决策树及其集成算法是分类和回归的机器学习任务的流行方法。决策树被广泛使用，因为它们易于解释，处理分类特征，扩展到多类分类设置，不需要特征缩放，并且能够捕获非线性和特征交互。诸如随机森林和提升树集成算法是分类和回归任务中表现较好的算法。

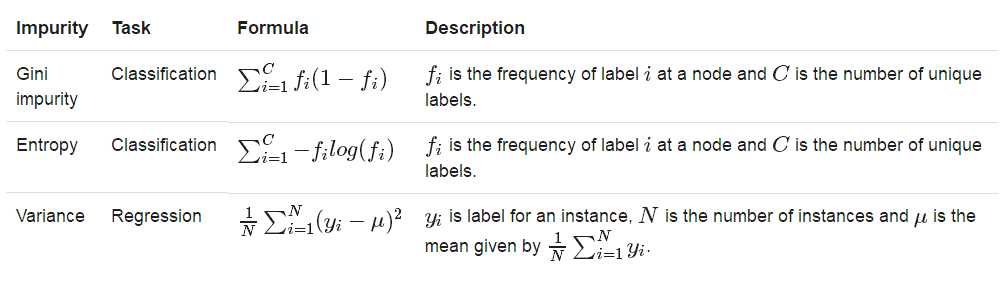
spark.mllib支持使用连续和分类特征进行二分类和多类分类以及回归的决策树。该实现按行分区数据，允许数百万个实例的分布式训练。

# 1.基本算法

决策树是一种贪婪算法，它在特征空间上递归地进行二分。树为每个最底层（叶子）分区预测相同的标签。通过从一组可能的分割中选择最佳分割来贪婪地选择每个分区，以便最大化树节点处的信息增益。换句话说，在每个树节点处选择的分割是从集合argmax**IG**（D，s）argmax**IG**（D，s）中选择的，其中IG（D，s）是信息增益。

1.1结点不纯度和信息增益

节点不纯度是节点处标签均匀性的量度。目前的实施方案提供了两种用于分类的不纯度测量（基尼不纯度和熵）和一种用于回归的不纯度测量（方差）。



信息增益是父节点不纯度与两个子节点不纯度的加权和之间的差。假设分割s将大小为N的数据集D分别分为大小为Nleft和Nright的两个数据集Dleft和Dright，信息增益为：



## 1.2 分割候选项

### 1.2.1 连续的特征

对于单机实现中的小型数据集，每个连续要素的拆分候选项通常是该要素的唯一值。某些实现对特征值进行排序，然后使用有序的唯一值作为快速树计算的分割候选。

对于大型分布式数据集，排序特征值是代价昂贵的。该实现通过对数据的采样部分执行分位数计算来计算一组近似的分割候选。有序拆分创建“bins”，并且可以使用maxBins参数指定此类bin的最大数量。

请注意，bin的数量不能大于N的实例数（由于默认的maxBins值为32，这是一种罕见的情况）。如果不满足条件，树算法会自动减少bin的数量。

### 1.2.2 分类特征

对于具有M个可能值（类别）的分类特征，可以提出2M-1-1分割候选者。对于二（0/1）分类和回归，我们可以通过按平均标签对分类特征值进行排序，将分割候选者的数量减少到M-1。例如，对于具有三个类别A，B和C的一个分类特征的二元分类问题，其标记1的对应比例为0.2,0.6和0.4，分类特征按A，C，B排序。两个分开的候选人是A | C，B和A，C | B， | 表示分割。

在多类分类中，尽可能使用所有2M-1-1可能的分割。当2M-1-1大于maxBins参数时，我们使用类似于二分类和回归的方法的（启发式）方法。M个分类特征值按不纯度排序，并考虑得到的M-1个分割候选项。

## 1.3 停止规则

当满足以下条件之一时，在节点处停止递归树构造：

* 节点深度等于maxDepth训练参数。
* 没有分割候选项导致信息增益大于minInfoGain。
* 没有分割候选项生成子节点，每个子节点至少具有minInstancesPerNode训练实例。

# 2. 使用提示

我们通过讨论各种参数总结出一些使用决策树的指南。下面大致按重要性递减的顺序列出参数。新用户应主要考虑“Problem specification parameters”部分和maxDepth参数。

## 2.1 Problem specification parameters

这些参数描述了您要解决的问题和数据集。 它们应该被指定，不需要调整。

* algo：决策树的类型，分类或回归。
* numClasses：类的数量（仅用于分类）。
* categoricalFeaturesInfo：指定哪些特征是分类的，以及每个特征可以采用的分类值。这是从特征索引到特征arity（类别数）的映射。 此映射中没有的任何特征都被视为连续的。

例如，Map（0 - > 2,4 - > 10）指定特征0是二进制（取值0或1），而特征4有10个类别（值{0,1，...，9}）。请注意，特征索引从0开始：要素0和4是实例特征向量的第1和第5个要素。

请注意，您不必指定categoricalFeaturesInfo。 该算法仍将运行并可能获得合理的结果。 但是，如果正确指定了分类特征，性能应该更好。

## 2.2 停止标准

这些参数确定树何时停止构建（添加新节点）。调整这些参数时，请小心验证保留的测试数据，以避免过度拟合。

* maxDepth：树的最大深度。 更深的树木更具表现力（可能允许更高的准确性），但它们的训练费用也更高，而且更容易过度拟合。
* minInstancesPerNode：对于要进一步分割的节点，其每个子节点必须至少接收此数量的训练实例。这通常与RandomForest一起使用，因为它们通常比单个树更深地训练。
* minInfoGain：对于要进一步分割的节点，分割必须至少改善这么多（就信息增益而言）。

## 2.3 可调参数

可以调整这些参数。 调整时要小心验证保持的测试数据，以避免过度拟合。

* maxBins：离散连续特征时使用的bin数。

增加maxBins允许算法考虑更多分割候选项并进行细粒度分割决策。 但是，它也增加了计算和通信代价。

请注意，maxBins参数必须至少为任何分类特征的最大类别数M。

* maxMemoryInMB：用于收集足够统计信息的内存量。

保守地选择默认值为256 MB，以允许决策算法在大多数情况下工作。增加maxMemoryInMB可以通过允许更少的数据传递来加快训练速度（如果内存可用）。但是，随着maxMemoryInMB增长，可能会有减少的回报，因为每次迭代的通信量可以与maxMemoryInMB成比例。

实现细节：为了加快处理速度，决策树算法收集有关要分割的节点组的统计信息（而不是一次收集1个节点）。可以在一个组中处理的节点数由内存要求（每个功能不同）决定。maxMemoryInMB参数指定每个工作程序可用于这些统计信息的内存限制（以兆字节为单位）。

* subsamplingRate：用于学习决策树的训练数据的分数。此参数与训练树基础算法（使用RandomForest和GradientBoostedTrees）最相关，其中对原始数据进行子采样非常有用。对于训练单个决策树，该参数不太有用，因为训练实例的数量通常不是主要约束。
* impurity：用于在候选分割项之间进行选择的不纯度度量（如上所述）。 此度量必须与算法参数匹配。

## 2.4缓存和检查点(Caching and checkpointing)

MLlib 1.2增加了几个功能，可以扩展到更大（更深）的树和树集成。 当maxDepth设置为较大时，打开节点ID缓存和检查点可能很有用。当numTrees设置为较大时，这些参数对RandomForest也很有用。

* useNodeIdCache：如果将其设置为true，则算法将避免在每次迭代时将当前模型（树或树）传递给执行程序。这对于深树（加速计算worker）和大型随机森林（减少每次迭代的通信）非常有用。

实现细节：默认情况下，算法将当前模型传递给执行程序，以便执行程序可以将训练实例与树节点进行匹配。当此设置打开时，算法将改为缓存此信息。

节点ID缓存生成一系列RDD（每次迭代1次）。这种较长的谱系可能会导致性能问题，但检查点中间RDD可以缓解这些问题。请注意，仅当useNodeIdCache设置为true时，检查点才适用。

* checkpointDir：检查点节点ID缓存RDD的目录。
* checkpointInterval：检查点节点ID缓存RDD的频率。设置太低会导致写入HDFS带来额外开销; 如果执行程序失败并且需要重新计算RDD，则将此设置得太高可能会导致问题。

# 3.Scaling

计算在训练实例的数量，特征的数量和maxBins参数中近似线性地缩放。 通信在特征数量和maxBins中大致呈线性关系。

实现的算法读取稀疏和密集数据。 但是，它没有针对稀疏输入进行优化。

# 4.代码示例

## 4.1分类

下面的示例演示了如何加载LIBSVM数据文件，将其解析为LabeledPoint的RDD，然后使用带有基尼不纯度作为不纯度度量且最大树深度为5的决策树执行分类。计算测试误差以测量算法精度。

|  |
| --- |
| SparkConf sparkConf = **new** SparkConf().setAppName("JavaDecisionTreeClassificationExample").setMaster("local");  JavaSparkContext jsc = **new** JavaSparkContext(sparkConf);  // 读取数据  String datapath = "F:\\Learning\\java\\project\\LearningSpark\\src\\main\\resources\\sample\_libsvm\_data.txt";  JavaRDD<LabeledPoint> data = MLUtils.*loadLibSVMFile*(jsc.sc(), datapath).toJavaRDD();  // 划分数据为训练集和测试集  JavaRDD<LabeledPoint>[] splits = data.randomSplit(**new** **double**[] { 0.7, 0.3 });  JavaRDD<LabeledPoint> trainingdata = splits[0];  JavaRDD<LabeledPoint> testData = splits[1];  // 参数设置  **int** numClasses = 2;  Map<Integer, Integer> categoricalFeaturesInfo = **new** HashMap<>();  String impurity = "gini";  **int** maxDepth = 5;  **int** maxBins = 32;  // 模型训练  DecisionTreeModel model = DecisionTree.*trainClassifier*(trainingdata, numClasses, categoricalFeaturesInfo,  impurity, maxDepth, maxBins);  // 模型测试  JavaPairRDD<Double, Double> predictionAndLabel = testData  .mapToPair(p -> **new** Tuple2<>(model.predict(p.features()), p.label()));  **double** testErr = predictionAndLabel.filter(pl -> !pl.\_1.equals(pl.\_2())).count() / (**double**) testData.count();  System.***out***.println("测试误差为:" + testErr);  System.***out***.println(model.toDebugString()); |

## 4.2 回归

下面的示例演示了如何加载LIBSVM数据文件，将其解析为LabeledPoint的RDD，然后使用方差作为不纯度度量和最大树深度为5的决策树执行回归。最后计算均方误差（MSE）以评估拟合优度。

|  |
| --- |
| SparkConf sparkConf = **new** SparkConf().setAppName("JavaDecisionTreeRegressionExample").setMaster("local");  JavaSparkContext jsc = **new** JavaSparkContext(sparkConf);  String datapath = "F:\\Learning\\java\\project\\LearningSpark\\src\\main\\resources\\sample\_libsvm\_data.txt";  JavaRDD<LabeledPoint> data = MLUtils.*loadLibSVMFile*(jsc.sc(), datapath).toJavaRDD();  // 划分数据为训练集和测试集  JavaRDD<LabeledPoint>[] splits = data.randomSplit(**new** **double**[] { 0.7, 0.3 });  JavaRDD<LabeledPoint> trainingdata = splits[0];  JavaRDD<LabeledPoint> testData = splits[1];  // 参数设置  Map<Integer, Integer> categoricalFeaturesInfo = **new** HashMap<>();  String impurity = "variance";  **int** maxDepth = 5;  **int** maxBins = 32;  DecisionTreeModel model = DecisionTree.*trainRegressor*(trainingdata, categoricalFeaturesInfo, impurity, maxDepth,  maxBins);  // 模型测试  JavaPairRDD<Double, Double> predictionAndLabel = testData  .mapToPair(p -> **new** Tuple2(model.predict(p.features()), p.label()));  **double** testMSE = predictionAndLabel.mapToDouble(pl -> {  **double** diff = pl.\_1() - pl.\_2();  **return** diff \* diff;  }).mean();  System.***out***.println("测试数据集的均方误差:" + testMSE);  System.***out***.println(model.toDebugString()); |