研究生算法课课堂笔记

上课日期：2016 年 11 月 7 日　　第(1)节课

组长学号及姓名：1601214496 赵栋杨

组员学号及姓名：1601111273 白宗磊　1601210098 董佶圣

内容概要

本节课主要讲授一个非常实用的机器学习模型——XGBoost 的原理、安装和实例。具体包括如下内容：

1. Ensemble 方法及其分类（肖老师）
2. XGBoost 原理
3. XGBoost 安装
4. XGBoost 应用实例

详细内容

1. Ensemble 方法及其分类

Ensemble 方法，简单来说，就是通过聚合多个单一模型的结果来提高学习效果。主要分为两种方法：

(1). Random Forest（随机森林）

Random Forest 是通过训练多个决策树，生成模型，然后综合利用多个决策树的结果。有两个特点：

一、所有树的训练、预测都是为了同一目标；

二、每棵树是独立的，单独预测（分类任务可理解为投票，回归任务即求平均值）。

以录取研究生投票（分类任务）为例，每个老师决定某个学生是否应该被录取时，受个人的喜好和对学校、地域、性别、性格、成绩等认知的偏差，难免使得录取结果不尽人意。Random Forest 的思想就是把这些老师的投票聚合到一起，得票多者被录取。

(2). Boosting 方法

Boosting 方法是一种提升弱学习算法准确度的方法。主要分为两种：

一、Adaboost 方法（本周四讲）；

二、GBM（Gradient Boosting Machines）方法（如 XGBoost）。

AdaBoost 方法训练一个预测模型的序列，每一个模型对上一模型的结果进行修正：降低正确分类样本的权重，同时提高错分样本的权重，从而弥补上一模型的不足之处，逐步提高分类的准确率。

以录取研究生为例，AdaBoost 的思想是每个老师依次考察来申请的学生，每个老师考察的属性可能并不相同，例如第一个老师主要看本科的学校，如果使用这样的单一标准，容易出现失误，有些好学生可能不被录取，也有些不那么优秀的学生会被录取。为解决这个问题，每个模型看前一模型的分类结果，正确分类的样本权重降低，错分的样本权重增加，继续进行分类。如果有某个样本在前面几轮中始终被分错，它的权重会不断被扩大，最后迫使当前模型越来越集中于那些以前被错分的样本。

2. XGBoost 原理

(1). XGBoost 特性

XGBoost 是基于 tree boosting 方法（决策树相关）的高效、通用、准确的机器学习系统，Github 上有许多在机器学习比赛上拿到冠军或很好名次的实例：https://github.com/dmlc/xgboost/blob/master/demo/README.md#machine-learning-challenge-winning-solutions

XGBoost 主要用于面向多种任务场景，如：

- 分类任务：包括 01 分类，多分类
- 回归任务：用模型拟合数据，做出预测
- 排序任务：搜索引擎网页排序，给定查询条件，返回网页顺序
- 用户自定义任务：见课件中 demo 链接

XGBoost 支持不同的主流编程语言，课上以 Python 为例。比较好用是在单机上写的程序可以直接做单机并行，或者在分布式平台上运行，可扩展性强。目前 XGBoost 已经支持 Hadoop，Spark 等分布式平台，其他平台的支持也正在建设中。

(2) 数学原理

XGBoost 的基本思想是基于第一部分所讲的 Boosting 方法

XGBoost 的数学原理是 Gradient Tree Boosting：

$$y_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = y_i^{(t-1)} + f_t(x_i)$$

记 $\hat{y}_i$ 为训练数据对应的真实值。训练过程为：每轮训练一棵树，$y_i^{(t-1)}$ 为训练 $t-1$ 轮过后得到的模型对数据的预测值，这两个值之间存在残差 $\hat{y}_i - y_i^{(t-1)}$。到第 t 轮（即训练第 t 棵树时），**第 t 棵树把上一轮的残差 $\hat{y}_i - y_i^{(t-1)}$ 作为训练的目标值，尽可能地拟合该残差**，即：

$$f_t(x_i) \to \hat{y}_i - y_i^{(t-1)}$$

以上是一个较为粗浅的理解，XGBoost 在实际上要复杂得多，课上不再赘述。

例 1：$X = \left\{ x_i \mid x_i = \left(x_i^1, x_i^2, \cdots, x_i^n \right) \right\}$ 是一组训练数据，$Y = \{ y_i \}$ 是其目标值。用 XGBoost 训练过程如下：

x_i^1	x_i^2	…	x_i^n	y_i	$f_1(x_i)$	$y_i - y_i^{(1)}$	$f_2(x_i)$	$y_i - y_i^{(2)}$	…
…	…	…	…	54	50	4	3	1	…
…	…	…	…	30	45	-15	-10	-5	…
…	…	…	…	100	102	-2	7	-9	…
…	…	…	…	…	…	…	…	…	…

表 1　XGBoost 训练过程实例

y 对各样本取值分别为 54，30，100……第一棵树拟合目标值，拟合方法与之前的回归树相同，假设三个样本分别拟合出的结果为 50，45，102。第二棵树拟合时，不像随机森林仍然拟合原始目标值，而是拟合第一棵树和真实值之间的残差，也就是 4，-15 和-2。假设

三个样本经过第二棵树的拟合结果分别为 3，-10，7，继续算残差为 1，-5，-9，并继续下一轮拟合。

例 2：Gradient boosting on CART

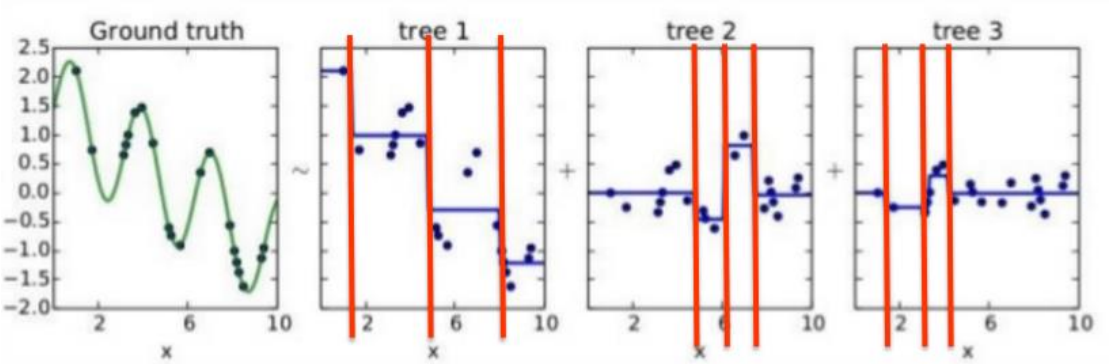

图 1　Gradient boosting on CART

如图，最左侧是真实数据分布 Ground truth，上面的点表示通过采样得到的训练数据；右边三个为训练树，横轴 x 为属性值（图中为单维变量，训练特征只有一维），纵轴 y（蓝线）为目标值，红线中间为决策树的叶结点（图中表示每棵树有四个叶结点）：

第一棵树中，蓝色点的分布和原始分布相同，即直接拟合目标值 y。决策树有四个叶结点，即相当于一个四段的分段函数.

第二棵树中样本点的分布与第一棵树不同，是因为对第二棵树来说，**训练的目标值都调整成了第一棵树的预测值和原始分布的残差**，继续训练，使得四个叶子的决策树尽量拟合目标残差。

如此继续下去，到第三棵树时可以发现残差已经很小了，虽然这些树都很简单（只有四个叶结点），但可以比较好地拟合出 Ground truth。

(3) Loss Function

损失函数（loss function）： 度量模型在给定数据集上数据的预测能力，越小表示模型对数据集上的预测能力越强。

举两个例子（支持的种类很多）：

(1) 平方 loss，用最小二乘法做线性回归就是用的平方损失：

$$L(y,\hat{y})=(y-\hat{y})^2$$

(2) 0-1 互信息 loss，基于之前课上讲的信息熵的方法，经常用于 0-1 分类：

$$L(y,\hat{y})=\hat{y}\log\frac{1}{y}+(1-\hat{y})\log\frac{1}{1-y}$$

XGBoost 的目标是使得训练样本的 loss function 与正则项之和最小，可以形式化的表达如下：

$$\min_{f_1,f_2,\cdots,f_t}\{\sum_{i=1}^{n}L(\sum_{j=1}^{t}f_j(x_i),\hat{y}_i)+\sum_{j=1}^{t}\Omega(f_j)\}$$

其中正则项 $\sum_{j=1}^{t}\Omega(f_j)$ 是为了限制模型的复杂度，避免过拟合。Ω 数值越大表示模型越复杂。

也就是说在 loss function 最小和模型简单之间做一个取舍。

3. XGBoost 安装

支持 Windows、Mac、Linux 操作系统，以 Linux 为例介绍，关于 Mac 和 Windows 的安装见课件后的教程链接。

环境要求：版本号>=4.6 的 g++编译器、代码管理 git、Python

版本号>=4.6 的 g++编译器使用了 c++11 的新版本语言特性，Linux 上可以运行命令：g++ --version 来查看 g++编译器的版本号。XGBoost 可以通过 OpenMP 支持单机并行，像 g++这样的 c++编译器是自动内置支持 OpenMP 的，需要注意 Mac 平台上的 clang 是没有 OpenMP 支持的，但这只牵扯多线程加速的问题，无关紧要。

如果熟悉在 Ubuntu/Debian 上操作：sudo apt-get install g++ python git

安装操作：

- 下载源代码（需开外网） git clone --recursive http://github.com/dmlc/xgboost 注意不能直接下载源码包，XGBoost 引用了 git 子模块的功能，如果单纯下载源代码不能编译，还需要装项目依赖，所以需要添加中间--recursive
- 编译共享库 cd xgboost; make –j4 可以直接输入 make，后面的-j4 代表四线程编译，如果电脑条件允许可以写-j8
- 安装相关 Python 科学计算包 sudo apt-get install python-numpy python-scipy
- 安装到 Python cd python-package; sudo python setup.py install

4. XGBoost 应用实例：蘑菇毒性测试（0-1 分类问题）

(1) 问题背景：根据蘑菇的 22 个离散型外观属性预测蘑菇是否可以食用。如：

1.伞形：钟形=b，锥形=c，凸=x，平面=f，把手=k，凹陷=s

2.伞面：纤维=f，沟槽=g，鳞屑=y，平滑=s

……

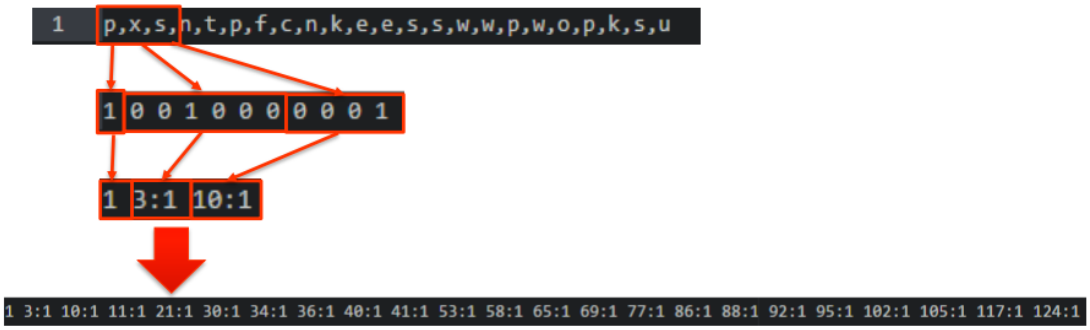

图 2　数据格式及预处理

(2) 数据格式：第 1 列代表蘑菇毒性，p 为有毒，e 为无毒；第 2-23 列代表蘑菇外观属性，例如第二列为 x，代表伞形为凸……

(3) 数据预处理：对于离散型特征，使用 **One Hot Encoding** 将数据转换为稀疏格式，如伞形共 6 个可能取值，编码为 6 维向量，取第几个值时对应位置就为 1，其余均为 0。以第一个特征为例，x 代表取第三种值，则数据处理为 0 0 1 0 0 0。将所有特征的 one-hot encoding 向量拼接为一个大的特征向量（例如图示中的数据拼接为[0 0 1 0 0 0 0 0 0 1 …]）。然后再将其转换为 XGBoost 数据格式，把元素下标和数值组（i：1）构成的键值对作为存储的数据。

(4) 用 XGBoost 进行训练和预测

```
import xgboost as xgb                                    导入模块 xgboost
dtrain = xgb.DMatrix('agaricus.txt.train')
dtest = xgb.DMatrix('agaricus.txt.test')                 DMatrix：xgboost 内部存储训练/测试数据的数据结构
param = {'max_depth': 3,                                 单颗决策树的最大深度
         'eta': 1.0,                                     学习步长/收缩因子，用来防止过拟合，取值范围(0, 1]
         'gamma': 1.0,                                   正则项，叶节点代价因子
         'min_child_weight': 1,                          单个叶节点最小允许的数据个数（权值和）
         'save_period': 0,                               每一轮不保存中间结果
         'booster': 'gbtree',                            模型：gradient boosted tree
         'objective': 'binary:logistic'}                 目标任务：0-1分类；Loss：0-1互信息Loss
num_round = 2                                            训练轮数
watchlist = [(dtest, 'eval'), (dtrain, 'train')]         每轮评测的训练集和测试集
bst = xgb.train(param, dtrain, num_round, watchlist)     训练
preds = bst.predict(dtest)                               预测
write_pred('pred2.txt', preds)                           输出预测结果
bst.dump_model('dump2.nice.txt', 'featmap.txt')          输出模型
```

图 3　用 XGBoost 进行训练和预测

代码解释：

param：Python 的字典数据结构，存储需要的变量。

eta：预测结果为 t 棵树结果的和，每棵树加入收缩因子 $\eta \in (0,1]$ 控制收缩力度。例如取 0.5 时，代表每棵树收缩到原来的一半，这样可以为下一棵树的训练留出空间，树的数目增多训练结果可能更好。如果不愿意用这个特性，直接取 eta=1，这样就不会收缩。

gamma：正则项，指叶节点代价因子，例如 $\gamma = 1$，那么每多分裂出一片叶子，就给整个模型加上 1 的惩罚。

min-child-weight：每个叶子中会存储一个平均值或其他值作为以后走到此叶子时返回的预测值，该参数表示，当分裂出一片新的叶子时，至少要有几条数据或样本走到这片叶子中，才会继续分裂出新的叶子，为了防止特殊数据甚至采样出现问题数据的异常情况而单独占用一片叶子。

save-period：取值为 0 或 1，表示是否保存每轮新训练树的结果，0 代表不保存，1 代表保存。

watchlist =[(dtest, ‘eval’),(dtrain, ‘train’)]

输出时，可以看到每轮模型在 test 和 train 上的表现结果

bst.dump_model(‘dump2.nice.txt’, ‘featmap.txt’)　输出模型

```
#!/bin/bash
# map feature using indicator encoding, also produce featmap.txt
python mapfeat.py
# split train and test
python mknfold.py agaricus.txt 1
# use xgboost to train, predict & dump model
python runxgb.py
```

图 4

代码解释：

dump2.nice.txt 打印树

featmap.txt　每个节点特征的名称（图 6，其中判断语句后面的字母代表类型，‘i’：布尔型判断；‘q’：浮点数大小判断；‘int’：整数大小判断）

mapfeat.py　数据处理的过程

mknfold.py　把数据分成训练数据和预测数据的过程

runxgb.py　XGBoost 训练测试

```
booster[0]:
0:[odor=pungent] yes=2,no=1
  1:[stalk-root=cup] yes=4,no=3
    3:[stalk-root=missing] yes=8,no=7
      7:leaf=1.90175
      8:leaf=-1.95062
    4:[bruises?=no] yes=10,no=9
      9:leaf=1.77778
      10:leaf=-1.98104
  2:[spore-print-color=orange] yes=6,no=5
    5:[stalk-surface-below-ring=silky] yes=12,no=11
      11:leaf=-1.98531
      12:leaf=0.808511
    6:leaf=1.85965
```

图 5　XGBoost 最终生成的决策树

```
0   cap-shape=bell   i
1   cap-shape=conical   i
2   cap-shape=convex   i
3   cap-shape=flat   i
4   cap-shape=knobbed   i
5   cap-shape=sunken   i
6   cap-surface=fibrous i
```

图 6　featmap.txt

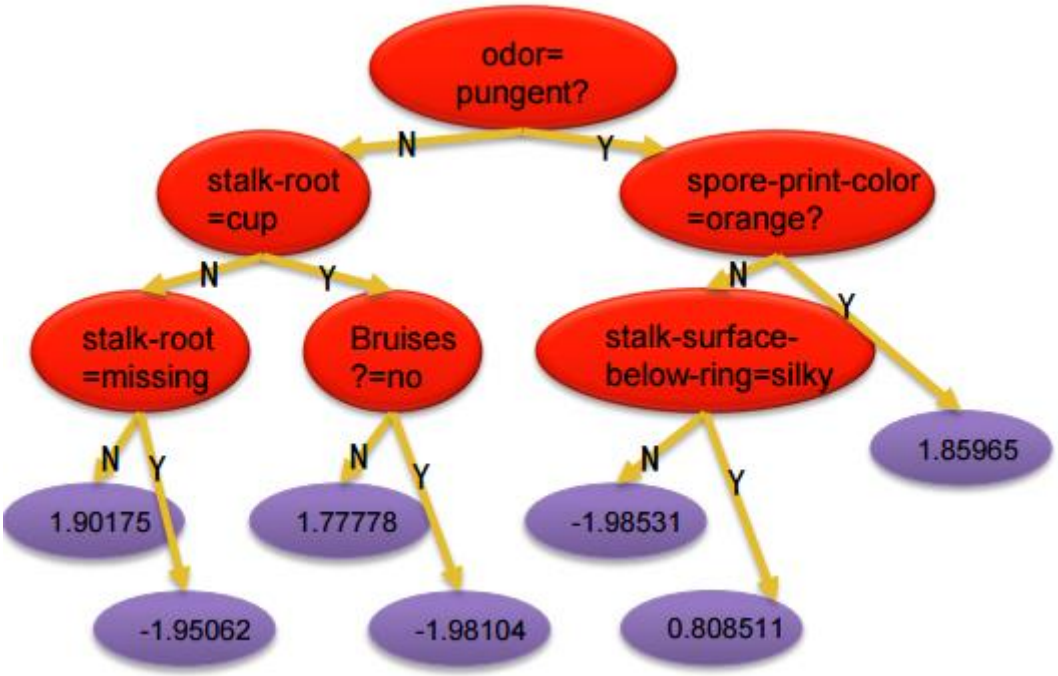

图 7　XGBoost 最终生成的决策树

结果分析：

图 7 中的树是图 5 中 XGBoost 最终生成的决策树的形象表示，红色节点代表内部节点，紫色节点代表外部节点。叶节点存储的值不是概率，而是 $P(y=1)=\frac{1}{1+e^{-z}}$ 中 z 的值。将 z 的值代入上式，即可得到该叶节点对应的概率。

在 XGBoost 的实际训练过程中，每一轮训练使用的 Loss Function 为：

$$L(z,\hat{y})=-\{\hat{y}=1\}\log\left(\frac{1}{1+e^{-z}}\right)-\{\hat{y}=0\}\log\left(1-\frac{1}{1+e^{-z}}\right)$$

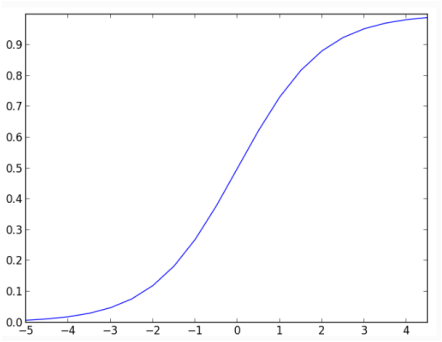

图 8

如图 8 所示，函数 $P(y=1)=\frac{1}{1+e^{-z}}$ 将取值范围为 $(-\infty,+\infty)$ 的 z 转化到 $(0,1)$ 之间，当 $z=0$ 时取值为 0.5，由于在 $z=0$ 时函数斜率较大，所以当叶节点的取值接近 ± 2 时，已经可以对应 70~80%的预测准确率了。