[**Weka算法Classifier-tree-J48源码分析（一）算法和基本数据结构**](http://blog.csdn.net/roger__wong/article/details/38987639)

Weka为一个[**Java**](http://lib.csdn.net/base/javase)基础上的[**机器学习**](http://lib.csdn.net/base/machinelearning)工具，上手简单，并提供图形化界面，提供如分类、聚类、频繁项挖掘等工具，本篇文章主要写一下分类器[**算法**](http://lib.csdn.net/base/datastructure)中的J48算法及其实现。

一、算法

J48是基于C4.5实现的决策树算法，对于C4.5算法相关资料太多了，笔者在这里转载一部分（来源：http://blog.csdn[**.NET**](http://lib.csdn.net/base/dotnet)/zjd950131/article/details/8027081）

 C4.5是一系列用在机器学习和数据挖掘的分类问题中的算法。它的目标是监督学习：给定一个数据集，其中的每一个元组都能用一组属性值来描述，每一个元组属于一个互斥的类别中的某一类。C4.5的目标是通过学习，找到一个从属性值到类别的映射关系，并且这个映射能用于对新的类别未知的实体进行分类。

    C4.5由J.Ross Quinlan在ID3的基础上提出的。ID3算法用来构造决策树。决策树是一种类似流程图的树结构，其中每个内部节点（非树叶节点）表示在一个属性上的[**测试**](http://lib.csdn.net/base/softwaretest)，每个分枝代表一个测试输出，而每个树叶节点存放一个类标号。一旦建立好了决策树，对于一个未给定类标号的元组，跟踪一条有根节点到叶节点的路径，该叶节点就存放着该元组的预测。决策树的优势在于不需要任何领域知识或参数设置，适合于探测性的知识发现。

    从ID3算法中衍生出了C4.5和CART两种算法，这两种算法在数据挖掘中都非常重要。下图就是一棵典型的C4.5算法对数据集产生的决策树。

数据集如图1所示，它表示的是天气情况与去不去打高尔夫球之间的关系。

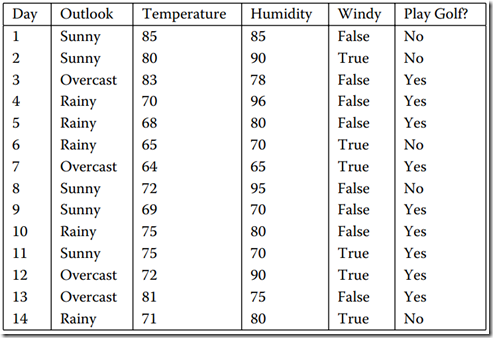
[](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246264711.png)

图1  数据集

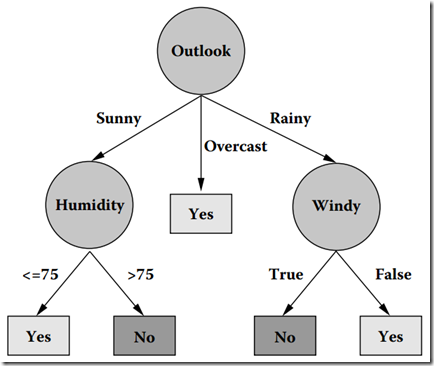
[](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246287319.png)

        图2   在数据集上通过C4.5生成的决策树

    算法描述

    C4.5并不一个算法，而是一组算法—C4.5，非剪枝C4.5和C4.5规则。下图中的算法将给出C4.5的基本工作流程：

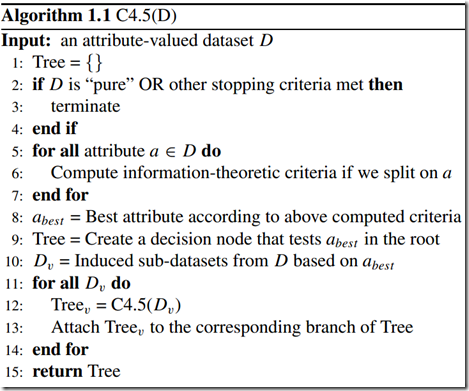
[](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246296022.png)

    图3  C4.5算法流程

    我们可能有疑问，一个元组本身有很多属性，我们怎么知道首先要对哪个属性进行判断，接下来要对哪个属性进行判断？换句话说，在图2中，我们怎么知道第一个要测试的属性是Outlook，而不是Windy？其实，能回答这些问题的一个概念就是属性选择度量。

    属性选择度量

     属性选择度量又称分裂规则，因为它们决定给定节点上的元组如何分裂。属性选择度量提供了每个属性描述给定训练元组的秩评定，具有最好度量得分的属性被选作给定元组的分裂属性。目前比较流行的属性选择度量有--信息增益、增益率和Gini指标。

    先做一些假设，设D是类标记元组训练集，类标号属性具有m个不同值，m个不同类Ci(i=1,2,…，m)，CiD是D中Ci类的元组的集合，|D|和|CiD|分别是D和CiD中的元组个数。

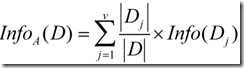
    （1）信息增益

    信息增益实际上是ID3算法中用来进行属性选择度量的。它选择具有最高信息增益的属性来作为节点N的分裂属性。该属性使结果划分中的元组分类所需信息量最小。对D中的元组分类所需的期望信息为下式：

[image](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246298597.png) （1）

Info(D)又称为熵。

    现在假定按照属性A划分D中的元组，且属性A将D划分成v个不同的类。在该划分之后，为了得到准确的分类还需要的信息由下面的式子度量：

[](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246308696.png)       （2）

    信息增益定义为原来的信息需求（即仅基于类比例）与新需求（即对A划分之后得到的）之间的差，即

[image](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246313745.png)       （3）

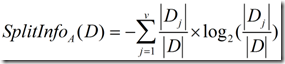
    我想很多人看到这个地方都觉得不是很好理解，所以我自己的研究了文献中关于这一块的描述，也对比了上面的三个公式，下面说说我自己的理解。

    一般说来，对于一个具有多个属性的元组，用一个属性就将它们完全分开几乎不可能，否则的话，决策树的深度就只能是2了。从这里可以看出，一旦我们选择一个属性A，假设将元组分成了两个部分A1和A2，由于A1和A2还可以用其它属性接着再分，所以又引出一个新的问题：接下来我们要选择哪个属性来分类？对D中元组分类所需的期望信息是Info(D) ,那么同理，当我们通过A将D划分成v个子集Dj(j=1,2,…,v)之后，我们要对Dj的元组进行分类，需要的期望信息就是Info(Dj),而一共有v个类，所以对v个集合再分类，需要的信息就是公式（2）了。由此可知，如果公式（2）越小，是不是意味着我们接下来对A分出来的几个集合再进行分类所需要的信息就越小？而对于给定的训练集，实际上Info(D)已经固定了，所以选择信息增益最大的属性作为分裂点。

    但是，使用信息增益的话其实是有一个缺点，那就是它偏向于具有大量值的属性。什么意思呢？就是说在训练集中，某个属性所取的不同值的个数越多，那么越有可能拿它来作为分裂属性。例如一个训练集中有10个元组，对于某一个属相A，它分别取1-10这十个数，如果对A进行分裂将会分成10个类，那么对于每一个类Info(Dj)=0，从而式（2）为0，该属性划分所得到的信息增益（3）最大，但是很显然，这种划分没有意义。

  （2）信息增益率

   正是基于此，ID3后面的C4.5采用了信息增益率这样一个概念。信息增益率使用“分裂信息”值将信息增益规范化。分类信息类似于Info(D)，定义如下：

[](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246314891.png)    （4）

这个值表示通过将训练数据集D划分成对应于属性A测试的v个输出的v个划分产生的信息。信息增益率定义：

[image](http://images.cnblogs.com/cnblogs_com/superhuake/201207/201207252246325273.png)         （5）

选择具有最大增益率的属性作为分裂属性。

二、算法说明

（1）我们是要构造一个决策树，很自然地，树的每一层代表一个属性的取值，最后的叶子节点指向划分的类。如图二所示。

（2）因此很自然的问题就是如何在每一层选择合适的节点去构造这个树使这个树的结构尽可能最优，也就是查找路径尽可能的短。

（3）因此最关键的问题就是如何在每一层，从剩下的还没被分配的节点中找出最合适的分裂节点。

（4）其中ID3算法选择最优节点的方式是：选出信息增益增益最高的属性。信息增益可以简单理解成使用某个属性划分后，不确定性的减少量。

（5）而C4.5算法做了一个改进，使用信息增益率最高的属性，这样做的好处是，可以避免树过宽。

（6）构建好了树之后还要进行一些剪枝的操作，当然这个不体现在算法主流行里，也没有做强求，但可以注意一下Weka是如何实现的。

三、算法中用到的主要[**数据结构**](http://lib.csdn.net/base/datastructure)

（1）Instances对象

一个Instances代表一张表，可以对应一个arff文件或者是一个csv文件，通过Instances对象可以取某一列的均值方差等，主要就是若干行记录的一个封装。

（2）Instance

一个Instance代表一行记录，换言之一个Instances的数据包含多个Instance。每个Instance会有一个特殊的列ClassIndex，该列值代表该Instance属于哪一类，具体来说就是图一里面的Golf。

（3）Classifier接口

Weka中每一个分类器都继承与这个接口（虽然从意义上来说是个接口但其实是个子类），该接口提供一个buildClassifier方法传入一个Instances对象用于训练，还有classifyInstance方法用于传入一个Instance来判断其属于哪个类。

（4）J48

分类器主类，实现了Classifier接口。

（5）ClassifierTree接口

代表树中的一个节点，维护和组成树的结构。其中J48用到的是C45PruneableClassifierTree和PruneableClassifierTree。

（6）ModelSelection接口

该接口负责判断和选取最优的属性，然后根据该属性将不同的Instance放到不同的subset中，ClassifierTree接口使用ModelSelection来生成树的结构。这种抽象方式还是很值得学习的，J48中用到的该接口的实现有BinC45ModelSelection和C45ModelSelection，通过名字大概也能看出来前一个是生成二叉树（即每个节点只含有是否两种回答），后一个是生成标准的C45树。

# [Weka算法Classifier-tree-J48源码分析（二）ClassifierTree](http://blog.csdn.net/roger__wong/article/details/39119701)

标签： [Weka](http://www.csdn.net/tag/Weka)[算法](http://www.csdn.net/tag/%e7%ae%97%e6%b3%95)[源码](http://www.csdn.net/tag/%e6%ba%90%e7%a0%81)[分类器](http://www.csdn.net/tag/%e5%88%86%e7%b1%bb%e5%99%a8)[机器学习](http://www.csdn.net/tag/%e6%9c%ba%e5%99%a8%e5%ad%a6%e4%b9%a0)

2014-09-07 15:43 3556人阅读 [评论](http://blog.csdn.net/roger__wong/article/details/39119701#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/roger__wong/article/details/39119701#report)

本文章已收录于：

[[](http://lib.csdn.net/base/2?source=blogtop)**机器学习知识库**](http://lib.csdn.net/base/2?source=blogtop)

http://static.blog.csdn.net/images/category_icon.jpg分类：

Weka算法及源码分析（15） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

一、问题

主要带着四个问题去研究J48的实现。

1、如何控制分类树的精度。

2、如何处理缺失的值（MissingValue）

3、如何对连续值进行离散化。

4、如何进行分类树的剪枝。

二、BuildClassifier

每一个分类器都会实现这个方法，传入一个Instances对象，在这个对象基础上进行来构建分类树。核心代码如下：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. public void buildClassifier(Instances instances)
2. throws Exception {
4. ModelSelection modSelection;
6. if (m\_binarySplits)
7. modSelection = new BinC45ModelSelection(m\_minNumObj, instances);
8. else
9. modSelection = new C45ModelSelection(m\_minNumObj, instances);
10. if (!m\_reducedErrorPruning)
11. m\_root = new C45PruneableClassifierTree(modSelection, !m\_unpruned, m\_CF,
12. m\_subtreeRaising, !m\_noCleanup);
13. else
14. m\_root = new PruneableClassifierTree(modSelection, !m\_unpruned, m\_numFolds,
15. !m\_noCleanup, m\_Seed);
16. m\_root.buildClassifier(instances);
17. if (m\_binarySplits) {
18. ((BinC45ModelSelection)modSelection).cleanup();
19. } else {
20. ((C45ModelSelection)modSelection).cleanup();
21. }
22. }

可以看到这段代码逻辑非常清楚，首先根据是否是一个二分树（即每个节点只有是否两种选择）来构造一个ModelSelection，随后根据是否有m\_reduceErrorPruning标志来构造相应的ClassifierTree，在这个tree上真正的构建模型，最后清理数据（主要是做释放指针的工作，防止Tree持有Instances指针导致GC不能在上层调用者想释放Instances的时候进行释放）。

三、C45PruneableClassifierTree

（1）该类也实现了BuildCClassifier方法来构建分类器，先看一下这个方法的主逻辑，代码如下：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. public void buildClassifier(Instances data) throws Exception {
3. // can classifier tree handle the data?
4. getCapabilities().testWithFail(data);
6. // remove instances with missing class
7. data = new Instances(data);
8. data.deleteWithMissingClass();
10. buildTree(data, m\_subtreeRaising || !m\_cleanup);
11. collapse();
12. if (m\_pruneTheTree) {
13. prune();
14. }
15. if (m\_cleanup) {
16. cleanup(new Instances(data, 0));
17. }
18. }

首先testWithFail是检测一下传入的data是否能用该分类器进行分类，比如C45只能对要分类的属性的取值是离散值的Instances进行分类，这个test就是检测诸如此类的逻辑。

接着清理一下instances里面的无效行（相应分类属性为空的行）。

在此数据上调用buildTree进行构建分类树。

调用collapse()进行树的“坍塌”（这里我不太知道学名应该怎么翻译）

如果有需要，则进行prune()剪枝。

最后清理数据。

（2）按照这个顺序首先来看buildTree函数

**[html]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. public void buildTree(Instances data, boolean keepData) throws Exception {
3. Instances [] localInstances;
5. if (keepData) {
6. m\_train = data;
7. }
8. m\_test = null;
9. m\_isLeaf = false;
10. m\_isEmpty = false;
11. m\_sons = null;
12. m\_localModel = m\_toSelectModel.selectModel(data);
13. if (m\_localModel.numSubsets() > 1) {
14. localInstances = m\_localModel.split(data);
15. data = null;
16. m\_sons = new ClassifierTree [m\_localModel.numSubsets()];
17. for (int i = 0; i < m\_sons.length; i++) {
18. m\_sons[i] = getNewTree(localInstances[i]);
19. localInstances[i] = null;
20. }
21. }else{
22. m\_isLeaf = true;
23. if (Utils.eq(data.sumOfWeights(), 0))
24. m\_isEmpty = true;
25. data = null;
26. }
27. }

该函数逻辑也比较简单（怎么都比较简单？！），首先根据传入参数来判断是否应该持有数据。

然后根据m\_toSelectModel来选择一个模型并把传入的数据集按相应的规则分成不同的subSet，这个selectModel是构造函数传入的，参见刚才描述的主流程。这一步如果对应上篇博客的[**算法**](http://lib.csdn.net/base/datastructure)描述，得到的subSet就是第10行的dv。

接着判断subSet的数量，如果只有一个，那么就是一个叶子节点，什么都不用做就返回了。

否则根据localModel将data分成不同的subInstances，接着为每一个subInstances建立新的ClassifierTree节点作为自己的孩子节点，并调用getNewTree函数来为每一个subInstances构造新的tree。

（3）采用DFS的方式接着去看一下getNewTree的逻辑

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. protected ClassifierTree getNewTree(Instances data) throws Exception {
3. ClassifierTree newTree = new ClassifierTree(m\_toSelectModel);
4. newTree.buildTree(data, false);
6. return newTree;
7. }

很简单，就是一个递归调用。

（4）重新回到C45PruneableClassifierTree.buildClassifier方法，来研究一下其中的collapse函数。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. /\*\*
2. \* Collapses a tree to a node if training error doesn't increase.
3. \*/
4. public final void collapse(){
6. double errorsOfSubtree;
7. double errorsOfTree;
8. int i;
10. if (!m\_isLeaf){
11. errorsOfSubtree = getTrainingErrors();
12. errorsOfTree = localModel().distribution().numIncorrect();
13. if (errorsOfSubtree >= errorsOfTree-1E-3){
15. // Free adjacent trees
16. m\_sons = null;
17. m\_isLeaf = true;
19. // Get NoSplit Model for tree.
20. m\_localModel = new NoSplit(localModel().distribution());
21. }else
22. for (i=0;i<m\_sons.length;i++)
23. son(i).collapse();
24. }
25. }

通过注释也可以看出，如果该节点的存在很多孩子节点，但这些孩子节点并不能提高这颗分类树的准确度，则把这些孩子节点删除。否则在每个孩子上递归的坍塌。通过collapse方法可以在不减少精度的前提下减少决策树的深度，进而提高效率。

简单说一下如何估计当前的节点的错误，也就是localModel().distribution().numIncorrect();

首先获得当前训练集上的一个分布，然后找出该分布里数量最多的那个属性的数量，认为是“正确的”，则其余的就是错误的。

getTrainingError就是对每个孩子节点做上述操作，然后结果相加。

（5）再来看看prune()方法，也是C45PruneableClassifierTree的BuildClassifier中的最后一个步骤。

该函数比较长，我就直接把对这个函数的分析写在注释里了。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. public void prune() throws Exception {
2. double errorsLargestBranch;//这个树节点的孩子节点中，肯定有一个分到的数据最多，该值记录该孩子节点分类错误的用例数
3. double errorsLeaf;//如果该节点成为了叶子节点，则分类错误的用例数量
4. double errorsTree;//<span style="font-family: Arial, Helvetica, sans-serif;">该节点目前情况下，错误用例数量</span>
5. int indexOfLargestBranch;//那个分到最多数据的孩子节点在son数组中的index
6. C45PruneableClassifierTree largestBranch;//son[indexOfLargestBranch]
7. int i;
9. if (!m\_isLeaf){

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //首先，如果是叶子节点，则先递归的队所有孩子几点进行prune()。
2. for (i=0;i<m\_sons.length;i++)
3. son(i).prune();

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //通过数据集的分布，很容易能找到indexOfLargetBranch
2. indexOfLargestBranch = localModel().distribution().maxBag();
3. if (m\_subtreeRaising) {

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //m\_subtreeRaising是一个标志，代表可否使用该树的子树去替代该树，如果有了这个标志，就去计算最大的子树的错误数量

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //否则就简单的标Double.Max\_Value

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //对于错误数量的估计不展开说了，简单来说依然是根据分布做一个统计（还要加一个基于m\_CF的修正），如果不是叶子节点则递

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //归的进行统计。
2. errorsLargestBranch = son(indexOfLargestBranch).
3. getEstimatedErrorsForBranch((Instances)m\_train);
4. } else {
5. errorsLargestBranch = Double.MAX\_VALUE;
6. }
8. //估计一下如果该节点成为了叶子节点，则错误数量大概有多少
9. errorsLeaf =
10. getEstimatedErrorsForDistribution(localModel().distribution());

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //估计该节点目前情况下，错误用例数量。
2. errorsTree = getEstimatedErrors();
4. //Utils.smOrEq是smaller or equal即<=的意思
5. if (Utils.smOrEq(errorsLeaf,errorsTree+0.1) &&
6. Utils.smOrEq(errorsLeaf,errorsLargestBranch+0.1)){
8. // 如果当前节点作为叶子节点的错误量比整棵树都要低，并且当前节点比最大的子树的错误量也低，那么就把当前节点作//为叶子节点一定是一个最优的选择。
9. m\_sons = null;
10. m\_isLeaf = true;
12. // Get NoSplit Model for node.
13. m\_localModel = new NoSplit(localModel().distribution());
14. return;//直接返回
15. }
17. // Decide if largest branch is better choice
18. // than whole subtree.
19. if (Utils.smOrEq(errorsLargestBranch,errorsTree+0.1)){

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. //如果当前节点的错误用例数大于最大子树，则用最大子树替代当前节点。
2. largestBranch = son(indexOfLargestBranch);
3. m\_sons = largestBranch.m\_sons;
4. m\_localModel = largestBranch.localModel();
5. m\_isLeaf = largestBranch.m\_isLeaf;
6. newDistribution(m\_train);
7. prune();
8. }
9. }
10. }

一句话总结collapse和prune：prune或许会影响精度，collapse不会。

四、PruneableClassifierTree

在J48主流程里，根据m\_reducedErrorPruning的不同会选择两个不同的ClassifierTree，刚才已经分析了一个，另外一个则是PruneeableClassifierTree。

（1）buildClassifier

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. public void buildClassifier(Instances data)
2. throws Exception {
4. // can classifier tree handle the data?
5. getCapabilities().testWithFail(data);
7. // remove instances with missing class
8. data = new Instances(data);
9. data.deleteWithMissingClass();
11. Random random = new Random(m\_seed);
12. data.stratify(numSets);
13. buildTree(data.trainCV(numSets, numSets - 1, random),
14. data.testCV(numSets, numSets - 1), !m\_cleanup);
15. if (pruneTheTree) {
16. prune();
17. }
18. if (m\_cleanup) {
19. cleanup(new Instances(data, 0));
20. }
21. }

和C45PruneableClassifierTree不同的是，buildTree的时候除了传入训练集，还传入了测试集，除此之外，少了Collapse步骤，其余都一样。

下面就看看传入了[**测试**](http://lib.csdn.net/base/softwaretest)集的build和之前分析的build有什么不同之处。

（2）buildTree

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. public void buildTree(Instances train, Instances test, boolean keepData)
2. throws Exception {
4. Instances [] localTrain, localTest;
5. int i;
7. if (keepData) {
8. m\_train = train;
9. }
10. m\_isLeaf = false;
11. m\_isEmpty = false;
12. m\_sons = null;
13. m\_localModel = m\_toSelectModel.selectModel(train, test);
14. m\_test = new Distribution(test, m\_localModel);
15. if (m\_localModel.numSubsets() > 1) {
16. localTrain = m\_localModel.split(train);
17. localTest = m\_localModel.split(test);
18. train = test = null;
19. m\_sons = new ClassifierTree [m\_localModel.numSubsets()];
20. for (i=0;i<m\_sons.length;i++) {
21. m\_sons[i] = getNewTree(localTrain[i], localTest[i]);
22. localTrain[i] = null;
23. localTest[i] = null;
24. }
25. }else{
26. m\_isLeaf = true;
27. if (Utils.eq(train.sumOfWeights(), 0))
28. m\_isEmpty = true;
29. train = test = null;
30. }
31. }

可以看到，代码基本一样，唯一不同的地方就是selectModel的时候会把test传进去，对于Model的实现会具体放到下篇博客中去讲述。

而prune也更为简单，去掉了subTreeRasing的特性。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39119701) [copy](http://blog.csdn.net/roger__wong/article/details/39119701)

1. public void prune() throws Exception {
3. if (!m\_isLeaf) {
5. // Prune all subtrees.
6. for (int i = 0; i < m\_sons.length; i++)
7. son(i).prune();
9. // Decide if leaf is best choice.
10. if (Utils.smOrEq(errorsForLeaf(),errorsForTree())) {
12. // Free son Trees
13. m\_sons = null;
14. m\_isLeaf = true;
16. // Get NoSplit Model for node.
17. m\_localModel = new NoSplit(localModel().distribution());
18. }
19. }
20. }

五、总结

至此，对两种ClassifierTree的buildClassifier的分析差不多就结束了，总体上来讲，ClassifierTree是通过传入的Model来构建并维护分类树的结构，除此之外在构建完毕后会按照不同的逻辑进行剪枝。

对于篇开头提出的问题，目前可以回答问题4，简而言之就是根据已有数据集的分布，判断该树、该树的最大子树、以及该树作为叶子节点时的正确率，在此基础上进行剪枝。

下篇文章主要分析Model的实现，也就是如何根据属性把已有的数据集分解subInstances

[**Weka算法Classifier-tree-J48源码分析（三）ModelSelection**](http://blog.csdn.net/roger__wong/article/details/39121709)

标签： [分类器](http://www.csdn.net/tag/%e5%88%86%e7%b1%bb%e5%99%a8)[Weka](http://www.csdn.net/tag/Weka)[源码](http://www.csdn.net/tag/%e6%ba%90%e7%a0%81)[算法](http://www.csdn.net/tag/%e7%ae%97%e6%b3%95)[C4.5](http://www.csdn.net/tag/C4.5)

2014-09-07 21:30 2787人阅读 [评论](http://blog.csdn.net/roger__wong/article/details/39121709#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/roger__wong/article/details/39121709#report)

http://static.blog.csdn.net/images/category_icon.jpg分类：

Weka算法及源码分析*（15）* http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

ModelSelection主要是用于选择合适的列对数据集进行分割，结合上一篇J48的主流程，发现用到的ModelSelection有 C45ModelSelection以及BinC45ModelSelection，先来分析C45ModelSelection。

一、C45ModelSelection

首先作为一个ModelSelection接口，实现的主要方法有两个，分别是selectModel(Instances)和selectionModel(Instances,Instances)。C45ModelSelection的后一个方法如下：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. public final ClassifierSplitModel selectModel(Instances train, Instances test) {
3. return selectModel(train);
4. }

可以看到就是忽略了test测试集直接调用selectModel方法而已，因此主要分词selectModel方法。

先放出整段代码，然后对该段代码进行分析：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. public final ClassifierSplitModel selectModel(Instances data){
3. double minResult;
4. double currentResult;
5. C45Split [] currentModel;
6. C45Split bestModel = null;
7. NoSplit noSplitModel = null;
8. double averageInfoGain = 0;
9. int validModels = 0;
10. boolean multiVal = true;
11. Distribution checkDistribution;
12. Attribute attribute;
13. double sumOfWeights;
14. int i;
16. try{
18. // Check if all Instances belong to one class or if not
19. // enough Instances to split.
20. checkDistribution = new Distribution(data);
21. noSplitModel = new NoSplit(checkDistribution);
22. if (Utils.sm(checkDistribution.total(),2\*m\_minNoObj) ||
23. Utils.eq(checkDistribution.total(),
24. checkDistribution.perClass(checkDistribution.maxClass())))
25. return noSplitModel;
27. // Check if all attributes are nominal and have a
28. // lot of values.
29. if (m\_allData != null) {
30. Enumeration enu = data.enumerateAttributes();
31. while (enu.hasMoreElements()) {
32. attribute = (Attribute) enu.nextElement();
33. if ((attribute.isNumeric()) ||
34. (Utils.sm((double)attribute.numValues(),
35. (0.3\*(double)m\_allData.numInstances())))){
36. multiVal = false;
37. break;
38. }
39. }
40. }
42. currentModel = new C45Split[data.numAttributes()];
43. sumOfWeights = data.sumOfWeights();
45. // For each attribute.
46. for (i = 0; i < data.numAttributes(); i++){
48. // Apart from class attribute.
49. if (i != (data).classIndex()){
51. // Get models for current attribute.
52. currentModel[i] = new C45Split(i,m\_minNoObj,sumOfWeights);
53. currentModel[i].buildClassifier(data);
55. // Check if useful split for current attribute
56. // exists and check for enumerated attributes with
57. // a lot of values.
58. if (currentModel[i].checkModel())
59. if (m\_allData != null) {
60. if ((data.attribute(i).isNumeric()) ||
61. (multiVal || Utils.sm((double)data.attribute(i).numValues(),
62. (0.3\*(double)m\_allData.numInstances())))){
63. averageInfoGain = averageInfoGain+currentModel[i].infoGain();
64. validModels++;
65. }
66. } else {
67. averageInfoGain = averageInfoGain+currentModel[i].infoGain();
68. validModels++;
69. }
70. }else
71. currentModel[i] = null;
72. }
74. // Check if any useful split was found.
75. if (validModels == 0)
76. return noSplitModel;
77. averageInfoGain = averageInfoGain/(double)validModels;
79. // Find "best" attribute to split on.
80. minResult = 0;
81. for (i=0;i<data.numAttributes();i++){
82. if ((i != (data).classIndex()) &&
83. (currentModel[i].checkModel()))
85. // Use 1E-3 here to get a closer approximation to the original
86. // implementation.
87. if ((currentModel[i].infoGain() >= (averageInfoGain-1E-3)) &&
88. Utils.gr(currentModel[i].gainRatio(),minResult)){
89. bestModel = currentModel[i];
90. minResult = currentModel[i].gainRatio();
91. }
92. }
94. // Check if useful split was found.
95. if (Utils.eq(minResult,0))
96. return noSplitModel;
98. // Add all Instances with unknown values for the corresponding
99. // attribute to the distribution for the model, so that
100. // the complete distribution is stored with the model.
101. bestModel.distribution().
102. addInstWithUnknown(data,bestModel.attIndex());
104. // Set the split point analogue to C45 if attribute numeric.
105. if (m\_allData != null)
106. bestModel.setSplitPoint(m\_allData);
107. return bestModel;
108. }catch(Exception e){
109. e.printStackTrace();
110. }
111. return null;
112. }

第一部分，主要是对局部变量的一些定义。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. double minResult;//最小的信息增益率
2. double currentResult;//当前信息增益率
3. C45Split [] currentModel;//存放所有未分类属性产生的模型
4. C45Split bestModel = null;//目前为止的最好模型
5. NoSplit noSplitModel = null;//代表不用分的模型
6. double averageInfoGain = 0;//各模型(currentModel)的平均信息增益
7. int validModels = 0;//是否存在有效模型
8. boolean multiVal = true;//是否多值
9. Distribution checkDistribution;//训练数据集的分布
10. Attribute attribute;//属性列集合
11. double sumOfWeights;//训练数据集的weight的和
12. int i;//循环变量

第二部分，递归出口。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. checkDistribution = new Distribution(data);
2. noSplitModel = new NoSplit(checkDistribution);
3. if (Utils.sm(checkDistribution.total(),2\*m\_minNoObj) ||
4. Utils.eq(checkDistribution.total(),
5. checkDistribution.perClass(checkDistribution.maxClass())))
6. return noSplitModel;

可以看到，如果当前数据集数量小于2\*m\_minNoObj(这个值默认是2)，或者当前数据集已经全在同一个分类中，就返回noSplitModel代表不用分，这就是整个C45分类树节点停止分裂的条件。

第三部分，判断是否是多值：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. if (m\_allData != null) {
2. Enumeration enu = data.enumerateAttributes();
3. while (enu.hasMoreElements()) {
4. attribute = (Attribute) enu.nextElement();
5. if ((attribute.isNumeric()) ||
6. (Utils.sm((double)attribute.numValues(),
7. (0.3\*(double)m\_allData.numInstances())))){
8. multiVal = false;
9. break;
10. }
11. }
12. }

如果属性中，任意一列是数值型，或者其取值的数量小于训练集数量\*0.3，则不是多值，否则按多值处理。是否是多值影响到后面某些逻辑。

第四部分，对于每一列属性构造Spliter。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. for (i = 0; i < data.numAttributes(); i++){
3. // Apart from class attribute.
4. if (i != (data).classIndex()){
6. // Get models for current attribute.
7. currentModel[i] = new C45Split(i,m\_minNoObj,sumOfWeights);
8. currentModel[i].buildClassifier(data);
10. // Check if useful split for current attribute
11. // exists and check for enumerated attributes with
12. // a lot of values.
13. if (currentModel[i].checkModel())
14. if (m\_allData != null) {
15. if ((data.attribute(i).isNumeric()) ||
16. (multiVal || Utils.sm((double)data.attribute(i).numValues(),
17. (0.3\*(double)m\_allData.numInstances())))){
18. averageInfoGain = averageInfoGain+currentModel[i].infoGain();
19. validModels++;
20. }
21. } else {
22. averageInfoGain = averageInfoGain+currentModel[i].infoGain();
23. validModels++;
24. }
25. }else
26. currentModel[i] = null;
27. }

对于每一列属性，如果不是存放分类的值得话，则构造C45Split对象，在该对象上进行分类，然后算出信息增益，相加到averageInfoGain上。对于C45Split的构造，稍后再看。

第五部分，选出最优模型。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. if (validModels == 0)
2. return noSplitModel;
3. averageInfoGain = averageInfoGain/(double)validModels;
5. // Find "best" attribute to split on.
6. minResult = 0;
7. for (i=0;i<data.numAttributes();i++){
8. if ((i != (data).classIndex()) &&
9. (currentModel[i].checkModel()))
11. // Use 1E-3 here to get a closer approximation to the original
12. // implementation.
13. if ((currentModel[i].infoGain() >= (averageInfoGain-1E-3)) &&
14. Utils.gr(currentModel[i].gainRatio(),minResult)){
15. bestModel = currentModel[i];
16. minResult = currentModel[i].gainRatio();
17. }

如果存在有效模型，则选出有效模型。注意这个选出最优模型的逻辑，并不是单纯的选出gainRatio最大的，而是在基础上必须还要大于平均信息增益，这也是和传统的c45算法不一样的一点。

从上述过程来看，Weka在实现C45的时候做了一个小的变动，并没有从“还没有使用的”属性列中找出最合理的列最为分割属性，而是在“所有的列”中找出最合理的列作为分割属性，虽然这二者在结果上肯定是等价的（之前是有过的属性不和能有很好的信息增益率），但效率上个人对Weka的做法持保留意见。

二、C45Spliter

在ModelSelection中真正根据属性对训练集进行分割、计算信息增益和信息增益率的是C45Spliter，首先也从其buildClassifier方法入手进行分析。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. public void buildClassifier(Instances trainInstances)
2. throws Exception {
4. // Initialize the remaining instance variables.
5. m\_numSubsets = 0;
6. m\_splitPoint = Double.MAX\_VALUE;
7. m\_infoGain = 0;
8. m\_gainRatio = 0;
10. // Different treatment for enumerated and numeric
11. // attributes.
12. if (trainInstances.attribute(m\_attIndex).isNominal()) {
13. m\_complexityIndex = trainInstances.attribute(m\_attIndex).numValues();
14. m\_index = m\_complexityIndex;
15. handleEnumeratedAttribute(trainInstances);
16. }else{
17. m\_complexityIndex = 2;
18. m\_index = 0;
19. trainInstances.sort(trainInstances.attribute(m\_attIndex));
20. handleNumericAttribute(trainInstances);
21. }
22. }

可以看到，对于枚举型和数值型的属性是分开处理的，枚举型调用handlEnumeratedAttribute，数值型调用handleNumericAttribute，值得注意的是，在处理数值型之前，按照相应列进行排序，同时设置m\_complexityIndex也就是期望分裂的节点数设定为2。

首先来看枚举类型是如何处理的。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. private void handleEnumeratedAttribute(Instances trainInstances)
2. throws Exception {
4. Instance instance;
6. m\_distribution = new Distribution(m\_complexityIndex,
7. trainInstances.numClasses());
9. // Only Instances with known values are relevant.
10. Enumeration enu = trainInstances.enumerateInstances();
11. while (enu.hasMoreElements()) {
12. instance = (Instance) enu.nextElement();
13. if (!instance.isMissing(m\_attIndex))
14. m\_distribution.add((int)instance.value(m\_attIndex),instance);
15. }
17. // Check if minimum number of Instances in at least two
18. // subsets.
19. if (m\_distribution.check(m\_minNoObj)) {
20. m\_numSubsets = m\_complexityIndex;
21. m\_infoGain = infoGainCrit.
22. splitCritValue(m\_distribution,m\_sumOfWeights);
23. m\_gainRatio =
24. gainRatioCrit.splitCritValue(m\_distribution,m\_sumOfWeights,
25. m\_infoGain);
26. }
27. }

大概流程是新建一个分布，遍历所有instance，如果该instance对应的分裂的属性不为空的话，则放到不同的bag里，之后检查一下这个分布是否满足要求，要求就是最多允许有一个bag里的数据数量小于m\_minNoObj，如果通过检查，就设置subset的数量，计算信息增益和信息增益率，否则subset默认会是0，上层调用checkModel就会返回false代表这是一个无效模型。

接下来看数值型是如何处理的：

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. private void handleNumericAttribute(Instances trainInstances)
2. throws Exception {
4. int firstMiss;//最后一个有效instance的下标
5. int next = 1;//下一个instance的index
6. int last = 0;//当前instance的index
7. int splitIndex = -1;//分裂点
8. double currentInfoGain;//当前信息增益
9. double defaultEnt;//分割之前的信息熵
10. double minSplit;
11. Instance instance;
12. int i;

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. //首先新建一个分布，数值型默认处理为2维分布，也就可以理解为小于某个值放到一个Bag里，其余的放到另外一个Bag里

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. m\_distribution = new Distribution(2,trainInstances.numClasses());
2. Enumeration enu = trainInstances.enumerateInstances();
3. i = 0;

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. <pre name="code" class="cpp">//注意instances传入的时候是排好序的，这个排序保证了missingValue放在最后面，所以读到了missingValue其之后肯定都是miss//ingValue，换言之，firstMiss在循环之后代表了最后一个有效的instance的下标。

while (enu.hasMoreElements()) { instance = (Instance) enu.nextElement(); if (instance.isMissing(m\_attIndex))break; m\_distribution.add(1,instance); i++; } firstMiss = i;//循环结束后，m\_distribution里放入了所有的有效instance，并全放入了bag1里。

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. //minSplit是最后分类好每个Bag里最小的数据的量，也就是0.1\*每个类的均值。
2. minSplit =  0.1\*(m\_distribution.total())/
3. ((double)trainInstances.numClasses());
4. if (Utils.smOrEq(minSplit,m\_minNoObj))
5. minSplit = m\_minNoObj;
6. else
7. if (Utils.gr(minSplit,25))
8. minSplit = 25;
10. //如果有效数据总量不到2\*minSplit，换言之无论怎么分均不能保证2个bag里的数量大于minSplit，就直接返回。
11. if (Utils.sm((double)firstMiss,2\*minSplit))
12. return;
14. //defaultEnt代表旧的信息熵，也就是对该属性进行分类之前，Indexclass对应的信息熵。
15. defaultEnt = infoGainCrit.oldEnt(m\_distribution);
16. while (next < firstMiss) {
18. if (trainInstances.instance(next-1).value(m\_attIndex)+1e-5 <
19. trainInstances.instance(next).value(m\_attIndex)) {
20. <pre name="code" class="cpp">//Instances里的记录是升序排列的，加上这个条件默认把值相差很小的Instance就当做同一个instance处理了

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. //last代表当前，next代表下一个，默认next=1,last=0，所以shiftRange可以理解成把当前记录从bag1移动到bag0中

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. <span style="font-family: Arial, Helvetica, sans-serif;">//注意一开始初始化时候所有的都是在bag1里面的。   </span>

m\_distribution.shiftRange(1,0,trainInstances,last,next);if (Utils.grOrEq(m\_distribution.perBag(0),minSplit) && //如果两个bag都满足最小数据集的数量minSplit Utils.grOrEq(m\_distribution.perBag(1),minSplit)) { currentInfoGain = infoGainCrit. splitCritValue(m\_distribution,m\_sumOfWeights, //算一下信息增益 defaultEnt);

**[cpp]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. if (Utils.gr(currentInfoGain,m\_infoGain)) {
2. m\_infoGain = currentInfoGain;//如果信息增益比当前最大的要大，则替换当前最大的值，并记录splitIndex
3. splitIndex = next-1;
4. }
5. m\_index++;
6. }
7. last = next;
8. }
9. next++;
10. }
12. if (m\_index == 0)
13. return; //执行到这里说明没找到一个合适的分裂点，直接返回。
15. // 计算最佳信息增益
16. m\_infoGain = m\_infoGain-(Utils.log2(m\_index)/m\_sumOfWeights);
17. if (Utils.smOrEq(m\_infoGain,0))
18. return; //如果信息增益是0也说明没找到合适的分裂点，直接返回。
20. //剩下的就是根据分裂点进行属性的划分。
21. m\_numSubsets = 2;
22. m\_splitPoint =
23. (trainInstances.instance(splitIndex+1).value(m\_attIndex)+
24. trainInstances.instance(splitIndex).value(m\_attIndex))/2;
26. // In case we have a numerical precision problem we need to choose the
27. // smaller value
28. if (m\_splitPoint == trainInstances.instance(splitIndex + 1).value(m\_attIndex)) {
29. m\_splitPoint = trainInstances.instance(splitIndex).value(m\_attIndex);
30. }
32. // Restore distributioN for best split.
33. m\_distribution = new Distribution(2,trainInstances.numClasses());
34. m\_distribution.addRange(0,trainInstances,0,splitIndex+1);
35. m\_distribution.addRange(1,trainInstances,splitIndex+1,firstMiss);
37. // Compute modified gain ratio for best split.
38. m\_gainRatio = gainRatioCrit.
39. splitCritValue(m\_distribution,m\_sumOfWeights,
40. m\_infoGain);
41. }

这个函数有点复杂，具体逻辑也写到代码注释里了。  
  
  
三、BinC45ModelSelection

该函数只负责生成二元分类树的模型，selectModel方法和C45ModelSelection几乎一样，不在多说，不同点在于其使用BinC45Spliter而不是C45Spliter。

四、BinC45Spliter

 handleNumericAttribute对于数值类型的属性处理和C45Spliter完全一样。下面只分析一下handleEnumeratedAttribute。

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. private void handleEnumeratedAttribute(Instances trainInstances)
2. throws Exception {
4. Distribution newDistribution,secondDistribution;
5. int numAttValues;
6. double currIG,currGR;
7. Instance instance;
8. int i;
10. numAttValues = trainInstances.attribute(m\_attIndex).numValues();
11. newDistribution = new Distribution(numAttValues,
12. trainInstances.numClasses());
14. // Only Instances with known values are relevant.
15. Enumeration enu = trainInstances.enumerateInstances();
16. while (enu.hasMoreElements()) {
17. instance = (Instance) enu.nextElement();
18. if (!instance.isMissing(m\_attIndex))
19. newDistribution.add((int)instance.value(m\_attIndex),instance);
20. }
21. m\_distribution = newDistribution;
23. // For all values
24. for (i = 0; i < numAttValues; i++){
26. if (Utils.grOrEq(newDistribution.perBag(i),m\_minNoObj)){
27. secondDistribution = new Distribution(newDistribution,i);
29. // Check if minimum number of Instances in the two
30. // subsets.
31. if (secondDistribution.check(m\_minNoObj)){
32. m\_numSubsets = 2;
33. currIG = m\_infoGainCrit.splitCritValue(secondDistribution,
34. m\_sumOfWeights);
35. currGR = m\_gainRatioCrit.splitCritValue(secondDistribution,
36. m\_sumOfWeights,
37. currIG);
38. if ((i == 0) || Utils.gr(currGR,m\_gainRatio)){
39. m\_gainRatio = currGR;
40. m\_infoGain = currIG;
41. m\_splitPoint = (double)i;
42. m\_distribution = secondDistribution;
43. }
44. }
45. }
46. }

可以看出，上一段代码根据该属性的不同的取值，在已有分布基础上，建立一个新的分布secondeDistribution，

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39121709) [copy](http://blog.csdn.net/roger__wong/article/details/39121709)

1. secondDistribution = new Distribution(newDistribution,i);

该分布包含两列，属性下标为i的，其余的，在这个分布的基础上计算信息增益和信息增益率，并选出最优的。

换句话说，离散值分类的二元化处理就是选出其中一列当做一个branch，其余的当做另外一个branch。虽然从结构上来讲这肯定不是最优的选择，但简单易用就够了。

到这里基本分析完了J48的两个ModelSelection，下一篇文章将对classifierInstance过程进行分析，并给出一个简单的总结。

[**Weka算法Classifier-tree-J48源码分析（四）总结**](http://blog.csdn.net/roger__wong/article/details/39136549)

标签： [算法](http://www.csdn.net/tag/%e7%ae%97%e6%b3%95)[源码](http://www.csdn.net/tag/%e6%ba%90%e7%a0%81)[Weka](http://www.csdn.net/tag/Weka)[机器学习](http://www.csdn.net/tag/%e6%9c%ba%e5%99%a8%e5%ad%a6%e4%b9%a0)[分类器](http://www.csdn.net/tag/%e5%88%86%e7%b1%bb%e5%99%a8)

2014-09-08 15:16 4438人阅读 [评论](http://blog.csdn.net/roger__wong/article/details/39136549#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/roger__wong/article/details/39136549#report)

http://static.blog.csdn.net/images/category_icon.jpg分类：

Weka算法及源码分析*（15）* http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

一、ClassifyInstance

首先先说一下构造好的分类树是如何对一个新的Instance进行区分。

直观上，会对树进行一个检索，从根节点根据属性的不同，最终走到叶子节点，得到具体的分类。

但Weka在实现上，是遍历了这个Instance属于不同的class的可能性，并从中选出了一个最大的，代码如下：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39136549) [copy](http://blog.csdn.net/roger__wong/article/details/39136549)

1. public double classifyInstance(Instance instance)
2. throws Exception {
4. double maxProb = -1;
5. double currentProb;
6. int maxIndex = 0;
7. int j;
9. for (j = 0; j < instance.numClasses(); j++) {
10. currentProb = getProbs(j, instance, 1);
11. if (Utils.gr(currentProb,maxProb)) {
12. maxIndex = j;
13. maxProb = currentProb;
14. }
15. }
17. return (double)maxIndex;
18. }

而getProbs函数关键代码如下：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39136549) [copy](http://blog.csdn.net/roger__wong/article/details/39136549)

1. if (m\_isLeaf) {
2. return weight \* localModel().classProb(classIndex, instance, -1);
3. } else {
4. int treeIndex = localModel().whichSubset(instance);
5. if (treeIndex == -1) {
6. double[] weights = localModel().weights(instance);
7. for (int i = 0; i < m\_sons.length; i++) {
8. if (!son(i).m\_isEmpty) {
9. prob += son(i).getProbs(classIndex, instance,
10. weights[i] \* weight);
11. }

如果是一个叶子节点，则直接从本节点的分布就能确定当前instance属于相关class的可能性，否则就要取出treeIndex进行判断，这里的treeIndex==-1代表这个Instance属于不同的branch，换句话说，本节点的Model不能对此Instance给出一个确定的划分。

在分类树中这种情况理论上是不会出现的，weka在这里其实是用treeIndex==-1来处理了相关属性缺失的情况。

whichSubset代码如下：

**[java]** [view plain](http://blog.csdn.net/roger__wong/article/details/39136549) [copy](http://blog.csdn.net/roger__wong/article/details/39136549)

1. public final int whichSubset(Instance instance)
2. throws Exception {
4. if (instance.isMissing(m\_attIndex))
5. return -1;
6. else{
7. if (instance.attribute(m\_attIndex).isNominal())
8. return (int)instance.value(m\_attIndex);
9. else
10. if (Utils.smOrEq(instance.value(m\_attIndex),m\_splitPoint))
11. return 0;
12. else
13. return 1;
14. }
15. }

如果属性缺失，则直接返回-1，如果是离散值，则根据属性进行选择，如果是连续值，则通过训练时得到的分裂点进行判断。

二、总结

接下来可以回答第二篇博客开头时提到的4个问题了。

1、如何控制分类树的精度。

答：使用minNoObj参数来控制分类树的精度，节点停止分裂的条件有5个：

（1）所有的instances已经属于同一个分类（selectModel里）

（2）instances数量小于2\*minNoObj（selectModel里）

（3）一个分裂产生的信息增益石0（selectModel里）

（4）对离散值进行分裂节点的计算时，超过一个的Bag里的instance数量小于minNoObj（spliter里）

（5）对连续值进行分裂计算时，有效instances数量小于2\*minNoObj（spliter里）

2、如何处理缺失的值（MissingValue）

答：对缺失值得处理分两种，其一是训练阶段的缺失值处理，方法是直接忽略掉这条记录。在分类阶段对缺失值得处理是，忽略该属性，使其在不同的branch上进行计算，最后将概率结果进行合并。

3、如何对连续值进行离散化。

答：Spliter首先对连续值进行排序，之后从前到后遍历每一个值当做分裂点，对分裂结果计算信息增益率最大的作为最终分裂点构建不同的branch。

4、如何进行分类树的剪枝。

答：略，参见第二篇博客的collapse方法和prune方法。  
  
关于J48差不多也就说到这里了，通过分析源码可以看到Weka的实现和原始的C4.5算法还是有相当多的不一样的地方。下一篇将对RandomForest算法进行分析。