().toDouble / test.count()
println(s"Test Error: $err")
```

RunCARTForClassification.scala

1.3.2 回归

```
// read training data and testing data from disk
val train = Points.readLibSVMFile(sc, data_dir + "cadata.train").cache()
val test = Points.readLibSVMFile(sc, data_dir + "cadata.test").cache()

// train a model of CART for regression
val cart_model = CART.train(
    train,
    impurity = "Variance",
    max_depth = 15,
    max_bins = 32,
    bin_samples = 10000,
    min_node_size = 10,
    min_info_gain = 1e-6)

// show structure of CART model
cart_model.printStructure()

// predict for testing data use the model
val preds = cart_model.predict(test)
// calculate testing error
println(s"Test RMSE: ${RMSE(preds.map(e => (e._2, e._3)))}")
```

RunCARTForRegression.scala

2 Gradient Boosting Decision Trees

2.1 算法基础

GBDT(Gradient Boosting Decision Trees) 算法用来解决 K 分类问题。它是一种梯度迭代算法 (Gradient Boosting)，每轮迭代的过程中会产生 K 棵回归树。

2.1.1 损失函数

使用 Log 对数损失作为损失函数，如 (7) 所示。

$$L\left(\{y_k, F_k\left(\mathrm{x}\right)\}_1^K\right) = -\sum_{k=1}^{K} y_k \log p_k(\mathrm{x}) \tag{7}$$

其中，$y_k = 1(class = k) \in \{0, 1\}$ 且 $p_k\left(\mathrm{x}\right) = Pr(y_k = 1|\mathrm{x})$。采用 logistic 变换得到概率函数，如 (8) 所示。

$$p_k\left(\mathrm{x}\right) = exp\left(F_k\left(\mathrm{x}\right)\right) / \sum_{l=1}^{K} exp\left(F_l\left(\mathrm{x}\right)\right) \tag{8}$$

2.1.2 残差计算

将 (8) 代入 (7) 中，通过求一阶导得到每轮迭代后的残差 (residuals)，如 (9) 所示。

$$\tilde{y}_{ik} = -\left[\frac{\partial L\left(\{y_{il}, F_l\left(\mathrm{x}_i\right)\}_{l=1}^K\right)}{\partial F_k\left(\mathrm{x}_i\right)}\right]_{\left\{F_l = F_{l,m-1}(\mathrm{x})\right\}_1^K} = y_{ik} - p_{k,m-1}\left(\mathrm{x}_i\right) \tag{9}$$

2.1.3 残差拟合

每一轮迭代中，通过使用 K 棵回归树拟合 $\tilde{y}_{ik}$ 来预测每个类别的残差。设每棵回归树有 J 个叶子节点，对应的样本区域为 $\{R_{jkm}\}_{j=1}^J$，以及每个区域的预测值为 $\{\gamma_{jkm}\}_{j=1}^J$。区域的预测值通过式 (10) 求解。

$$\{\gamma_{jkm}\} = \arg\min_{\{\gamma_{jk}\}} \sum_{i=1}^{N}\sum_{k=1}^{K} \phi\left(y_{ik}, F_{k,m-1}\left(\mathrm{x}_i\right) + \sum_{j=1}^{J} \gamma_{jk} 1\left(\mathrm{x}_i \in R_{jm}\right)\right) \tag{10}$$

其中，$\phi\left(y_k, F_k\right) = -y_k \log p_k$。

由于式 (10) 没有解析解 (closed form solution)，因此通过式 (11) 求取其近似解。

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{\mathrm{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathrm{x}_i \in R_{jkm}} |\tilde{y}_{ik}| \left(1 - |\tilde{y}_{ik}|\right)} \tag{11}$$

2.1.4 伪代码

用于 K 分类的 GBDT 算法的伪代码如 (1) 所示。

Algorithm 1 $L_K\_TreeBoost$

$F_{k0}(\mathrm{x}) = 0, k = 1 \to K$
for $m = 1 \to M$ do
  $p_k(\mathrm{x}) = exp(F_k(\mathrm{x})) / \sum_{l=1}^{K} exp(F_l(\mathrm{x})), k = 1 \to K$
  for $k = 1 \to K$ do
    $\tilde{y}_{ik} = y_{ik} - p_{k,m-1}(\mathrm{x}_i), i = 1 \to N$
    $\{R_{jkm}\}_{j=1}^{J} = J - terminal\ node\ tree\left(\{\tilde{y}_{ik}, \mathrm{x}_i\}_1^N\right)$
    $\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{\mathrm{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathrm{x}_i \in R_{jkm}} |\tilde{y}_{ik}|(1-|\tilde{y}_{ik}|)}, j = 1 \to J$
    $F_{km}(\mathrm{x}) = F_{k,m-1}(\mathrm{x}) + \sum_{j=1}^{J} \gamma_{jkm} 1(\mathrm{x} \in R_{jkm})$
  end for
end for

2.1.5 预测结果

根据伪代码 (1)，得到 $\{F_{kM}(\mathrm{x})\}_1^K$，通过式 (8) 计算得 $\{p_{kM}(\mathrm{x})\}_1^K$。最终，通过式 (12) 得到分类结果。

$$\hat{k}(\mathrm{x}) = \arg \min_{1 \leqslant k \leqslant K} \sum_{k'=1}^{K} c(k, k') p_{k'M}(\mathrm{x}) \tag{12}$$

2.2 使用示例

```
// read training data and testing data from disk
val train = Points.readLibSVMFile(sc, data_dir + "cadata.train").cache()
val test = Points.readLibSVMFile(sc, data_dir + "cadata.test").cache()

// train a model of GBDT for multiple classification
val gbdt_model = GBDT.train(train,
  impurity = "Variance",
  max_depth = 15,
  max_bins = 32,
  bin_samples = 10000,
  min_node_size = 10,
  min_info_gain = 1e-6,
  num_round = 20)

// predict for testing data using the model
val preds = gbdt_model.predict(test)
// calculate testing error
val err = preds.filter(r => r._2 != r._3).count().toDouble / test.count()
println(s"Test Error: $err")
```

RunGBDT.scala

3 Gradient Boosting Regression Trees

3.1 算法基础

GBRT(Gradient Boosting Regression Trees) 算法用来解决回归问题。它和 GBDT 类似，也是一种梯度迭代算法 (Gradient Boosting)，每轮迭代的过程中会产生一棵回归树。

3.1.1 损失函数

使用平方损失作为损失函数，如式 (13) 所示。

$$L\left(y, F\left(\mathrm{x}\right)\right) = \left(y - F\left(\mathrm{x}\right)\right)^2 / 2 \tag{13}$$

3.1.2 残差计算

通过求一阶导得到每轮迭代后的残差 (residuals)，如式 (14) 所示。

$$\tilde{y}_i = -\left[\frac{\partial L\left(y_i, F\left(\mathrm{x}_i\right)\right)}{\partial F\left(\mathrm{x}_i\right)}\right]_{F=F_{m-1}(\mathrm{x})} = y_i - F_{m-1}\left(\mathrm{x}_i\right) \tag{14}$$

3.1.3 残差拟合

每一轮迭代中，通过建立一棵回归树拟合 $\tilde{y}_i$ 来预测本轮残差。设每棵回归树有 J 个叶子节点，对应的样本区域为 $\{R_{jm}\}_{j=1}^{J}$，以及每个区域的预测值为 $\{\gamma_{jm}\}_{j=1}^{J}$。区域的预测值通过式 (15) 求解。

$$\{\gamma_{jm}\}_1^J = \arg\min_{\{\gamma_j\}_1^J} \sum_{i=1}^{N} L\left(y_i, F_{m-1}\left(\mathrm{x}_i\right) + \sum_{j=1}^{J} \gamma_j 1\left(x \in R_{jm}\right)\right) \tag{15}$$

由于决策树的 J 个样本区域互不重叠，因此区域的预测值可进一步推导，得到式 (16) 。

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\mathrm{x}_i \in R_{jm}} L\left(y_i, F_{m-1}\left(\mathrm{x}_i\right) + \gamma\right) \tag{16}$$

最终，γ_{jm} 的解析解如 (17) 所示。

$$\gamma_{jm} = mean_{\mathrm{x}_i \in R_{jm}}\{y_i - F_{m-1}\left(\mathrm{x}_i\right)\} = mean_{\mathrm{x}_i \in R_{jm}}\{\tilde{y}_i\} \tag{17}$$

Algorithm 2 *LS_TreeBoost*

1: $F_0(\mathrm{x}) = mean\{y_i\}_1^N$
2: for $m = 1 \to M$ do
3: $\quad \tilde{y}_i = y_i - F_{m-1}(\mathrm{x}_i), i = 1 \to N$
4: $\quad \{R_{jm}\}_{j=1}^J = J - terminal\ node\ tree\left(\{\tilde{y}_i, \mathrm{x}_i\}_1^N\right)$
5: $\quad \gamma_{jm} = mean_{\mathrm{x}_i \in R_{jm}}\{\tilde{y}_i\}, j = 1 \to J$
6: $\quad F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) + \sum_{j=1}^J \gamma_{jm} 1(\mathrm{x} \in R_{jm})$
7: end for

3.1.4 伪代码

用于回归的 GBRT 算法的伪代码如 (2) 所示。

3.1.5 预测结果

根据伪代码 (2) ，得到 $F_M(\mathrm{x})$，即为预测值。

3.1.6 正则化

在预测问题中，过度拟合训练集往往会事与愿违。因此，我们对算法 (2) 的第 (6) 步引入正则化 (Regularization) 来防止过拟合 (over-fitting)。如式 (18) 所示。

$$F_m(\mathrm{x}) = F_{m-1}(\mathrm{x}) + \nu \sum_{j=1}^{J} \gamma_{jm} 1(\mathrm{x} \in R_{jm}), 0 < \nu \leq 1 \tag{18}$$

3.2 使用示例

```
// read training data and testing data from disk
val train = Points.readLibSVMFile(sc, data_dir + "cadata.train").cache()
val test = Points.readLibSVMFile(sc, data_dir + "cadata.test").cache()

// train a model of GBRT for regression
val gbrt_model = GBRT.train(
  train,
  Array(("test", test)),
  impurity = "Variance",
  max_depth = 15,
  max_bins = 32,
  bin_samples = 10000,
  min_node_size = 10,
  min_info_gain = 1e-6,
  num_round = 100,
  learn_rate = 0.1)

```

```
// predict for testing data using the model
val preds = gbrt_model.predict(test)
// calculate testing error
println(s" Test RMSE: ${RMSE(preds.map(e => (e._2, e._3)))}" )
```

RunGBRT.scala

4 Random Forest

4.1 基本思想

随机森林 (Random Forest) 算法可以用来解决分类和回归问题。它以 CART 模型作为弱学习器 (weak learners)，并对它们的结果进行融合。

4.1.1 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。

4.1.2 伪代码

随机森立的执行过程如伪代码 (3) 所示。

Algorithm 3 Random forest

1: for $m = 1 \rightarrow M$ do
2: Sample, with replacement from X, Y; call these $\mathrm{X_m}$, $\mathrm{Y_m}$.
3: Fit a CART model to the targets $y_{im}(y_{im} \in \mathrm{Y_m})$ giving terminal regions $R_{jm}, j = 1, 2, ..., J_m$(Sample features without replacement during nodes splitting of the CART model).
4: for $j = 1 \rightarrow J_m$ do
5: $\gamma_{jm} \leftarrow argmin_\gamma \sum_{x_i \in R_{jm}} L(y_i, \gamma)$
6: end for
7: end for

4.1.3 预测结果

随机森林算法生成很多棵决策树 (CART)，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”。

- 对分类问题，随机森林输出所有分类树结果的众数。
- 对回归问题，随机森林输出所有回归树结果的均值。

4.2 使用示例

4.2.1 分类

```
// read training data and testing data from disk
val train = Points.readLibSVMFile(sc, data_dir + "a1a").cache()
val test = Points.readLibSVMFile(sc, data_dir + "a1a.t").cache()

// train a model of Random Forest for classification
val rf_model = RandomForest.train(
  train,
  impurity = "Gini",
  max_depth = 10,
  max_bins = 32,
  bin_samples = 10000,
  min_node_size = 15,
  min_info_gain = 1e-6,
  row_rate = 0.6,
  col_rate = 0.6,
  num_trees = 100)

// predict for testing data using the model
val preds = rf_model.predict(test)
// calculate testing error
val err = preds.filter(r => r._2 != r._3).count().toDouble / test.count()
println(s"Test Error: $err")
```

RunRandomForestForClassification.scala

4.2.2 回归

```
// read training data and testing data from disk
val train = Points.readLibSVMFile(sc, data_dir + "cadata.train").cache()
val test = Points.readLibSVMFile(sc, data_dir + "cadata.test").cache()

// train a model of Random Forest for regression
val rf_model = RandomForest.train(
  train,
  impurity = "Variance",
  max_depth = 15,
  max_bins = 32,
  bin_samples = 10000,
  min_node_size = 10,
  min_info_gain = 1e-6,
  row_rate = 0.6,
  col_rate = 0.6,
  num_trees = 100)

// predict for testing data use the model
val preds = rf_model.predict(test)
// calculate testing error
```

```
21 println(s"Test RMSE: ${RMSE(preds.map(e => (e._2, e._3)))}")
```

RunRandomForestForRegression.scala

参考文献

[1] riedman J H. Greedy function approximation: a gradient boosting machine[J]. Annals of statistics, 2001: 1189-1232.