机器学习中的算法(1)-决策树模型组合之随机森林与GBDT

版权声明：

前言：

决策树这种算法有着很多良好的特性，比如说训练时间复杂度较低，预测的过程比较快速，模型容易展示（容易将得到的决策树做成图片展示出来）等。但是同时，单决策树又有一些不好的地方，比如说容易over-fitting，虽然有一些方法，如剪枝可以减少这种情况，但是还是不够的。

模型组合（比如说有Boosting，Bagging等）与决策树相关的算法比较多，这些算法最终的结果是生成N(可能会有几百棵以上）棵树，这样可以大大的减少单决策树带来的毛病，有点类似于三个臭皮匠等于一个诸葛亮的做法，虽然这几百棵决策树中的每一棵都很简单（相对于C4.5这种单决策树来说），但是他们组合起来确是很强大。

在最近几年的paper上，如iccv这种重量级的会议，iccv 09年的里面有不少的文章都是与Boosting与随机森林相关的。模型组合+决策树相关的算法有两种比较基本的形式 - 随机森林与GBDT((Gradient Boost Decision Tree)，其他的比较新的模型组合+决策树的算法都是来自这两种算法的延伸。本文主要侧重于GBDT，对于随机森林只是大概提提，因为它相对比较简单。

在看本文之前，建议先看看机器学习与数学(3)与其中引用的论文，本文中的GBDT主要基于此，而随机森林相对比较独立。

基础内容：

这里只是准备简单谈谈基础的内容，主要参考一下别人的文章，对于随机森林与GBDT，有两个地方比较重要，首先是information gain，其次是决策树。这里特别推荐Andrew Moore大牛的Decision Trees Tutorial，与Information Gain Tutorial。Moore的Data Mining Tutorial系列非常赞，看懂了上面说的两个内容之后的文章才能继续读下去。

决策树实际上是将空间用超平面进行划分的一种方法，每次分割的时候，都将当前的空间一分为二，比如说下面的决策树：

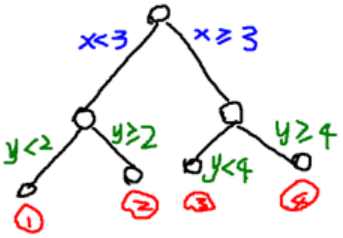
导航
博客园
首页
联系
订阅XML
管理

我的标签
机器学习(7)
搜索引擎(7)
Lucene(6)
machine learning(4)
pymining(3)
mathmatics(3)
人工智能(3)
数据挖掘(3)
PCA(2)
Model Combining(2)
更多

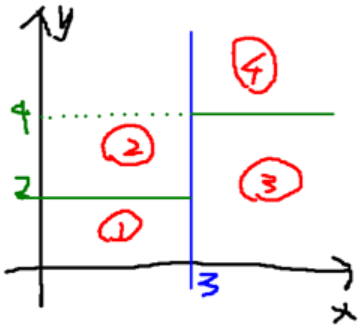
随笔分类
Hadoop(2)
Lucene C++重写心得(2)
Lucene JAVA心得(9)
安排，计划，总结(2)
分布式存储(2)
机器学习(7)
结构设计(1)
数学(7)

随笔档案
2012年2月 (1)
2011年9月 (1)
2011年8月 (1)
2011年5月 (5)
2011年3月 (1)
2011年2月 (1)
2011年1月 (3)
2010年12月 (2)
2010年11月 (1)
2010年10月 (1)
2010年9月 (2)
2010年8月 (1)
2010年1月 (12)
2009年12月 (2)
2009年11月 (4)

最新评论
1. Re:机器学习中的数学(5)-强大的矩阵奇异值分解(SVD)及其应用
很好！！谢谢
--他们是青年
2. Re:机器学习中的数学(3)-模型组合(Model Combining)之Boosting与Gradient Boosting
楼主，你好，请问你的那些图片使用什么工具得到的呢？
--hi，daring



就是将空间划分成下面的样子：



这样使得每一个叶子节点都是在空间中的一个不相交的区域，在进行决策的时候，会根据输入样本每一维feature的值，一步一步往下，最后使得样本落入N个区域中的一个（假设有N个叶子节点）

随机森林(Random Forest):

随机森林是一个最近比较火的算法，它有很多的优点：

- 在数据集上表现良好
- 在当前的很多数据集上，相对其他算法有着很大的优势
- 它能够处理很高维度（feature很多）的数据，并且不用做特征选择
- 在训练完后，它能够给出哪些feature比较重要
- 在创建随机森林的时候，对generlization error使用的是无偏估计
- 训练速度快
- 在训练过程中，能够检测到feature间的互相影响
- 容易做成并行化方法
- 实现比较简单

随机森林顾名思义，是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。

在建立每一棵决策树的过程中，有两点需要注意 - 采样与完全分裂。首先是两个随机采样的过程，random forest对输入的数据要进行行、列的采样。对于行采样，采用有放回的方式，也就是在采样得到的样本集合中，可能有重复的样本。假设输入样本为N个，那么采样的样本也为N个。这样使得在训练的时候，每一棵树的输入样本都不是全部的样本，使得相对不容易出现over-fitting。然后进行列采样，从M个feature中，选择m个(m << M)。之后就是对采样之后的数据使用完全分裂的方式建立出决策树，这样决策树的某一个叶子节点要么是无法继续分裂的，要么里面的所有样本的都是指向的同一个分类。一般很多的决策树算法都一个

重要的步骤 - 剪枝，但是这里不这样干，由于之前的两个随机采样的过程保证了随机性，所以就算不剪枝，也不会出现over-fitting。

按这种算法得到的随机森林中的每一棵都是很弱的，但是大家组合起来就很厉害了。我觉得可以这样比喻随机森林算法：每一棵决策树就是一个精通于某一个窄领域的专家（因为我们从M个feature中选择m让每一棵决策树进行学习），这样在随机森林中就有了很多个精通不同领域的专家，对一个新的问题（新的输入数据），可以用不同的角度去看待它，最终由各个专家，投票得到结果。

随机森林的过程请参考Mahout的random forest 。这个页面上写的比较清楚了，其中可能不明白的就是Information Gain，可以看看之前推荐过的Moore的页面。

Gradient Boost Decision Tree:

GBDT是一个应用很广泛的算法，可以用来做分类、回归。在很多的数据上都有不错的效果。GBDT这个算法还有一些其他的名字，比如说MART(Multiple Additive Regression Tree)，GBRT(Gradient Boost Regression Tree)，Tree Net等，其实它们都是一个东西（参考自wikipedia – Gradient Boosting)，发明者是Friedman

Gradient Boost其实是一个框架，里面可以套入很多不同的算法，可以参考一下机器学习与数学(3)中的讲解。Boost是"提升"的意思，一般Boosting算法都是一个迭代的过程，每一次新的训练都是为了改进上一次的结果。

原始的Boost算法是在算法开始的时候，为每一个样本赋上一个权重值，初始的时候，大家都是一样重要的。在每一步训练中得到的模型，会使得数据点的估计有对有错，我们就在每一步结束后，增加分错的点的权重，减少分对的点的权重，这样使得某些点如果老是被分错，那么就会被“严重关注”，也就被赋上一个很高的权重。然后等进行了N次迭代（由用户指定），将会得到N个简单的分类器（basic learner），然后我们将它们组合起来（比如说可以对它们进行加权、或者让它们进行投票等），得到一个最终的模型。

而Gradient Boost与传统的Boost的区别是，每一次的计算是为了减少上一次的残差(residual)，而为了消除残差，我们可以在残差减少的梯度**(Gradient)**方向上建立一个新的模型。所以说，在Gradient Boost中，每个新的模型的简历是为了使得之前模型的残差往梯度方向减少，与传统Boost对正确、错误的样本进行加权有着很大的区别。

在分类问题中，有一个很重要的内容叫做Multi-Class Logistic，也就是多分类的Logistic问题，它适用于那些类别数>2的问题，并且在分类结果中，样本x不是一定只属于某一个类可以得到样本x分别属于多个类的概率（也可以说样本x的估计y符合某一个几何分布），这实际上是属于Generalized Linear Model中讨论的内容，这里就先不谈了，以后有机会再用一个专门的章节去做吧。这里就用一个结论：如果一个分类问题符合几何分布，那么就可以用**Logistic**变换来进行之后的运算。

假设对于一个样本x，它可能属于K个分类，其估计值分别为F1(x)...FK(x)，**Logistic**变换如下，logistic变换是一个平滑且将数据规范化（使得向量的长度为

1）的过程，结果为属于类别k的概率pk(x)，

$$p_k(\mathbf{x}) = \exp(F_k(\mathbf{x})) \Big/ \sum_{l=1}^{K} \exp(F_l(\mathbf{x}))$$

对于Logistic变换后的结果，损失函数为：

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

其中，yk为输入的样本数据的估计值，当一个样本x属于类别k时，yk = 1，否则yk = 0。

将Logistic变换的式子带入损失函数，并且对其求导，可以得到损失函数的梯度：

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

上面说的比较抽象，下面举个例子：

假设输入数据x可能属于5个分类（分别为1,2,3,4,5），训练数据中，x属于类别3，则y = (0, 0, 1, 0, 0)，假设模型估计得到的F(x) = (0, 0.3, 0.6, 0, 0)，则经过Logistic变换后的数据p(x) = (0.16,0.21,0.29,0.16,0.16)，y - p得到梯度g：(-0.16, -0.21, 0.71, -0.16, -0.16)。观察这里可以得到一个比较有意思的结论：

假设gk为样本当某一维（某一个分类）上的梯度:

gk>0时，越大表示其在这一维上的概率p(x)越应该提高，比如说上面的第三维的概率为0.29，就应该提高，属于应该往"正确的方向"前进

越小表示这个估计越"准确"

gk<0时，越小，负得越多表示在这一维上的概率应该降低，比如说第二维0.21就应该得到降低。属于应该朝着"错误的反方向"前进

越大，负得越少表示这个估计越"不错误 "

总的来说，对于一个样本，最理想的梯度是越接近**0**的梯度。所以，我们要能够让函数的估计值能够使得梯度往反方向移动（>0的维度上，往负方向移动，<0的维度上，往正方向移动）最终使得梯度尽量=0），并且该算法在会严重关注那些梯度比较大的样本，跟**Boost**的意思类似。

得到梯度之后，就是如何让梯度减少了。这里是用的一个迭代+决策树的方法，当初始化的时候，随便给出一个估计函数F(x)（可以让F(x)是一个随机的值，也可以让F(x)=0），然后之后每迭代一步就根据当前每一个样本的梯度的情况，建立一棵决策树。就让函数往梯度的反方向前进，最终使得迭代N步后，梯度越小。

这里建立的决策树和普通的决策树不太一样，首先，这个决策树是一个叶子节点数J固定的，当生成了J个节点后，就不再生成新的节点了。

算法的流程如下:（参考自treeBoost论文）

Algorithm 6: L_K_TreeBoost

$F_{k0}(\mathbf{x}) = 0, \quad k = 1, K$

For $m = 1$ to M do:

$p_k(\mathbf{x}) = \exp(F_k(\mathbf{x})) / \sum_{l=1}^{K} \exp(F_l(\mathbf{x})), \quad k = 1, K$

For $k = 1$ to K do:

$\tilde{y}_{ik} = y_{ik} - p_k(\mathbf{x}_i), \quad i = 1, N$

$\{R_{jkm}\}_{j=1}^{J} = J$-terminal node $tree(\{\tilde{y}_{ik}, \mathbf{x}_i\}_1^N)$

$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} |\tilde{y}_{ik}| (1 - |\tilde{y}_{ik}|)}, \quad j = 1, J$

$F_{km}(\mathbf{x}) = F_{k,m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jkm} 1(\mathbf{x} \in R_{jkm})$

endFor

endFor

end Algorithm

0 1 2 3 4 5 6 7

0. 表示给定一个初始值

1. 表示建立M棵决策树（迭代M次）

2. 表示对函数估计值F(x)进行Logistic变换

3. 表示对于K个分类进行下面的操作（其实这个for循环也可以理解为向量的操作，每一个样本点xi都对应了K种可能的分类yi，所以yi, F(xi), p(xi)都是一个K维的向量，这样或许容易理解一点）

4. 表示求得残差减少的梯度方向

5. 表示根据每一个样本点x，与其残差减少的梯度方向，得到一棵由J个叶子节点组成的决策树

6. 为当决策树建立完成后，通过这个公式，可以得到每一个叶子节点的增益（这个增益在预测的时候用的）

每个增益的组成其实也是一个K维的向量，表示如果在决策树预测的过程中，如果某一个样本点掉入了这个叶子节点，则其对应的K个分类的值是多少。比如说，GBDT得到了三棵决策树，一个样本点在预测的时候，也会掉入3个叶子节点上，其增益分别为（假设为3分类的问题）：

(0.5, 0.8, 0.1)，(0.2, 0.6, 0.3)，(0.4, 0.3, 0.3)，那么这样最终得到的分类为第二个，因为选择分类2的决策树是最多的。

7. 的意思为，将当前得到的决策树与之前的那些决策树合并起来，作为新的一个模型(跟6中所举的例子差不多)

GBDT的算法大概就讲到这里了，希望能够弥补一下上一篇文章中没有说清楚的部分：）

实现：

看明白了算法，就需要去实现一下，或者看看别人实现的代码，这里推荐一下wikipedia中的gradient boosting页面，下面就有一些开源软件中的一些实现，比如说下面这个：http://elf-project.sourceforge.net/

参考资料：

除了文章中的引用的内容（已经给出了链接）外，主要还是参考Friedman大牛的文章：Greedy function approximation：A Gradient Boosting Machine

标签: 数据挖掘, 人工智能, Boosting, Gradient, GBDT, Model Combining, 随机森林, random forest

LeftNotEasy
关注 - 16
粉丝 - 785
+加关注

« 上一篇：支持中文文本的数据挖掘平台开源项目PyMining发布
» 下一篇：机器学习中的算法(2)-支持向量机(SVM)基础

posted on 2011-03-07 23:53 LeftNotEasy 阅读(74456) 评论(18) 编辑 收藏

评论

#1楼 2011-03-08 08:01 Wuya

很好的文章，顶。

支持(0) 反对(0)

#2楼 2011-03-08 10:00 深蓝医生

我也支持一个！

支持(0) 反对(0)

#3楼 2011-05-04 13:55 yangyangcv

想问下这个GBDT与Adaboost什么关系？是不是可以说成boost思想的两种不同的实现？

支持(0) 反对(0)

#4楼[楼主] 2011-05-04 17:28 LeftNotEasy

@yangyangcv
可以说是两种不同的实现吧，boost其实只是一个非常宽泛的一种思想

支持(0) 反对(0)

#5楼 2011-08-10 17:14 华秋实

你好，我想请问一下，在随机森林一栏中，“进行列采样，从M个feature中，选择m个(m << M)”是每棵树的每个节点都要从M个feature中随机出m个，并从其中选择决策属性吗？就是说每个节点的m个feature都不一样吗？

支持(1) 反对(0)

#6楼[楼主] 2011-08-11 10:13 LeftNotEasy

@华秋实
不是的，是每棵树有m个feature，树与树之间的feature set不一样

支持(0) 反对(0)

#7楼 2011-08-11 10:21 华秋实

不好意思，这块我确实没太搞明白，但是在Leo Breiman的**Random Forests**一论文中，说的是“The simplest random forest with random features is formed by selecting at random, **at each node**, a small group of input variables to split on.”，而且在你推荐的Random Forests算法实现的网页中（https://cwiki.apache.org/MAHOUT/random-forests.html）“How to grow a Random Forest”一栏，说的也是“if there are M input variables, a number m << M is specified such that **at each node,** m variables are selected at random out of the M and the best split on these m is used to split the node.”，请教您~非常感谢！

支持(0) 反对(0)

#8楼[楼主] 2011-08-12 08:53 LeftNotEasy

@华秋实
我建议你可以看看Mahout那篇文章中的伪代码，里面还是说得比较明白的，for j = 1..m这个循环体就是在选择出的m个feature集合中找出一个information gain最大的一个，作为一个节点分裂的依据

支持(0) 反对(0)

#9楼 2011-08-12 10:59 华秋实

你推荐的Mahout那篇文章中的伪代码很详细，非常感谢，不过你可能没有认真看这篇伪代码，“How to grow a Decision Tree”一栏给出的是一个递归程序LearnUnprunedTree(X,Y)，它是针对每一个节点的递归程序，如果你不这么认为那我们继续往下看，在“if...if...else...”中else一开始就“select m variables at random out of the M variables”，然后“For j = 1 .. m”找出nformation gain最大的属性作为决策属性，试想是在每一个节点找决策属性呢，还是在每一棵树找决策属性？继续往下，j*的每个值都作为一个child，再递归上述过程LearnUnprunedTree(Xv,Yv)，于是又会在这个child节点处继续“select m variables at random out of the M variables”，所以我觉得这个伪代码的含义如我前述是每个节点都要从M个属性中随机出m个再择优选择；另外，经同学介绍，在维基百科中（http://en.wikipedia.org/wiki/Random_forest），“Learning algorithm”一栏的第4条说的也比较明白，“For each node of the tree, randomly choose m variables on which to base the decision at that node. Calculate the best split based on these m variables in the training set.”，请指教，感谢不吝时间~~

支持(0) 反对(0)

#10楼[楼主] 2011-08-12 18:18 LeftNotEasy

@华秋实
呵呵，我觉得你说的是正确的，之前确实没有认真的看过这段代码，直接自己这样认为了，之前我也实现过一个rf玩玩，但是发现它在稀疏的数据下退化非常的严重（树很深，接近线性了），导致正确率很低很低，可能就是因为这个原

因导致的。不过是否是这样原因还得靠实验来证明。
按这样说的话，random forest和普通的决策树的区别就很大了，因为之前还看过gbdt的代码，跟普通决策树还是非常接近的。
很感谢你这次提了这个问题，让我对这个关键的地方认识更清楚了。呵呵~

支持(0) 反对(0)

#11楼 2011-08-13 13:59 华秋实

@LeftNotEasy
呵呵，讨论问题共同进步~

支持(0) 反对(0)

#12楼 2011-11-11 18:00 **laomadadupi**

@华秋实
楼主在b公司做learntorank么~

支持(0) 反对(0)

#13楼 2011-11-11 18:00 **laomadadupi**

@LeftNotEasy
楼主在b公司做learntorank么~
看起来很像

支持(0) 反对(0)

#14楼 2013-01-29 10:31 孙队长

值得一看

支持(0) 反对(0)

#15楼 2013-06-05 16:01 **xuanyoumeng**

楼主 有木有介绍关于 系统推荐的文章？

支持(0) 反对(0)

#16楼 2013-08-24 15:36 **DM张朋飞**

哇咔咔

支持(0) 反对(0)

#17楼 2015-10-02 18:29 计算机的潜意识

很不错，赞一个。

支持(0) 反对(0)

#18楼 2015-10-25 15:37 **leo1005**

老大 Mahout的random forest 连接访问不了 求助

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。