第六课 Spark ML决策树/随机森林/GBDT算法

目录

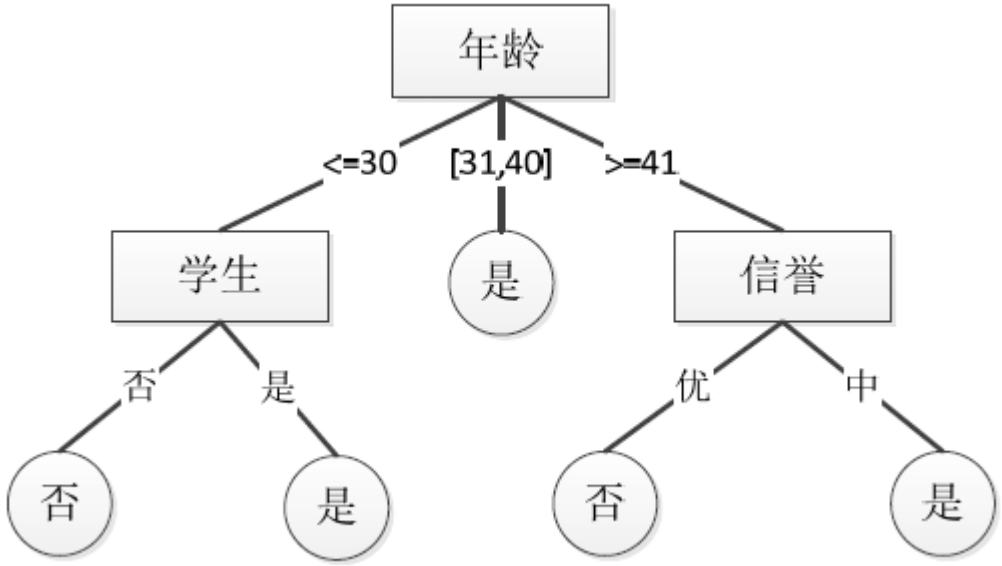
- 决策树算法
- 随机森林算法
- GDBT算法
- ML树模型参数详解
- ML实例

目录

决策树

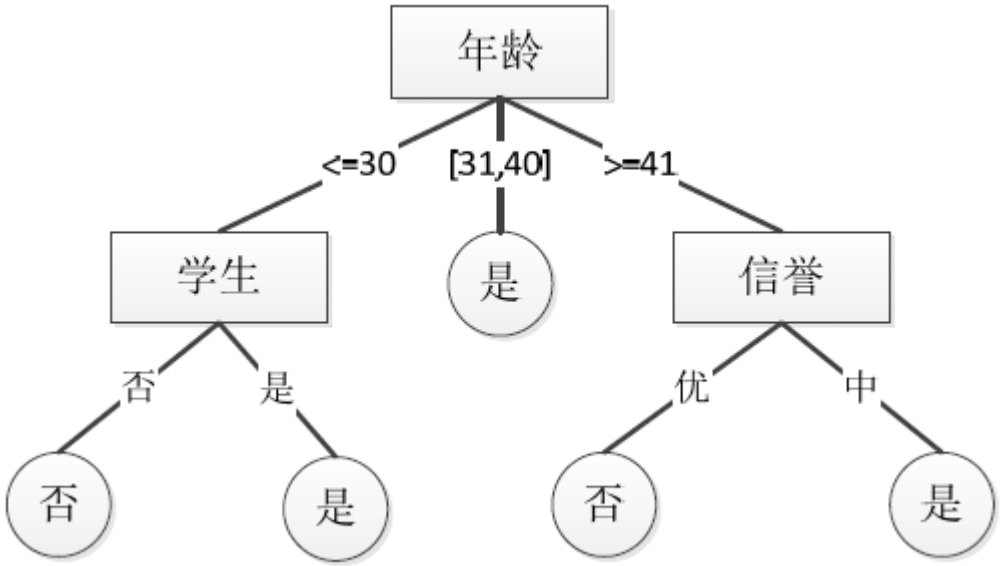
决策树

- 决策树定义：
- 决策树（decision tree）是一个树结构，决策树由节点和有向边组成。
- 节点有两种类型：内部节点和叶节点，内部节点表示一个特征或属性，叶节点表示一个类。
- 其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出。



决策树

- 决策树学习过程：
- 决策树学习的本质是从训练数据集上归纳出一组分类规则，通常采用启发式的方法：局部最优。
- 具体做法就是，每次选择feature时，都挑选当前条件下最优的那个feature作为划分规则，即局部最优的feature。
- 决策树学习通常分为3 个步骤：特征选择、决策树生成和决策树的修剪。



特征选择

- 选择特征的标准是找出局部最优的特征 ，判断一个特征对于当前数据集的分类效果。也就是按照这个特征进行分类后，数据集是否更加有序（不同分类的数据被尽量分开）。
- 衡量节点数据集合的有序性（纯度）有：

  熵　　（分类）

  基尼　（分类）

  方差　（回归）

特征选择：熵

1）信息量

信息量由这个事件发生的概率所决定。经常发生的事件是没有什么信息量的，只有小概率事件才有信息量，所以信息量的定义：

$$I_e = -\log_2 p_i$$

例如，英语有 26 个字母，假如每个字母在文章中出现的次数是平均数，那每个字母的信息量为：

$$I_e = -\log_2 \frac{1}{26} = 4.7$$

而汉字常用的有 2500 个，假如每个汉字在文章中出现的次数是平均数，那每个汉字的信息量为：

$$I_e = -\log_2 \frac{1}{2500} = 11.3$$

汉字的信息量要远大于英语字母的信息量，所以汉语比英语更加难学。

特征选择：熵

2）信息熵

熵，就是信息量的期望，信息熵的公式为：

$$H(x) = E(I(x)) = \sum_{i=1}^{n} p(x_i) I(x_i) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

条件熵的公式为：

$$H(x \mid y) = -\sum_{i=1}^{n} p(x_i \mid y) \log_b p(x_i \mid y)$$

特征选择：熵

3）信息增益

分类前，数据中可能出现各种类的情况，比较混乱，不确定性强，熵比较高；分类后，不同类的数据得到比较好的划分，那么在一个划分中大部分是同一类的数据，比较有序，不确定性降低，熵比较低。信息增益就是用于这种熵的变化 。

信息增益的定义为：特征 A 对训练数据集 D 的信息增益 $g(D, A)$，定义为集合 D 的经验熵 $H(D)$与特征 A 给定条件下 D 的经验条件熵 $H(D|A)$之差，即：

$$g(D,A) = H(D) - H(D \mid A)$$

其中，$H(D)$根据信息熵的公式计算得到。

而 $H(D|A)$，D 根据 A 分为 n 份 $D_1 \ldots D_n$，那么 $H(D|A)$就是所有 $H(D_i)$ 的期望（平均值）：

$$H(D \mid A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i)$$

特征选择：熵

4）信息增益比

单纯的信息增益只是个相对值，因为这依赖于 $H(D)$的大小，所以信息增益比更能客观地反映信息增益。

信息增益比的定义为：特征 A 对训练数据集 D 的信息增益比 $g_R(D,A)$ 定义为其信息增益 $g(D,A)$ 与分裂信息熵 split_info(A)之比：

$$g_R(D,A)=\frac{g(D,A)}{\text{split\_info}(A)}$$

其中，$\text{split\_info}(A)=H(A)=-\sum_{j=1}^{v}\frac{|D_j|}{|D|}\log_2(\frac{|D_j|}{|D|})$。

特征选择：熵

```
 object Entropy extends Impurity {
  private[tree] def log2(x: Double) = scala.math.log(x) / scala.math.log(2)
/**
   * :: DeveloperApi ::
   * information calculation for multiclass classification
   * @param counts Array[Double] with counts for each label
   * @param totalCount sum of counts for all labels
   * @return information value, or 0 if totalCount = 0
   */
  @Since("1.1.0")
  @DeveloperApi
  override def calculate(counts: Array[Double], totalCount: Double): Double =
{
}
```

特征选择：熵

```
override def calculate(counts: Array[Double], totalCount: Double): Double = {
    if (totalCount == 0) {
      return 0
    }
    val numClasses = counts.length
    var impurity = 0.0
    var classIndex = 0
    while (classIndex < numClasses) {
      val classCount = counts(classIndex)
      if (classCount != 0) {
        val freq = classCount / totalCount
        impurity -= freq * log2(freq)
      }
      classIndex += 1
    }
    impurity
  }
```

特征选择：基尼

- 基尼指数是另一种数据的不纯度的度量方法，其公式为：

$$Gini(D) = 1 - \sum_{i}^{c} p_i^2$$

- 其中 c 表示数据集中类别的数量，Pi 表示类别 i 样本数量占所有样本的比例。
- 从该公式可以看出，当数据集中数据混合的程度越高，基尼指数也就越高。当数据集 D 只有一种数据类型，那么基尼指数的值为最低 0。

特征选择：基尼

```
object Gini extends Impurity {

  /**
   * :: DeveloperApi ::
   * information calculation for multiclass classification
   * @param counts Array[Double] with counts for each label
   * @param totalCount sum of counts for all labels
   * @return information value, or 0 if totalCount = 0
   */
  @Since("1.1.0")
  @DeveloperApi
  override def calculate(counts: Array[Double], totalCount: Double): Double =
{
```

特征选择：基尼

```
override def calculate(counts: Array[Double], totalCount: Double): Double = {
  if (totalCount == 0) {
    return 0
  }
  val numClasses = counts.length
  var impurity = 1.0
  var classIndex = 0
  while (classIndex < numClasses) {
    val freq = counts(classIndex) / totalCount
    impurity -= freq * freq
    classIndex += 1
  }
  impurity
}
```

特征选择：方差

- 方差公式是一个数学公式，用来度量随机变量和其数学期望（即均值）之间的偏离程度，方差越小，代表这组数据越稳定，方差越大，代表这组数据越不稳定。

```
/**
 * :: DeveloperApi ::
 * variance calculation
 * @param count number of instances
 * @param sum sum of labels
 * @param sumSquares summation of squares of the labels
 * @return information value, or 0 if count = 0
 */
@Since("1.0.0")
@DeveloperApi
override def calculate(count: Double, sum: Double, sumSquares: Double): Double = {
  if (count == 0) {
    return 0
  }
  val squaredLoss = sumSquares - (sum * sum) / count
  squaredLoss / count
}

/**
```

ID3算法

1. ID3 算法

ID3 算法就是在每次需要分裂时，计算每个属性的增益率，然后选择增益率最大的属性进行分裂。算法过程如下。

输入：训练数据集 D、特征集 A、阈值 ε。

输出：决策树 T。

1）若 D 中的所有实例属于同一类 C_k，则 T 为单节点树，并将类 C_k 作为该节点的类标记，返回 T；

2）若 $A=\varnothing$，则 T 为单节点树，并将 D 中实例数最大的类 C_k 作为该节点的类标记，返回 T；

3）否则，按照特征选择算法（熵、Gini、方差等）计算 A 中各特征对 D 的信息增益，选择信息增益最大的特征 A_g；

a）如果 A_g 的信息增益小于阈值 ε，则置 T 为单节点树，并将 D 中实例数最大的类 C_k 作为该节点的类标记，返回 T；

b）否则，对 A_g 的每一可能值 a_i，依据 $A_g=a_i$ 将 D 分割为若干非空子集 D_i，将 D_i 中实例数最大的类作为标记，构建子节点，由节点及其子节点构成树 T，返回 T；

4）对第 i 个子节点，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用 1）~3），得到子树 T_i，返回 T_i。

决策树生成实例

ID3 算法的主要思想就是每次计算出各个属性的信息增益，选择最大者为分裂属性。下面举例说明（见表 9-1），为简单起见，10 条数据，分为两个维度。

表 9-1　决策树实例数据 1

性别（T_1）	1	0	1	0	1	0	1	0	1	1
套餐类别（T_2）	A	B	A	A	C	C	B	A	A	C
是否购买	true	false	true	false	true	true	false	true	true	true

根据公式，信息熵计算方式如下：

$$H(U) = -\sum_{i=1}^{n} p(u_i)\log_b p(u_i) = -\frac{7}{10}\log_2(\frac{7}{10}) - \frac{3}{10}\log_2(\frac{3}{10}) = 0.881$$

根据公式，条件熵计算如下：

决策树生成实例

性别（T_1）	1	0	1	0	1	0	1	0	1	1
套餐类别（T_2）	A	B	A	A	C	C	B	A	A	C
是否购买	true	false	true	false	true	true	false	true	true	true

$$H(U \mid T_1) = \frac{6}{10}(-\frac{5}{6}\log_2(\frac{5}{6}) - \frac{1}{6}\log_2(\frac{1}{6})) + \frac{4}{10}(-\frac{2}{4}\log_2(\frac{2}{4}) - \frac{2}{4}\log_2(\frac{2}{4})) = 0.790$$

$$H(U \mid T_2) = \frac{5}{10}(-\frac{4}{5}\log_2(\frac{4}{5}) - \frac{1}{5}\log_2(\frac{1}{5})) + \frac{2}{10}(-\frac{2}{2}\log_2(\frac{2}{2}) - \frac{0}{2}\log_2(\frac{0}{2}))$$
$$+ \frac{3}{10}(-\frac{3}{3}\log_2(\frac{3}{3}) - \frac{0}{3}\log_2(\frac{0}{3})) = 0.361$$

根据公式，信息增益计算如下：

$$g(U, T_1) = H(U) - H(U \mid T_1) = 0.091$$
$$g(U, T_2) = H(U) - H(U \mid T_2) = 0.520$$

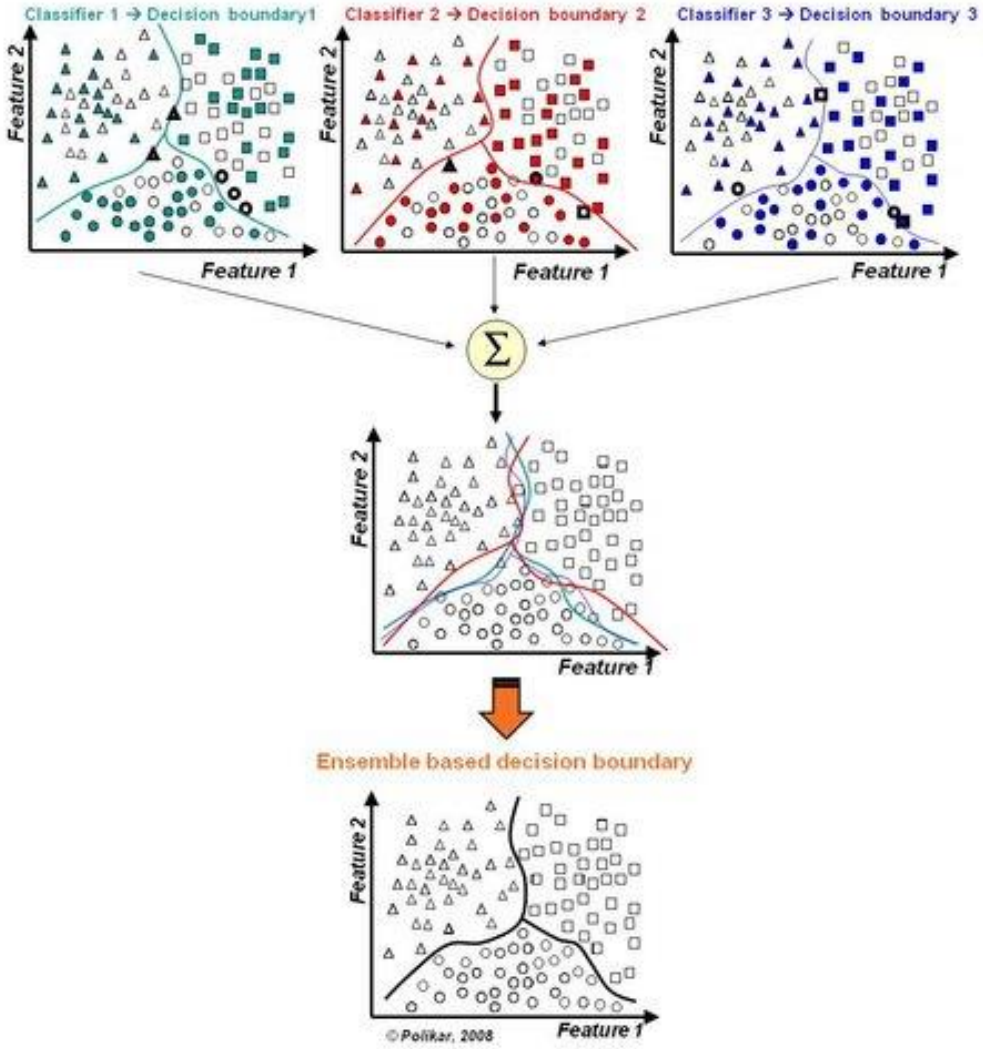
根据 ID3 的算法，将会选择 T_2 作为最佳分组变量。

目录

随机森林、GBDT

Ensemble learning概述

集成学习（ Ensemble learning ）是一种机器学习范式，它使用多个弱学习器来解决同一个问题，可以提高分类和回归的准确性。而最具代表性的就是Bagging和Boosting方法。



Bagging：随机森林

Boosting：AdaBoost， GBDT

Ensemble learning概述

- 集成分类器-使用多个弱分类器构建一个强分类器
  - **Bagging方法**

有放回抽样得到S个样本集，用同一个分类算法分别作用于每个样本集得到S个分类器，选择分类器投票结果最多的类别作为分类结果，代表有随机森林。

  - **Boosting方法**

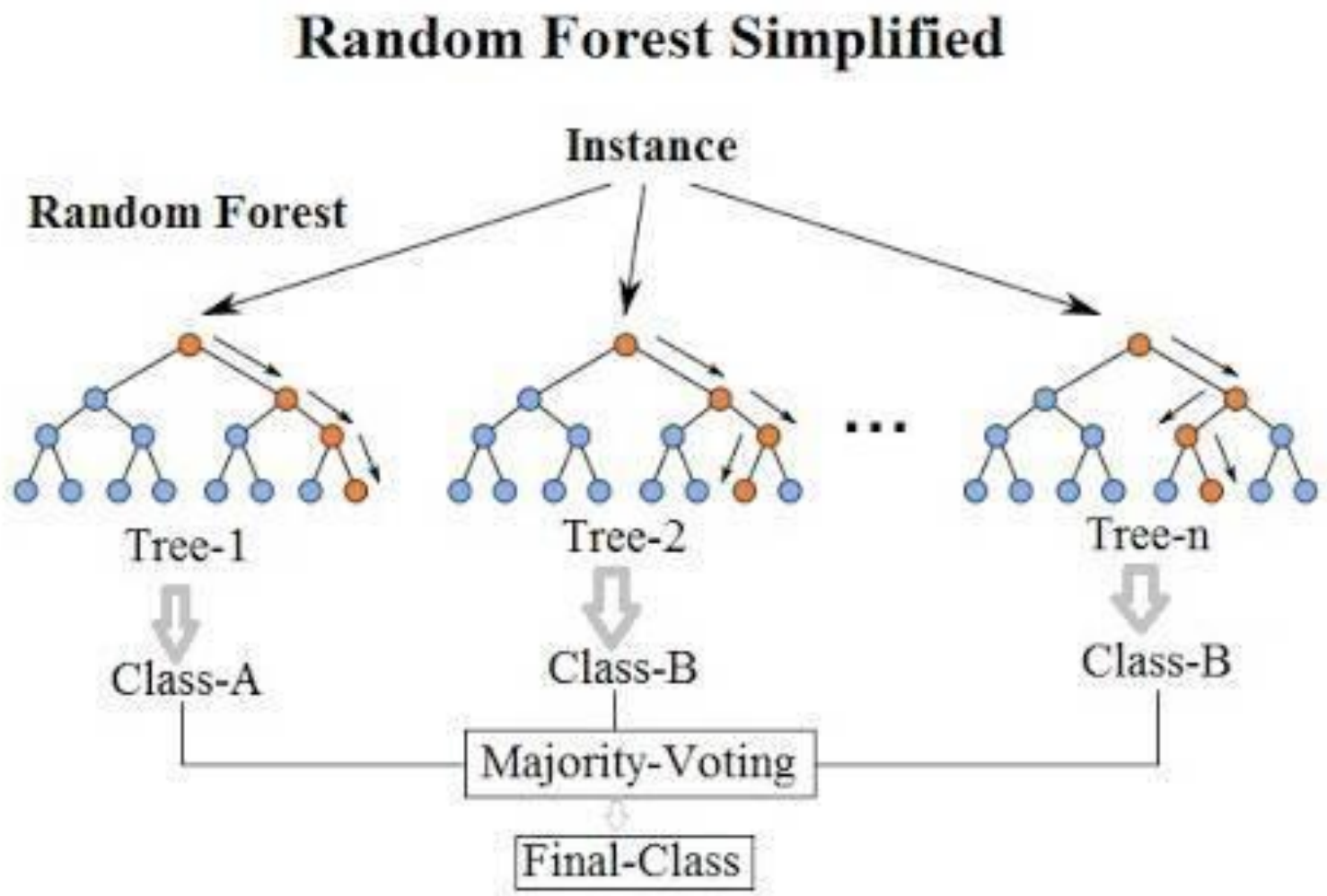
基于错误，不断迭代修正提升分类器性能。代表有Adaboost（Adaptive boosting）和GBDT（Gradient Boosting Decision Tree）

Ensemble learning概述

- Bagging方法和Boosting方法比较
  - **相同点**

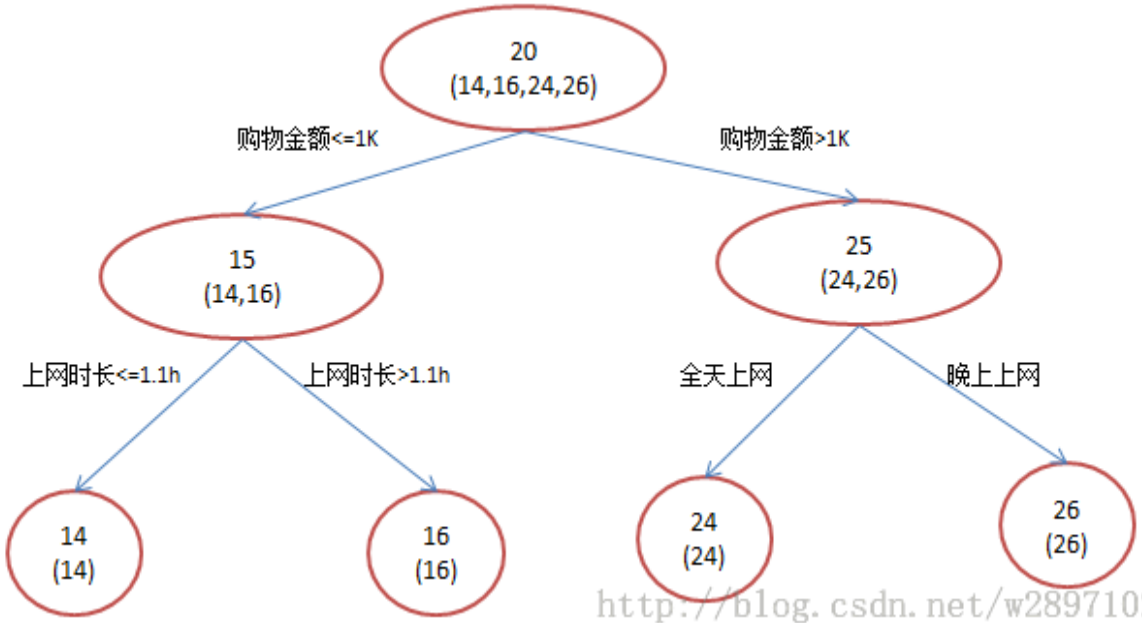
    所使用的多个分类器的类型是一致的
  - **不同点**
    - Bagging中不同分类器是并行得到的，而Boosting中每个分类器通过串行训练得到(关注被已有分类器错分的样本获得新分类器)
    - Bagging中各个分类器权重相同，Boost则是不同的（boosting分类的结果基于所有分类器的加权求和结果的）

随机森林

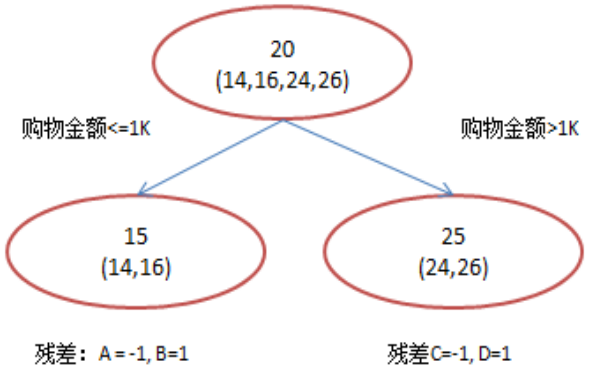

Random Forest Simplified
Instance
Random Forest
Tree-1
Tree-2
Tree-n
Class-A
Class-B
Class-B
Majority-Voting
Final-Class

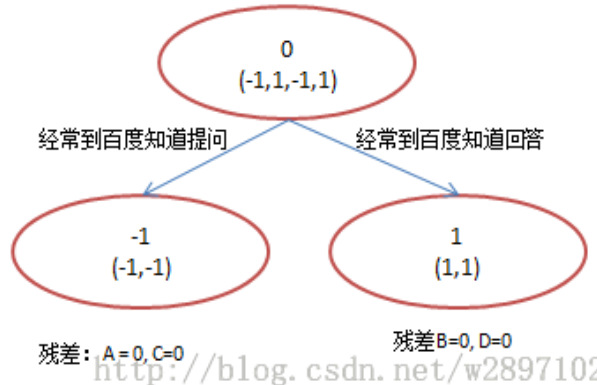
GBDT

1.普通决策树



2． ＧＢＤＴ基于残差进行学习





每一棵树是基于前面n棵树的残差进行学习，使得尽可能的整体残差最小，所以每增加一棵树，就是在这个残差优化走一小步，最终找到最优值。

GBDT介绍

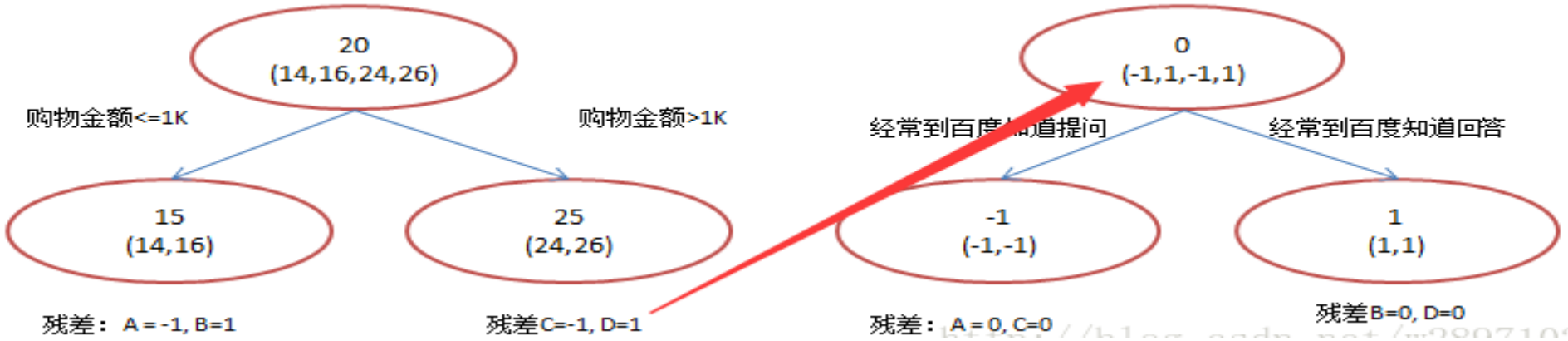
- 基本概念
  - 残差
    - $y_i - F(x_i)$ (These are the parts that existing model F cannot do well)
  - 损失函数

Loss	Task	Formula	Description
Log Loss	Classification	$2\sum_{i=1}^{N}\log(1+\exp(-2y_iF(x_i)))$	Twice binomial negative log likelihood.
Squared Error	Regression	$\sum_{i=1}^{N}(y_i - F(x_i))^2$	Also called L2 loss. Default loss for regression tasks.
Absolute Error	Regression	$\sum_{i=1}^{N}\lvert y_i - F(x_i)\rvert$	Also called L1 loss. Can be more robust to outliers than Squared Error.

  - 梯
    - 所有样本预测值向什么方向移动一小步,使得损失函数以最快的速度达到最小

GBDT过程

- 基于残差Boosting



A: 14岁高一学生，购物较少，经常问学长问题；预测年龄A = 15 – 1 = 14
B: 16岁高三学生；购物较少，经常被学弟问问题；预测年龄B = 15 + 1 = 16
C: 24岁应届毕业生；购物较多，经常问师兄问题；预测年龄C = 25 – 1 = 24
D: 26岁工作两年员工；购物较多，经常被师弟问问题；预测年龄D = 25 + 1 = 26

GBDT过程

- 基于梯度下降Boosting

Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), \ldots, F(x_n)$.
Notice that $F(x_1), F(x_2), \ldots, F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

update F based on residual , update F based on negative gradient

目录

ML树模型参数详解

ML 参数

DecisionTreeClassifier	
参数	说明
setCheckpointInterval(value: Int)	指定检查点缓存的频率。 仅当cacheNodeIds为true并且在org.apache.spark.SparkContext中设置了检查点目录时，才使用此方法。 必须至少为1.（默认= 10）
setFeaturesCol(value: String)	**指定特征列**
setImpurity(value: String)	**信息增益计算方法，支持: "entropy" 和 "gini". (默认 = gini)**
setLabelCol(value: String)	**指定标签列**
setMaxBins(value: Int)	**最大Bins数量，（默认= 32）**
setMaxDepth(value: Int)	**树的最大深度（> = 0）。 深度0表示1个叶节点; 深度1表示1个内部节点+ 2个叶节点。 （默认= 5）**
setMinInfoGain(value: Double)	节点在分裂时的最小信息增益。 应该>= 0.0。 （默认= 0.0）
setMinInstancesPerNode(value: Int)	**节点分裂后子节点必须具有的最小实例数。 如果分裂导致左或右子节点的MinInstancesPerNode小于minInstancesPerNode，则当前分裂将被丢弃为无效。 应为> = 1.（默认= 1）**
setPredictionCol(value: String)	指定预测列
setProbabilityCol(value: String)	指定预测概率值列
setRawPredictionCol(value: String)	指定原始预测列
setSeed(value: Long)	随机种子
setThresholds(value: Array[Double])	控制分类的阈值

ML 参数

RandomForestRegressor	
参数	说明
setCheckpointInterval(value: Int)	指定检查点缓存的频率。 仅当cacheNodeIds为true并且在org.apache.spark.SparkContext中设置了检查点目录时，才使用此方法。 必须至少为1.（默认= 10）
setFeatureSubsetStrategy(value: String)	节点每次分裂时选择特征的方法： “auto” ：自动选择：如果numTrees == 1，则设置为“全部”。 如果numTrees> 1（森林），如果是分类则设置为“sqrt”，如果是回归则设置为“onethird。 “全部” ：使用所有特征 “onethird” ：使用1/3的特征 “sqrt” ：使用sqrt（特征数量） “log2” ：使用log2（特征数量） “n” ：当n在0,1.0范围内时，使用n *个特征，当n在范围（1，特征个数）内时，使用n个特征（默认=“auto”）。

ML 参数

setFeaturesCol(value: String)	**指定特征列**
setImpurity(value: String)	**信息增益计算方法，支持: "variance". (默认 = variance)**
setLabelCol(value: String)	**指定标签列**
setMaxBins(value: Int)	**最大Bins数量，（默认= 32）**
setMaxDepth(value: Int)	**树的最大深度（> = 0）。 深度0表示1个叶节点; 深度1表示1个内部节点+ 2个叶节点。 （默认= 5）**
setMinInfoGain(value: Double)	节点在分裂时的最小信息增益。 应该>= 0.0。 （默认= 0.0）
setMinInstancesPerNode(value: Int)	**节点分裂后子节点必须具有的最小实例数。 如果分裂导致左或右子节点的MinInstancesPerNode小于minInstancesPerNode，则当前分裂将被丢弃为无效。 应为> = 1.（默认= 1）**
setNumTrees(value: Int)	**树的棵数**
setPredictionCol(value: String)	指定预测列
setSeed(value: Long)	随机种子
setSubsamplingRate(value: Double)	用于学习每个决策树的训练数据的抽样比例，范围为（0，1]。（默认值= 1.0）

GBTClassifier	
参数	说明
setCheckpointInterval(value: Int)	指定检查点缓存的频率。 仅当cacheNodeIds为true并且在org.apache.spark.SparkContext中设置了检查点目录时，才使用此方法。 必须至少为1.（默认= 10）
setLossType(value: String)	GBDT的最小化的损失函数。 支持： “logistic” （默认=logistic）
setFeaturesCol(value: String)	**指定特征列**
setImpurity(value: String)	**信息增益计算方法，支持: "variance". (默认 = variance)**
setLabelCol(value: String)	**指定标签列**
setMaxBins(value: Int)	**最大Bins数量，（默认= 32）**
setMaxDepth(value: Int)	**树的最大深度（> = 0）。 深度0表示1个叶节点; 深度1表示1个内部节点+ 2个叶节点。 （默认= 5）**
setMinInfoGain(value: Double)	节点在分裂时的最小信息增益。 应该>= 0.0。 （默认= 0.0）
setMinInstancesPerNode(value: Int)	**节点分裂后子节点必须具有的最小实例数。 如果分裂导致左或右子节点的MinInstancesPerNode小于minInstancesPerNode，则当前分裂将被丢弃为无效。 应为> = 1.（默认= 1）**
setNumTrees(value: Int)	**树的棵数**
setSeed(value: Long)	随机种子
setSubsamplingRate(value: Double)	用于学习每个决策树的训练数据的抽样比例，范围为（0，1]。（默认值= 1.0）
setPredictionCol(value: String)	指定预测列
setProbabilityCol(value: String)	指定预测概率值列
setRawPredictionCol(value: String)	指定原始预测列
setSeed(value: Long)	随机种子
setThresholds(value: Array[Double])	控制分类的阈值

ML 参数

(expert-only) Parameters

A list of advanced, expert-only (hyper-)parameter keys this algorithm can take. Users can set and get the parameter values through setters and getters, respectively.

final val **cacheNodeIds**: BooleanParam

If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each insta
the cache be checkpointed or disable it by setting checkpointInterval. (default = false)

Definition Classes DecisionTreeParams

final val **maxMemoryInMB**: IntParam

Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may e

Definition Classes DecisionTreeParams

(expert-only) Parameter setters

def **setCacheNodeIds**(value: Boolean): GBTClassifier.this.type

Definition Classes GBTClassifier → DecisionTreeParams
Annotations @Since("1.4.0")

def **setMaxMemoryInMB**(value: Int): GBTClassifier.this.type

Definition Classes GBTClassifier → DecisionTreeParams
Annotations @Since("1.4.0")

目录

ML 实例

实例代码讲解

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{ RandomForestClassificationModel, RandomForestClassifier }
import org.apache.spark.ml.classification.{ DecisionTreeClassifier, DecisionTreeClassificationModel }
import org.apache.spark.ml.classification.{ GBTClassificationModel, GBTClassifier }
import org.apache.spark.ml.evaluation.{ MulticlassClassificationEvaluator, BinaryClassificationEvaluator }
import org.apache.spark.ml.{ Pipeline, PipelineModel }
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.ml.linalg.{ Vector, Vectors }
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.Encoder
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}
import org.apache.spark.ml.classification.{DecisionTreeClassifier, DecisionTreeClassificationModel}
import org.apache.spark.ml.classification.{GBTClassificationModel, GBTClassifier}
import org.apache.spark.ml.evaluation.{MulticlassClassificationEvaluator, BinaryClassificationEvaluator}
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.ml.linalg.{Vector, Vectors}
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.Encoder
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

实例代码

```
import spark.implicits._

//1 训练样本准备
val data = spark.read.format("libsvm").load("hdfs:/[redacted]/sample_libsvm_data.txt")
data.show
```

```
import spark.implicits._
data: org.apache.spark.sql.DataFrame = [label: double, features: vector]
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(692,[127,128,129...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[124,125,126...|
|  1.0|(692,[152,153,154...|
|  1.0|(692,[151,152,153...|
|  0.0|(692,[129,130,131...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[99,100,101,...|
|  0.0|(692,[154,155,156...|
|  0.0|(692,[127,128,129...|
|  1.0|(692,[154,155,156...|
|  0.0|(692,[153,154,155...|
|  0.0|(692,[151,152,153...|
```

实例代码

```
//2 标签进行索引编号
val labelIndexer = new StringIndexer().
  setInputCol("label").
  setOutputCol("indexedLabel").
  fit(data)
// 对离散特征进行标记索引，以用来确定哪些特征是离散特征
// 如果一个特征的值超过4个以上，该特征视为连续特征，否则将会标记得离散特征并进行索引编号
val featureIndexer = new VectorIndexer().
  setInputCol("features").
  setOutputCol("indexedFeatures").
  setMaxCategories(4).
  fit(data)

//3 样本划分
val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))
```

```
labelIndexer: org.apache.spark.ml.feature.StringIndexerModel = strIdx_e23e5857cc21
featureIndexer: org.apache.spark.ml.feature.VectorIndexerModel = vecIdx_5ad78a1f6fa0
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: double, features: vector]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [label: double, features: vector]
```

实

```
//4 训练决策树模型
val dt = new DecisionTreeClassifier().
  setLabelCol("indexedLabel").
  setFeaturesCol("indexedFeatures")

//4 训练随机森林模型
val rf = new RandomForestClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures")
  .setNumTrees(10)

//4 训练GBDT模型
val gbt = new GBTClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures")
  .setMaxIter(10)

//5 将索引的标签转回原始标签
val labelConverter = new IndexToString().
  setInputCol("prediction").
  setOutputCol("predictedLabel").
  setLabels(labelIndexer.labels)
```

```
dt: org.apache.spark.ml.classification.DecisionTreeClassifier = dtc_6ea81fb1ad46
rf: org.apache.spark.ml.classification.RandomForestClassifier = rfc_6526a5323038
gbt: org.apache.spark.ml.classification.GBTClassifier = gbtc_cea8465437df
labelConverter: org.apache.spark.ml.feature.IndexToString = idxToStr_d94dfe0acc71
```

实例代码

```
//6 构建Pipeline
val pipeline1 = new Pipeline().
  setStages(Array(labelIndexer, featureIndexer, dt, labelConverter))

val pipeline2 = new Pipeline().
  setStages(Array(labelIndexer, featureIndexer, rf, labelConverter))

val pipeline3 = new Pipeline().
  setStages(Array(labelIndexer, featureIndexer, gbt, labelConverter))

//7 Pipeline开始训练
val model1 = pipeline1.fit(trainingData)

val model2 = pipeline2.fit(trainingData)

val model3 = pipeline3.fit(trainingData)
```

```
pipeline1: org.apache.spark.ml.Pipeline = pipeline_5b21e630a9da
pipeline2: org.apache.spark.ml.Pipeline = pipeline_e0c67046c322
pipeline3: org.apache.spark.ml.Pipeline = pipeline_3430c74b38fb
model1: org.apache.spark.ml.PipelineModel = pipeline_5b21e630a9da
model2: org.apache.spark.ml.PipelineModel = pipeline_e0c67046c322
model3: org.apache.spark.ml.PipelineModel = pipeline_3430c74b38fb
```

```
    //8 模型测试
    val predictions = model1.transform(testData)
    predictions.show(5)

    //8 测试结果
    predictions.select("predictedLabel", "label", "features").show(5)
```

```
predictions: org.apache.spark.sql.DataFrame = [label: double, features: vector ... 6 more fields]
+-----+--------------------+------------+--------------------+-------------+-----------+----------+--------------+
|label|            features|indexedLabel|     indexedFeatures|rawPrediction|probability|prediction|predictedLabel|
+-----+--------------------+------------+--------------------+-------------+-----------+----------+--------------+
|  0.0|(692,[100,101,102...|         1.0|(692,[100,101,102...|   [0.0,30.0]|  [0.0,1.0]|       1.0|           0.0|
|  0.0|(692,[123,124,125...|         1.0|(692,[123,124,125...|   [0.0,30.0]|  [0.0,1.0]|       1.0|           0.0|
|  0.0|(692,[125,126,127...|         1.0|(692,[125,126,127...|   [0.0,30.0]|  [0.0,1.0]|       1.0|           0.0|
|  0.0|(692,[126,127,128...|         1.0|(692,[126,127,128...|   [0.0,30.0]|  [0.0,1.0]|       1.0|           0.0|
|  0.0|(692,[126,127,128...|         1.0|(692,[126,127,128...|   [0.0,30.0]|  [0.0,1.0]|       1.0|           0.0|
+-----+--------------------+------------+--------------------+-------------+-----------+----------+--------------+
only showing top 5 rows
+--------------+-----+--------------------+
|predictedLabel|label|            features|
+--------------+-----+--------------------+
|           0.0|  0.0|(692,[100,101,102...|
|           0.0|  0.0|(692,[123,124,125...|
|           0.0|  0.0|(692,[125,126,127...|
|           0.0|  0.0|(692,[126,127,128...|
```

实例化

```
//9 分类指标
// 正确率
val evaluator1 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("accuracy")
val accuracy = evaluator1.evaluate(predictions)
println("Test Error = " + (1.0 - accuracy))
// f1
val evaluator2 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("f1")
val f1 = evaluator2.evaluate(predictions)
println("f1 = " + f1)
// Precision
val evaluator3 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("weightedPrecision")
val Precision = evaluator3.evaluate(predictions)
println("Precision = " + Precision)
// Recall
val evaluator4 = new MulticlassClassificationEvaluator().
  setLabelCol("indexedLabel").
  setPredictionCol("prediction").
  setMetricName("weightedRecall")
val Recall = evaluator4.evaluate(predictions)
println("Recall = " + Recall)
```

实例代码

```
evaluator1: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_a97495cbb15f
accuracy: Double = 0.9666666666666667
Test Error = 0.033333333333333326
evaluator2: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_17908b0eb374
f1: Double = 0.9667789001122334
f1 = 0.9667789001122334
evaluator3: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_99b448bc6ae1
Precision: Double = 0.969047619047619
Precision = 0.969047619047619
evaluator4: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = mcEval_21575c4799ed
Recall: Double = 0.9666666666666667
Recall = 0.9666666666666667
```

实例代码

```
// AUC
val evaluator5 = new BinaryClassificationEvaluator().
  setLabelCol("indexedLabel").
  setRawPredictionCol("prediction").
  setMetricName("areaUnderROC")
val AUC = evaluator5.evaluate(predictions)
println("Test AUC = " + AUC)

// aupr
val evaluator6 = new BinaryClassificationEvaluator().
  setLabelCol("indexedLabel").
  setRawPredictionCol("prediction").
  setMetricName("areaUnderPR")
val aupr = evaluator6.evaluate(predictions)
println("Test aupr = " + aupr)
```

```
evaluator5: org.apache.spark.ml.evaluation.BinaryClassificationEvaluator = binEval_304a133c6c06
AUC: Double = 0.9705882352941176
Test AUC = 0.9705882352941176
evaluator6: org.apache.spark.ml.evaluation.BinaryClassificationEvaluator = binEval_446d26711645
aupr: Double = 0.9642857142857143
Test aupr = 0.9642857142857143
```

实例代码

```
    //10 决策树打印
    val treeModel = model1.stages(2).asInstanceOf[DecisionTreeClassificationModel]
    println("Learned classification tree model:\n" + treeModel.toDebugString)
```

```
treeModel: org.apache.spark.ml.classification.DecisionTreeClassificationModel = DecisionTreeCl
Learned classification tree model:
DecisionTreeClassificationModel (uid=dtc_421cc4def13a) of depth 1 with 3 nodes
  If (feature 434 <= 0.0)
   Predict: 1.0
  Else (feature 434 > 0.0)
   Predict: 0.0
```

实例代码

```
    //11 模型保存与加载
    model1.save("hdfs://[redacted]/mlv2/dtmodel")
    val load_treeModel = PipelineModel.load("hdfs://[redacted]/mlv2/dtmodel")
```

```
java.io.IOException: Path hdfs://[redacted]/mlv2/dtmodel already exists. Please use
  at org.apache.spark.ml.util.MLWriter.save(ReadWrite.scala:107)
  at org.apache.spark.ml.util.MLWritable$class.save(ReadWrite.scala:154)
  at org.apache.spark.ml.PipelineModel.save(Pipeline.scala:292)
  ... 58 elided
```

Took 1 sec. Last updated by cunbowhuang at August 11 2017, 5:16:20 PM

Thanks

FAQ时间